THE UNIVERSITY *of* EDINBURGH

Low-resource Learning in Complex Games

Mihai Sorin Dobre

Doctor of Philosophy

Institute for Language, Cognition and Computation

School of Informatics

University of Edinburgh

2018

Abstract

This project is concerned with learning to take decisions in complex domains, in games in particular. Previous work assumes that massive data resources are available for training, but aside from a few very popular games, this is generally not the case, and the state of the art in such circumstances is to rely extensively on hand-crafted heuristics. On the other hand, human players are able to quickly learn from only a handful of examples, exploiting specific characteristics of the learning problem to accelerate their learning process. Designing algorithms that function in a similar way is an open area of research and has many applications in today's complex decision problems.

One solution presented in this work is design learning algorithms that exploit the inherent structure of the game. Specifically, we take into account how the action space can be clustered into sets called *types* and exploit this characteristic to improve planning at decision time. Action types can also be leveraged to extract high-level strategies from a sparse corpus of human play, and this generates more realistic trajectories during planning, further improving performance.

Another approach that proved successful is using an accurate model of the environment to reduce the complexity of the learning problem. Similar to how human players have an internal model of the world that allows them to focus on the relevant parts of the problem, we decouple learning to win from learning the rules of the game, thereby making supervised learning more data efficient.

Finally, in order to handle partial observability that is usually encountered in complex games, we propose an extension to Monte Carlo Tree Search that plans in the Belief Markov Decision Process. We found that this algorithm doesn't outperform the state of the art models on our chosen domain. Our error analysis indicates that the method struggles to handle the high uncertainty of the conditions required for the game to end. Furthermore, our relaxed belief model can cause rollouts in the belief space to be inaccurate, especially in complex games.

We assess the proposed methods in an agent playing the highly complex board game Settlers of Catan. Building on previous research, our strongest agent combines planning at decision time with prior knowledge extracted from an available corpus of general human play; but unlike this prior work, our human corpus consists of only 60 games, as opposed to many thousands. Our agent defeats the current state of the art agent by a large margin, showing that the proposed modifications aid in exploiting general human play in highly complex games.

Lay Summary

The most prominent successes in learning to play complex games, such as Go or Poker, rely on the availability of massive data resources where the number of examples are in the order of millions. However, human players are able to quickly learn from only a handful of examples by exploiting certain characteristics of the learning problem. Similarly, young children learn to separate objects in certain categories given the objects attributes and previous research has shown that such categorisation could aid in reducing the cognitive effort. Furthermore, we apply our previous knowledge about the world when we learn new skills, for example we use our knowledge of gravity when learning certain basketball shots.

In this thesis, we design our algorithms to function in a similar way such that these can learn from a handful of examples. Our first solution is to exploit the structure of the game during learning and planning. For this purpose, we take advantage of how the rules of the game naturally carve up a structure in complex domains. Another successful approach is to design an accurate model of the environment that can be used by the learning algorithms to reduce the complexity of the problem. Such a model allows us to decouple learning to win from learning the rules of the game, thereby making learning more data efficient. Finally, we extend our algorithms to handle what is known as partial observability in games, i.e. the players cannot be certain of the complete description of the current game state. As before, the games structure permits creating efficient methods that focus only on specific attributes when reasoning over what is known as the agents belief of the game state.

We assess the proposed methods in an agent playing the highly complex board game Settlers of Catan. Building on previous research, our strongest agent combines planning with knowledge extracted from a set of example human play; but unlike this prior work, our dataset consists of only 60 games. Our agent defeats the current state of the art agent by a large margin, showing that the proposed modifications aid in exploiting general human play in highly complex games.

Acknowledgements

First and foremost, I would like to thank my primary supervisor, Alex Lascarides, for her continuous support and guidance, for the many insightful discussions on my topic and life of a researcher, and for her pure honesty which helped me develop my own critical thinking. I can't even imagine what my Ph.D. would look like without her. I would also like to thank my second supervisor, Subramanian Ramamoorthy. His comments and suggestions helped me explore the field and develop my own ideas.

I wish to mention the people I have met in the department for the enjoyable moments spent together. Thank you to Sam, Philip, Valentin and many others. Special thanks go to Federico and Daniel, with who I shared many experiences and typical Ph.D. frustrations. I wish to also express my appreciation for the people in my office, especially to Craig and Ondrej, with who I had many technical discussions that piqued my curiosity for other branches of Machine Learning.

A special mention goes to the team I had the chance to work with on the Alexa challenge: Federico, Daniel, Marco, Joachim, Ben and Manny. It was one of the few collaboration experiences during my Ph.D. and I learnt so much from each of you.

On a more personal note, I'd like to thank my parents for their faith in me and for always being there for me. Their teachings to never give up have been essential in withstanding the obstacles of a Ph.D.. I am also very grateful to my older brother, Alex, for sharing his opinions and nurturing my way of thinking. The countless enlightening books he suggested took my mind off my concerns so I can see the bigger picture.

Finally, the most important acknowledgement goes to my dear wife, Simona. I thank you for encouraging me to take on this challenge, and I am also grateful for your care, love and devotion throughout these years. Your infectious and sometimes inexplicable optimism has kept me going.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mihai Sorin Dobre*)

Table of Contents

1 Introduction **1**

1.1 Motivation . . . 2

1.2 Hypotheses . . . 5

1.3 Contribution . . . 9

1.3.1 Code . . . 11

1.3.2 Limitations . . . 11

1.3.3 Publications . . . 12

1.4 Summary of Thesis . . . 13

2 Background **15**

2.1 Game Theory . . . 15

2.2 Multiagent Systems . . . 17

2.2.1 Opponent Modelling . . . 18

2.3 Reinforcement Learning . . . 19

2.3.1 Markov Decision Process . . . 20

2.3.2 Partially-observable Markov Decision Process . . . 20

2.3.3 Function Approximations . . . 21

2.3.4 Planning at Decision Time . . . 23

2.4 Mining Human Knowledge . . . 28

2.4.1 Imitation Learning . . . 28

2.4.2 Learning from Demonstration . . . 29

2.4.3 Human Intervention . . . 30

2.4.4 Heuristics . . . 31

2.5 Conclusion . . . 32

3 Resources and Domain Analysis **33**

3.1 Settlers of Catan . . . 33

3.2 Game Analysis and Motivation 35
3.2.1 Structure in Complex Games 38
3.2.2 Game Modifications 40
3.3 Software 42
3.4 The human corpus 43
3.5 Experimental methodology 45

4 Simple Non-parametric Method for Mining the Corpus 49
4.1 Non-parametric Method for Value Estimation 49
4.1.1 Combining with Flat Monte Carlo Tree Search 53
4.1.2 Results and Analysis 54
4.2 Conclusion 57

5 Monte Carlo Tree Search in Observable Settlers of Catan 59
5.1 Monte Carlo Tree Search 60
5.2 Parallel Monte Carlo Tree Search 65
5.2.1 General Performance 66
5.2.2 Fine-tuning other Parameters 67
5.3 Afterstates 68
5.4 Exploiting Domain Structure 70
5.4.1 Action Types 73
5.4.2 Extending the Rollout Policy 75
5.4.3 Learning a Type Distribution from Human Games 77
5.4.4 Empirical Evaluation against the State of the Art 83
5.5 Trading and Negotiation 83
5.5.1 Results 85
5.6 Conclusion 87

6 Monte Carlo Tree Search in Partially-Observable Settlers of Catan 89
6.1 Introduction 89
6.2 Tracking Belief in Settlers of Catan 92
6.3 The Imperfect Information Game Model 93
6.4 Partially Observable Monte Carlo Planning 95
6.4.1 Extensions 96
6.5 Information Set Monte Carlo Tree Search 99
6.6 Belief Monte Carlo Tree Search 103

6.7 Results . . . 106
6.7.1 Rollouts with the Human Type Distribution . . . 109
6.7.2 Experiments with Action Legality Probability . . . 111
6.7.3 Experiments with Belief Chance Nodes . . . 114
6.8 Error Analysis . . . 115
6.9 Conclusions . . . 118

7 Supervised Learning of Policies from Game Play 121
7.1 Datasets . . . 123
7.2 Representation . . . 124
7.3 Frequency Baseline . . . 125
7.4 Single Model . . . 126
7.4.1 Evaluation on Synthetic Dataset . . . 131
7.4.2 Logistic Regression Baseline . . . 131
7.5 Mixture of Experts Model . . . 132
7.5.1 Evaluation on Synthetic Dataset . . . 134
7.5.2 Evaluation on Human Dataset . . . 135
7.6 Introducing Knowledge via Label Smoothing . . . 138
7.7 Training an Opponent Policy for the Partially Observable Game . . . 139
7.8 Transfer Learning between Experts via Pre-training . . . 141
7.8.1 Evaluation on Human Dataset . . . 143
7.9 Conclusion . . . 144

8 Combining Extracted Policies with Monte Carlo Tree Search 147
8.1 Algorithms for Informing the Search . . . 148
8.2 Implementation Details . . . 150
8.3 Observable Version of Settlers of Catan . . . 151
8.3.1 Seeding with Neural Networks Trained on the Synthetic Data 153
8.3.2 Seeding with Neural Networks Trained on the Human Data . . 154
8.3.3 Comparison between Datasets . . . 155
8.3.4 Seeding with the Action Type Distribution . . . 156
8.4 Partially-observable Version of Settlers of Catan . . . 160
8.4.1 Seeding with Models Trained on Human Dataset . . . 161
8.5 Final Agent . . . 164
8.6 Conclusions . . . 167

9 Conclusion and Future Work **169**
9.1 Main Findings 169
9.2 Future Work 172
9.2.1 Human Evaluation 174
9.2.2 Hierarchical Approaches 175
9.2.3 Multi-agent Perspective 176
9.2.4 Reinforcement Learning 177

A Settlers of Catan Action Space **179**

B Features for the Non-parametric Method **181**

C Monte Carlo Tree Search Parameter Tuning **183**

D Factored Belief Model **187**

E Settlers of Catan Partially-observable Game Model **191**

F Abstract Belief Representation **195**

G Neural Network Input Features **197**

Bibliography **201**

Chapter 1

Introduction

This thesis is concerned with learning to play highly complex games. Exact methods are not feasible due to the size of state and action space. Online approaches address this by sampling the decision space to perform approximate inference about optimal strategies. Even in this case it is well known that time forbids sufficient search of the space, and that random sampling is not informative enough.

Prior knowledge of the problem can suggest which parts of the game tree are likely to be lucrative to explore. Unfortunately, this is generally very difficult to design and evaluate manually, because specifying heuristics that determine the lucrative moves accurately for all possible states places a large burden on the developer. Even so, there is strong evidence that one can learn directly from human play (Silver et al., 2016). In spite of their cognitive bounds, human players learn from their experience and reach expert level. Learning from expert play has proven useful in modelling Go which was until recently considered one of the biggest challenges in the field. However, there are at least two shortcomings to this prior work. First, Go presents only some features that are characteristic to very complex domains. Secondly, they rely on mining tens of millions of games of human play to estimate the lucrative portions of the game tree, and these extremely high data requirements are not typically available for other games. The aim of this thesis is two fold: to demonstrate that learning from human play is beneficial in even more complex environments; and to show that even for more complex games than Go, the agent can usefully exploit very small amounts of data on human play—of the order of 50 games rather than millions.

In this work we show how previously developed algorithms can be modified and adapted in order to scale to such challenging learning problems. Finally, previous research assumes that one has access to almost unlimited data and computational re-

sources. The strongest methods presented here run on standard CPU machines, but could also be easily extended if more computing power is available.

1.1 Motivation

Just as children and adults use games to understand the world and develop their skills, board games and computer games are perfect tools for developing and evaluating autonomous agents. Games provide an ideal research test-bed (Schaeffer and van den Herik, 2002; Sawyer and Rejeski, 2002) as they model various problems concerning optimal decision making that are encountered in real life; e.g. commercial transactions or where to go on holiday. Some of these also model human behaviour in specific scenarios that relate to many daily activities (e.g. logistics, planning or negotiations). Therefore, many games (e.g. Chess, Backgammon, Poker and Go) have been used as benchmarks for Artificial Intelligence performance. Another advantage is that these games can easily be simulated with high confidence, a characteristic that permits evaluating the learning algorithms in a controlled fashion.

Just as children improve and develop new abilities while encountering increasingly challenging problems, we must attempt to solve increasingly complex games in order to push the boundaries of Artificial Intelligence. There are many ways in which a game can be complex, however the most common way to define complexity is in terms of the size of the game, e.g. the large number of possible situations that one may encounter while playing it. This aspect also increases the number of ways one can satisfy the objective of the game, so an algorithm has to evaluate a large number of strategies in order to find the best one. But there are other aspects of a game that make it challenging such as the uncertainty caused by certain aspects of the game being hidden. This uncertainty increases the number of possibilities, but also raises other challenging questions such as how can we create information seeking algorithms that perform optimal exploration? Another characteristic that increases the complexity of a game is the number of participating players. Again, this results in an increase in the size of the game in terms of number of possible situations. However, the possible interactions between players poses other (social) questions, e.g. how can a player form coalitions that would be beneficial in the long run? Despite this, the comunity has focused its attention on the effect of complexity in terms of how it increases the size these games. Furthermore, it has addressed this issue with mostly brute force solutions with the objective to create *tabula rasa* methods.

Following this brief description of complexity, we can observe that games like Chess, Poker and Go are all complex environments, providing some interesting decision problems that are similar to those in real life. However, they each have distinct interesting characteristics. For example, Backgammon displays a large degree of non-determinism given the many dice rolls, Poker contains elements of imperfect information, Go has a very large branching factor due to the board size, while each of the pieces in Chess has a different capability (and importance) resulting in different available strategies depending on the current state. Aside from the fact that the algorithms are developed for one of these specific decision problems and may not scale well to others, many real-world decision problems display all these characteristics together.

The environment we have chosen to evaluate our models is the board game *Settlers of Catan* since it displays all the complexity characteristics encountered in the games presented above. The game is a popular multi-player board game, where players race to achieve the victory conditions, i.e. 10 victory points. These points are achieved through a variety of actions (e.g. building pieces on the board or buying development cards that offer certain benefits). The requirements for these actions and the sparsity of the available resources force players to interact with each other. Players must race to be the first to access the available resources, but they must also exchange resources with the other players if certain resources are not available. As a result, they must find a tradeoff between how aggressive they are in their play on the board versus how much they are willing to pay for this agressiveness when they need the scarce resources. Players keep their hands hidden making negotiations even more challenging. One exchange could decide the course of the game, especially close to the end of the game. The board is modular and the initial configuration is randomised before the start of the game, so designing heuristics to address these tradeoffs in every possible configuration is not feasible.

We provide a more detailed description of the game rules in Chapter 3. For now, we show a comparison to Go in Table 1.1. Go has been extensively used as a benchmark for learning agents (e.g. Silver et al., 2016). The figures in the table on Settlers of Catan are determined by the game rules, except for the depth and branching factors which are estimated by running games with the rule-based agent. We did not use the human corpus since a large number of games contained fewer players than the maximum of 4. The table shows that Settlers of Catan presents a much more challenging problem than Go. The size of the state and action space is much larger. It also contains other challenging elements such as imperfect information and chance events (i.e. it is

stochastic). Its tree does have a reduced branching factor, but the depth is almost two orders of magnitude greater (11715 average depth) if random sampling is used. The reduced branching is caused by a minor modification that we have introduced to limit the trading possibilities (making the action space finite in the process). This modification is required to evaluate our agents against the current baselines, which also do not handle the complete game. Learning the unrestricted game is an interesting path to explore in future work and may require devising methods that incorporate life-long learning.[1]

Of course, there are other characteristics that Settlers of Catan does not feature such as simultaneous actions, real-time play and continuous actions. However, solving tasks with these features, while a highly interesting and important challenge, is beyond the scope of this work. Even though Settlers of Catan does not display all characteristics found in daily life, it is a major step forward from the environments previously explored.

Property	GO	Settlers of Catan
Incomplete Information	Yes	Yes
Stochastic	No	Yes
Partially Observable Moves	No	Yes
Imperfect Information	No	Yes
Avg. depth	250	250 (11715 random π)
Branching factor	250	65
Initial state space	1	$\approx 1.2 * 10^{15}$
Action Space	361	1882[2]
Types of actions	2	8
Number of players	2	4
Number of actions per turn	1	≥ 2
Turn-based	Yes	Yes (mostly)

Table 1.1: Comparison of Go and Settlers of Catan. The depth of SoC was computed by simulating games with the heuristics agents or via random sampling.

[1] Negotiations in a corpus of human play show that people make complex offers, involving specific promises on the contingency of specific possible future states, that may not be within anyone's deliberations about possible moves in the game—a case of having to adapt to unforeseen possibilities.

[2] The action space is calculated as shown in Appendix A given the minor game modifications presented later in Section 3.2.2.

1.2 Hypotheses

We have highlighted how learning to play Settlers of Catan is a very challenging task. This document presents a series of improvements brought to a learning agent which achieves state of the art performance in this difficult environment. These improvements are based on the following two core ideas:

Emergent Structure Hypothesis (ESH): Highly complex games have a clear structure that can be exploited to aid the learning process;

Model-based Abstraction Hypothesis (MAH): One can use a model of the environment to reduce the complexity and improve the performance of the learning agent.

Certain traits of complex games cause a structure to emerge. For example, a large branching factor causes differences and similarities to be noticeable between the available actions so these can easily by clustered into what we refer to as *types*. In fact, game developers make use of these types to communicate the rules of the game more concisely in the game's manual. Settlers of Catan has 4 times more action types than Go (see Table 1.1). An action *type* is the trade action, while an action *token* of this type is the description of the exchange, e.g. wood for ore. This categorisation aids the algorithm generalise to certain situations that are similar. Furthermore, the game rules enforce a legality over both the set of types and their tokens (e.g. a player cannot give away wood without owning any). This legality reduces the space of possibilities but also forces the players to choose from the less valuable moves that are available. In this thesis, we aim to increase the generalisable capability of learning methods by taking both of these characteristics into account.

The defined structure highlights an additional issue: certain action types are more common than others. In most turn-based games for example, there is a single way one could end their turn, while there could be multiple methods to build or move around the board before you end it. Consequently, if learning involves exploring the space by sampling from action tokens as opposed to action types, then the learning problem is inherently skewed towards these common types. For instance, an agent is in danger of never learning that ending the turn is (probably) optimal, because ending the turn is only 1 of 600 possible actions and the agent never happens to sample the consequences of this move! One should avoid this skewness in situations where the optimal behaviour is either a combination of several action types or it does not contain

the common types.

The rules of the game also naturally carve up the game into a set of phases that may require different strategies to solve. For example, Settlers of Catan has an initial set-up phase where players build their first two settlements. These phases can be clustered following the categorisation into types and the action type legality, since only a portion of the action types can be executed in a particular phase. In addition to these clusters, large games are either very long or have a large average branching factor, or both. In long games, the optimal strategy at the beginning may dramatically differ to the one closer to the end of the game. For example, players are more likely to collaborate with their opponents in the early stages of the game, when such collaboration would aid both parties without any immediate adverse consequences. Later in the game, aiding an opponent that is close to winning is very risky. The game's inventor encourages the players to use these definitions to learn the game because there are major benefits in being aware of them. Previous work devised rules to account for this game trait, e.g. the heuristics-based agent that plays Settlers of Catan, developed by Thomas (2004), does not trade with opponents that are 2 victory points away from ending the game.

Enforcing strict rules may result in sub-optimal behaviour since the agent is not allowed to explore the entire space of possibilities. Instead, we design our algorithms such that the resulting agent is aware of this structure. In particular, we employ the "divide and conquer" principle when designing our learning agent which permits specialising parts of the algorithm to certain phases. The presentation of these structures in the game rules encourages the players to employ this principle. Disregarding the phases of the game is similarly problematic to ignoring that certain action types are more common, since players spend a large portion of the game in certain phases.

Furthermore, this emergent structure of the domain could be useful for abstraction purposes. Human decision makers are limited due to their available information, the difficulty of the problem and their cognitive bounds in terms of both time and memory (Gigerenzer and Selten, 2001). Despite this, some human players become experts and are able to take better decisions than novice human players that lack knowledge of the game (although expert players can execute apparent bad moves which later in the game will prove decisive). To overcome their limitations, humans most likely exploit known structural regularities in the environment (Gigerenzer and Selten, 2001) and so are able to generalise from previous experience. Rosch (1978) concludes that there is a high correlational structure in the world, and that humans utilise this to define categories in order to reduce the cognitive effort. Children learn these categories during early stages

of development which suggests that categorisation may have an impact on learning to interact with the world. One way to generate these categories is based on their common attributes. Similar to how the large number of actions results in a structure that helps to cluster them into types, the large dimensionality of the game state description and the large number of possible states one may encounter encourages abstracting over the complete description of a game state. There is only a small set of features that are relevant to the task at hand and that repeat across all possible states. Keeping these features while dropping the additional information can dramatically reduce the space a learning algorithm needs to search for the optimal strategy.

All these aspects described by the ESH can be observed in many other complex games, such as Monopoly, Civilisation, Battlerstar Galactica etc. There is a clear categorisation of the actions, e.g. Civilisation has build type actions or move units type actions. Some of the available actions are more common then others, e.g. the number of negotiation actions one can make during bargaining in Monopoly is very high compared to the number of actions for building houses on the owned properties. Finally, the rules present a clear separation of the game phases, e.g. there are hidden cylons among the human players that are revealed after the first part of the Battlestart Galactica game. Given this large number of similarities, the "divide and conquer" principle and the hierarchical planning approach could also be applied in learning to play in many other board games that present similar challenges to Settlers of Catan.

The second idea (MAH) is inspired by model-based reinforcement learning (Sutton and Barto, 1998), which in turn is inspired by psychology (Spelke et al., 1992; Hegarty, 2004; Gläscher et al., 2010). The human brain creates an internal model of the world that is a good approximation of the physical properties of the real word, like gravity. We develop this model during the early periods of our childhood while honing our motor skills and handling objects. We later use this model to make inference about new situations. For example, if we observe an object falling when there is nothing to support it, we can infer that other objects that are not supported would also fall. Generalising that there is always a force that attracts the objects towards the earth allows us to focus on other properties of the object that are relevant and may influence the fall. We learn that a heavier object can reach a higher total speed, while a paper plane may form a circular motion due to the increased air friction given its wings. In turn, knowing these properties allows us to quickly learn new skills as well as transfer them to slightly different situations, e.g. throwing a basketball through a hoop versus throwing a tennis ball. Our internal model of the world allows us to focus on developing new skills

without the need to relearn the physics of the world. Similarly, model-based learning in games allows us to decouple learning the game rules from learning to win. Just as a human expert player doesn't relearn what actions are legal before making decisions, our offline learning models only consider legal actions when evaluating which is the optimal one. Such an approach permits defining simpler models that can generalise better in the low-resource scenario as we show in Chapter 7.[3]

Another well-known benefit of the internal model is that one can exploit it to simulate what might happen without *experiencing* it. For example, human adults can predict that a glass on the edge of the table is likely to fall and shatter without actually trying it. Similarly, a model of the game allows the player to evaluate which of the possible playing sequences are more likely to end up in a win or a loss, without performing the required exploration in the real game. In addition, model-based planning at decision time exploits this benefit to quickly estimate the optimal play in a given situation. This is particularly useful in complex games, where learning a general good strategy is very challenging if not impossible. Despite the game states being very similar, there are large differences between strategies that are optimal in the different phases of the game. Furthermore, one cannot be certain that the learned policy performs well in every possible situation due to the sheer size of such games. Building on the previous success with model-based planning at decision time (Szita et al., 2010; Silver et al., 2016), we believe such an approach is needed in our chosen domain and in the domains previously enumerated.

However, simulated play is more useful the more accurate its predictions about outcomes are and these benefit from strong and accurate guidance of a good prior model. Following on MAH, existing work in games utilise an exact model of the environment. The game rules provide an additional knowledge about legal and illegal moves (ESH) which can be used to extend the model of the environment and further guide the learning algorithm. Such an approach is especially useful for efficient planning and permits planning to be performed online during game play. While an exact model is beneficial, we believe it is not sufficient for complex games with very sparse reward functions, where pure random walks take too long to encounter states with high rewards and are very likely to fall prey to cycles (i.e. encountering the same state multiple times during simulations). Some form of opponent modelling is required and, taking advantage of the ESH, we can perform abstractions followed by extracting high-level preferences

[3] Our work is also different to that on the Atari environment (Mnih et al., 2015), where the agent is allowed to try illegal actions and learn that these are not modifying the game state. Therefore, the goal there was also to also learn the game rules in addition to learning to play well.

from a corpus of general play. A basic opponent model made of general preferences is useful in simulating more realistic play and avoid the pitfalls of complex environments.

Finally, previous work on Settlers of Catan rely on complicated heuristics created using expert human knowledge to either develop the full agent or to inform parts of the learning process (Thomas, 2004; Pfeiffer, 2003; Szita et al., 2010). But, other previous research has shown that learning directly from human examples can be very fruitful: inverse reinforcement learning (Ng and Russell, 2000), apprenticeship learning (Abbeel and Ng, 2004), learning by demonstration (Argall et al., 2009), learning to predict expert play via supervised learning (Clark and Storkey, 2015; Maddison et al., 2015), active learning (Settles, 2009) or human computation (Quinn and Bederson, 2011). Most of this work employed only expert demonstrations and safely assumed the play is optimal. But learning from humans solving highly complex tasks presents particular challenges for AI agents, because the corpus of human choices will inevitably contain a fair portion of suboptimal play. Nevertheless, as long as decent human moves can be separated from poor ones, valuable information can be extracted from human play. Based on this intuition, the hypothesis that inspired the work on this thesis is that learning from human data can be beneficial even if the amount and the quality of the data available is considerably lower than that of the datasets previously used.

1.3 Contribution

The aim of this project is to present new techniques for deciding on optimal actions in complex games in extensive form, when available resources, both computational and data wise, are limited. In line with prior work on learning strategies for complex agents (Branavan et al., 2012; Silver et al., 2016), we deploy algorithms that support planning at decision time. In particular, we adapt the learning model based on Monte Carlo Tree Search (MCTS) so that it reasons simultaneously over the data generated via simulated play and over suggestions provided by a policy extracted from the human corpus data during an offline phase. We show that this combination can be applied to even more complex environments if the two parts are designed based on the two hypotheses, ESH and MAH, presented earlier.

First, we present the large benefits that can be gained by taking into account the structure of the game during the sampling phase of MCTS. Similar to hierarchical methods, our approach samples the type space first followed by sampling the specific description of the chosen action type. This simple modification is the most important

modification required for extending our agent to take decisions over the most complicated aspect of the game: trading. By parallelising MCTS and incorporating several well-known methods for sharing statistics, the agent is able to defeat the current state of the art models in the fully-observable version of the game. We then extract from our corpus a probability distribution over action types and condition it on type legality. Sampling the action types during planning based on this distribution, which only represents the conditional preferences of a standard player, boosts the performance of the agent significantly.

Second, supervised learning can only be applied to such a complex problem if it learns over an abstracted version of the game and ignores the illegal parts of the action space. These techniques reduce the hypothesis space sufficiently to allow training on a much smaller and noisier dataset of example play than previously tackled. Our evaluation indicates that using a model of the environment to batch the input and design the output layer increases the performance significantly. In addition, we observed that training models on a dataset of mixed play is beneficial as this approach has an inherent regularisation effect. Training on a dataset generated by a deterministic synthetic agent resulted in overfitting to the trajectories the agent is more likely to explore, and the learned policy was not useful when combined with planning.

We also evaluate the agents in a tournament setting where we pitch each of our proposed models against several baselines and against each other. The results illustrate that, despite the difficulty of the learning problem, it is useful to combine planning with knowledge of what humans did in similar states. Such a model outperforms any model that uses only one of these information sources. Combining MCTS with a Neural Network, that was trained in a supervised fashion on the human corpus to predict what a human would do in the current game, performed well. But, our best agent is the one that uses Maximum Likelihood Estimation on the human corpus to estimate only the action *type* (in contrast to the neural network which was designed to estimate the action token) that a human would choose given the available types. This result indicates that simpler approaches or methods that reason over an abstracted version of the decision space are more appropriate in a low-resource setting.

Finally, we implemented several extensions to MCTS that can handle imperfect information. We propose combining MCTS with a factored belief model such that our agent can plan in the Belief Markov Decision Process. The exact method has proven weak, while an approximation did not outperform the existing sampling based implementations (Silver and Veness, 2010). We provide a detailed analysis that shows

our approach struggles to handle the uncertainty of key information that is required for deciding when the game ends during rollouts. We also observed that our relaxed belief model, even though it is very accurate, reduces the effectiveness of the rollouts performed by MCTS. Nonetheless, all agents that reason over imperfect information defeat the current open-source state of the art agent in our evaluation environment and could potentially be a challenging opponent for human expert players.

1.3.1 Code

The code for running the experiments presented in this thesis is available as part of 3 Java projects:

- MCTS: https://github.com/sorinMD/MCTS. This project contains the code for the planning agents presented in Chapter 5 and 6.
- DeepCatan: https://github.com/sorinMD/deepCatan. This project contains the code for training and evaluating the Neural Networks developed in Chapter 7.
- StacSettlers: https://github.com/sorinMD/StacSettlers. This project contains the modified JSettlers framework developed by the STAC project STAC (2018) and all the other agents used as baselines in game simulations.

1.3.2 Limitations

The aim was not to develop the world's best player for Settlers of Catan. As a result, the final agent has not been extensively optimised, either for performance or for efficiency. The focus of this research is only to evaluate the hypotheses described above. Therefore, we have extensively explored the parameter space in each ablation study to confirm the benefits of each method presented in this work, but we have not explored all possible parameter combinations for the final agent presented in Section 8.5. There are methods that can help further optimise the parameters of the planning method and of the offline learning method presented in this thesis (e.g. Coulom (2011); Snoek et al. (2012)). Due to resources constraints, such optimisations are left for future work and are very likely to further increase the performance.

On the same note, the presented method was not developed from a multi-agent perspective, despite the clear characteristic of the chosen environment that makes it highly likely that reasoning about the player type(s) of the opponents would further improve

performance on the game. Our agent, however, doesn't reason about other players' goals or beliefs, and it doesn't factor in detailed opponent types. The assumption is that each of the opponents play optimally in the sense that each tries to maximise their score and win the game. We reduce the chances of falling into local optima only by balancing exploitation with exploration during planning. Future work could introduce behavioural profiles of the current opponents in the sampling methods to further improve the performance and reduce this risk.

1.3.3 Publications

The work presented in this thesis has been included in the folowing publications:

- Mihai Dobre and Alex Lascarides (2018). POMCP with human preferences in Settlers of Catan. *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, Edmonton, Canada. This paper describes the Partially-Observable Monte Carlo Planning extensions from Chapter 6 and the evaluation of different combinations of Partially-Observable Monte Carlo Planning with the offline learning methods from Chapter 8.

- Mihai Dobre and Alex Lascarides (2017). Exploiting action categories in learning complex games. *Proceedings of the IEEE SAI Intelligent Systems Conference (IntelliSys)*, London, UK. The work on extending Monte Carlo Tree Search algorithm with sampling over the action types in the observable version of the game (see Chapter 5) is contained in this publication.

- Mihai Dobre and Alex Lascarides (2017). Combining a Mixture of Experts with Transfer Learning in Complex Games. *Proceedings of the AAAI Spring Symposium: Learning from Observation of Humans*, Stanford, USA. This paper presents the proposed neural network architecture that permitted extracting policies from our sparse corpus (see Chapter 7).

- Mihai Dobre and Alex Lascarides (2015). Online Learning and Mining Human Play in Complex Games. *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, Tainan, Taiwan. This publication describes the combination of the non-parametric method with a variant of Flat Monte Carlo Tree Search that learns to execute one action in the game, i.e. the placement of the second settlement during the initial placement phase (see Chapter 4 for details).

1.4 Summary of Thesis

Chapter 2 is the background section that contains the basic definitions of the methods and a discussion on related work. Chapter 3 presents a quick introduction of the resources used and a more detailed analysis of Settlers of Catan. Chapter 4 presents a simple non-parametric model for extracting advice from the corpus and combining with a simplified version of Monte Carlo Tree Search. Chapters 5 and 6 describe the planning at decision time algorithm applied in an observable version of the game and in the full version of the game. These chapters contain the modifications required to extend Monte Carlo Tree Search to handle the challenges in Settlers of Catan. Chapter 6 also contains a detailed analysis of why planning in the Belief Markov Decision Process is not appropriate for Settlers of Catan. We then introduce the offline learning model and its extensions in Chapter 7. We combine planning with the offline models and evaluate the resulting agents in Chapter 8. Finally, Chapter 9 contains the conclusion and a brief description of possible future work.

Chapter 2

Background

This chapter provides an overview of the required theory and algorithms implemented in this thesis. We also aim to place our work within the existing research literature by highlighting similarities and differences to prior work. This is not an exhaustive presentation, but it does provide a basis to discuss the proposed methods.

2.1 Game Theory

Decision theory combines utility theory and probability theory to define a Savagean model (Savage, 1954) of rational action (Russell and Norvig, 2009): i.e., a rational agent chooses its action via an optimal trade off between what it prefers (as defined by utility theory) and what it believes it can achieve (as defined by a probabilistic model of belief). Specifically, the Maximum Expected Utility (MEU) principles states that rational agents choose actions a to maximise their expected utility (see Equation 2.1) given the evidence e. Utility U is a measure of how much the agent wants something, represented as a mapping of the outcome $Result_i(a)$ of an action to a real number, and "expected" refers to the agent's belief as to whether it can achieve the outcome $P(Result_i(a)|Do(a),e)$ given the evidence e and what other outcomes are possible, hence the summation over all i outcomes.

$$EU(a|e) = \sum_i P(Result_i(a)|Do(a),e)U(Result_i(a)) \tag{2.1}$$

The MEU principle can be extended to sequential decision making via the Bellman equation (see Equation 2.2). The idea is that the utility of a state in sequential planning is not only dependent on its immediate reward, but also on the rewards of those states that are reachable from it via the agent's action. Specifically, Equation 2.2 states that

the expected utility of a state s is its immediate reward $R(s)$ plus the discounted utility of the next state s' given that the agent chooses the optimal action a.

$$EU(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) EU(s') \tag{2.2}$$

The degrading factor $\gamma \in [0,1]$ describes the agent as anything between myopic ($\gamma = 0$) to far-sighted ($\gamma = 1$). As before, a transition model is used to quantify the expectation. It provides $P(s'|s,a)$ which is the probability of s' being the outcome state given that a is performed in s. There are methods for computing optimal solutions as defined by the Bellman equation, such as backward chaining, which reasons backwards from the end states of a problem in order to find the optimal sequence of actions. However, these are exact methods which are not tractable in large environments.

Game theory studies decision theory in situations where multiple agents interact to produce specified outcomes (von Neumann and Morgenstern, 1944). These situations are posed as games: a set of established rules that define the interaction. Games are categorised into several types. Of particular relevance to this project is games of *extensive form*: in other words, completing the game requires the players to perform an *extended sequence* of actions. Games can be further categorised via the following dimensions (Browne et al., 2012; Shoham and Leyton-Brown, 2008):

- *Cooperation*: Whether players have binding commitments;
- *Pay-off*: What is the sum of the players' utilities;
- *Sequential*: Whether players execute their actions simultaneously or in turn;
- *Information*: Only in sequential games, whether all players can fully observe their opponents' moves;
- *Determinism*: Whether chance factors play a part;
- *Symmetry*: Whether the goal is the same for each participant;

There are other characteristics, but these are the most relevant for presenting and comparing Settlers of Catan with other similar domains. We will revisit these notions when we present the game in Chapter 3.

There are several solution concepts for games, such as *Nash equilibrium* (Nash, 1950), in which no player has an incentive to deviate unilaterally from their own strategy given the strategies of the other players. Nash equilibrium is related to the MEU

principle, if we assume that the players know the other players' strategies and that these are fixed. A player's best response strategy is the one that maximise the player's utility, and no rational agent would deviate from it. Finding Nash equilibria in complex domains is an active area of research, but the space of possible strategies increases exponentially with the number of players, action and state space. As it is the case with backward chaining this is not appropriate in Settlers of Catan and we need to develop approximate methods.

2.2 Multiagent Systems

The goal of multiagent learning is to develop and analyse agents that learn to interact with each other. In order to be successful in Settlers of Catan, players need to negotiate and may even form temporary coalitions with other players. Therefore, the large body of knowledge in the multiagent field Shoham and Leyton-Brown (2008); Young (2004) is very relevant to our chosen domain. Despite this, we do not implement methods from a multiagent perspective and focus on our initial goal of learning a single policy from a sparse and noisy dataset. It is also very likely that reasoning about interactions with other players would not provide additional gains given that the baselines we compare against do not have any capabilities to react. If we evaluate against human players in future work, these methods might prove effective. Since Settlers of Catan is a multiplayer game, we must at least tackle the fact that opponents contribute to the game's progress (i.e., they have the capacity to change the game state). We make the safe assumption that all players act to maximise their expected utility measured purely as a function of the score they achieve. The methods that we develop plan via what is known as self-play: i.e., the same algorithm that acts to maximise the score plays against itself.

There is a small amount of work in Settlers of Catan that analyses the multi-agent characteristic of the game. Guhe and Lascarides (2014b) evaluate the effects of persuasion on win percentage and number of moves. The authors define a small set of player types to represent how ingenious the persuaders are or how gullible the recipients of persuasion are, followed by running game simulations between different agent types. The current state of the art rule-based agent (Guhe and Lascarides, 2014a) was developed by improving parts of the overall strategy and tuning the parameters of the existing heuristic agent (Thomas, 2004). The resulting agent was evaluated in a tournament setting versus multiple versions or types of the heuristic agent. Branca and

Johansson (2007) develop a multi-agent system that represents a single player by separating the heuristics of an existing agent (Thomas, 2004) into different agents, each handling different aspects of the game. In addition to not reasoning about interactions with the opponents, decentralising the agent was unsuccessful which is expected as it adds the additional challenge of coordination between the parts.

2.2.1 Opponent Modelling

Even though our agent doesn't interact or coordinate with the opponents over and above what the action space allows, our planning method could obviously benefit from knowledge of the opponent's strategy. In opponent modelling, the algorithms learn a model of an opponent by observing its behaviour. The trained model can predict moves which help to compute the opponent's optimal response during planning. One of the first opponent modelling methods proposed is known as fictitious play Brown (1951), which tracks the opponent's action frequency under the assumption that the distribution describing the opponent's preferences is stationary. Other more flexible methods account for the fact that the distribution may not be stationary, similar to the Bayesian Game setting (Harsanyi, 1967, 1968). These methods estimate the opponents current player type, given an extended history of evidence of their behaviour (and the game states). Instead of directly modelling the distribution describing the player's preferences, type-based opponent modelling can quickly adapt to changes in the opponent's behaviour by modifying the distribution over types that describes the opponent. Due to these benefits, there has been a large interest in type-based modelling in the literature (Gal et al., 2005, 2008; Hindriks and Tykhonov, 2008) as well as research into how to model the uncertainty over the definition of the hypothesis space of possible player types (Albrecht, 2015).

Most of the existing methods update the model's parameters during game play. This is complimentary to the work in this thesis and all the algorithms presented here can be combined with such an online method for modelling the opponent. Furthermore, it doesn't make any sense to learn distributions over player types from a database of example play unless the agent is encountering the same opponents or can describe the new opponents via mixtures of the learned types. It is a very challenging and possibly unrealistic task to learn and differentiate the possible player types from the amount of data that we have available when the state and action spaces are massive. It is also very difficult to define these models without a strong knowledge of the game. Most

of the previous work in Bayesian games is done on relatively simple games—fully observable, with hundreds (not thousands) of possible states, with only a handful of options at each state (on average) instead of hundreds. It would be interesting to see whether some of the models mentioned above scale up to inform optimisation in more complex environments, but we leave this to future work.

However, models that learn a general move prediction model without reasoning over types (Sutskever and Nair, 2008; Clark and Storkey, 2015) may be more applicable in our case. The extracted player model collapses all player types into a single type, so we should require less data to learn such a general estimation. However, the assumption that the data was generated by a single player type may not hold since different player types can generate play that is contradictory. Previous work bypasses this issue by learning from large datasets of expert play where it is likely that the optimal moves outnumber the sub-optimal ones. In this work we evaluate if training these methods on a sparse dataset that also contains sub-optimal play is possible. We evaluate their utility in biasing a planning agent and we also compare their performance against simpler approaches, similar to Bitan and Kraus (2017), who extract preferences over an abstraction of the action space. As future work we will look into reasoning over the player types both in planning and when learning from data.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a part of the broader field of machine learning that formalises the learning from experience problem (Sutton and Barto, 1998). In this setting, an autonomous agent learns to reach a predefined goal by interacting with the environment and observing the reward signals. The agent achieves its goal by optimising a function defined over these signals (e.g. average of the rewards), which results in learning a policy to take optimal decisions. A policy is a mapping π from each state s to a probability distribution $P(A)$ over the action space A, and is denoted as $\pi(s)$. The whole probability mass is concentrated on a single action in a deterministic policy, while in a stochastic policy, the mass is spread over multiple actions. A rational agent facing a decision problem is tasked with finding the optimal policy: i.e. the policy that maximises a function over the reward signals. In general this function is quantified using state value functions $V^{\pi}(s)$ or action value functions $Q^{\pi}(s,a)$ given the current policy. The optimal policy is π such that $V^{*}(s) = \max_a Q^{*}(s,a) = \max_{\pi} V^{\pi}(s)$. The two value function represent how good a state or an action that can be executed in the

current state are. Following on the decision theory, these functions are quantified by computing the expected reward given the agent's policy and the environment's dynamics (as shown in Equation2.3 in the following Section).

2.3.1 Markov Decision Process

A sequential reinforcement learning task that satisfies the Markov property can be modelled as a Markov Decision Process (MDP). The Markov property assumes that a state and expected reward of a state of the process depends only on the previous state and the action performed (Russell and Norvig, 2009). A MDP is a tuple $< S, A, T, R, \gamma >$ which specifies the states S of the environment, the actions A the agent can take from each state, the transition function $T(s,a,s')$ that provides probabilities $P(s'|s,a)$ of reaching any outcome state s' given the current state s and the chosen action a, the reward function R that maps states in S to a numerical value and the discount factor γ. Given this definition, the Bellman optimality equation in an MDP is:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s') + \gamma V^*(s') \tag{2.3}$$

2.3.2 Partially-observable Markov Decision Process

Part of the complexity of domains such as Settlers of Catan is attributed to being partially observable, which causes an exponential increase in the search space (i.e. due to the curse of dimensionality). Furthermore, these domains cannot be modelled using an MDP since the agent doesn't know what the state of the environment is. Kaelbling et al. (1998) introduces the Partially Observable Markov Decision Process (POMDP) that extends the MDP framework by adding a set of observations Ω and defining an observation model $O(s',a,o)$ that provides the probabilities $P(o|s',a)$ of observing an observation $o \in \Omega$ given the action executed is a and the outcome state that cannot be observed directly is s'. A POMDP is a framework for modelling what is known as the Belief MDP by marginalising over the possible current states s to compute the probabilities over the possible resulting states s' (Equation 2.4 shows the transition function in a POMDP). A Belief MDP is an MDP $< B, A, \tau, \rho, \gamma >$ defined over the continuous space of belief states $b \in B$ where each b is a probability distributions over S, the same set of actions A, the belief transition function $\tau(b',a,b)$ (see Equation 2.5) and a reward function $\rho(b)$ (see Equation 2.6). Every belief update is deterministic since $P(b'|a,b,o)$ output depends on the deterministic SE function (Equation 2.7). The SE

function is known as the state estimator and it outputs the new belief state depending on the previous belief state, the action and the observation (Kaelbling et al., 1998).

$$P(s'|b') = \sum_o P(o|a,s') \sum_s P(s'|a,s) \tag{2.4}$$

$$\tau(b',a,b) = P(b'|b,a) = \sum_o P(b'|a,b,o)P(o|a,b) \tag{2.5}$$

$$\rho(b) = \sum_s P(s|b)R(s) \tag{2.6}$$

$$P(b'|a,b,o) = \begin{cases} 1 & \text{if } b' = SE(b,a,o) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Exact planning in a POMDP is impossible in complex environments due to Bellman's curse of dimensionality, which is further aggravated by having to consider an additional dimension: belief. The most successful approaches are based on sampling (Kurniawati et al., 2008; Silver and Veness, 2010; Somani et al., 2013) or by planning in abstract representation of the belief (Kaelbling and Lozano-Prez, 2013). We implemented a modified version of POMCP (Silver and Veness, 2010) and compared it with a model that can be seen as an extension of the approach in Kaelbling and Lozano-Prez (2013) to forward planning in multi-agent non-cooperative environments (see Chapter 6 for details). However, our algorithm plans in an almost exact representation of the belief by making use of a factored representation (Paquet et al., 2005; Williams, 2005). Interactive POMDPs (I-POMDP) are an extension of POMDP to the multi-agent setting by integrating player types (Gmytrasiewicz and Doshi, 2005). Finding a solution becomes even more expensive as the model explicitly reasons over nested beliefs. We therefore use the standard POMDP framework to present the algorithms and make the assumption that the opponent's have the same strategy as our own agent, i.e. trying to maximise their expected return. This is known as self-play.

2.3.3 Function Approximations

A tabular representation for the value functions is clearly not appropriate in large environments since it doesn't allow any form of generalisation, despite being exact. Approximate methods learn functions over the continuous space of a different representation (e.g. sparse feature vectors) by learning the parameters of the function instead

of the exact state or action to value mapping. There is a variety of ways to define these functions depending on the domain (Sutton and Barto, 2018): linear functions, radial basis functions, tile coding, non-linear functions etc.

Linear function approximations have been shown to give good results in large or continuous games where the state space is too large (Silver et al., 2008). To perform such approximations, one first needs to transform the game data into a different representation, e.g. feature vectors. This pre-processing has proven to be a very important step in the performance of the final systems (Bishop, 1995). It involves feature selection and feature extraction. The former is the process of creating a subset of features from the raw data set by eliminating any irrelevant or redundant ones. Feature extraction is a special form of dimensionality reduction and could create new features from an existing set, by performing some mathematical calculations. Generalisation over binary features has been applied with success to a multitude of domains: Go (Silver et al., 2008), Civilisation 2 (Branavan et al., 2012), Checkers (Neto et al., 2014) and Backgammon (Tesauro, 1995). In general, the features were chosen manually based on the author's domain knowledge. However, (Neto and Julian, 2007) show that a simple automatic process using a Genetic Algorithm can greatly improve the agent's performance. We are manually building the representation used by our offline methods, by employing both feature selection and extraction (see Chapter 7 for details). We want to underline the fact that we do not include any knowledge of state or action values, and we only aim to reduce the hypothesis space compared to that of a model that uses the raw representation. Such an approximation is obviously needed given the complexity of our domain and the lack of data, but we did not do a thorough exploration of the feature space.

2.3.3.1 Neural Networks

Neural Networks are computational models composed of basic units called neurons (nodes) intended to imitate the operation of a single brain cell (Haykin, 1998). Each network is tailored to its task, which could be classification or regression tasks. A multi-layer network contains at least one hidden layer which enables the network to compute non-linear functions. The output o_j corresponding to the jth neuron of the current layer with activation function g and having as input the output of neuron i from the previous layer is given by the following formula, where w_{ij} are the weights (including a bias term connected to a fixed input) between the previous neurons i and

the current neuron j:

$$o_j = g(\sum_{i=0}^{n} w_{ij} o_i) \tag{2.8}$$

The training process is called weight synaptic adjustment. It consists of updating the weights in order to produce the desired outcome. Updating the weights involves minimising the difference (called error) between the desired and the actual results. There are a multitude of algorithms that can be applied, some of which employ a form of gradient descent (Bishop, 1995). Deep Neural Networks (DNN) (Hinton, 2007) are multi-layer neural networks that have more than 2 hidden layers which permits learning even more complex functions. These are trained via a process called *backpropagation* which takes advantage of the fact that each layer can be seen as a function and these are chained together to form the network. Using the chain rule of calculus, the error can be propagated back to the initial layers of the network.

Neural Networks have been used in games for varied purposes. For instance the Neurogammon program (Tesauro, 1990) learns to take decisions using a multi-layer network and is the first learning program to win a tournament. Sutskever and Nair (2008) use a network to model the opponents while Branavan et al. (2012) trained a network to approximate a Q-function by combining information taken from a manual with information extracted from the state encountered in the real game. Using DNNs as function approximations lead to the creation of a field known as Deep Reinforcement Learning that is concerned with developing algorithms that are able to overcome the difficulties in training these functions using only the reward signal. Deep Reinforcement Learning has proven very successful in games (Riedmiller, 2005; Mnih et al., 2015; Silver et al., 2016), however these methods are notorious for the amount of data and computational resources they require especially in highly sparse complex games. In contrast, this thesis focusses on methods that will learn effectively when vast data and computational resources for training are not available.

2.3.4 Planning at Decision Time

Planning has been extensively studied in the field of artificial intelligence, especially in model-based reasoning systems that define the environment and how the agent can interact with it using similar formalisms to propositional or first-order logic (Russell and Norvig, 2009). Several languages have been proposed such as STRIPS (Fikes and Nilsson, 1971) and ADL (Pednault, 1987). Uncertainty and partial-observability was

usually handled via contingency planning or by replanning if a plan failed in the real world. Most of the classical planning frameworks were only intended to find a plan and not necessarily the best plan. Also, the agent was not learning while planning. In general, a reinforcement learning agent learns from real experience, but there are reinforcement learning algorithms that perform planning using a model of the environment. In this case, the agent can simulate game playing and learn to play optimally during the simulated play, and execute the best action or sequence of actions in the real environment when it finished planning. The most popular approaches only estimate the optimal policy since these are sampling-based procedures for forward panning (Kearns et al., 2002; Kocsis and Szepesvari, 2006). Planning at decision time is sometimes referred to as online planning or online learning since it is generally fast and applied during game play whenever a new state is encountered. We will stick to the name of planning to avoid confusion with learning in a continuous or life-long setting.

2.3.4.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an extremely successful method for creating agents that play complex games. To avoid the need to learn a policy applicable in every possible state in a vast state space, the agent learns to plan at decision time. It combines Monte Carlo methods for sampling the decision space with strong policies for balancing exploitations and exploration in multi-armed bandits. In addition, it slowly builds the game tree making use of one of the most influential ideas in reinforcement learning: bootstrapping (Sutton and Barto, 2018). It uses the statistics from previous iterations to select longer sequences of actions thus being able to plan over several time-steps and slowly converge towards the optimal sequence as the number of iterations is increased.

Abramson (1990) demonstrated that Monte Carlo methods might be useful to approximate the game-theoretic value of a move. The action value function is the average expected reward r given the number of times $N(s,a)$ the action a was taken from state s (Gelly and Silver, 2011):

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} r_i \tag{2.9}$$

Multi-armed bandit problems are a class of sequential decision problems, in which the agent needs to choose amongst k actions in order to maximise the cumulative reward (Robbins, 1952). Auer et al. (2002) showed that methods that calculate the upper

confidence bounds (UCB) achieve a logarithmic expected regret. UCB was adapted to MCTS and renamed to Upper Confidence Bound for Trees (UCT). It is the most popular policy for MCTS as it performs better than ε-greedy methods in which exploration is achieved choosing uniformly at random from the available options (Sturtevant, 2008; Balla and Fern, 2009). We direct the reader to the survey done by Browne et al. (2012) for alternative tree policies. UCT combines the value function with an additional term that represents the confidence in the current statistics based on how much the action was explored compared to other available options (this is the second term in the sum shown in Equation 2.10). This confidence value in combination with a constant C are used to decide on the amount of exploration:

$$UCT(s,a) = Q(s,a) + C\sqrt{\frac{2\ln N(s)}{N(s,a)}} \tag{2.10}$$

The standard MCTS algorithm can be separated into four steps (see Figure 2.1): selection (starting from the root that represents the current state in the real game, select the nodes based on the UCT policy until a leaf node is reached), expansion (one or more children are added to the leaf node), rollout (from the new node the game is played following the default random policy) and backpropagation (the result is used to update the statistics at every chosen node in the tree).

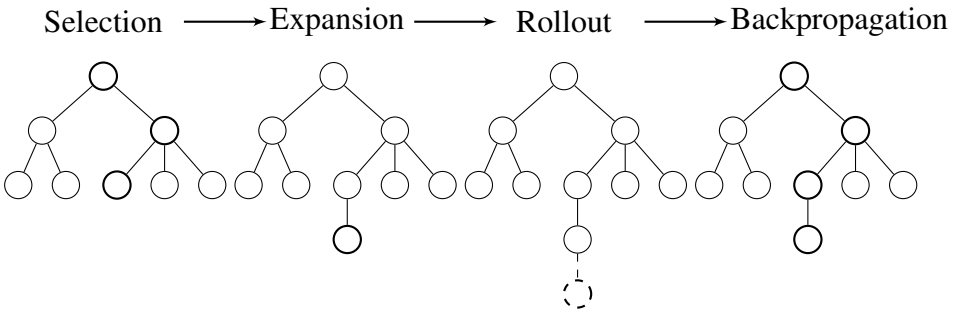


Figure 2.1: MCTS algorithm; Repeat the steps n times or until a given time limit.

MCTS suffers from the same issues in large complex games like any other reinforcement learning algorithms. If the space is too large and there are only small differences in the values of the available actions, it requires a huge number of samples to converge to an optimal policy. If it doesn't encounter any high valued node in the selection step, also known as the tree phase, it is unable to bootstrap and differentiate the available branches. In addition, the estimated return computed with Monte Carlo methods has a large variance (Sutton and Barto, 2018), especially in stochastic

environments such as Settlers of Catan. There are other characteristics of games that aggravate this issue, such as the action space being dominated by action types that either do not get the players closer to winning the game or generate a cyclic behaviour. To overcome these issues, one can either perform importance sampling or bias the algorithm with some form of control variate. An example for the latter is how neural networks trained on previous experience were used to bias the tree search in AlphaGo (Silver et al., 2016).

In our experiments on the observable version of Settlers of Catan (see Chapter 5), we partly address the issue in the tree phase via standard techniques by sharing results via transpositions (Childs et al., 2008) as well as combining results of actions that lead to the same outcome state. The latter is known as either afterstates or successor-states in literature (Sutton and Barto, 1998; Szepesvari, 2010). We depart from the standard way of handling partial-observablity by representing nodes using the history in the partially-observable game (Silver and Veness, 2010; Cowling et al., 2012a) due to the obvious impossibility to share results. Instead, we use an abstract representation of belief states to represent nodes. We also develop a game model that provides a belief transition function and, following on the work of (Kaelbling and Lozano-Prez, 2013), we do not sample fully-observable states. The result is the first MCTS algorithm that plans entirely in the Belief MDP in stochastic partially-observable games.

Another popular way to overcome the issues of random sampling in large domains is by introducing prior knowledge into either the tree phase or the rollout phase of MCTS. Similar to the work in contextual bandits (Wang et al., 2005; Li et al., 2011; Beygelzimer et al., 2011; Bubeck and Cesa-Bianchi, 2012; Caron et al., 2012), where a context that provides information on the reward structure is used, UCT can be improved via a multitude of ways: initialising the action values (Szita et al., 2010; Silver and Veness, 2010) based on common knowledge or general heuristics, initialising with statistics from previous runs (Gelly and Silver, 2007), combining with an opponent model learned via fictitious play (Heinrich and Silver, 2015), interpolating with a probability distribution that describes general preferences over abstract actions from a database of gameplay (Bitan and Kraus, 2017), weighting with a move-prediction model trained on a database of expert games (Graf and Platzner, 2016) or with a policy trained via reinforcement learning (Silver et al., 2007, 2016). On the other hand improving the rollout policy could have similar benefits and this is generally known as using heavy rollouts (James et al., 2017). Again there are a variety of ways to improve the rollouts by combining with: heuristics (Silver and Veness, 2010; Whitehouse et al.,

2013), general preferences over abstract actions (Bitan and Kraus, 2017) or a policy trained via supervised or reinforcement learning (Silver et al., 2016). In general, the strategies used for informing the rollout policy need to be more efficient than the ones used in the tree phase since the algorithm spends the majority of the time performing rollouts. Following on this large literature, we also combine MCTS with knowledge extracted from a corpus of human play and inform both policies. However, our corpus is much smaller in comparison to the human corpus of game play used by Silver et al. (2016), and it contains general play instead of expert play. We therefore show the benefits of adding general preferences over the standard policy mapping states to action probabilities in this case. We define these preferences over an abstract description of the action space that is provided by the game rules, known as action types, instead of defining our own abstraction as Bitan and Kraus (2017). In addition we explore the benefits of learning these preferences conditional on type legality which encapsulates strong preferences of certain types over others. This information aids in preventing the cycles encountered in the random rollouts caused by one type dominating the action space.

We want to briefly mention here that the algorithms developed in this thesis use a single accurate model of the environment when planning. One of the major limitations of MCTS is the dependency on a model of the environment for planning via forward sampling. In some real-time strategy games, researchers focus their attention on games where a fast forward model can be created, e.g. Moraes et al. (2018). Looking at robotics or real-world applications, where the physics of the world must be simulated and the robot response time must be in miliseconds, creating such a model is an even more challenging task. However, boardgames (e.g. Settlers of Catan, Go, Chess) have a set of clear rules and the effects of the actions can be quickly simulated. Therefore, we do not attempt to learn or improve an existing model of the game as in Bayesian model-based reinforcement learning, where the agent may be uncertain of the environment dynamics. There are methods that deal with this uncertainty (Castro and Precup, 2010; Asmuth and Littman, 2011; Guez et al., 2013; Menashe and Stone, 2015) in a sampling-based fashion, however this adds unnecessary burden for the learning algorithm when we can define an accurate model for Settlers of Catan.

2.4 Mining Human Knowledge

The methods presented before, in their standard implementation, are random walk approaches and there is a risk of getting stuck in local maxima. To prevent this, one must balance guided search with random exploration. We want to avoid manually specifying rules since the massive state and action space makes such methods highly labour intensive and error prone. Automatically extracting information on what humans do in similar situations would be a better solution. Techniques for exploiting human play to learn to play a game have proved useful for improving the performance of agents in several applications, including complex games. We provide here a short survey of existing methods for mining human knowledge by categorising these into: imitation learning, learning from demonstration and human intervention. For completeness we also cover the work done on designing heuristics by manually analysing human play or based on expert advice.

2.4.1 Imitation Learning

The most simple and well-known approach to imitation learning is by training a standard classifier (Bishop, 1995) that learns a policy as a mapping from states to actions while following an expert's trajectory in the environment. The label or correct actions are the actions that the expert policy chose at each state in the trajectory. Such methods have been applied with some success since Neurogammon (Tesauro, 1990), whose doubling algorithm was trained on 3000 positions taken from 64 expert games. Classification via statistical methods require the samples to be independent and identically distributed, while following expert trajectories generates samples that are strongly correlated. Some well known attempts to address this issue are the SEARN (Daumé et al., 2009) and the DAGGER (Ross et al., 2011) algorithms which both train the classifier on datasets generated from a policy that is an interpolation between the expert policy and the continuously improving classifier. Another benefit of these methods is that it bootstrap with the expert policy to avoid the difficulty of learning in large spaces.

However, these aggregation methods require access to the expert policy, which is not available in most cases. As a result, most methods rely on just randomly shuffling the existing data to break correlations. These methods are sometimes referred to as move-prediction methods, since they are not necessarily able to generate sequences of actions that resemble those of the expert but can provide an indication on the most likely actions given the current state. These methods have been very successful in Go

(Sutskever and Nair, 2008; Clark and Storkey, 2015; Maddison et al., 2015) and have proven useful in combining with MCTS when trained on expert play (Silver et al., 2016). In our experiments we also do not have access to the expert's policy, so we will follow this research in training a policy via supervision in an offline manner. In this thesis, we motivate and evaluate some modifications that are needed to help these methods tackle training on mixed play and with a smaller set of training examples.

2.4.2 Learning from Demonstration

Learning from demonstration involves extracting a policy from a set of examples, or demonstrations. Argall et al. (2009) definition of demonstration includes any form of knowledge, while we focus on human knowledge only and in particular in the form of an existing corpus of game play. Of most relevance to our task is the inverse reinforcement learning approach (IRL) (Ng and Russell, 2000). IRL tries to recover the reward function that best explains the demonstration behaviour given evidence of the model of the environment in which the demonstrations were generated in. The recovered reward function can then be used in a reinforcement learning algorithm to learn the optimal policy. An alternative is to use the recovered reward to enhance the environments reward function via reward shaping (Ng et al., 1999; Randløv and Alstrøm, 1998; Suay et al., 2016). This new reward function could be used in reinforcement learning or even in online planning. Finally, Apprenticeship Learning (AL) (Abbeel and Ng, 2004) creates a mixture of the policies explored by IRL during learning such that the trajectories generated by it resemble that of the expert.

Unfortunately, these methods have several challenges that increase the already difficult task tackled in this thesis. First of all, IRL chases two objectives during learning: recovering the reward function and generating the optimal policy under this reward. This cycle increases the hypothesis space dramatically. Secondly, the assumption that a reinforcement learning algorithm can find the optimal policy in such a game is unrealistic. In order to make it work, previous work focused on extracting rewards for specific actions in a shooter game and also created simple specific scenarios for collecting the demonstration trajectories (Tastan and Sukthankar, 2011). In addition, we require good function approximation to represent the reward, the policy and the feature expectations. This is especially problematic since these have to account for the trajectories' outstanding space of possibilities and ambiguity. Most IRL algorithms also assume ergodicity of the MDP but this is not true for Settlers of Catan because of

the way the initial state is generated. Finally, there is no guarantee that the recovered policy is similar to the one that generated the demonstration, rather than just generating trajectories that look the same.

There is a large body of work that attempts to address these challenges; many of these constitute highly complex algorithms (Ratliff et al., 2006; Ziebart et al., 2008; Babeş-Vroman et al., 2011; Boularias et al., 2011). It is very likely that we would still need to combine the learned policy with a planning method, which will add additional parameters. Unfortunately, our dataset already presents a large number of challenges that we need to focus on first. As a result we chose develop a good function approximation and extract a policy via the simpler approach of supervised learning. We believe IRL and AL are interesting paths to explore in future work.

An approach worth mentioning that is sometimes considered part of the learning from demonstration field is combining expert demonstrations with another learning mechanism. For example, Thrun (1995) shows that an agent can be trained on a combination of self-play and database play. Alternatives include bootstrapping reinforcement learning by either training the initial policy via supervised learning first Silver et al. (2016) or by combining the loss of Deep Q-learning with a loss function that encourages imitating the expert Hester et al. (2017). Even though these approaches have accelerated the learning, there was still a very large computational requirement required to train deep reinforcement learning on simpler games than Settlers of Catan. In addition, these approaches would make it hard to quantify the value of our data, so we chose to focus on a pure imitation learning method as a means of extracting a policy from our corpus which we will use to bootstrap an online planning method.

2.4.3 Human Intervention

The term human computation was coined to describe the use of human processing power to solve problems; however the output is generally used without performing any further computation. Therefore, we believe that a more accurate description would be “human intervention” and we will be using this term instead. Human intervention problems fit the general paradigm of computing using human participation directed by the system to guide its computations (Quinn and Bederson, 2011). So it is a human-machine collaboration scenario, in which human processing is used to solve problems that computers cannot solve yet. Crowdsourcing and human intervention overlap but are not the same: “Whereas human computation replaces computers with

humans, crowdsourcing replaces traditional human workers with members of the public" (Quinn and Bederson, 2011). An example of human intervention in games is the work by Knox and Stone (2008) who developed a framework called Training an Agent Manually via Evaluative Reinforcement (TAMER). Their framework allows humans to provide feedback in the form of a reward to the agent's actions in the game of Tetris, reducing the number of games required and improving the agent's performance.

On the other hand, data mining involves applying algorithms for extracting patterns from data (Fayyad et al., 1996) which may have been created by humans. One can notice that human intervention and data mining are different, because the former directly uses human processing power, while the latter performs its own processing on data resulting from human processes. Another very relevant field is that of active learning (Settles, 2009), where the learning algorithm can help reduce the bottleneck of data labelling by learning to query an oracle only when a certain condition is met, e.g. the learning algorithm is uncertain about the label of a sample. This approach can help reduce the data requirements significantly, at the expense of having an oracle available to answer the algorithm's query.

The only methods that fit the human intervention paradigm and are related to this thesis are that of data mining. Our corpus is already collected and the only post-processing we could do is that of annotating based on the quality of the play. We would need to crowdsource this task to expert players that are capable of analysing such data. Annotating the games, as we will show in Chapter 3, is quite challenging so we cannot guarantee the results will have a high quality. Furthermore, annotating or using an oracle involve additional expense that this thesis aims to avoid by developing algorithms that can learn from sparse and noisy datasets. Finally, if we have access to experts, we could just ask them to play the game and generate more example play. We conclude that this field is not the most appropriate for finding solutions to our problem.

2.4.4 Heuristics

As mentioned before, manually coding rules to reflect what a human player would do in similar situations provides no guarantees. Even so, improving the play with heuristics has been extensively done in every game for which an AI agent has been developed. It is the simplest form of integrating human knowledge. There are also many ways to achieve this and we will only mention a few relevant ones. Applying MCTS in the domain of Go has received a lot of attention in recent years and, until recent work on

alphaZero (Silver et al., 2017), the best agents made use of patterns describing board positions (Gelly et al., 2006) or opening moves (Chaslot et al., 2009), both written following expert players advice. Both the tree and the rollout phases of MCTS can also be improved using strong heuristics (Browne et al., 2012; Gelly and Silver, 2007; Silver and Veness, 2010; Whitehouse et al., 2013). Regarding Settlers of Catan, the strongest agent (Guhe and Lascarides, 2014a) is an extension to the initial rule-based one (Thomas, 2004) implemented in the JSettlers framework. Keizer et al. (2017) shows that this agent is strong enough to be competitive against human players. A pure reinforcement learning method has not been applied yet on the full game and all of the existing implementations either use a combination of heuristics with reinforcement learning or have simplified the rules of the game (Szita et al., 2010; Roelofs, 2012). Pfeiffer (2003) shows that one needs heuristics to structure the high level strategies or medium-term goals and use reinforcement learning from the low-level decisions.

2.5 Conclusion

The literature review presented in this chapter highlights the strengths and deficiencies of many popular methods with respect to the problem we aim to solve: learn to play in a complex game under a low-resource constraint. Previous work has highlighted the need for approximate and adaptable methods in such complex environments. As a result, we implement an online planning algorithm that can take decisions by simulating the set of possibilities given the state the agent finds itself in. Since sampling based algorithms require a large number of samples in order to form a good estimate (due to the variance of the returns), we bias planning with policies extracted offline from a corpus of previous play. Unfortunately, previous research indicates that the strongest offline learning methods also have large data requirements. Therefore, we perform certain abstractions that would permit us to utilise the sparse corpus we have available, and we evaluate methods that make use of such abstractions versus neural network based methods that do not.

Chapter 3

Resources and Domain Analysis

This chapter presents the domain chosen for evaluating the proposed models. It also contains a brief description of the software used, the corpus of human games and the experimental methodology, including our method for evaluating the performance of our various learning algorithms. The goal of this chapter is to provide a quick introduction of the environment in which the experiments were conducted and to present the challenges this setting poses to learning effective game strategies.

3.1 Settlers of Catan

Settlers of Catan is a multi-player win-lose game. We focus on the core board game with 4 players shown in Figure 3.1. The players build roads, settlements and cities on the board, which is formed of hexagonal tiles. The first player to reach 10 Victory Points (VP) wins the game. One obtains victory points in a variety of ways (e.g. a settlement is worth 1 point and a city is worth 2 points). The hexagonal board tiles represent one of the five resources (Clay, Ore, Sheep, Wood and Wheat), desert, water or ports. Each of the resource producing tiles has an associated number between 2 and 12 (but not 7). Players obtain resources via the location of their buildings and dice rolls that start each turn of the game: e.g., if a player rolls a 3 and a 6, then all players who have a settlement (or city) on a hex marked with a 9 receive 1 (or 2) of that hex's resources (unless there is a robber on the hex in question, in which case no resource is produced until the robber is moved). One needs different combinations of resources to build different pieces (e.g. a road costs 1 clay and 1 wood). In addition to dice rolls, players can acquire resources through trades with the bank (at a 4:1 ratio), or with a port if they have built a settlement or city there (3:1 or 2:1, depending on the port) or

through negotiated trades with other players.

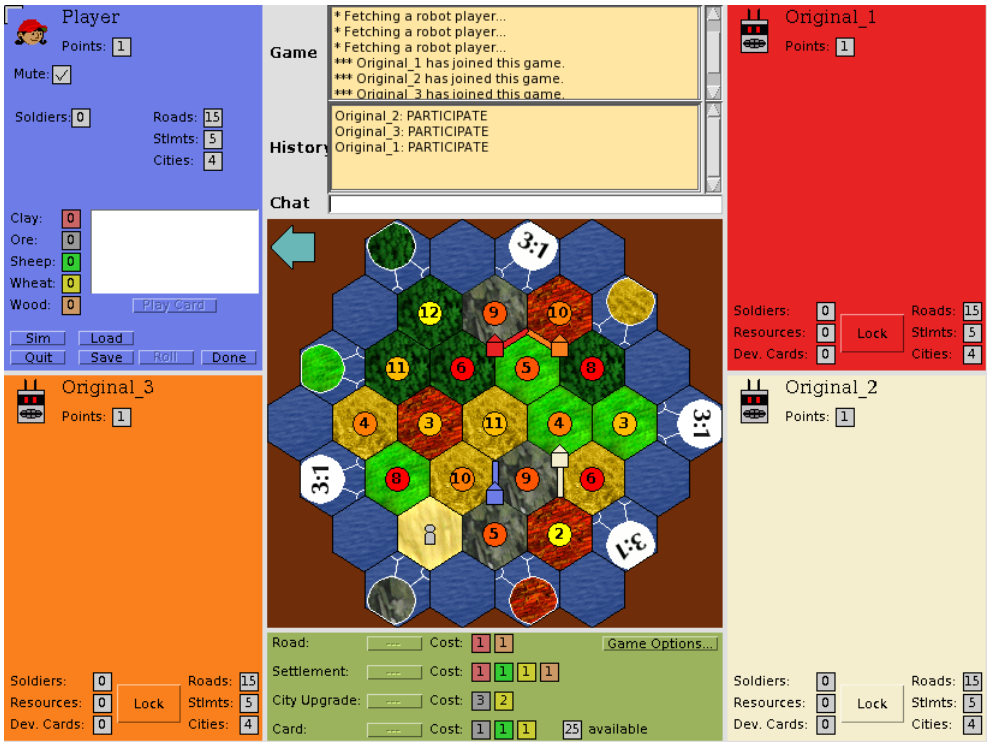


Figure 3.1: JSettlers game interface

There are many special actions which increase the complexity of the game, such as: *(i)* buying and then playing development cards that each give different advantages, *(ii)* moving the robber and stealing resources from other players and *(iii)* gaining victory points via the longest road or largest army. When an agent plays a development card it receives one of the following benefits depending on the type of card: build two free roads (i.e. the road building card), move the robber followed by stealing a resource from another player (i.e. the knight card), receive two free resources (i.e. the discovery card) or get every resource of that type from the other players (i.e. the monopoly card). The victory point development card gives the owner one victory point as soon as it is bought, however this is kept hidden from the other players until the end of the game. There are two awards that each give their owner two victory points if certain conditions are met. The longest road award requires the player to have the longest road made of at least 5 connected road pieces. The largest army award requires the player to have played the most or at least 3 knight cards. It is worth mentioning that the game rules encourage trading with other players. Hoarding resources until the player possess the required amount to execute certain actions is risky, since players that have over 7 resources must discard half of them when a 7 is rolled. We have provided a brief

description of every aspect of the game, however we encourage the reader to read the complete rules that can be found at `www.catan.com`.

Figure 3.1 illustrates the interface that a human player sees when using JSettlers. JSsettlers is an open-source implementation of Settlers of Catan, which we will describe in more detail in Section 3.3. The square in the centre is the board, where the players build the settlements, cities and roads, as well as move the robber. The coloured side panels are the players' panels which contain information about their hands. The current interface is linked to the blue player, hence this player has access to additional information such as what type of resources or development cards it posesses. The buttons inside the panel aid the player execute certain commands, e.g. the *Done* button is used for ending the turn. The box above the board is the chat interface and the box below the board is the helper box with buttons for buying pieces or development cards. The chat interface is used for trading by typing a set of specific utterances which the built-in agent (as well as the server who executes the trade) can understand, e.g. I give 1 ore for 2 wood.[1] Finally, the teal arrow pointing to the blue panel indicates that it is the blue player's turn to act.

3.2 Game Analysis and Motivation

From a game theoretic perspective, Settlers of Catan is a very complex game (see Table 1.1 in Chapter 1). In addition to the incomplete information (i.e. the opponents' policies), the game contains elements of imperfect information. Specifically, opponents cannot see the type of unplayed development card that others posses (nor know which development cards are still in the pile that can be bought). Further, since the type of resource that gets robbed or discarded is hidden to the unaffected players, players generally have only partial information about which type of resources their opponents possess as well. Further, the game is stochastic (e.g. dice rolls determine the players' resources). The generation of the board is done by shuffling the 19 land hexes and the 9 port hexes, so the game has a huge space of possible initial states ($\approx 1.2 * 10^{15}$ compared to 1 for the game of Go). However, local features such as smaller groups of hexes appear across many games, therefore Settlers of Catan presents an ideal problem to evaluate the generalisation capabilities of learning agents. Even though there is a large set of heuristics and suggestions for playing the game, there is no exact analytic

[1] StacSettlers' documentation contains a list of the allowed utterances. See link to the project in Section 1.3.1.

solution for every possible game state as far as we are aware.

The game tree has a large branching factor, e.g. there's a wide range of negotiation actions one can employ to trade the necessary resources with others. In fact, the large depth and branching factor of Settlers of Catan are mainly due to trading as we show in Table 3.3. Table 3.1 contains approximations of these factors, given 3 different sampling policies (where in each trial, all players are the same; i.e. human, or a heuristic agent, or a random agent). The results corresponding to the human and agent policies have been averaged over 60 human games (Afantenos et al., 2012) and 1000 simulated games respectively. Due to how these games have been logged, trades are considered a single exchange action and the preceding negotiations are not taken into account, hence the depths in Table 3.1 are smaller than in reality.

Policy	Branch	Depth
Heuristic	73	249
Human	65	159
Random	65	11715

Table 3.1: Average branching and depth. Many of the human games have 2 or 3 players, hence the smaller average depth. The values in this table are collected from games that run with the minor modifications described later in Section 3.2.2.

The length of the games generated using a random policy differs dramatically to that of the games generated with a more reasonable policy, e.g. a set of heuristics. As we will show in Table 3.3 (see page 41), this is mainly due to the trading actions. Disallowing trades in a random policy reduces the space size of the game tree significantly. The growth in the depth of the game is partly due to the cycles created by trading. But these cycles are actually a result of the agent's inability to properly evaluate the value of the trade actions. It is very difficult to differentiate between these actions in the planning phase since agents can trade back the resources and easily revert to the original state without any penalty. Such behaviour is unnatural in human games, and sub-optimal: an opponent may learn to exploit this eagerness to trade. Therefore, any learning method based purely on random sampling of the space would be inefficient and very inaccurate when evaluating the value of states or actions.

Despite the massive state and action space, high-level strategies are the same across games. For example, there are well-known suggestions for beginners to play towards the longest road, which is worth 2 VPs, by building settlements quickly and expanding

on the board, or towards the largest army (again worth 2 VPs) by building cities and buying development cards. These two are known in the community as wood-clay or ore-wheat strategies based on the names of the main resource requirements for achieving them. This game playing advice points to the fact that a player must be able to plan ahead and their action sequences must be consistently getting them closer to their end goal. However, this is not always possible due to how congested the board is, so players must be able to adapt and modify these two strategies to be successful. Combined with the lengthy games and difficulty to rank the legal actions according to their value, makes this game a very challenging problem for both planning at decision time and methods that learn general policies over the whole game.

One aspect that has a large impact on the complexity of the game is that it is a multiagent environment. As we mentioned before, we do not explicitly model the other agents in our implementation, however our agent must still be able to account for the fact that other agents have a say in how the game advances. Given that it is a win-lose game, the game is adversarial so all agents race to achieve the victory condition. Any rational agent will try to minimise the gains of the agent that is in the lead or at least ensure that it is not aiding the leading player and thus throwing the game away. If multiple agents display such rational behaviour, a coordination against the leading player arises even if the agents do not explicitly aim to form coalitions. This can be observed if multiple baseline agents play in the same game since these players will target the leading player when moving the robber. Human players are capable of forming more complex coalitions and can betray each other in the most unexpected way, e.g. trade many cards of a specific resource followed by playing the monopoly card to get everything back. Furthermore, negotiations moves are more advanced as human players are capable of forming complex persuasion moves, e.g. “If you trade your ore for my wood you will be able to build a settlement at location x”, and blocking moves, e.g. “Do not accept that trade or he will be able to get the longest road and win the game!”. Unfortunately, these interesting aspects of the game can only be observed in human games since the current state of the art agents do not have the capabilities to even respond to such moves. As a result we decided to learn under the assumption that all agents act to maximise their return without considering other agents’ preferences. As we will see in several of our results, such an assumption reduces the performance of our strongest agent which indicates that a bayesian game formulation (i.e. reason over player types) is an interesting path to explore as future work.

3.2.1 Structure in Complex Games

While Settlers of Catan is highly complex, it also incorporates a clear structure to its decision space. Actions can be grouped into several types, e.g. trade actions versus build road actions. The cardinality of one class almost always dominates those of others: e.g. there are always many trade options, but only one action of the type "end turn". The game can also be separated into 6 phases based on what action types are possible, such that there is a minimum overlap between the phases. We will refer to the following phases as **tasks** and use their names or corresponding indices throughout the document:

0. **Free road building state**: this is either the initial state where the agent places two free roads, or the two free roads that stem from having just played a 'road building' development card;
1. **Free settlement building state**: placing settlements during the initial placement part of the game;
2. **The normal state**: this is a state where the agent can build, trade, buy or play development card, or end their turn;
3. **Before rolling the dice state**: the agent has to decide between playing a development card or rolling the dice;
4. **Discard state** due to a 7 being rolled: the agent has to discard half of the resources it is holding;
5. **Moving robber state** due to a 7 being rolled: the agent has to move the robber and steal one resource at random from an opponent.

Out of these tasks, the normal one is the most challenging and it also contains the highest diversity of action types. Table 3.2 enumerates these types alongside statistics that show how likely it is that each action type is legal, the average number of possibilities (i.e. action tokens) when it is legal and the maximum number of possibilities. These were collected from 10k games played using a uniform random policy. The high values for the trade action type indicates that players have to generally decide between many trade moves and a few moves belonging to other types. Due to this skewness, a uniform random policy over the action tokens will be very biased towards trading. This table also shows that there are very few examples in the corpus for each of the

types that describe playing a development card. Combined with the small size of our corpus, this increases the difficulty of learning a strong policy for the complete action space.

Action type	Probability	Average	Maximum
Build road	0.0547	5.48	19
Build settlement	0.0331	2.82	13
Build city	0.0122	2.71	5
Buy development card	0.0114	1	1
Trade with opponent	0.9982	64.8	512
Trade with bank/port	0.2361	4.47	20
Play knight card	0.0001	16.16	29
Play monopoly card	0.0010	3.92	5
Play discovery card	0.0005	15	15
Play free road card	0.0023	6.35	34
End turn	1	1	1

Table 3.2: The list of action types during the normal phase of the game, the probability of the type of action being available, the average number of actions when the type is available and the maximum number of actions of each type. These numbers are collected from 10K games with 4 players, where each player plays randomly.

Following our previous game analysis, Settlers of Catan presents a very challenging problem. To overcome these we design our algorithms so that they exploit the structure inherent in the game rules. We specialise offline learning models to the tasks described above to alleviate the skewness of our data towards the normal phase in which players spend majority of the time. Furthermore, we reason over action types during planning to handle the ubiquitous characteristics of the game: negotiation and trading. There are many other board games that present similar game play characteristics—e.g. Monopoly, Civilisation, Diplomacy, Battlestar Galactica to name a few—for which the techniques developed in this work could aid the learning process.

In addition to this structure, only certain action types are legal depending on the current state description. This type legality provides a very compact representation of game states and allows us to cluster them. This is especially powerful since we have a very sparse dataset and learning a tabular representation or a general approximate representation may prove too challenging. We have integrated this knowledge of legal-

ity in the design and implementation of all our algorithm and it has proven key to the success of our agent given our empirical evaluations.

3.2.2 Game Modifications

Our final agent is tested in a game that has only a few minor modifications to the rules defined in `www.catan.com`. The goal of this modification is to level the playing field between our agents and the baseline heuristic agent. The state of the art rule-based agent that we evaluate our agent against only considers 1:1, 1:2 and 2:1 resource trades between players so we also limit our agents' set of legal actions to these. There are still a large set of possible trades (up to 540). Of course a side effect is that it reduces the branching factor dramatically by not including trades such as 2:2, 2:3, 3:2 and so on, but it is obvious that these trades are a composition of the allowed ones. The only benefit of this modification is that we do not include 3:1 or 4:1 trades. These are equivalent to bank or port trades and it is obvious to any player who aims to win that it is better to not trade with opponents at these rates. This is a very minor heuristic that only slightly reduces the space the agent needs to explore.

A simplification worth mentioning is that we do not allow any agent to perform any form of negotiations above offer, accept, reject and counter-offer. In addition, the current player must initiate the negotiations. These modifications prevent very interesting aspects of human games and there are perhaps many moves of high strategic importance that we ignore. Unfortunately, coordinating the agents to allow such moves is highly complex and would deviate too much from the original goal of this research. There is some interesting initial research into the value of persuasion moves in Settlers of Catan by Guhe and Lascarides (2014b) (e.g., "If you accept this trade offer, then you will get wood and be able to build a road"), but these are restricted to a small set of rules or templates. Human players are also able to change the way they formulate the negotiation utterances based on the current context and opponent's types. This is a very interesting problem that delves very close to Natural Language Generation for task oriented dialogue, but where the current goal should be decided by the agent's decision making algorithm. This is a very interesting path to explore in future work.

Previous research has either attempted to combine machine learning with rule-based methods (Pfeiffer, 2003) to reduce the responsibility of the learning algorithm, or has focused entirely on a single aspect of the game such as what resources to trade while the rest was controlled by the heuristic agent (Cuayáhuitl et al., 2015), or has

simplified the game such that a pure machine learning algorithm is able to handle the reduced space (Szita et al., 2010). Szita et al. (2010) have removed the agent's ability to trade in the real game, but also removed trading from the game model used during planning. Table 3.3 shows the large effects of these modifications on the game branching and depth factors. This modification doesn't impede their agent from playing the game, because being able to trade is beneficial but not a requirement. As a result, the agent presented by Szita et al. (2010) is still able to defeat the best heuristic agent of the time (Thomas, 2004). At the same time, Cuayáhuitl et al. (2015) show large performance gains when modifying a single aspect of the current best heuristic agent's trading behaviour. This underlines the need to handle trading in order to be very successful in the game.

Trade	Branch	Depth
None	10	1090
1:1 only	17	3085
1:1, 1:2 same and 2 same:1	26	5371
all trades	65	11715

Table 3.3: Average branching and depth when trading is not allowed or when only specific exchanges are allowed. The third option contains all 1:2 and 2:1 trades when the 2 resources are of the same type (e.g. 1 clay for 2 ore), while the last includes all 1:1, 1:2 and 2:1 including trades such as 1 clay for 1 ore and 1 sheep.

Finally, the existing planning based algorithms (Szita et al., 2010; Roelofs, 2012) have further simplified the game by either ignoring the imperfect information or by making the game observable to all agents. Ignoring the unknown information is clearly a sub-optimal approach, while making the game observable reduces the difficulty of the game. Our final planning-based agent is able to reason over the imperfect information. We also extend the current state of the art planning algorithms to handle the opponent's trading behaviour and include all trading actions described above in the planning phase. Therefore, our agent is the first pure machine learning based agent that can handle the rules of the full game.[2]

[2]Without being able to negotiate with human players due to not having natural language generation and understanding capabilities.

3.3 Software

The starting point for conducting our experiments is an open-source implementation called JSettlers (Thomas, 2004). JSettlers is a client-server system: a server maintains the game state and passes messages between each of the players' clients. Clients can be any combination of human players or computer agents (with a maximum of 4 players overall). The game interface for human players (including the board and each of the player's information) is shown in Figure 3.1. Guhe and Lascarides (2014a) improved the original JSettlers heuristics-based player, to create what we call the *Stac* agent. To our knowledge this is the strongest open-source rule-based Settlers of Catan agent and, like ours, it can play according to the game's complete set of options. So this heuristic agent tackles both negotiation and trading, and also the fact that the game is a game of imperfect information. Keizer et al. (2017) shows that this agent is strong enough to be competitive against human players. The JSettlers software was extended further with a simulation environment that permitted previous research (Guhe and Lascarides, 2014a; Guhe et al., 2013; Keizer et al., 2017; Cuayáhuitl et al., 2015) to compare agents with different parameters in a tournament-like setting.

JSettlers logging is a delta update system which stores only modifications to the current state. It is impossible to quickly access the details of a specific state without parsing the whole log file. In addition, JSettlers has undergone several modifications to its message parsing code without backwards compatibility, so it is not always straightforward to parse some of the old logs. To avoid the expensive look-up as well as losing data, we have created a database logging system which stores every information on each state and action in a game. The stored information is everything required to be able to load a game state stored in the corpus or played by synthetic or even human versus synthetic players.[3] We also created a save-load mechanism which can be used either by human players or synthetic agents in most game states (excluding some situations such as during negotiations). Unfortunately, JSettlers has a similar delta update system for the agents' state and belief representations, making it impossible for a synthetic agent to replace either a human player or an agent with different capabilities. Nonetheless, the save-load function is sufficient to run the models presented in Chapter 4 which only required reloading the same state with the same type of agents or with agents with fewer dependencies.

[3]The database does not contain the messages in the chat interface since this was not relevant to the current research.

In addition to the framework that allows us to evaluate the agents by running games between them, we need a lightweight simulation environment to perform the forward sampling required by MCTS. The game characteristics (e.g. discrete, no physics simulation required, clear rules etc.) allows us to build a fast and accurate model. Furthermore, the SmartSettlers framework (Szita et al., 2010) can quickly simulate a simplified version of the game, i.e. fully-observable and without trading. However, extending this framework to handle the remaining aspects of the game as well as connecting it with the existing JSettlers framework has proven very time-consuming. First of all, JSettlers agents are built into the JSettlers framework and an agent interface is not well-defined. There are many special cases that are not mentioned in the game rules which we discovered as we were combining the two frameworks. For example, players can gain 3 victory points by building a settlement that breaks the longest road of an opponent and allows them to get the longest road award. Furthermore, the partial observability requires the forward sampling model to be synchronised with the JSettlers framework that serves the real game without providing access to the true game state. Finally, trading and negotiations create scenarios where the result of actions may be different in the real game. For example, an offer may be encoutered by a different response than the one anticipated. We will present the details on how we handle trading in Section 5.5.

The deep learning models were built using the Deeplearning4j library version 0.4-rc3.10 (Deeplearning4j, 2016). Using an established deep learning library ensures gradient calculations, backpropagation and regularization methods are implemented correctly and efficiently. This permitted quick testing of various implementations. Furthermore, the choice for a deep learning library in Java was due to the need to keep the agent's decision time to a minimum. Creating a fast and seamless interface between programming languages is not always straightforward.

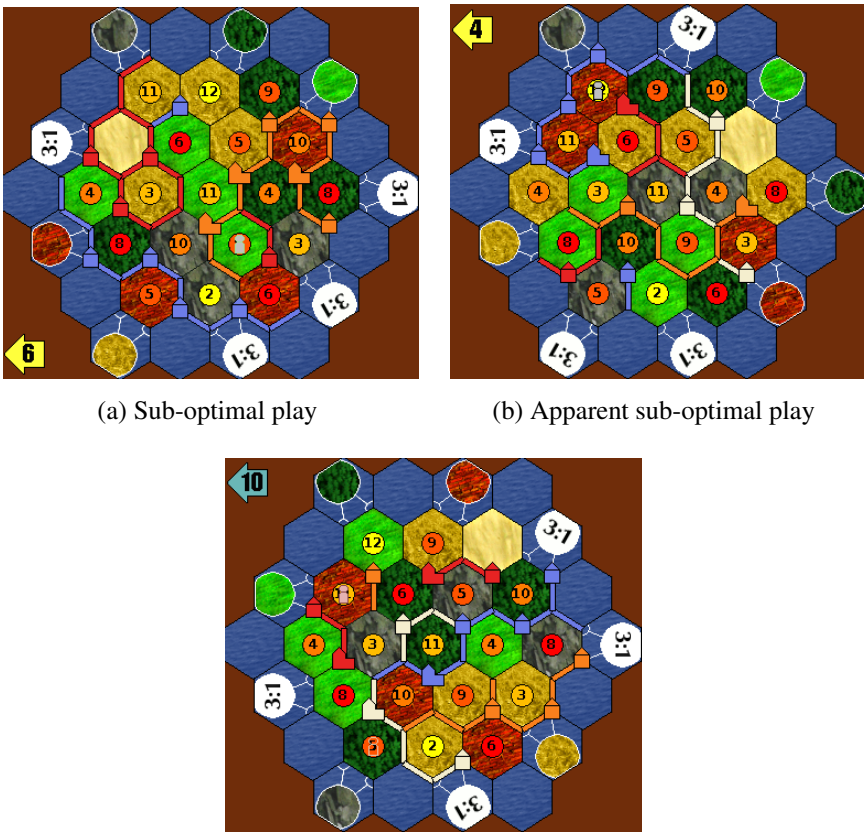
3.4 The human corpus

Our human data is taken from a corpus of humans playing Settlers in a competition that ran for three seasons. Two of the seasons are split into different leagues based on the participants' previous experience and level of play. Each participant provided their own estimation of their playing ability by completing a questionnaire. In these leagues each player plays multiple games and a basic ranking system is produced. The third season contains 21 games in which players of different levels are mixed together

and each of them has played a single game. The corpus consists of 59 games played on different board layouts and containing different number of players (i.e. 2, 3 or 4 players), from a total of 100 participants with an age range of 18-54 and 25% of them were women. Overall, this corpus ensures the data we have collected contains a large variety of scenarios. Despite this, the data is very sparse and it is highly unlikely that an exact state that is attested in the corpus will be encountered in a future game. In fact, it is unclear if existing learning methods would be able to generalise in such a complex game from such a small amount of data. Furthermore, filtering of sub-optimal plays based on the resulting inaccurate ranking system or on the subjective responses provided by the participants in the questionnaire would most likely only reduce the amount of data available instead of improving the overall quality of the data.

Following a more in-depth analysis of the corpus we have found 3 interesting examples that represent the type of game play contained in the corpus (see Figure 3.2). Example 3.2a shows a clearly sub-optimal play if the objective of the red player was to win the game. The player built settlements next to the desert despite having other options. At the same time it seems the player focused its attention entirely on winning the longest road, despite clearly struggling to compete with others that have access to the required resources. Another possibility is that this player lost an earlier race to a second settlement and gave up on trying to win the game. The second example (3.2b) also illustrates what is generally weak play. The blue player built most of its settlements on the edge of the board, a location that generally provides less resource production then the centre of the board. However this player has won the game and it is unclear if this is due to luck or if avoiding locations contested by the other players was the best play. It could also be the result of what is known in Game Theory as a mixed strategy. The final example (3.2c) shows a strong example play by the blue player that results in gaining the longest road award while ensuring no other player could contest it for the rest of the game: blocking opponents from building roads and settlements is an important part of one's strategy for winning the game.

These examples illustrate the variety of play included in our corpus. As with many experiments that involve human participation, it is impossible to decipher the participant's intentions only by observing the produced data. The characteristics of the game, such as non-determinism and being a multi-agent environment, exacerbate the player's uncertainty about the consequences of his or her decisions. This short analysis underlines the difficulty in interpreting the game play and classifying it as good or bad examples. It is very challenging for a human expert to reach a conclusion, let alone



(a) Sub-optimal play

(b) Apparent sub-optimal play

(c) Interesting (possibly optimal) play

Figure 3.2: A few examples of game play taken from the human corpus.

for standard machine learning algorithms. The diversity included in the corpus poses an interesting question: Is it possible to extract any useful advice without additional expensive pre-processing?

3.5 Experimental methodology

We are evaluating our methods by implementing agents that play the game in the JSettlers environment similar to the approach of Guhe and Lascarides (2014a). Running several game simulations in an evaluation is necessary because of the stochastic nature of the game: even a weak player can get lucky! The performance of an agent is measured by running simulations of 2000 games between 4 players: one of the players is the (modified) agent we are evaluating and the other 3 are baseline agents. So, a player that is of equal strength to the baseline agent would win 25% of the games. We tested the significance of win rates against this null hypothesis using the z-test and a threshold $p < 0.01$. This makes any win rate between 22.5-27.5% not significantly different

from the null hypothesis (i.e. a win rate of 25%). In order to keep the tables clear, z scores are not included.

The setting of evaluating one modified agent versus 3 baseline agents of the same type is a controlled experiment where we also have a well defined null hypothesis. On the other hand, an experiment where agents are all different is not a controlled experiment. There can be many interactions between the agents, which means we cannot form a null hypothesis. Other variations such as 2 versus 2 can also be tricky due to the same reason. A round-robin experiment requires searching a large space of possible combinations in order to change one variable at a time in. Such a setting makes it impossible to run all experiments in a timely manner.

Settlers of Catan is a multi-player game in which human players often form coalitions or betray each other. If synthetic agents would present the same behaviour, other experimental settings (e.g. round-robin) may reveal these interactions. However, none of the baseline agents or our agents explicitly reason about these interactions. As mentioned before in Section 1.3.2, our agent doesn't include any behavioural reasoning in its planning either. Nonetheless, there could still be implicit interactions (e.g. a certain type of agent may collaborate better with agents of the same type, or some types of agents may have a counter-strategy to the baseline agent's one). In order to account for these interactions without running the expensive round-robin experiment, we also evaluate one baseline agent versus 3 modified agents. In future work on introducing type-based reasoning in the planning agents, round-robin experiments may be required.

Finally, there is the concern that the same type of agent may win more games than the expected 25% win rate in the 1 versus 3 setting. Given that there are no explicit interactions, the only other culprit could be the implicit interactions such as coordinations between agents caused by built-in rules (e.g. never trade with the player that is winning the game). Any interactions of such form would affect all agents equally since these agents are of the same type. The large number of games (2000) and the randomisations performed (i.e. the board layout and the agents position are randomised before each game) ensure that the experiment is fair. Nonetheless, we performed a 1 vs 3 experiment where all four agents were the state of the art rule-based agent. We observed that the difference from 25% win rate is always less than 0.3% so we did not repeat the experiment for the other agents.

We evaluate the modified agents against several baselines:

- *Stac* agent—the state of the art heuristic agent which can handle the full game

(Guhe and Lascarides, 2014a);

- *SmartSettlers* agent—the state of the art planning agent which can handle a simplified version of the game (Szita et al., 2010);

- several ablated versions of our agent to evaluate the effects of each modification. We will introduce these and how we refer to them when presenting the algorithms used in constructing our final agent.

In a typical game of Settlers of Catan, a player has access to a limited amount as well as types of resources. Furthermore, some of the resources are finite (e.g. there are only 25 development cards in the deck) and there is a race between the players to use these resources to achieve the winning condition. To avoid situations where the modified agent has a successful strategy only because it focuses on aspects of the game that are ignored by the baseline agent, we also evaluate the performance of the baseline agent versus 3 of the modified agents.

SmartSettlers agents don't trade, so our agents playing against them are not allowed to trade either. Otherwise, the *SmartSettlers* agent would be disadvantaged; limiting an agent's trade capability handicaps it (Guhe et al., 2013). Furthermore, *SmartSettlers* agents are unable to handle the imperfect information so we only evaluate our agent against it on the fully-observable version of the game. We evaluate against the *Stac* agent and against the ablated versions of our agent on the game that contains the complete set of actions.

An important aspect of the experimental methodology is the choice to not introduce a time constrain for the planning agents. The agents developed in this thesis employ what is known as online planning via forward sampling (i.e. extensions of the MCTS algorithm). These methods are expensive so the emphasis is generally on making them more efficient. However, we have not spent additional time on micro-optimising all of our methods. Introducing a time constrain, would result in an unfair comparison in this case. Instead we have individually timed the algorithms and discussed if these have comparable time requirements. Methods which have comparable performance (i.e. win rate) but require more time, would be weaker if time constraints were introduced. This can be inferred from the performance and individual timing experiments, and we also highlight this when discussing the results in text.

The majority of experiments was performed on the "Eddie" computing cluster of the Edinburgh Compute and Data Facility (ECDF). Game simulations with one of our most expensive agents versus 3 *Stac* agents can be run on a normal desktop machine

with an Intel-I5 hyper-threaded processor and 8GB of RAM. However, the 2000 games required for evaluating each modification requires running many jobs on the Eddie cluster and aggregating the results at the end. The Neural Networks were trained on Nvidia GTX TITAN X GPUs, but these could have been trained on CPUs given the small amount of data and size of our networks. Even with the access to such computational power, the amount of exploration one needs to do to highly tune the parameters of complex implementations is an impossible task given our limited time. As a result we have performed a mixture of grid search with a large coarseness over a fixed range specific to each of the agent's parameters, followed by manual tuning at a more granular level in the most promising range.

Chapter 4

Simple Non-parametric Method for Mining the Corpus

Work on similar complex games, e.g. Branavan et al. (2012); Silver et al. (2016), has shown that combining an online planning algorithm with additional sources of information, such as a manual presenting general good advice or example play executed by expert players, can be very succesful. The planning method is bootstrapped and thus biased to explore the areas of the space that are more likely to be promising. In turn, planning can escape regions of local minima and learn corrections to the general advice received given its ability to perform exploration. However, previous work assumes that the advice or previous play are reasonable. On the other hand, our corpus is made of mixed play which it is very likely to contain sub-optimal play. It is unclear if anything useful can be extracted from our corpus. Furthermore, would combining the extracted advice with a planning method result in a stronger player?

4.1 Non-parametric Method for Value Estimation

To answer these questions, we introduce a simple method for mining the corpus that we implemented to learn a single action of the game: the placement of the second free settlement in the initial phase of the game. The inspiration for the first implementation is the well known classification method of nearest neighbours (Duda et al., 2001). Nearest neighbours classifies a sample based on the distance to a predefined and correctly classified set of samples. This set is sometimes referred to as the training set, although the base algorithm doesn't process it or construct any representation that can explain the data distribution. The training set is instead stored and used as it is for classify-

ing of new points. The data is defined as a set of n pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i$ is the representation of the data sample i in some metric space X and y_i is the label that indicates to which class the sample belongs. Nearest Neighbours requires the definition of an appropriate distance metric d (e.g. Euclidean distance) to measure the distances between the representation of the new samples $\mathbf{x}'$ and the correctly classified samples $\mathbf{x}$. The nearest neighbour is the sample $\mathbf{x}_i$ that minimises this distance: $\min d(\mathbf{x}_i, \mathbf{x}')$, $i = 1, 2, \ldots, n$. A modification to the algorithm by computing the majority vote of k-nearest neighbours allows for reducing the variance of the standard algorithm. In general the value is chosen based on the performance of different values on the training data evaluated via cross-validation.

We have chosen to represent both states and actions as vectors of numerical features. The vector $\mathbf{x}_a$ representing an action is constructed via vector difference between the vector $\mathbf{x}_{s'}$ representing the resulting state s' and the vector $\mathbf{x}_s$ representing the state s in which the action is performed: $\mathbf{x}_a = \mathbf{x}_{s'} - \mathbf{x}_s$. Due to the sparsity of our corpus data, we need a set of features that does not necessarily contain the exact description of the state (i.e. coordinates of pieces on the board), but rather specific characteristics or relations (e.g. number of pieces on board, distances between them, access to resources etc). Furthermore, it is very difficult to transpose the actions due to parts of the game configuration varying a lot between games (e.g. hexes on the board are randomised in the beginning). Therefore, our features are intended to capture the abstract properties that best represent the current player's state and focus on what effects his actions may have. These features are chosen based on general knowledge of the game and the set of features was kept small since a low dimensionality aids generalisation. The list of features composing the vectors are described in Appendix B.

In contrast to previous approaches, we have chosen to underline the importance of similarity of both states and actions. Such similarities have been shown to speed up convergence of tabular reinforcement learning such as Q-learning (Rosenfeld et al., 2017). But, the main reason is the sparsity of our data and the complexity of the game as described in Chapter 3. The board is different from one game to another since it is generated by shuffling the 19 hexes and 9 port hexes. The position and order of play are also randomised. It is very unlikely that we will encounter the exact state again, however an action's effects are more likely to be similar since these effects are local and also depend on small portions of the board. In the initial placement for example, building a settlement will provide a player with access to resources based on the 3 surrounding hexes. The exact board may not be encountered, but the same 3 hexes

combination or a subset of this combination will very likely be encountered in a different game. This idea is related to the concept of advantage function in reinforcement learning, which tries to capture the benefits of an action by separating it from the value of the state (Sutton and Barto, 2018; Wang et al., 2016):

$$A(s,a) = Q(s,a) - V(s) \tag{4.1}$$

The difference to other research in reinforcement learning is that we do not compute this advantage function as a function of the state representation $\mathbf{x}_s$, but only as a function of the action representation: $A(a)$ using $\mathbf{x}_a$. Using an action representation permits a richer alternative representation to the abstract one presented in the game rules. Consider again the example of placing a settlement on the board at a specific coordinate somewhere in the middle of the board. Depending on the board description, this action may provide access to three resources or to two resources if one adjacent hex is the desert. If only one of the hexes is different between these two board descriptions then the former is clearly a better action despite having the same coordinate based on the abstract description provided by the game rules. The same action effect could be encountered in a different board setting and with placing a settlement closer to the border of the island. As a result, our action representation offers great flexibility in deciding when to utilise the advice from the corpus since we can use it for generalisation purposes.

Still, reinforcement learning theory states that the action value depends on the state this action was executed in and we should take into account the current state. Returning again to the settlement placement example, placing the second free settlement close to a desert may be beneficial if it is close to the first free settlement, since it may be easier to connect roads and get the longest road award. Such an effect could be encoded in the action representation, but this is tedious and we will slowly start describing an infinite set of interactions. Instead, we will use both state and action representations in our method.

To implement a method similar to nearest neighbours, we need to define an appropriate distance metric. We calculate the relevance of the states and actions by computing their vectors' *cosine distance* (where $\mathbf{x}$ and $\mathbf{y}$ are the vectors describing the candidate states or actions):

$$d(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=1}^{n}\mathbf{x}_i \times \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n}(\mathbf{x}_i)^2} \times \sqrt{\sum_{i=1}^{n}(\mathbf{y}_i)^2}} \tag{4.2}$$

Cosine similarity presents multiple properties that make it a good metric in our case. It is bounded and it handles high dimensionality spaces better than euclidean distance. Furthermore, it is bounded in $[-1,1]$ providing information on when states are similar (value close to 1), opposite (value close to -1) and not alike at all (value close to 0). In Settlers of Catan, the vectors describing the states are in positive space, so their cosine distance is in $[0,1]$. Action vectors are the result of the difference between the resulting state and the initial state, so these could contain negative values for different actions of the game. However, during the initial building phase that we run these initial experiments in, the action representation cannot contain any negative values. This permits using the similarity value to influence the estimation of the value function without any further processing. In the case of negative values, a threshold can be used to ignore pairs from the corpus instead of down voting current actions based on relevance only.

We use the vector of features representing the states or actions stored in the corpus (s_c and a_c) to compute the relevance to the newly encountered state and action (s_n, a_n) as shown in Equation 4.3. The similarity between states and actions is averaged since there is no indication which of the two carries more weight. As future work, one could finetune these weights following an empirical evaluation.

$$rel((s_c,a_c),(s_n,a_n)) = \frac{d(\mathbf{x}_{s_c},\mathbf{x}_{s_n}) + d(\mathbf{x}_{a_c},\mathbf{x}_{a_n})}{2} \tag{4.3}$$

This relevance value is used as a weight when evaluating the new states and actions, s_n and a_n. Equation 4.4 estimates their action value, $Q_i(s_n,a_n)$, using only sample i from the corpus, i.e. using state s_c and action a_c.

$$Q_i(s_n,a_n) = rel((s_c,a_c),(s_n,a_n))U(s_c,a_c) \tag{4.4}$$

We do not estimate the utility of the state–action pairs gathered in the corpus; instead, we trust the decision making of the person who played the action. To reflect this assumption, the utility of every pair in the corpus $U(s_c,a_c)$ is initialised to 1. Even though our corpus contains play from novices and experts, we have observed that human players will win approximately 50% of the games when playing against 3 *original* agents (also confirmed by Thomas (2004)). So the assumption that humans are always better at choosing locations for the initial placement than the heuristic agent is naive, but largely reasonable. If utilities were available, one could potentially weight the utility of a state by the relevance to the states stored in the corpus and the same for actions

before combining the result. Also, another set of weights could be used to decide if the relevance between the states or actions is more important.

Finally, we need to take into account that multiple pairs from the corpus may suggest the same action as the best next move. As a result, the final estimated $Q(s_n, a_n)$ value is averaged over all k suggestions:

$$Q(s_n, a_n) = \frac{\sum_{i=0}^{k} Q_i(s_n, a_n)}{k} \tag{4.5}$$

We want to underline that the value of each state–action pair from the corpus is used only once to influence the most similar pair from the current options. The result is that we are not evaluating all the pairs encountered in the current game, but rather we bias the search towards a smaller part of the game tree deemed good by the information extracted from the corpus. We also do not rely entirely on the advice from the corpus. This bias is used only to guide the initial exploration of the MCTS algorithm and MCTS can learn local corrections given sufficient samples.

4.1.1 Combining with Flat Monte Carlo Tree Search

We have combined this method with a simpler version of MCTS, known in literature as Flat MCTS (see Figure 4.1). This pilot experiment tests only one action in the game, so we want for now to avoid having MCTS (see Figure 2.1) selecting actions based on how good a specific part of a branch continuing from the current node is. We removed the expansion part of the algorithm turning this into a similar approach as Monte-Carlo Search (Browne et al., 2012). Flat MCTS has proven to be very strong in the domains of Bridge (Ginsberg, 2001) and Scrabble (Sheppard, 2002). We also want to keep the benefits of UCT, so our method treats the tree as a one step multi-armed bandit. Such an approach is also known as *Flat Upper Confidence Bounds* (Browne et al., 2012).

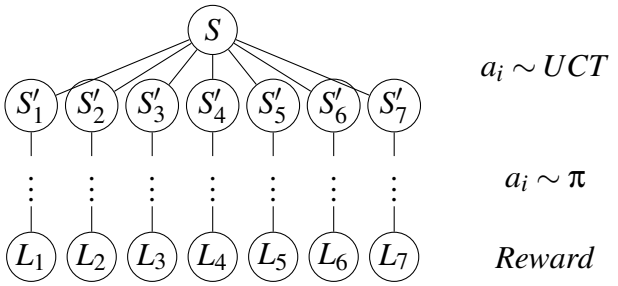


Figure 4.1: Flat-MCTS algorithm. Repeat until a budget limit.

Flat-MCTS with random rollouts doesn't perform any opponent modelling, because UCT is only used in the tree level (i.e. only from the current player's perspective), while the other players take only random decisions during rollouts. To overcome this, we are replacing the random rollout policy with an ε-greedy policy π (with $\varepsilon = 0.2$), in which an action is chosen based on the *Stac* heuristic in $1 - \varepsilon$ proportion of time and uniformly at random in the remaining ε proportion. All players use the ε-greedy policy in the rollout phase. Whitehouse et al. (2013) have shown that existing heuristics can be re-purposed inside MCTS to retain the personality of the original agent and potentially create a stronger player. But the ε-greedy policy will also allow the model to explore the game space as the *Stac* agent's decisions are deterministic.

Finally we have chosen a fixed number of 1000 roll-outs for the Flat MCTS method, with each roll-out terminating at an end state of the game (i.e. the state where the first player acquires 10 victory points). The action selected to be played in the real game is the one that maximises $Q(s,a)$ in the UCT formula (see Equation 2.10), with $C = 1$. We introduce the estimation performed via the nearest neighbour approach in the tree by setting the initial value of $Q(s,a)$ to the value computed in Equation 4.5 (as the value is in the range $[0,1]$) and $N(s,a) = 10$. This means that the value of the node is equal to the value estimated by the non-parametric method after 10 visits. Initialising the number of visits estimates the weight of the seeding in comparison to the estimation performed via sampling by the search method. Note that the UCT algorithm will try the nodes that were not initialised at least once before the seeded value will influence the search.

4.1.2 Results and Analysis

For this particular experiment only, we measured the performance of a modified agent by running 10000 game simulations with 4 players; one of the players is the modified agent and the other 3 are baseline agents (the details of the baseline agents are given shortly). Therefore, a player that is of equal strength to the baseline agent would win 25% of the games. We performed a Z-test to test the significance of win rates against this null hypothesis and we chose a threshold $p < 0.01$. Results between 24-26% are not considered significant.

We ran our experiments using the *Stac* agent briefly presented in Chapter 3 as an upper-bound baseline: i.e. an agent that plays according to a set of sophisticated, but hand-crafted rules. Results against the lower-bound baseline (random player) are not

reported in the table: any of the presented agents win 99% of the games against a random agent. The modified agent was replaced with different versions of the Flat-MCTS model with the intention to test each addition on its own and then assess the joint model. Furthermore, we explore the effects of biasing the search with the evaluation performed by the *Stac* agent. Therefore we have the following modified agents:

1. Flat-MCTS without any prior knowledge;

2. Flat-MCTS seeded with prior information extracted from the *Stac* heuristics;

3. Flat-MCTS seeded with the Corpus suggested policy;

4. An agent that follows the Corpus suggested policies without performing simulations;

Agent 1 samples the space following the uninitialised UCT formula (see Equation 2.10) and chooses the action that yielded the best result during rollouts to play in the real game. This player doesn't use any prior information. Agent 4 builds the tree and seeds in the values computed from the corpus. It doesn't perform any rollouts, but just chooses the action with the highest value. Agent 3 combines these two approaches. Agent 2 computes the values of actions following the *Stac* heuristic agent estimates and only the 5 nodes corresponding to the best 5 actions are initialised. The tree is seeded in the same manner as for the other two players that make use of prior knowledge, by initialising the $Q(s,a) = 1$ and $N(s,a) = 10$. All agents, aside from the last one, perform rollouts using the ε-greedy policy described in the previous section, where the optimal move is chosen by the *Stac* heuristic. These 4 agents choose a single action in the real game (i.e. the placement of the second settlement during the initial set up), while the remainder of the game is played as if they were *Stac* agents.

Agent type	**Win percentage**	**Number of games**
Stac (baseline)	*25%*	*10000*
Flat-MCTS	28.74%	10000
Flat-MCTS seeded with *stac* heuristics	28.48%	10000
Flat-MCTS seeded with Corpus	30.43%	10000
Corpus	19.11%	10000

Table 4.1: Results

Table 4.1 contains the performance in terms of win percentage of each of these agents versus 3 Stac agents. As expected, Flat-MCTS shows a large increase in win rate (3.7%) over the baseline and informing the search with a policy that averages over the related advice from the corpus further increases the performance by another 1.7%. These results illustrate the potential value of mining our human corpus. The increase in performance may seem small, however the model only tests this on one action in what is a very long game (recall from Chapter 1 that average game depth is 11715). Furthermore, the strong rollout policy could also be a cause for such a large increase in performance resulting from applying the Flat-MCTS method. Unsurprisingly, the agent that doesn't run any rollouts performs poorly, decreasing the win rate by 6%. The corpus data is too sparse to be the only source of information for guiding the decision making. Further, without sampling, the agent is not afforded the chance to learn that the chosen move that's based on human observation may have been suboptimal (because the human choices in the corpus were). The method for assessing the relevance presented above is an approximation and averaging over the given advice only partly filters the noise. Without any rollouts to correct any poor suggestions or to differentiate between the suggestions this is not enough.

This experiment has also shown that a simple non-parametric method could be used to improve policies in complex games. In addition, these methods are very flexible and there are many extensions that could be done. For example separating how the utility and relevance of the action are evaluated from those of the state, as previously mentioned. Another option is to include certain thresholds and use only samples that are very similar to the current options. Combining the suggestions with a simple average is also problematic and initialising the utility of the samples from the corpus based on our confidence in the player's ability could also further improve the model. Mining a very small corpus of examples has also proven successful, but only for initialising a planning method. It is very unlikely that the advice from such a small set of examples could be used on its own to find solutions to such complex games as Settlers of Catan.

Unfortunately, there are a few drawbacks that are problematic for extending to the full game. First of all, such a method is very slow and expensive. It is also well known that non-parametric methods require a lot of memory to store the training data. We have tested the performance of the approach on a typical desktop machine (Intel-I5 hyper-threaded processor with 4GB of RAM). The JSettlers environment runs 10000 games with 4 *Stac* players in approximately 1 hour on a typical desktop machine. Running the 1000 simulations required for the Flat-MCTS method with the JSettlers

environment takes approximately 5 minutes, which is too slow for an online method. The seeding method requires less than a second to finish and can be further improved, however there are a total of only 390 samples for the initial placement phase in the corpus. There are 7k samples for the normal phase of the game, where players spend the majority of the game. We could decide to use only part of the corpus instead of the whole, similar to the well-known k-nearest neighbour algorithm, but iterating over the whole dataset to decide on the k samples is still required. Moreover, deciding which k samples to use based on relevance alone is not sufficient and we would need to approximate the utility of the moves in the corpus beforehand. Finally, the abstraction may be well suited for the initial phase of the game and cosine similarity has proven a good metric, but it is not clear if this will extend well to the full game.

4.2 Conclusion

We applied a nearest neighbour approach to learn a policy for one of the most important actions in the game. The results have shown that the corpus data contains some useful advice but this method is not sufficient on its own to play optimally. Due to the amount of data available combined with a naive distance metric, the method cannot generalise to unseen situations even for a single action. This raises some concerns to how well this will extend to the full game. In addition, this approach is very expensive and approximations to this method would be difficult without evaluating the corpus data beforehand. Performing an accurate evaluation is itself a very challenging if not impossible task given the nature of our corpus and the game's stochasticity.

The results pointed out the benefits of planning at decision time. MCTS has proven to be highly successful, both on its own and to improve on the advice from the mining method. It may be that, given our corpus, we can only extract general advice and we will never be able to extract a policy that is able to generalise to the full game and create long sequences of actions. As a result MCTS is a crucial part of the final agent. Even though the rollouts handle the full game, we have only applied the tree phase of the algorithm on a single action in the observable phase of the game. Therefore, we focus on extending MCTS to handle trades and imperfect information in the next two chapters, before we combine with a different model of extracting a policy from the corpus.

Chapter 5

Monte Carlo Tree Search in Observable Settlers of Catan

In this chapter, we focus on the MCTS extensions needed to overcome the challenges encountered in the observable version of Settlers of Catan (i.e. a version where all players can see each other's resource cards). As discussed in Chapter 3.2, these are mainly due to a large branching factor where (approximate) methods for estimating their utility assign them largely the same values (as we'll see shortly). In Settlers of Catan, trading is the main culprit and, in addition to increasing the difficulty in evaluating the available actions, it generates a cyclic behaviour as the effects of trading can be easily undone. This results in a tree depth larger than ever treated before in similar work on complex board games. The experiments and analysis presented in this chapter increases our understanding of the game and demonstrate that trading is the most challenging aspect.

First, we evaluate if the standard MCTS implementation can scale to a game of such size and what are its limitations. We present a few well-known approaches to increase its efficiency: parallelisation and sharing statistics in the tree. Despite their well-known benefits, our experiments show that these improvements are not enough. The goal of this chapter is to evaluate if we can exploit the structure of the game (see ESH in Description 1.2) to address the challenges presented by trading. We adapt MCTS to reason over an abstract version of the game (i.e. action types) to increase the efficiency and strength of the agent. Finally, we address the thesis objective to extract useful policies from a corpus under the low-resource constraint. Therefore, we learn a simple policy for the abstracted game and modify how MCTS evaluates the leaf nodes. Following the ESH, we evaluate if integrating preferences with action legality

provided by the game rules increases the performance of the agent.

In order to remove the need to reason over hidden information in Settlers of Catan, we make the players hands visible to every other player and treat the action of drawing a development card as a chance event that informs every player of the outcome. Given this modification, we will limit our discussion to the MDP framework in this chapter (Chapter 6 will present a model that learns the full version of the game, which incorporates the partially observable states and actions).

5.1 Monte Carlo Tree Search

We will now present MCTS in more detail following the brief description in Chapter 2. It is a planning method for finding optimal solutions that combines random sampling of the decision space with the precision of a search tree (Coulom, 2006; Browne et al., 2012). The high-level structure is presented in Algorithm 1. It performs forward search to evaluate the current state of the game using statistics from simulated experience. This experience is generated using a model of the environment. Such simulated experience is preferable to avoid performing exploration in the real environment, especially in a competitive setting.

create n root node;
while *within computational budget* **do**
  $n \leftarrow$ TREE_POLICY(n);
  $r \leftarrow$ ROLLOUT_POLICY(n);
  BACKPROPAGATION(n, r);
end
return *BEST_ACTION(n)*;

Algorithm 1: The basic MCTS algorithm

MCTS only needs access to the samples and can work with "black-box" models. Unfortunately, like any model-based learning method, it is not guaranteed to find the optimal policy if the game model is imperfect. In our application, we have access to an exact model of the environment, so the search will not be affected by the approximation error resulted from learning the environment. But, inferences about which action is optimal is still approximate and prone to error because, while sampling converges on true solutions in the limit, finite resources mean you may not ever reach those limits. Furthermore, Settlers of Catan has a big branching factor and it is very likely that

only a set of the available actions are worth exploring. Focusing the search towards promising areas of the space makes better use of the available budget. This is the motivation for using a policy extracted from the corpus of human play and at the same time for modifying MCTS to reason over action types as we will show in this chapter.

MCTS presents additional characteristics that are beneficial to the problem addressed in this thesis. The most important ones are: *(a)* it is an anytime algorithm, and *(b)* it builds an asymmetric tree (Browne et al., 2012). The thesis is concerned with finding policies in complex games given a strict computational budget and limited human resources, so being able to stop the algorithm at anytime and return an approximation to the optimal policy is crucial. Building an asymmetric tree suggests that the algorithm is able to focus on the most promising area of the game tree in time. This is a characteristic we would like to exacerbate when combining with prior knowledge taken from previous play. MCTS' policy also improves in time due to its bootstrapping behaviour. This is achieved by iteratively adding new nodes to the tree in combination with a policy that offers a good balance between exploring less visited branches and exploiting promising ones such as UCT. The standard tree policy is given in Algorithm 2. EXPAND(n) adds the children nodes to node n and returns one child sampled uniformly. SELECT_UCT(n) uses Equation 2.10 to select the child node that results from executing the action a with the maximum UCT value in the state s corresponding to node n. We make the reasonable assumption that game states and tree nodes are somehow linked, and that the game state can be retrieved given access to the tree node.

```
while n is non-terminal do
    if n is a leaf node then
        return EXPAND(n);
    else
        n ← SELECT_UCT(n);
    end
end
return n;
```

Algorithm 2: Standard TREE_POLICY

The rollout step follows and it utilises the game model G (i.e. access to step(s,a) function, reward function and knowledge of when the game has ended) to play the game until a terminal state is reached as shown in Algorithm 3. The policy ($\pi(s)$)

chooses an action uniformly at random from the list of legal actions and is called the default policy. Finally, the statistics stored in the nodes are updated based on the result of the rollout. When the computational budget limit is reached, the best action is executed in the real game (i.e. BEST_ACTION(n)). There are multiple methods for choosing this action, e.g. max child, robust child (Browne et al., 2012). For our implementation we select the action that yields the highest reward average, i.e. $\max Q(s,a)$, to be played in the real game (see Appendix C for an alternative approach for the BEST_ACTION(n) function).

$s \leftarrow n$.getState();
while *s is non-terminal* **do**
  $a \sim \pi(s)$;
  $s \leftarrow G.step(s,a)$;
end
return *reward for s*;

Algorithm 3: Standard ROLLOUT_POLICY

MCTS has been applied to two different environments: standard MDP environments where illegal actions are allowed but the environment doesn't advance, or multi-player game environments where illegal actions are generally not allowed. Since Settlers of Catan is a multi-player board game, we don't allow the execution of illegal actions. We briefly show here that illegal actions have no effect on the expected return in an MDP given some conditions. Therefore, excluding illegal actions would reduce the space a planning method needs to search without modify the optimal policy.[1] The two conditions are that the reward function provides highly sparse rewards, i.e. $R = \{0,1\}$ received at the end of the game, and that the game is an undiscounted MDP, i.e.$\gamma = 1$. Both conditions are true in our implementation. Illegal actions can be defined in an MDP using a deterministic transition function, such that $\sum_{s' \neq s} P(s'|a,s) = 0$ and $P(s|a,s) = 1$. This definition requires that there is at least one legal action or the game cannot continue. In general, this requirement is satisfied by the game rules which make sure that states without legal actions do not exist or that the game ends in such a state.

[1] We make the assumption here that the agent is rational and acts to maximise it's expected return.

Given these conditions, the Bellman equation for an illegal action a^i is:

$$\begin{aligned} Q(s,a^i) &= \sum_{s'} P(s'|a^i,s)[R(s') + \gamma \max_{a'} Q(s',a')] \\ &= \sum_{s'} P(s'|a^i,s)[R(s') + V(s')] \\ &= \sum_{s'} P(s'|a^i,s)V(s') \\ &= V(s) \end{aligned} \tag{5.1}$$

We can drop the reward function $R(s')$, since players cannot finish a game with an illegal action so it will always be 0 for illegal actions. Given this equation, the expected return over two trajectories, where one contains several illegal actions a^i and the other only legal actions a^l (e.g. $(s_1,a_1^i),(s_1,a_1^i),(s_1,a_1^l),\dots s_n,a_n^l$ versus $(s_1,a_1^l),\dots s_n,a_n^l$), would be the same. This means that illegal actions only increase the length of trajectories so it would be beneficial to ignore illegal actions in this case. Luckily, we have access to a function provided by the game model that can be used to recognise illegal actions during the planning phase:

$$\mathbb{I}(s,a) = \begin{cases} 1 & \text{if } a \text{ is legal} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

We can change the Bellman equation by multiplying each action value with the output from this function that checks action legality: $Q^l(s,a) = \mathbb{I}(s,a)Q(s,a)$. This ensures that the value of illegal actions is $Q^l(s,a^i) = 0$. The only time illegal actions would still be considered is if $\arg\max_{a_i} Q^l(s,a_i) = \arg\max_{a_l} Q^l(s,a_l) = 0$ (breaking ties randomly) since the value of legal actions is obviously unchanged. This means we can still generate trajectories with illegal actions if there is no better option. MCTS applied to game environments takes this further and disallows illegal actions altogether. This would be equivalent to modifying the function that checks action legality as following:

$$\mathbb{I}(s,a) = \begin{cases} 1 & \text{if } a \text{ is legal} \\ -\infty & \text{otherwise} \end{cases} \tag{5.3}$$

To conclude, we do not allow illegal actions in the tree because these only increase the size of the tree, but we prefer the first definition of the action legality function when computing the Q-values. Firstly, the second option is incorrect because it is

equivalent to saying that executing an illegal action would cause the agent to lose the game. In reality, a server (or the peers in human games) would inform the agent that the action is illegal without any consequences. So nothing happens if an illegal action is executed and the game continues normally.[2] Secondly, we use this function to compute a different quantity in the partially observable version of the game and the former is more appropriate in that setting as we will show in Chapter 6.

As previously described, there are a set of characteristics specific to complex games that Settlers of Catan features. Accordingly, we add a set of standard techniques to the base MCTS algorithm to address the game's non-determinism and multi-player characteristics. Sections 5.4 then presents further novel techniques that tackle the large branching factor and depth of the game, the cyclic behaviour that arises from simulating the game, the varied cardinality of the sets of actions that are of a given type (we explain why this cardinality is problematic unless it is tackled head on), and extracting conditional preferences over types from the corpus.

MCTS can easily be extended to multi-player games by keeping track of the statistics for each player in the nodes and modelling the turn change. Selecting an action in the tree level is done by selecting the maximum UCT value of the player whose turn it is. Non-deterministic actions can be handled by introducing chance nodes, where the subsequent move is a nature move. These select the outcome of the player's action based on a distribution that is given by the game rules (e.g. the player rolls a die and nature uniformly chooses one of the 6 outcomes). In addition, the planning model should not have access to information specific to the hidden part of a game state. In Settlers of Catan for example, we need to handle the order of development cards available in the deck. This can be achieved by shuffling the deck before each MCTS iteration and treating the action of buying a development card as an action that has a non-deterministic outcome (i.e. you buy the development card, but then nature chooses which card you bought).

Some games present actions that may have a very large set of resulting states. In Settlers of Catan this sometimes happens with the discard action, where the number of resulting states is equal to all the possible combinations of resources a player can discard. Given that our search budget is very limited, the number of iterations may not be enough to make an informed decision. As a result, we treat the discard action as a chance node with a uniform distribution over the space of outcome states when

[2] In human games attempting illegal actions could be a strategic move that misinforms the opponents of the player's intentions, but we consider a tightly controlled environment where we do not allow such misdirections.

the player has to discard more than 8 resource cards. If an agent must discard so many cards, it already has access to abundant resources and a random discard will barely harm it. This situation is also rarely encountered so this check only ensures that experiments and planning do not take too long.

Finally, the game tree of a complex game usually contains multiple duplicate nodes. These duplications could arise from either when the same state is encountered multiple times in the same trajectory through the tree or when the same state can be reached via different paths through the tree. It is best to reuse the information we have on these states and this is achieved via a transposition table (Childs et al., 2008), where the statistics are attached to the game state and multiple nodes could map to the same state. To avoid impairing UCT's ability to balance between exploration and exploitation, we only update a node once per iteration no matter how many times it was accessed in that iteration. The alternative is known as every-visit search (Sutton and Barto, 2018) which we have not evaluated in our experiments.

In Section 5.2 we present how we parallelised MCTS. In each of the subsequent subsections, we analyse the modifications we brought to the standard MCTS by comparing our final best model with a version of itself that does not contain the modification. All of our MCTS agents are multi-threaded as described in Section 5.2 and contain the standard techniques discussed above. Section 5.4.4 presents an empirical evaluation of our best MCTS agent against the current state of the art models in Settlers of Catan. The last section analyses the effect trading has on the performance of the agent and the difficulty of the game.

5.2 Parallel Monte Carlo Tree Search

MCTS is easily parallelisable. We exploit this since we wish to balance the need of sampling the game space to yield decent strategies with the need to decide on the next move within a time interval that human opponents would tolerate. There are three alternatives: leaf parallelisation, root paralellisation and tree parallelisation (Chaslot et al., 2008a).

Leaf parallelisation performs multiple parallel roll-outs starting from a leaf node and backpropagates all the results through the tree once all have finished. It is the simplest to implement, but has a few disadvantages. First of all, the algorithm needs to wait for all threads to finish before backpropagating the results. So it is inefficient in its use of resources. Secondly, it is difficult to decide on the exploration term in UCT

if the statistics are incremented in steps greater than 1.

Root parallelisation involves replicating the whole tree a number of times equal to the number of threads used. The results are combined at the end of the search, before deciding on the next move to make in the real game. It generally performs better than leaf parallelisation, but it requires a good combination of the final results. It is memory inefficient as it stores multiple trees and it duplicates a lot of the effort.

Tree parallelisation addresses the above issues by keeping a single copy of the tree and using mutexes to lock the branches accessed by the threads. The locks are global locks, where the whole tree is locked, or local locks, where each node requires the accessing thread to get a lock. The latter has a better overall performance. It is also more efficient since threads do not lock branches or sub-trees, so it is less likely that threads will wait for lock releases. We have chosen a tree parallelisation with local mutexes due to its advantages over leaf or root parallelisations. A lock-free version implementation could be implemented (Silver et al., 2016; Chaslot et al., 2008a), but it would probably provide a minor improvement in speed while sacrificing a number of results. Our agents run a relatively small number of iterations compared to other applications of MCTS (e.g. AlphaGo Silver et al., 2016), so we wanted a synchronised approach to avoid any overwrites. To reduce the overhead caused by the synchronisation, we are also using virtual loss to discourage multiple threads from following the same path through the tree.

5.2.1 General Performance

We will briefly mention the performance of the multi-threaded version of our best MCTS agent on a server with Intel Xeon quad-core CPUs at 3GHz frequency. This agent requires a single CPU and 4GB of RAM. Multiple identical CPUs are used when the number of threads is greater than 4. The time it takes to run 10k iterations as presented in Table 5.1 can be tolerated by human opponents. As expected, the more iterations are run, the more useful it becomes to increase the number of threads. We didn't evaluate the effect of the number of threads on the agents' win rates, so we fixed this number to 4 for all the MCTS agents in this thesis.

The times presented above represent how long our best MCTS agent requires to plan the first move at the start of the game, where the rollouts are the longest. As the game progresses, the length of the rollout diminishes so the planning time reduces. As a result, we decided to measure a small set of statistics over 10 games in which our

Iterations	Threads			
	1	4	8	16
10k	7518	2701	1747	1418
30k	21563	7548	4428	3954
40k	27302	9995	6862	5229
50k	36386	12743	8092	6682

Table 5.1: Time in milliseconds spent by the best MCTS algorithm on the first decision of the game averaged over 100 Catan games.

best MCTS agent faces 3 Stac agents. Our MCTS agent performed 10k iterations and used 4 threads. As Table 5.2 shows, the mean and median time are much lower than the times presented above. Despite this, there are other factors that affect the planning time such as branching factor. The maximum of 4 seconds is encountered in the normal phase of the game, where there is a large number of possible trades.

Agent	Min	Max	Mean	Median
MCTS	45	3852	1433	1445

Table 5.2: The planning time in milliseconds of the best MCTS agent across the whole game. Statistics are gathered over 10 games.

5.2.2 Fine-tuning other Parameters

For the sake of space and interest, we present only a partial exploration of the full parameter space. Better models could potentially be discovered if one were to perform a more detailed search of all parameter combinations. We have performed the minimum required optimisation and have analysed two important parameters: the number of MCTS iterations and the C parameter in UCT. The former shows how our algorithm scales in our environment with the increase of the available budget, while the latter is a parameter that defines the exploration that is specific to each environment.

Table 5.3 contains the performance of our best MCTS agent against 3 (state of the art, hand-crafted, rule-based) Stac agents when varying the number of iterations. Even though the improvements get smaller, these are still noticeable even after 30k iterations; but obviously increasing iterations comes at the expense of increasing the decision time. We aimed to have a low decision time (so that human opponents will

find our agents tolerable to play against) and we have fixed the number of iterations to 10k given the time it takes to run these. The large number of experiments and the computation time per move that can be tolerated by a human player (see Table 5.2) are the main reasons for choosing 10k as the iteration limit and the number of threads to 4. The best value for the exploration parameter C was 0.5: higher values would reduce the performance of the agent. See Appendix C for the effects of parallelisation and C parameter on performance. All the unbiased MCTS agents presented in this thesis used these values for the parameters, unless otherwise specified.

5k	10k	20k	30k	40k	50k
18.8%	36.2%	45.75%	55.95%	59.4%	61.01%

Table 5.3: Win rates of the best MCTS agent against 3 Stac agents, while varying the number of MCTS iterations.

5.3 Afterstates

The initial UCT selection criterion was proposed by Kocsis and Szepesvari (2006) as an application of the Upper Confidence Bounds (UCB) algorithm to trees. UCB is used for selecting the next arm to pick in bandit based problems, therefore the natural translation is to apply UCT to select the next action, instead of the next child node (i.e. state). The literature has since branched out in two methods: the one presented in Chapter 2, which selects the *actions*, and a method for selecting the outcome *state* as shown in equation 5.4. $V(s')$ is the outcome state value estimated as the number of wins out of the number of simulated plays when action a was chosen in state s and leads to s', $N(s)$ is the number of times s was encountered before and $N(s')$ is the number of times the outcome state s' was encountered. These two approaches would be equivalent if the nodes in the tree are unique. This is typically not the case in complex games, and Settlers of Catan is no exception (for instance, an agent can exchange resources with another player and then exchange them back again).

$$UCT(s,a) = V(s') + C\sqrt{\frac{\ln N(s)}{N(s')}} \tag{5.4}$$

The resulting state immediately after executing an action is known in the literature as an *afterstate* (Sutton and Barto, 1998) or a *post-decision state* (Szepesvari, 2010)

(see Figure 5.1 for a simple illustration). These states account for non-deterministic actions since they represent states immediately after a decision was taken but right before any of the stochastic effects are performed (i.e. $P(s'|a,s)$). In addition to separating the deterministic effect from the stochastic effect of an action, the state space (therefore also the afterstate space) is much smaller than that of the state–action pairs. As a result, using the value of the states could be more economical and efficient than using the action value (Szepesvari, 2010). Another benefit of afterstates is that different state–action pairs could produce the same outcome state, so their values must be the same (Sutton and Barto, 1998). Computing the action value using Equation 5.4 allows the results of the rollouts following either of the two state–action pairs to be shared.

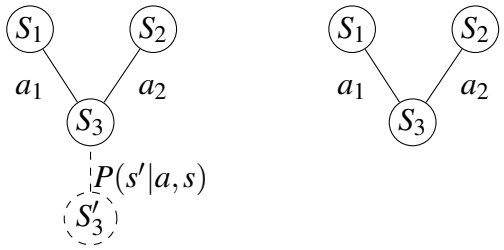


Figure 5.1: Simple example of afterstates or post-decision states. The first graph from the left accounts for the stochastic effect or other unknown effects of the environment (including the opponent model). Node S_3 from the first graph is the state prior to the stochastic effect of the action. Two different non-deterministic actions, executed in different states, could have the same afterstate if their set of successor states and transition probabilities are the same. The second graph shows the simplified case when the actions are deterministic.

We now present an empirical evaluation of the effect of afterstates. We compare the performance of our best MCTS agent with a version of itself that does not use afterstates. The model that uses afterstates as in Equation 2.10 is referred to as *MCTSA*, while the model that does not use afterstates is *MCTS*. We have re-tuned the exploration parameter C of the *MCTS* agent and observed that the best value for this agent is 2. Table 5.4 outlines the large benefit of afterstates. The table contains the performance of the modified agents that are specified on the first column of the table. To clarify, MCTSA wins 36.2% of the games versus 3 Stac agents, while the MCTS agent wins only 3.42% of the games against 3 MCTSA. The MCTS agent is much weaker than both the baseline Stac agent and its counterpart with afterstates. This result backs up the hypothesis that using the value of the outcome state during UCT calculations

can significantly reduce the complexity of the space. Every MCTS agent presented in future experiments use Equation 5.4 so we avoid mentioning afterstates in their names.

Modified	Baseline		
	Stac	MCTS	MCTSA
MCTS	9.23%	–	3.42%
MCTSA	36.2%	71%	–

Table 5.4: Win rates of the MCTS algorithm with and without afterstates, collected over 2000 games.

5.4 Exploiting Domain Structure

Despite MCTS characteristics that make it a better choice over the alternative reinforcement learning methods, it still degrades with the length of the planning horizon because the complexity of the planning problem increases exponentially (this is known as the curse of history). One solution for handling this problem is to define a hierarchy subroutine, also known as Macro actions or options which are composed of a sequence of primitive actions. Dietterich (2000) introduced the MAXQ framework which recursively decomposes the overall value function into a collection of value functions for the subtasks, subsubtasks and so on. Experiments on the taxi problem show that the MAXQ model can converge much faster than Q-learning. Vien and Toussaint (2015) have similarly extended the MCTS planning framework, via a hierarchy of pre-defined subtasks that in turn reduce the set of policies that can be considered. As a result, the computational cost shrinks considerably, with the effect proportional to the length of the macro actions (He et al., 2010).

These hierarchical models assume that subgoals can be easily identified by the developers. Dietterich (2000) provides a detailed discussion on how this approach relies heavily on careful reasoning when designing certain parts of the hierarchical models, such as defining the subgoals or the use of state abstraction. In a game as complex as Settlers of Catan, the smallest mistakes may cause such an approach to fail. There are methods that can automatically generate these subgoals (Stolle and Precup, 2002; He et al., 2010; Chaganty et al., 2012; Vezhnevets et al., 2016; Fox et al., 2017), but using such methods on a vast action space such as that in Settlers of Catan will most likely result in defining an inaccurate game model. Furthermore, macro actions

impose a structure on the task and having an incomplete definition of the possible subgoals limits the possible policies that the agent can learn and the optimal policy may be external to the set of possibilities generated by the limited set of subgoals the agent is aware of. Another limitation of these methods is that they rely entirely on a reward structure that provides feedback during the game. This is not the case in our environment, where rewards are sparse and delayed (only binary rewards are received at the end of the game).

The most closely related work is that of Xie et al. (2014) applied to improving the exploration of a greedy best-first search algorithm. Their method clusters states or nodes based on specific characteristics or heuristics which they refer to as a type-based system. Their type-based system is then used to ensure that the search algorithm doesn't get stuck trying nodes of a specific type. Lelis et al. (2013) have applied a similar type-based system to a tree search algorithm and noticed the modification made the search more efficient. Unlike this previous work, we cluster the actions into types rather than the states. We apply this system to the rollout phase of MCTS and notice efficiency gains as well as performance gains measured in an increase in the agent's win rate. Furthermore, clustering actions allows us to extract a high-level strategy that further improves how the rollouts evaluate leaf nodes. Such a system would not be possible with a state type-based system due to the state space being much larger than the action space.

There are also well developed ways of dealing with a large branching factor of some complex games in the MCTS literature, such as grouping the moves according to some characteristic and combining their statistics in the tree. This approach is known as Move Groups, in which the moves are clustered based on a specific characteristic defined in advance (Childs et al., 2008). Another very similar method is to group the chance actions that can be executed from one state into a single group, which is called node-groups (Jouandeau and Cazenave, 2014a,b). This approach works well when there is some correlation between the moves such that the information on one move gained during the search generalises to the other moves belonging to the same group. These grouping techniques have only been applied to the tree level part of MCTS. We also do not share the statistics between the actions of the same type since belonging to the same type doesn't imply that the actions have similar values.

In other complex games, splitting the decision into multiple steps and choosing the order of the steps has greatly increased the performance of the algorithm (Kloetzer et al., 2007; Cowling et al., 2012a,b). This approach requires expert knowledge of the

domain to choose the parts as well as the order these are executed in. Cowling et al. (2012b) mention that the important decisions should be higher in the tree. Since the aim of this method is to also reduce the branching factor of the tree it resembles the Move Group method, but these are two different approaches.

There are also many methods for improving the rollouts in the MCTS algorithm, but these make use of existing rule-based implementations or hand-crafted heuristics to bias the search. Previous methods employed an existing agent to search the game (Dobre and Lascarides, 2015; Branavan et al., 2012) or define their own rules based on expert knowledge (Cowling et al., 2012b; Chaslot et al., 2009). These methods are sometimes referred to as pseudorandom games (Kloetzer et al., 2007). The rules used in previous implementations need to be devised in advance and may not generalise to all cases in the game. Furthermore, the resulting policy is a deterministic policy that could get stuck in local minima. Subramanian et al. (2016) constrained the action space by integrating rules to prevent known bad situations (e.g. avoid ghost in PacMan) during rollouts in MCTS. The authors also combined these with options in the tree phase of the algorithm. The options were generated via crowdsourcing (Subramanian et al., 2011) while the constraints were created following a detailed analysis of varied human play (Irani, 2015).

In this work we present an approach to sampling the decision space of a complex planning problem, which increases both performance and efficiency. Instead of defining macro-actions or grouping actions based on a specific metric, we exploit the natural hierarchy of the actions as it is defined by the game rules (i.e. Emergent Structure Hypothesis described in Chapter 1 on page 5). We therefore sample from the set of legal action *types* before sampling from the specific actions of the (sampled) type during rollouts in MCTS. This results in a reduced branching factor and a reduced depth by avoiding redundant behaviour. Such an approach has numerous benefits, particularly in large domains where the sets of distinct action types vary a lot in size. For example, it is unlikely to explore the benefits of ending the turn when it is one action in a set of almost 600 legal actions. This approach is related to the Hierarchical Bayes model, where a prior distribution over the hyperparameters yields a more expressive model (Koller and Friedman, 2009). The hierarchy that we have defined is also similar to a semantic hierarchy which can be used to reduce the search space, e.g. in computer vision, such a semantic hierarchy can be exploited to acquire labels more efficiently during annotation tasks (Deng et al., 2014).

5.4.1 Action Types

The base MCTS algorithm relies on Monte Carlo sampling of the space to ensure that an accurate estimation of the current state's value can be performed given a sufficient computational budget. The default sampling policy assumes a uniform distribution over the legal actions given the current state. These actions could belong to the same class of actions for simple games or could belong to one of many classes in more complex games. In the case where there is a single class, or if the number of actions belonging to each class is similar, the uniform sampling over all the legal actions would ensure a sufficiently unbiased estimation. But, if the cardinality of different classes is very different (e.g. the average number of trade moves is 64.8, while the average for ending the turn is 1 in Catan), the resulting policy is more likely to execute an action that belongs to a dominant class. The resulting estimation would be useful only if the opponents in the real game have a similar policy (i.e. one where they tend to perform actions from a dominant class). This is highly unlikely if there isn't a huge benefit to executing one of the actions from the dominant class over those of other classes.

Looking at Settlers of Catan, these types are encountered in what we call the *normal phase* of the game (Dobre and Lascarides, 2017) (i.e. the phase in the game where the player has rolled the dice, and now has a choice of trading, building, buying or playing a development card, or ending the turn). Most of the time is spent in this phase in which players usually have to decide between multiple types of actions. We have copied the table with the list of action types introduced earlier (see Table 5.5). The table also includes the probability of actions belonging to a type being legal, the average number of actions belonging to the type when this is present, and the maximum number of actions for each type. These are computed over all the decisions made in this phase during 10k games. The games were generated using the standard random policy of selecting uniformly at random from all legal actions indifferent of their type. The high values for the trade actions indicates that players have to generally decide between many trade moves and a few moves belonging to other classes. A uniform random policy over this set is much more likely to choose a trade action. This policy is weak since this action type is not sufficient: trades are necessary to quickly gain access to scarce resources, but executing other action types is required to win the game.

Another characteristic that causes difficulties in turn-based complex games is that the end turn action is always present and the type cardinality will almost always be smaller than the other classes. In Go, a simple heuristic can handle this case since it

Action type	Probability	Average	Maximum
Build road	0.0547	5.48	19
Build settlement	0.0331	2.82	13
Build city	0.0122	2.71	5
Buy development card	0.0114	1	1
Trade with opponent	0.9982	64.8	512
Trade with bank/port	0.2361	4.47	20
Play knight card	0.0001	16.16	29
Play monopoly card	0.0010	3.92	5
Play discovery card	0.0005	15	15
Play free road card	0.0023	6.35	34
End turn	1	1	1

Table 5.5: The list of action types during the normal phase of the game, the probability of the type of action being available, the average number of actions when the type is available and the maximum number of actions of each type. These numbers are collected from 10K games with 4 players, where each player follows the default random policy over all legal actions.

is well known that it is generally better to not pass your turn. In Settlers of Catan, ending the turn has a high strategic importance: player's may choose not to execute actions in order to avoid informing the other players of their plans. There are also other situations where ending the turn might be desirable over other actions (e.g. a player can build a road, but may want to keep its resources so it can build a settlement in the following turn). This small cardinality compared to the other types means that the action is unlikely to be tried if other types are also available.

There are many other large complex games, such as most of the multi-player board games or video games (e.g the Civilisation series, Diplomacy, Battlestar Galactica, Monopoly), that have a large branching factor and present a clear structure in the rules of the game. These games have similar challenges to Settlers of Catan as the cardinality of specific action types would be greater than that of others (e.g. negotiations in the Monopoly game).

5.4.2 Extending the Rollout Policy

We now present the formal details. Let T be the set of action types and $t \in T$. a is an action option from the set A_t of actions belonging to type t. n is a tree node and s is the corresponding game state. Algorithm 4 shows how the rollout policy can incorporate a step of selecting the action type based on a policy $\pi_t(T)$ then select the action description based on a policy $\pi_a(A_t)$.[3] In Settlers of Catan for example, road building is a different type of action to city building; but the specific description of such actions include where to place the piece on the board. In the simple case, which is also what we evaluate in our experiments, these two policies select uniformly at random from the available actions.

A uniform prior would be sufficient to address the concerns presented in the previous section. However, this model can easily be combined with an opponent model or a better sampling strategy if one is available, just by training a set of parameters that define the distribution over types and the distribution over action descriptions for a given type. Just as a Hierarchical Bayes model defines richer priors, this separation allows a more expressive opponent model to be implemented and permits more interesting combinations of Monte Carlo planning with data driven models. Another side effect of the method is that it reduces the branching factor of the game: only the actions belonging to one type are listed as available and the number of classes is inherently smaller than the number of all actions indifferent of their type.

$s \leftarrow n$.getState();
while *s is non-terminal* **do**
  $T \leftarrow G.action\_types(s)$;
  $t \sim \pi_t(T)$);
  $A_t \leftarrow G.actions\_of\_type(s,t)$;
  $a \sim \pi_a(A_t)$);
  $s \leftarrow G.step(s,a)$;
end
return *reward for s*;

Algorithm 4: Extended ROLLOUT_POLICY

Table 5.6 shows the effects of introducing the second step in the sampling policy on

[3] The algorithm makes the reasonable assumption that the game developer can provide the functions $action\_types(s)$ and $actions\_of\_type(s,t)$ to categorise the actions into types. The game rules must permit such a categorization as it is the case in Settlers of Catan.

the depth and time of the random rollouts. Despite the extra sampling step, the gains are massive. To illustrate the differences, a game between 4 players, where one is the standard (untyped) MCTS agent and 3 are Stac, finishes in 17 minutes. In comparison, a game with one typed MCTS (TMCTS) agent and 3 Stac agents finishes in under 3 minutes. Note that running 10k iterations of the typed MCTS algorithm (see Table 5.1) is cheaper than just running 10k of random typed rollouts. We believe this is caused by the focused approach of the tree selection policy in combination with the importance of the free initial placement stage. Comparing the rollouts of the 3 MCTS agents (vanilla, SmartSettlers and the typed version), the typed MCTS algorithm presents the most balanced and diverse sampling of the decision space. The standard algorithm prefers trades over all given the probabilities included in Table 5.5, while SmartSettlers has a preference towards building pieces over the other action of the game given the heuristics used by the authors (Szita et al., 2010).

Policy	time	depth
Default	583384	11715
Typed	8530	420

Table 5.6: Comparing the typed sampling against the single step sampling over 10k Catan games. The time spent is in milliseconds. Both methods are single-threaded.

The results presented here show the benefits of the action type hierarchy over the vanilla MCTS method. Due to the expensive planning of the (untyped) MCTS method, we compared it only against our proposed model (TMCTS) and the Stac agent. Also, the legal trades actions have been reduced to only 1 for 1 exchanges for this experiment. Otherwise, the standard MCTS method would take to long to finish the search, given the time required to perform random rollouts as shown in Table 5.6. We have performed this experiment twice: once when the number of iterations was fixed to 10k for both planning methods (Table 5.7); and the second with a budget limit of 1.5 seconds or 50K iterations, whichever is reached first (Table 5.8). The 1.5 seconds budget was chosen based on the average time our strongest agent spends planning (see Table 5.2), and the 50k limit is the upper limit that we used in all our experiments. As before, the modified agents are specified on the first column and the corresponding baseline on the second row.

Modified	Baseline		
	Stac	MCTS	TMCTS
MCTS	22.34%	–	9.58%
TMCTS	36.2%	54.3%	–

Table 5.7: Win rate of the MCTS agents with and without the type categorisation, and a limit of 10k rollouts. Each result is measured over 2000 games.

Modified	Baseline		
	Stac	MCTS	TMCTS
MCTS	6.55%	–	0.81%
TMCTS	27.98%	88.57%	–

Table 5.8: Win rate of the MCTS agents with and without the type categorisation, and a limit of 1.5 seconds (up to a maximum of 50k rollouts). Each result is measured over 2000 games.

5.4.3 Learning a Type Distribution from Human Games

Our end goal is to combine the planning method with a method that extracts a policy from our human corpus. We aim to use this policy to improve both the tree phase and the rollout phase of MCTS. One simple way to improve the rollouts is by improving the policy used to select action types $\pi_t(T)$ over the standard uniform policy. This can be achieved by learning a preference distribution over action types from the game play stored in the corpus as we suggested in Dobre and Lascarides (2017). The simplest approach is to estimate this distribution via Maximum Likelihood Estimation (MLE) which counts the number of times an action type was selected in the corpus over the number of times this action type was legal. We use a function C to represent the counts, a function $type(a)$ to represent the type of the chosen action a and a function $legal(t)$ to indicate that a type t is legal (i.e. there is at least one legal action of type t) or not (there are no legal actions of type t):

$$\pi_t^u(T) = P(T = t) = \frac{C(type(a) = t)}{C(legal(t) = True)} \tag{5.5}$$

A very similar approach was performed by Bitan and Kraus (2017) in Cheat, the main difference is that we use the game rules that define what an action type is while they partition their action space manually. Bitan and Kraus (2017) have an abstract

representation for each action to avoid including the full space of possibilities: instead of including the exact value of the rank claimed by an opponent, they cluster all higher ranks and all lower ranks together. Given the game's termination conditions, they are also able to run rollouts in this abstracted space. Neither the abstraction nor running such rollouts is possible in Settlers of Catan where we need the exact state description to access the end game conditions. Another major difference is that they learn this distribution by counting the total number of times an abstract action was selected without taking into consideration if the action was legal or not. We found this rather odd since the authors take into consideration the possibility of actions being illegal and normalise accordingly when performing the actual rollouts.

Ideally, one would like to also condition on the state representation, such that this preference would resemble an action value function $Q(s,a)$. It is clearly impossible to use a tabular representation in a game of the size of Settlers of Catan when we have access to such a small set of samples (60 games). We used a non-linear function approximation to condition this Q-value estimation on a state representation in Chapter 7. For now we want to include any additional information without conditioning on aspects that would clearly result in an extremely sparse space such as states or sequences of actions. None of the previous research has considered learning a policy over action types as well as conditioning on the set of n legal action types given the current state s. We use a function $\Gamma(s) =< legal(t_1), legal(t_2), \ldots, legal(t_n) >$ to represent this set of legal types. Therefore we modify the previous estimation in Equation 5.5 by using a list that contains the legal types as a condition and counting the number of times a type was selected when this condition was true in the corpus:

$$\pi_t^c(T,\Gamma(s)) = P(T = t|\Gamma(s)) = \frac{C(type(a) = t)}{C(\Gamma(s))} \tag{5.6}$$

In words, this approach captures the fact that a human player's policy includes preferences of executing certain types over other types when both are legal. The unconditioned distribution may be very skewed towards the more common types that are more likely to be legal and at the same time more likely to be executed in a game rather than preferred. Such a skewed distribution will not reflect the fact that certain types that are known to be good by players are less likely to be encountered in a game. A simple example in Settlers of Catan would be to consider how often the player selected to build something over trading when both actions are available. Trading is a means to acquire the requirements for executing other actions which in turn will get the player closer to winning the game. The conditioned distribution encapsulates these preferences as well

as others such as preferences (or possibly indifference) of building settlements over building cities which such a conditioning can capture. These preferences can be seen as a weak form of defining options or macro actions. However the preferences are not compulsory milestones as it is the case with defining sub-goals, overcoming the issues we presented earlier with enforcing a structure. Avoiding sub-goals is especially useful in a multi-agent environment, where these goals may not be achievable due to agents racing for the limited resources. Another benefit of our approach over pre-defining macro actions, is that these preferences can be easily extracted automatically from a small corpus of mixed play, while learning macro actions is a very challenging task.

Capturing user preferences is known to be very useful in decision making and a very popular method for representing them is a Conditional Preference Net (CP-Net) (Boutilier et al., 2004). Simple preferences can be represented as: $a \succ b$ (a is strictly preferred over b), $a \succeq b$ (a is equally or more preferred to b) or $a \sim b$ (the agent is indifferent to either a or b). CP-Nets are graphs which can represent more complex preferences by slowly incorporating more variables based on an ordering that shows how parent variables affect preferences over other variables. Therefore a CP-Net can represent conditional preferences of the form $a : b \succ c$ (if a is true then the agent prefers b over c). It is obvious that Equation 5.6 learns a probabilistic version of a shallow CP-Net, known in literature as Probabilistic CP-Nets (PCP-Net) (Bigot et al., 2013). The learned PCP-Net is 2 levels deep: the first level provides preferences of the form $a \succ \neg a$ (agent prefers executing action a over not executing action a), while the second level contains preferences of the form $ab : c \succ \neg c$ (if a and b are also legal, then the agent prefers doing c over the other actions). The benefits of PCP-Nets is that they account for possible noisy preferences or when the whole set of variables that affect the user's preferences is unknown. In our case, a probabilistic representation accounts for the fact that there are multiple players that generated the data and we collapse this to a single preference representation for a standard player. There is some work in combining preference learning with reinforcement learning (Fürnkranz et al., 2012), but none of the previous approaches integrate conditional preferences as well as combine these preferences with a type-based system.

Despite the fact that the action type space is small, our corpus is still highly sparse and we still do not have enough examples to learn a good estimation for each possible condition. Therefore we learn both the unconditioned and the conditioned policies and when we encounter an unseen condition or we do not have any counts of the action being selected when the conditioning is true, we fall back to the unconditioned case.

Due to time limitation, we have not explored any smoothing strategies (other than the backoff strategy just given) to account for fewer or no counts for certain types. Our data is sufficient to at least have some counts for every type in the unconditioned policy. It is worth mentioning that we normalised the two policies to sum to 1 since the types are not always legal.

We have run an empirical evaluation where we pitched these modified agents versus 3 Stac agents. In Table 5.9 we have 3 modified agents:

- *uniform* which uses the typed rollouts with a uniform policy over types $\pi_t(T)$;
- *unconditioned* which uses the typed rollouts with the policy over types $\pi_t^{\mu}(T)$ learned using Equation 5.5;
- *conditioned* which uses the typed rollouts with the policy over types $\pi_t^{c}(T)$ learned using Equation 5.6, and falling back to $\pi_t^{\mu}(T)$ from Equation 5.5 when the condition was not encountered in the corpus (i.e. $C(\Gamma(s)) = 0$).

Modified	Baseline			
	Stac	uniform	unconditioned	conditioned
uniform	36.2%	–	23.60%	17.60%
unconditioned	30.85%	25.00%	–	17.22%
conditioned	41.45%	34.40%	35.34%	–

Table 5.9: Win rates of the TMCTS agents while varying the distribution over types; each result is collected over 2000 games.

All 3 agents are able to easily win when pitched against 3 Stac agents, but sampling from the unconditioned distribution yields the weakest result (30.85%). In fact, this approach is not even able to perform significantly different to the baseline performance when playing versus 3 agents that use the uniform distribution to sample action types, winning exactly 25% of the games. The agent using the conditioned distribution is the best performing agent since it easily defeats every other agent, and none of the other agents are able to win over 25% of the games when playing versus 3 agents of this type. As before we will integrate reasoning over types in the rollouts in all our future MCTS agents. However, we want to be able to evaluate the benefits of future modifications independent of those brought by learning the distribution from the corpus. As a result we will use the uniform distribution over types, unless otherwise specified.

The win rate of agents shows that certain modifications are improving the agent, but it does not provide any indication as to why these modifications succeeded. As shown in Figure 3.2, analysing game play is very challenging, therefore cherry-picking state–action pairs from the experiments may provide a biased explanation. Instead, we take advantage of the structure of the environment (following the ESH introduced in Description 1.2) and indicate how the average behaviour of the agent changes following the modifications we make to the algorithm. For example, we can show what type of actions the agent prefers as the percentage of that type the agent executed out of the total actions the agent executed. In this analysis, we only include build road, build settlement, build city and play development card actions. In the interest of clarity, we do not include trading, stealing, discarding or rolling dice actions. However, we include other statistics: average number of times the agent achieved the largest army or longest road awards, and the average number of resources the player received from the roll dice action. The average number of times the agent received an award can be over 1, since the player may have lost and regained the award multiple times per game.

Table 5.10 contains one such analysis for the experiments presented in Table 5.9. These statistics show how different the effects of the unconditioned policy are compared to the conditioned one. The conditioned policy biases the agent towards building more cities (12.96% from 10.94%), and dramatically reduces the focus on roads (50.93% from 52.39%) including aiming for the longest road award. On the other hand, the unconditioned policy increased how many roads the agent built more than the building of settlements and cities. However, building cities and settlements is known to be a good strategy as it is the only way to access the required resources from dice rolls. As can be seen, the production of resources from dice rolls is increased when the agent uses the conditioned rollout policy (52.68) over the unconditioned policy (51.53).

The policy represented by the shallow PCP-Net learned via MLE is a very quick method for integrating prior knowledge into rollouts. This shallow and quick method for utilising human data was effective, even though we had access to only very small amounts of human data, namely 60 games. Table 5.11 contains a comparison of the conditioned distribution to the uniform distribution over types. Since we want to keep the agent decision time low, a quick heavy rollout is preferable to methods that include complex heuristics or make use of Neural Networks. In fact the need for quick rollouts is well known and there is a lot of research in this area. For example, Xiao and Müller (2017) use a Factorization Bradley-Terry model to learn the interaction strength between the features used to define the legal actions. AlphaGo (Silver et al., 2016) uses

Statistic	Agent		
	uniform	unconditioned	conditioned
Avg number of actions	14	13.78	13.76
Percentage build road	52.39%	53.07%	50.93%
Percentage build sett.	12.55%	13.66%	14.81%
Percentage build city	10.94%	11.46%	12.96%
Percentage play card	24.13%	21.81%	21.30%
Avg LA count	0.28	0.23	0.25
Avg LR count	1.04	1.01	0.98
Avg rss. from dice	52.27	51.53	52.68

Table 5.10: Statistics for the modified agents from Table 5.9 when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic). For example, the uniform agent builds 52.39% roads from the 14 pieces it builds on average per game.

a smaller Neural Network that represents a policy during the rollout phase of MCTS.

Agent	Min	Max	Mean	Median
uniform	45	3852	1433	1445
conditioned	59	4149	1470	1477

Table 5.11: The planning time in milliseconds of the best MCTS agent while varying the type distribution. These values are measured over 10 games.

A possible extension is to use the distribution learned via MLE as a prior that is updated during planning or during the real game play if we want to model the opponents. Rosman and Ramamoorthy (2012) presented an interesting approach to accelerate reinforcement learning by defining priors over actions conditioned on states or observations in a tabular representation. The authors used the prior to perform a more informed exploration step in an ε-greedy control policy. They use a Dirichlet distribution and update the counts only when the action was chosen in a greedy manner. In this way the distribution averages the Q-functions and bootstraps exploration with this knowledge. The extension of initialising the counts according to a set of trajectories followed by updating it during learning was actually proposed by the authors for learning a prior

over actions (Rosman and Ramamoorthy, 2012) instead of action types.

5.4.4 Empirical Evaluation against the State of the Art

Table 5.12 shows the performance of our uniform (typed) MCTS agent (TMCTS) against the two current state of the art agents, Stac and SmartSettlers (see Section 3.5 for a description of the two agents). As a reminder, SmartSettlers cannot trade so our agent does not trade with opponents either when playing against SmartSettlers, while trading is allowed in games versus Stac agents. Table 5.12 shows the performance of the modified agents which are specified on the first column of the table. To clarify, TMCTS agent wins 36.2% of the games versus 3 Stac agents, while one Stac agent wins only 15.18% of the games against 3 TMCTS. Evaluating the agent both ways guards against the possibility of the modified agent winning only due to having a different policy to the three baseline agents. To level the playing field between the two planning agents, we aimed to limit the time each agent can take to deliberate before executing an action. Unfortunately, SmartSettlers does not contain an option to set this limit, so we had to limit the number of rollouts instead. 10k rollouts for our agent takes approximately the same amount of time as 2k rollouts for the SmartSettlers one.

Modified	Baseline		
	Stac	SmartSettlers	TMCTS
Stac	–	–	15.18%
SmartSettlers	–	–	18.67%
TMCTS	36.2%	36.39%	–

Table 5.12: Win rates of the TMCTS agent and the 2 state of the art agents: Stac and SmartSettlers. Trading is allowed in the games against Stac, but not in the games against SmartSettlers. Each result is measured over 2000 games.

5.5 Trading and Negotiation

We now describe in more detail how negotiations and trades are handled and how the planning method interfaces with the real game. This is a challenging task. Negotiations are composed of the actions of offer, reject and accept. Both in the real game and during planning, when a player accepts another's offer, the exchange of resources is

executed. This means that the participating players must agree before a trade is executed. In the real game, the other players have their own policy to decide on what is the best action, e.g. the Stac agents accept offers based on their internal heuristic evaluation of the action. When an MCTS agent plans, the best action (i.e. accept or reject) is chosen based on the UCT value for the player that must respond to the offer. The main difference in negotiations between planning and the real game is that counter-offers are completely ignored during planning. The list of original offers is already exhaustive, so any counter-offer will already be contained in this list. This action would not bring any benefit aside from increasing the depth of the tree. In future work, this option could be explored again in combination with reasoning about player types.

In addition to a large increase in the complexity of the search space when trades are added (e.g. branching factor increased to 65 from 10 when trading is allowed), the coordination between players is difficult to solve. For example, the planned offer may fail to result in a trade in the real game because the opponent can reject the offer). Despite MCTS's assumption that the opponents play optimally (i.e. they try to maximise their expected return), the opponent's evaluation of the best trade may be different than the approximation performed by the planning method. The usual application of MCTS is to plan and select one best action to execute in the game, then repeat for each subsequent state. But, with negotiations in the real game, this search strategy carries a risk of cyclic behaviour that will block the progression of the real game: the planning model may choose the same trade offer repeatedly and the opponent may always reject it.[4] One obvious solution is to implement a tabu list that keeps track of the failed offers while replanning. Replanning from the same state would however replicate the same effort several times and would slow down the experiments. In our implementation we rank all available actions according to their value, which was already estimated in the first search, and try to execute these actions in decreasing order. The agent makes a new plan when an action is successful (indifferent of the action's type). If it receives a counter-offer to any of its offers, our agent treats it as a reject action to speed up the negotiations in the real game. If the received counter-offer exists in the list of evaluated actions, it will be considered by our agent only if nothing better is successful.

Since Settlers of Catan is a turn-based game, we must also consider how our agent reacts to offers received during the opponent's turn. We are not allowing our MCTS agent to make counter-offers during an opponent's turn due to timing issues and the complexity of handling such a behaviour in JSettlers. Allowing these would also in-

[4]This issue could potentially be alleviated with an accurate opponent model.

crease the time required to run experiments. Instead, the planning agent must choose between either accepting or rejecting offers received during the opponent's turn. Our results show that our agent with these limits still outperforms the *Stac* agent, which doesn't have these limits.

As explained before, the same state may be encountered repeatedly during planning. This is problematic when the new nodes have not been explored enough and uncertainty in their current value is high. Due to the tree policy of selecting the action with the highest UCT value, the current search could get stuck in an infinite loop. Again a tabu list or certain restrictions to disallow this cyclic behaviour seem like the obvious solution. However, we have chosen a much simpler and cheaper alternative, which takes advantage of how the virtual loss is used during multi-threading. Instead of adding a virtual loss per thread, we increment a counter of virtual losses every time a thread enters a node. In time, this cyclic behaviour is discouraged. This counter is reset during updates.

5.5.1 Results

Trades add a lot of depth and complexity to the game. In a normal Settlers of Catan game, trading with other players is desirable because it is more resource efficient than trading with bank or port trades and has the potential to provide the needed resources immediately in contexts where bank or port trades are not possible. Furthermore, due to the layout of the board and the rules that decide how the players choose their initial positions on the board, it is highly unlikely that a player will have access to all five resources in a sufficient amount to achieve their goals without interacting with the other players.

To highlight the above, we run a quick set of experiments, where we compared our uniform TMCTS agent's performance against 3 Stac agents when trades are allowed in the game versus when trades are not allowed (i.e. either both the Stac agents and the TMCTS agent are allowed to trade, or none of the agents are allowed to trade). In the latter, the performance of the TMCTS agent would cap at 43% with 30k rollouts, as can be observed in Table 5.13. With trades, the game is much more complex and we can still observe an increase even at 50k rollouts.

We observed that the TMCTS agents make on average a large number of offers when trades are allowed. Also, this number is only slightly reduced as the number of rollouts is increased. Only 15% of the offers are accepted by the opponents in the

Trades	Iterations				
	5k	10k	20k	30k	50k
no trades	33.9%	39.9%	40.8%	43.8%	42.85%
with trades	18.8%	36.2%	45.75%	55.95%	61.01%

Table 5.13: Win rates of the TMCTS agent against 3 Stac agents with or without trades allowed in the game, while varying the number of iterations. Each result is measured over 2000 games.

real game. Since such a behaviour may seem annoying to a human opponent, we have tried two different ways of limiting it. Looking at the games collected in the corpus by Afantenos et al. (2012), a decent human player makes on average far fewer offers compared to novice players. Another possible cause for the reduced performance when the number of offers are unlimited could be that the Stac opponents may benefit from our agent's eagerness to trade. Even though the Stac agents use rule-based policies and would only accept trades that are relevant to their current plan, proposing so many exchanges increases the chances of making one that is relevant to their plan. We believe a human player would also be able to exploit the unlimited agent's behaviour. So, we have implemented a version that limits the total number of offers per turn. We also made the game model used by TMCTS aware of this limit. Table 5.14 illustrates the performance of the agent with this limitation. This limit affects the agent's performance slightly, but it also reduces the total number of trades to a number that would be tolerated by a human player.

Max offers per turn	1	2	3	4	5	unlimited
Win Rate	30.50%	31.10%	31.20%	33.75%	33.85%	36.2%
No. offers	18	36	45	57	70	527

Table 5.14: TMCTS agent's win rates and average number of offers per game given the number of offers it is allowed to make in total per turn. These are collected over 2000 games.

Appendix C shows that the trade limit has a higher impact on performance if the algorithm is allowed to perform more than 10k iteration. As future work, it would be interesting if the TMCTS algorithm could be modified and informed of a global constraint, e.g. the agents are only allowed a maximum of n offers over the whole

game. The goal of this modification would be to help the agent learn when to trade. However, the current search method only optimises the behaviour of the agents in the tree level (i.e. UCT) followed by long random rollouts to quickly evaluate leaf nodes. This means the current algorithm is likely to use the allowance early instead of partitioning it over the duration of the game.

5.6 Conclusion

This chapter presented the modifications necessary for scaling MCTS to the the full action space of the fully-observable version of Settlers of Catan. Standard methods for parallelising the algorithm and sharing results in the tree have proven useful, however the biggest gains are due to extending the rollout phase with an action type based system. Our best performing agent combines this type-based system with preferences extracted from the corpus via MLE, highlighting the need for prior knowledge to extend sampling methods to complex domains. Conditioning the preferences on type legality was crucial to improving over a uniform distribution. The benefits of the type-based system and of the conditioning support the ESH presented in Chapter 1 on page 5.

Overall, the very good results indicate the need of a planning at decision time method to win in complex games despite not using heuristics as in Chapter 4 to generate heavy rollouts. We have shown how even a sparse corpus that contains mixed play can be used to improve the planning. Even if the advice is not optimal, the extracted preferences represent how a standard player would play the game. Sampling from this high-level advice enforces trajectories to be more similar to those encountered in the corpus. Unfortunately, the proposed modifications were not enough to reduce the agent's eagerness to trade with the opponents. We need better ways to inform the planning of the consequences of trading as a human player may take advantage of this behaviour. Our current solution is to limit the number of trades the agent is allowed to perform and we leave solving this problem to future work.

Chapter 6

Monte Carlo Tree Search in Partially-Observable Settlers of Catan

A characteristic of Settlers of Catan that makes the game a challenging environment for learning agents is the imperfect information. Due to the uncertainty over which is the true state of the environment, an agent may need to consider a large number of possibilities in order to take an informed decision. Therefore, the complexity of the game is increased. In the previous chapter, we have showed that exploiting the game structure can alleviate the explosion of possibilities caused by trading. In this chapter, we evaluate if we can construct similar solutions to handle the imperfect information in the game. Following the MAH (see Description 1.2), we build a model of an agent's belief and utilise it during planning. We also aim to exploit the ESH in building this model since only certain portions of the state need to be modelled. Finally, the experiments in this chapter will evaluate the different implementations by measuring the performance of the agents in the full version of the Settlers of Catan game. The goal of these experiments is to show the merits of each method and which aspects of the game explain their performances.

6.1 Introduction

The game starts fully-observable, with the only unknown aspect being the opponents' player types and the order of development cards in the deck. We leave modelling the opponents to future work, while the order of development cards in the deck is bypassed by treating the action of buying a development card as a chance node. As the game plays out, however, certain action effects are not observable by all the players and the

true game state is hidden. For instance, when one player robs a resource from another, the unaffected players know the robbing has taken place but don't know what's been stolen. Further, when an opponent has to discard resources, they don't know which resources the player discarded. Fortunately, the list of all possible outcomes can be easily deduced given the state that the action was executed from and the rules of the game. For example, when a player buys a development card it can only draw one of: knight, monopoly, year of plenty, road building or victory point cards. There is a fixed and known number of each in the deck initially. Given that we know the exact quantities and the game rules, we can develop a game model that provides the exact probabilities of the transition functions. Such a model will alleviate the steepness of the learning curve, since learning has a single objective: finding the best policy to win the game. This is in contrast to many decision problems that humans face in daily life, where they must tackle learning the model as well as what is the optimal action. For example, learning to drive has these (more complex) aspects to the learning task. A beginner may initially be unaware that service vehicles can drive on the opposite side in case of an emergency. This missing detail from their naive model makes it impossible to plan in advance and sometimes also very difficult to know how to react.

For this reason, our agent has access to a complete and exact game model. As it is generally the case in games, e.g. Szita et al. (2010); Silver et al. (2017), an accurate game model can be developed due to the games' clear structure described in the rules. In other environments, one could have an abstract symbolic description of the relevant aspects, e.g. Kaelbling and Lozano-Prez (2013) or attempt to learn a model at the same time, e.g. Guez et al. (2013). Despite having access to an accurate game model, the agent won't know the true game state as the game progresses (even if the agent has the luxury of infinite memory), and instead it has a belief of what the true state might be. This belief model is represented as a distribution over the possible game states in S. The size of S is massive given that certain actions (e.g. stealing and discard) have a huge outcome range that also grows with the size of the initial belief. To make matters worse, Settlers of Catan is a 4 player game which tends to increase the number of states that are assigned a non-zero probability, given the current observations (for any given player will be uncertain about the resources possessed by all 3 opponents).

A popular approach to address this issue is to sample fully-observable states, plan in the observable version of the game, and then aggregate the results. But, making the game fully-observable is inappropriate since the fact that some information is hidden has strategic importance. For example, the development cards are always hidden until

these are played, so a player could hold them in their hand until the right moment. Furthermore, victory point cards are never shown and the other players cannot be certain of how close an opponent is to winning the game. Even hidden resources can have an impact in human games because a player can hide the fact that it has a specific resource to negotiate a better trade. Overall, then, the ability to hide information from others is part of what makes a strong Settlers of Catan player, and so we must be able to include these strategies in a player's decision making. This ability is extremely important in commercial scenarios in mergers and acquisitions (e.g., you hide from a hostile bidder for your company that you have a better offer from another more friendly company). Sampling fully-observable states changes the behaviour of agents in both single and multi-agent environments. Kaelbling and Lozano-Prez (2013) has observed multiple benefits to integrating uncertainty in robot planning, while sampling and incorrectly combining the results gives rise to what is known as *strategy fusion* in games (Frank and Basin, 1998).

The algorithms presented in this chapter attempt to plan in what is known as the *Belief MDP*. A POMDP can be translated into a Belief MDP as long as the belief transitions are Markovian, i.e. each outcome belief can be generated from the information in the current belief. We provided a formal definition of the Belief MDP in Section 2.3.2. As mentioned earlier, many current extensions to MCTS sample fully-observable states at the root node in order to access the game model. The results of planning with those (fully-observed) samples are then aggregated in the Belief MDP based on the trajectory of these samples. Following Kaelbling and Lozano-Prez (2013), we depart from the sample-based approach and we evaluate if planning entirely in the belief space is appropriate for non-cooperative multi-player environments. Kaelbling and Lozano-Prez (2013) claim that this has multiple benefits over the current approaches that exploit sampling fully-observable states, since it encourages "intelligent information-gathering behavior". In addition, we hypothesise that sampling struggles in large and highly uncertain situations because the planning will always be done over a very sparse tree. If the support of the distribution representing the belief is massive or it has a high entropy, then (perhaps) too many samples will be needed to estimate the true distribution.

Reasoning over the belief of the agent allows reasoning over all possible actions and at the same time aggregates the possible states. This permits sharing of certain results such as winning or losing the game where the hidden part of the state doesn't necessarily contribute to the result. Furthermore, transitions in the belief space are de-

terministic modifications to the current belief distribution so the variance of the result would be reduced. But, in order to plan in the belief space, we require two parts:

- a model to represent and track the belief both in the real game and during planning;
- a model of the game that provides a reward and transition probabilities given the belief of a player.

Before we present these two parts and the algorithms, we must briefly remind the reader how we handle the multi-agent characteristic of the game. The POMDP framework presented in Chapter 2 is appropriate, and as mentioned there we present the algorithms from a single player's perspective by making the assumption that each agent acts to maximise their outcome. Therefore, planning is done via self-play, and we do not include player types or reason over nested beliefs.

6.2 Tracking Belief in Settlers of Catan

The most popular methods for tracking the belief of an agent are to either use a particle filter (Silver and Veness, 2010), or to create a model of the environment such as a hidden markov model or a dynamic bayesian network (Russell and Norvig, 2009). An alternative is to create an abstract representation of the game and track only the relevant or more important aspects of the environment (Kaelbling and Lozano-Prez, 2013). The latter generally requires some knowledge of the domain. Given that our task is one where the agent starts the learning process knowing all game rules, we adopt this method of creating an abstract representation. Our goal is to have a belief model that is as accurate as possible such that we can be certain of the performance of the algorithms that use it. It is worth mentioning that all final algorithms presented in this chapter make use of the agent's complete and accurate knowledge of the game rules to represent and update the belief of the agent.

Our implementation resembles that of a *Factored* POMDP (FPOMDP) (Paquet et al., 2005; Williams, 2005). In a FPOMDP, the belief is not represented as a distribution over a single state variable, instead states are represented as a set of M random variables $s = X_1 = x_1, \ldots, X_M = x_M$ and a belief is represented as a joint probability distribution over these variables $b = P(X_1, \ldots, X_M)$. One approach is to assume complete independence $b = \prod_k^M P(X_k)$ (Paquet et al., 2005), but this is quite a strong assumption that may not work very well in domains such as ours, where *sets* of variables determine

actions legality as well as actions effects. For instance, the requirement for building a road is to hold 1 wood and 1 clay, while the effect of stealing depends on the victim's entire hand. As a result we only assume independence between some of the variables. We included a more detailed description of the belief model and belief transition function $b' = \tau(b,a,o)$ that are specific to Settlers of Catan in Appendix D. As it is the case with the observable game model, a belief model can be designed and developed due to the clear structure described in the game rules.

There are three major benefits in factorising the belief in this way: belief update is faster, enumerating possible legal actions is also faster and belief representation requires a lower memory. The first and the second benefits are mostly due to how these only target specific portions of the state as well as specific players. The latter is true because (as with Bayes Nets) the factored approach offers a highly compact representation of the joint probability distribution over all possible states. These benefits are also reported by Paquet et al. (2005). Kaelbling and Lozano-Prez (2013) show that failed actions in the real environment can be used to update the belief followed by replanning, such that the robot can advance and achieve its goal. In Settlers of Catan, action legality depends on part of the game that is observable from the current player's perspective. This means that players always know what is legal and what is not, so we do not need to worry about actions that fail in the real game.

6.3 The Imperfect Information Game Model

Following the discussion in Section 6.1, our hypothesis is that planning in the Belief MDP without marginalising over states is beneficial for several reasons:

1. it turns the transition function into a deterministic one, so it reduces the number of chance nodes and the uncertainty as a result. In a Belief MDP, the transition function $P(b'|b,a,o)$ is deterministic (see Section 2.3.2).

2. it speeds up convergence to the true action value expectations in situations where the belief distribution domain is massive and the entropy of the distribution is very high.

3. it allows agents to learn strategically important moves since they can reason about what opponents might want to keep hidden or reveal.

Kaelbling and Lozano-Prez (2013) showed that planning entirely in the belief space results in intelligent and more realistic behaviour in single-agent environments. To

achieve this, the authors represented the belief as a set of fluents that are important for the agent's goal and belief update as a set of modifications to these fluents. They also defined a set of observations and their likelihoods according to the robot configuration and the sensors they had at their disposal. We used the factored representation presented earlier. We present in Appendix E how we implemented the set of action effects and observations in the game model according to the game rules.

A further complexity in Settlers of Catan that we must address is the (hidden) possibility that an opponent's next action will result in the game being over. In the real game, a server has access to the true state and it can inform all agents of the outcome. During planning this is not possible since the algorithm has access only to one of the agent's belief. There are cases where the game has ended in some of the states that are possible according to the belief model, but not in others. In other words, the agent is uncertain about whether the game has ended! This is caused by the game rules that dictate victory point cards are kept hidden until the player has gathered at least 10 victory points, including the points from these cards. There are many board games that have this characteristic, such as Small World or Oregon, and there are also games in which players have a hidden victory condition from a pool of possible ones, such as Mission Risk (a variant of the well-known Risk game) or Ticket to Ride. In either of these scenarios, the planning method must be able to take into account the possibility that the game has ended and an opponent won.

So we need to introduce an extra chance node for the cases where a player's observable victory points in combination with the number of unplayed development cards could reach the required threshold of 10 for the game to end. For example, the current player receives the longest road award that increases this player's score to 9 victory points. The player also has 2 unplayed development cards and our current player believes there are 3 victory points either in the development deck or in an opponent's hand. Given this information, we compute the probability that the player has at least one victory point card ($P(X_v^j = 1)$) using simple combinatorics. This probability indicates how likely it is the game has ended (and $1 - P(X_v^j = 1)$ indicates how likely it is the game has continued). The chance node samples one of these outcomes accordingly and, if the game continues, the development model is updated with the information that the current player cannot have any victory point cards in their hand. This chance node is encountered in the planning phase only, but the update to the development model is performed in the real game as well when the server doesn't end the game. We will refer to this new chance node as the win/continue chance node through the rest of this

chapter.

Finally, Kaelbling and Lozano-Prez (2013) did not have to deal with action legality and could permit planning over illegal actions as long as these provide observations to update the agent's belief. As shown in Chapter 5, we cannot afford such a luxury due to the massive action space. As a result we created our game model that can enumerate all possible actions. The possible actions are defined to be the actions that are legal in at least one state from those that have a non-zero probability mass in the agent's current belief (i.e. $P(s|b) > 0$). From now on we refer to the set of actions that are allowed in the game as the action space, the set of actions that are allowed in a fully-observable state as legal actions and the set of actions that are allowed in a belief state as possible actions.

6.4 Partially Observable Monte Carlo Planning

Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness, 2010) was originally proposed as an approximate planning method in POMDPs. In undiscounted POMDPs with highly sparse rewards such as our game, it is a straightforward extension of the MCTS algorithm presented in Chapter 5: the main difference is that the nodes in the tree are histories h instead of true states s. Histories are list of tuples $h_t =< (a_1,o_1),(a_2,o_2),\ldots,(a_t,o_t) >$ which are collected from interacting with the environment. The UCT equation can easily be adapted to use histories by replacing the counts over states with the counts over the histories: $Q(h,a) = Q(h,a) + C\sqrt{\frac{\log N(h)}{N(h,a)}}$. Before each MCTS iteration, a fully observable state s is sampled from the agent's belief given the history at the root node ($s_t \sim B(h_t)$). In order to generate the history during the planning phase, the state is used as input alongside the action with the maximum UCT value to a black box model of the game $G(s,a)$. This model returns the next fully observable state s_{t+1}, an observation o_{t+1} (used in combination with the action a to update the history) and a reward r_{t+1} (which is 1 only when the game ended and the agent won). Therefore, the sampled state is updated during the tree phase of the algorithm. In the original paper, the belief of the agent is represented using an unweighted particle filter and particle reinvigoration as random noise is used to generate new particles.

The algorithm is very quick since it never updates the full belief during the planning phase and it only updates the current sampled state. It adds the sampled state to the current history node if it is missing, so the algorithm slowly builds up the belief of the

agent at each of the non-root nodes. Silver and Veness (2010) extend the rollout phase of MCTS to the partially-observable case by creating a heavy rollout policy defined over the history $\pi(h) = P(a_{t+1}|h = h_t)$. This policy is a probability distribution that prefers certain actions given a small set of manually-defined heuristics. While introducing domain knowledge helped the algorithm considerably (as one would expect), the set of legal actions given the history is the same as the set of legal actions given the current sampled fully-observable state in their environment. The authors do not reason over action legality, since this is not state dependent in the partially-observable version of PacMan that they used to evaluate the algorithm. As a result, the state policy $\pi(s)$ and the history policy $\pi(h)$ are the same because these are the same distribution defined over the same set of legal actions. This is true for both the policy that prefers certain actions and for the default uniform distribution. Unless additional reasoning over the history is performed, the rollouts in POMCP are the same as the ones performed in a standard MCTS algorithm over the fully-observable game.

6.4.1 Extensions

We want to compare the performance in our domain of tackling planning with imperfect information with the two distinct approaches we described above: i.e., the belief transition model, in which uncertainty about the current state is maintained while following how the game might progress in MCTS versus the 'sampling' POMCP model, in which the agent starts by sampling a fully observable state from its belief about the current state, and follows the consequences of that in MCTS. We need to make several modifications to POMCP. The most obvious one is the belief representation using a particle filter. Particle reinvigoration is not straightforward in Settlers of Catan since applying random noise to the state description could easily result in illegal states. We would need a detailed model in order to be able to do this safely, perhaps by defining certain bounds for each feature according to the current history. Unfortunately, this would be just as complex as the factored belief model we described earlier. We have therefore decided to replace the particle filter with our factored belief model.

A weakness of POMCP is using histories to represent nodes, especially in Settlers of Catan. Transpositions can still be used by creating a hash map where the keys are the hash values of the histories, but results are no longer shared. Hashing histories enforces nodes to be unique. The action-observation pairs and their order must be the same for two nodes represented with histories to be the same. Sharing statistics with

transpositions has been known in the literature to be useful (Childs et al., 2008). The results in Chapter 5 have also shown the large benefits of sharing action values using afterstates in the observable version of Settlers of Catan. Obviously, neither afterstates or transpositions can be implemented with histories. We need a representation for the nodes that combines the observable part of the state with the agent's belief. Hashing the factored belief in the current format is expensive and redundant. So we use an abstract representation of the belief as input to the hashing function. Similar to the state representation, this abstract representation of the belief is a vector of features (see Appendix F) that provide a sufficient definition to differentiate from beliefs in other nodes. This is combined with the observable description of the state to represent nodes.

Hashing this representation permits sharing of results when different actions or sequences of actions lead to the same belief state b'. This is often the case in the first half of the game, before discard and stealing occurs, when the belief model contains only the true state of the game. But it is also encountered when the sequences of actions do not reveal anything additional to what the agent already knows, such as the cases where opponents trade or use resources that our agent is already certain they own. Finally, it is possible that different actions executed from different belief state would end up in the same outcome node as in the fully-observable version of the game. Sharing results and using transpositions allows us to compute the UCT value of an action a that results in belief b', when executed in belief b (see Equation 6.1). This is similar to how afterstates are used in the observable case (see Equation 5.4).

$$UCT(b,a) = V(b') + c\sqrt{\frac{\log N(b)}{N(b')}} \tag{6.1}$$

Even though this modification is beneficial, it requires access to the exact belief to generate the abstract representation for the corresponding node. As a result, we further depart from the original POMCP algorithm and we update the agent's belief as we traverse the tree phase of the algorithm. We still sample a fully-observable state at the root node to advance the game using the observable game model $G(s,a)$, but we update the belief using the observation and our transition function $b' = \tau(b,a,o)$ at the same time.

Updating the belief makes POMCP more expensive. However, this update only takes place during the tree phase of the algorithm, while rollouts use the observable game model. There is only a small set of updates for each iteration and we believe the

benefits of sharing statistics, as shown before, outweigh its costs. An obvious extension would be to define a cheap update step so that we can update the abstract representation directly. This would be an approximation of the full belief update step and would be much quicker. Since the other algorithms presented in this chapter use the exact update rule and the exact belief model, we decided not to make this modification such that we have a fair comparison between the algorithms.

Silver and Veness (2010) have applied POMCP to environments where the agent does not need to reason over the opponents' action legality. Their environment is a single player game, where the agent always has complete knowledge of which actions are legal (in spite of the current game state being hidden). In Settlers of Catan, we need to reason over what opponents can and cannot do, because a failed action would inform the other players of what the agent doesn't have. Similar to how we expand the nodes with the set of legal actions in the fully-observable case, we expand the nodes when the opponent is next to move with the set of possible actions given our agent's current belief. Based on the sampled state, we then mask out the edges that correspond to the actions that are illegal. This is only a design choice. The alternative is to enumerate all legal actions given the current state and slowly expand the node until all possible actions are added. Both approaches achieve the same goal of not allowing to plan over actions that are illegal given the state sampled at the beginning of the iteration.

A very popular and effective approach to improve any MCTS algorithm is to reuse the statistics over the branch that was chosen in the real game in subsequent planning phases. This has been done in the original POMCP algorithm, but we have not allowed the reuse of statistics and each planning phase starts from the root node. This modification would clearly improve any MCTS algorithms presented in this thesis, but we left it to future work due to time constraints. A final modification is that we use the typed policy during rollouts as presented in Chapter 5. This modification has proven crucial to the success in the observable version of the game. We have not compared the original POMCP to this modified version because all modifications presented here are clearly improving the original agent at the expense of slightly slowing down the tree phase. This modified version resembles the algorithm presented by Bai et al. (2016) who present state abstractions in MDP as turning the problem into a POMDP. However, Bai et al. (2016) have also included options to create a hierarchical method. This is not straightforward in Settlers of Catan as we discussed in Section 5.4. Our approach also contains other differences such as: the typed policy that we use for rollouts, sharing statistics by aggregating them in the outcome belief node, the factored representation

of the belief model that provides the abstraction function, and the use of action legality function to allow extension to a multi-player scenario. We use this modified version in the experiments and we refer to it as *TPOMCP* (Typed POMCP).

Algorithm 5 presents TPOMCP. The observable game model G is extended such that it can also enumerate the possible actions given a belief state, $possible\_actions(b)$, in addition to enumerating the legal action types given a state, $action\_types(s)$, and enumerating the legal actions of a given type in a state, $actions\_of\_type(s,t)$. We included the typed rollout and the use of afterstates in this presentation. It is worth mentioning that we can quickly access the statistics stored in a node corresponding to a belief-state b' because the observation model is deterministic as described in Appendix E. Therefore, we drop o to keep the algorithm clear. If the observation model was not deterministic, we would need to delay $P(O|s,a)$ and $\tau(b,a,o)$ to a chance node after the deterministic modifications of the action are executed. This is similar to how afterstates are implemented in a stochastic fully-observable game (see Figure 5.1). Another related aspect is that we do not need to recompute $b' = \tau(b,a)$ for every a in the selection phase of the algorithm (i.e. TreePolicy function). These are stored during the expansion step, where each child node is connected to the parent. Finally, we do not include the Backpropagation function as this is only incrementing the statistics in the nodes that were visited in the corresponding iteration.

6.5 Information Set Monte Carlo Tree Search

Another very popular and simple extension to the standard MCTS algorithm in order to handle games with imperfect information is Determinized MCTS (Bjarnason et al., 2009; Browne et al., 2012). The approach is to sample a fully-observable state or a *determinization* at the root node followed by building a tree for each different sampled states. The planning is always done in the fully-observable version of the game, so a standard MCTS algorithm can be used. At the end of the planning phase, the results are merged and the action with the maximum aggregated value according to some metric is selected in the real game. This approach has a few weaknesses:

- *Nonlocality*: Players have different beliefs of what the true state of the game could be (Frank and Basin, 1998). So, agents must consider the opponents' beliefs when reasoning on how they will direct the game towards what is most favourable for them;

Function `Search` (*G, Tree, b*) **:**

$n \leftarrow Tree.new\_node(b))$

while *within budget* **do**

$s \sim b$

$n,s,b \leftarrow$ `TreePolicy` (s, b, n, G, $Tree$)

$s \leftarrow$ `Expand` (s, b, n, G)

$r \leftarrow$ `Rollout` (s, G)

`Backpropagate` (r, $Tree$)

end

$a \leftarrow \arg\max_a V(b')$, where $b' \leftarrow \tau(b,a)$

return a;

Function `TreePolicy` (*s, b, n, G, Tree*) **:**

while *n has children* **do**

$a \leftarrow \arg\max_{a_l} V(b') + C\sqrt{\frac{\log N(b)}{N(b')}}$, where $a_l \in \{a|\prod(s,a) = 1\}$ and

$b' \leftarrow \tau(b,a_l)$

$s \leftarrow G.step(s,a)$

$n \leftarrow Tree.get\_node(\tau(b,a))$

end

return n,s,b

Function `Rollout` (*s, G*) **:**

while *s is non-terminal* **do**

$t \sim \pi_t(G.action\_types(s))$

$a \sim \pi_a(G.actions\_of\_type(s,t))$

$s \leftarrow G.step(s,a)$

end

return $G.reward(s)$

Function `Expand` (*s, b, n, G*) **:**

foreach $a \in G.possible\_actions(b)$ **do**

$Tree.new\_node(b')$ where $b' \leftarrow \tau(b,a)$

end

$a \sim \{a|\prod(s,a) = 1\}$ # uniform random choice

return $G.step(s,a)$

Algorithm 5: The TPOMCP algorithm

- *Duplication*: The technique does not take advantage of nodes being duplicated across trees, spending resources for recomputing their values (Cowling et al., 2012a);

- *Strategy fusion*: There is a constraint that the agent must have the same strategy indifferent of the sampled determinization (Frank and Basin, 1998). This is not true in any algorithm that samples determinizations and combines the strategies post-planning. So these would learn a sub-optimal policy.

Information Set Monte-Carlo Tree Search (ISMCTS) (Cowling et al., 2012a) was proposed to address these issues observed with Determinized MCTS. It constructs a single tree with nodes representing information sets rather than exact states to address the duplication issue. Information sets are sets of all the possible determinizations. When combined with a probability distribution over the determinizations, they are equivalent to belief states. At the root node, the algorithm chooses a determinization from the ones available in the information set and follows a path down the tree based on which actions are available given the chosen determinization. At every node it builds up the information set by adding the determinization if it is missing. Strategy fusion is partly solved by performing the selection step of the algorithm using the results aggregated over all determinizations. In fact, the standard ISMCTS is equivalent to the POMCP algorithm save for the UCT calculation, which is modified to avoid over-exploring unlikely actions. Since ISMCTS samples determinizations, different subsets of actions are available for each pass through a node. So, each action selection in the tree becomes what they refer to as a subset-armed bandit problem. The authors modify the UCT formula to accommodate this by replacing the number of times $N(b)$ a node has been visited with the number of times $N(b|legal(a) = True)$ the node was visited and the action was legal. So, ISMCTS is also adapted to multi-player game environments and reasons over action legality by modifying the UCT calculation as following:

$$UCT(b,a) = V(b') + c\sqrt{\frac{\log N(b|legal(a) = True)}{N(b')}} \tag{6.2}$$

Cowling et al. (2012a) present several improvements to the ISMCTS algorithm. One of them is to keep the opponent's partially observable actions hidden in the tree and replace them with a chance node that selects their effect at random. The motivation for this is that it reduces strategy fusion at the risk of weakening the opponent model as the opponent cannot decide the outcome. We do not agree with the use of

chance nodes because we believe these generate a sparse tree when the results could be aggregated. So we have instead integrated all possible effects in our belief transition function. However, the only partially observable move that the agent has full control over is the discard action. In this case, the others can only observe the number of cards that were discarded. When a player other than the one whose beliefs we're modelling performs a discard action, we assume in that belief model that this player had no preference over its available actions. So similar to chance nodes, this is also a weaker opponent model, but all resulting outcomes are aggregated. Player preferences can be easily integrated into the belief transition function if we were to reason over player types.

Another extension proposed, known as multiple observer ISMCTS (MO-ISMCTS), is to keep track of the opponents' beliefs as well. This version keeps a separate tree and a separate belief for each participating player. During planning, all trees are traversed in parallel and the algorithm selects the actions using the statistics from the tree whose player is next to move. MO-ISMCTS attempts to address the non-locality issue as it allows our agent to reason over opponents' beliefs. However, this is an estimation. The opponents' beliefs and knowledge of the game state are based on our agent's belief of the game state. Therefore, the opponent's actions will be an estimate of what their optimal play should be. It is obviously impossible to accurately track an opponent's real knowledge of the game state without providing access to the true game state. There are methods that estimate or correct the belief distribution using the utilities of certain actions estimated via MCTS (Wang et al., 2015), under the assumption that players act rationally and aim to maximise their score. They increase the probability mass on states that are more beneficial to the opponent because they expect the opponent can decide which state they prefer.

MO-ISMCTS may result in our agent choosing not to rush into playing certain actions because it can reveal certain information to the opponent. This may have a high strategic importance if the negotiation moves would be more advanced and agents would be able to react to these. However, MO-ISMCTS is more expensive compared to ISMCTS as it must track at least the belief of multiple players and it must store and traverse multiple trees. We aimed for the simplest possible model given that this thesis centres on tackling complex games with limited computational resources (as well as limited data resources). As a result, we only implemented the standard single-observer ISMCTS algorithm with the partially observable actions effects integrated in the belief updates to our factored model. We also introduced the typed rollouts in the ISMCTS

algorithm since the approach has proven essential in the previous experiments. We refer to this agent as *TISMCTS* (Typed ISMCTS). We do not include an algorithm for TISMCTS as it is the same as the TPOMCP algorithm with the exception of the exploration term in the UCT calculation performed in the TreePolicy function.

6.6 Belief Monte Carlo Tree Search

POMCP and ISMCTS both sample a fully-observable state followed by masking out the illegal portion of the tree. So the algorithms are always traversing sparse trees and rely on sampling to be sufficient to cover the full tree in time. In massive state spaces, when the support of the belief distribution is large, sampling may not be sufficient to get a good estimation if the number of samples is small. Furthermore, an even larger number of samples is required if the entropy of the distribution is high. Finally, even if the number of samples are sufficient to acquire a reasonable estimate of the belief distribution, it would definitely not be enough to get an accurate estimate of the expected value of each state in the belief. This is caused by the way MCTS already samples the game tree to estimate the value of each state, so POMCP and ISMCTS combine two sampling phases: one to estimate the current belief and one to estimate the expected return for each sample. This means the number of iterations required to get reasonable estimates of expected utilities is further increased.

Planning in the belief space means that we do not need to marginalise over states to estimate the belief distribution. Our modified game model and belief transition function allows us to propagate the agent's belief down the tree, without the need to sample states. We in effect avoid this slow convergence to the true distribution at every node in the tree. Wang et al. (2015) claim to achieve the same objective of planning in the belief space. However, they sample observable states before the search over the game tree and so their algorithm resembles POMCP and ISMCTS. Their main contribution is that they use the statistics from planning to update the belief distribution such that it slowly changes towards a distribution that reflects what states a rational opponent is more likely to visit. As a result, the belief corrections they make affect what states are more likely to be sampled at the root node. This is an interesting approach, but is complementary to what we aim to achieve. We model the belief transition function to reflect the game dynamics and use it during planning, while they still sample states to slowly converge towards this distribution. We handle the stochastic effects of partially-observable actions while they apply their method to

phantom games that do not have any stochastic effects. So they assume the players can always decide the outcomes of actions. An opponent model such as theirs can further improve our belief transition function for the actions where we assume the opponent is indifferent (e.g. discard).

Similar to TPOMCP and TISMCTS, we represent the nodes as belief states. The main difference is that we do not sample states at the root node and we do not mask any of the possible actions as a result. This approach has a large risk of over-sampling unlikely actions because all possible actions given the current belief are allowed with equal probability. We need to somehow weight the actions according to how likely these are to be legal given the current belief. Let's assume for the moment that we allow illegal actions to be executed in states and that all agents receive one of the two observations after each action $\Omega = \{Success, Failure\}$. The Bellman equation in a Belief MDP is:

$$Q(b,a) = \rho(\tau(b,a,o)) + \gamma \sum_{o \in \Omega} P(o|b,a)V(\tau(b,a,o)) \tag{6.3}$$

Given we have the two observations mentioned before, we can rewrite the equation as following (we also drop the discount parameter as before):

$$\begin{aligned} Q(b,a) = \rho(\tau(b,a,o)) &+ P(o = Success|b,a)V(\tau(b,a,o)) \\ &+ P(o = Failure|b,a)V(\tau(b,a,o)) \end{aligned} \tag{6.4}$$

As we have shown in Chapter 5, illegal actions would just increase the length of trajectories in a non-discounting scenario with highly sparse rewards ($\{0,1\}$ on only end states to the game). One will not receive a reward after an illegal action, since the game cannot end following an illegal action. Similar to the observable case, the value of an illegal action would be equal to the value of the belief this was executed in. Executing an illegal action can also provide valuable information to the opponents as we discussed before. Since all players know what actions are legal and what are not during their turn, it does not make any sense to include these actions.[1] In the end we want the game model to accurately represent the real environment and JSettlers is a server-client where the server keeps track of the true state. Only the player that

[1] Making trade offers where the receivable part is illegal in the true state of the game is a legal move, since even the strongest players cannot know the true state. In this case, the recipient is not allowed to accept the offer since it doesn't have the requested resources. On the other hand, a player is not allowed to offer resources it doesn't possess as this can be a move that attempts to misinform the opponents. As mentioned in Chapter 5, we do not allow such moves in our experiments.

tried to execute the action will be informed that the action failed. As a result, we can drop the second part of the equation. Since the players only observe *Success* when the action is legal, we can compute this probability using the action legality function $\amalg(s,a)$ that we defined in Equation 5.1 in Chapter 5 (we use $legal(a)$ to indicate if the action a is legal or not): $P(o = Success|b,a) = P(legal(a) = True|b) = \mu \sum_{s \in b} P(s|b) \amalg(s,a)$. Since the action legality probabilities are independent, we need to normalise the final distribution. Calculating $P(legal(a) = True|b)$ is quite expensive if we were to enumerate the full state space, so we utilise our factored implementation to quickly compute the probability of the player having the requirements for the action to be legal. The new UCT calculation is shown in Equation 6.5. This weighting is only required during the opponents' turns. During its own turn, our agent is certain of what are its legal actions given its belief. For the same reason we do not need to weight the decision made once the planning is over, i.e. the agent chooses the action with the maximum value, $\arg\max_a V(b')$ as in TPOMCP.

$$UCT(b,a) = P(legal(a) = True|b)[V(b') + c\sqrt{\frac{\log N(b)}{N(b')}}] \tag{6.5}$$

Rollouts can also be done using the belief model by enumerating all possible actions and representing the policy $\pi(b) = P(A = a|b)$ as a distribution over them. But, we must take into account how certain actions are legal more often then others. As a result, the default uniform policy should be modified by replacing the probability of selecting each action with the probability that the action is legal $P(legal(a) = True|b)$. Each action legality probability is independent of the other possible actions' legality so we need to normalise them such that the sum equals 1. When using the uniform typed rollout, we replace the probability of selecting a type $\pi_t(T) = P(T = t)$ with the maximum legality probability $\max_a P(legal(a) = True|b)$ of the action a from the possible set that are of type t. The policy $\pi_a(A_t)$ that selects the action description is modified to take into account the actions' legality $P(legal(a) = True|b)$, and the list A_t now contains the possible actions of type t rather than just the legal ones. The typed rollout algorithm is the same as the one in Algorithm 4 (see page 75) where each mention of s is replaced with b. We also list all possible actions or action types instead of just the legal ones. We call this algorithm Typed Belief MCTS (TBMCTS) because it plans in the belief space. We also refer to the agent that plans using it as *TBMCTS*. TBMCTS is shown in Algorithm 6. For conciseness we do not show how the lists T and A_t are formed in the *Rollout* function. These are created during the computation of $P(t)$, by

enumerating over the possible actions given by $G.possible\_actions(b)$.

As we will observe, the belief rollouts are by far the most expensive part of the algorithm because games in Settlers of Catan tend to be long (i.e., the tree depth is large). The algorithm performs many belief updates and the sampling policy encounters belief states where agents hoard resources, further increasing the time required to perform the updates. A simple modification to the TBMCTS algorithm is to sample a fully-observable state at the leaf nodes of the tree and perform the typed rollouts using the observable game model similar to TPOMCP and TISMCTS. We would still propagate the belief down the tree during the tree phase and aggregate the possible states. We call this TBMCTS with Observable Rollouts (TBMCTSOR) and we refer to the agent as *TBMCTSOR*.

Algorithm 7 presents TBMCTSOR. This method is a combination of TBMCTS (i.e. the $TreePolicy$ and $Expand$ functions) and TPOMCP (i.e. the $Rollout$ function). Performing observable rollouts is acceptable since we do not use player types, or reason over opponent's knowledge and beliefs. As a result, belief rollouts would not improve how leaf nodes are evaluated more than just aggregating trajectories. Observable rollouts can be easily turned into heavy rollouts just by using as policy the distribution over types learned from our corpus with MLE. Belief rollouts could also be extended with this knowledge by multiplying the belief distribution with the distribution over types.

6.7 Results

Before we present the performance of the agents, we remind the reader about the modifications to the MCTS agents that we described from Chapter 5. All agents use the typed rollout with the uniform distribution over types. Also, we limited the offers the agents can make per turn to 3, otherwise the experiments would take too long and the agent we build would not be tolerated by human opponents. Due to time restrictions we focused on evaluating the presented algorithms versus the Stac agent only and we did not run a tournament-like experiment to find the best agent. Table 6.1 contains the results while varying the number of iterations. The performances of the TPOMCP, TISMCTS and TBMCTSOR agents versus 3 Stac agents are not significantly different. On the other hand, the results clearly show that the TBMCTS agent is the weakest. The issue lies with performing rollouts in the belief space, since TBMCTSOR differs from TBMCTS only because it performs observable rollouts as TPOMCP and TISMCTS.

Function `Search` (*G, Tree, b*) **:**

$n \leftarrow Tree.new\_node(b))$

while *within budget* **do**

$n,b \leftarrow$ `TreePolicy` $(b, n, Tree)$

$b \leftarrow$ `Expand` (b, n, G)

$r \leftarrow$ `Rollout` (b)

`Backpropagate` $(r, Tree)$

end

$a \leftarrow \arg\max_a V(b')$ where $b' \leftarrow \tau(b,a)$

return a;

Function `TreePolicy` (*b, n, Tree*) **:**

while *n has children* **do**

$a \leftarrow \arg\max_a P(legal(a) = True|b)[V(b') + C\sqrt{\frac{\log N(b)}{N(b')}}]$ where

$b' \leftarrow \tau(b,a)$

$n \leftarrow Tree.get\_node(\tau(b,a))$

end

return n,b

Function `Rollout` (*b*) **:**

while *b is non-terminal* **do**

$P(t) \leftarrow \max_a P(legal(a) = True|b)$, where $a \in A_t \leftarrow \{a|type(a) = t\}$

$t \sim \pi_t(T)$ # using $P(t)$

$a \sim \pi_a(A_t)$ # using $P(legal(a) = True|b)$

$b \leftarrow \tau(b,a)$ # includes win/continue chance node

end

return $reward(b)$ # from observable score or w/c chance node

Function `Expand` (*b, n, G*) **:**

foreach $a \in G.possible\_actions(b)$ **do**

$Tree.new\_node(b')$ where $b' \leftarrow \tau(b,a)$

end

$a \sim G.possible\_actions(b)$ # uniform random choice

return $\tau(b,a)$

Algorithm 6: The TBMCTS algorithm

Function `Search` (*G, Tree, b*) **:**

$n \leftarrow Tree.new\_node(b))$

while *within budget* **do**

n,$b \leftarrow$ `TreePolicy` (b, n, $Tree$)

$b \leftarrow$ `Expand` (b, n, G)

$s \sim b$

$r \leftarrow$ `Rollout` (s, G)

`Backpropagate` (r, $Tree$)

end

$a \leftarrow \arg\max_a V(b')$ where $b' \leftarrow \tau(b,a)$

return a;

Function `TreePolicy` (*b, n, Tree*) **:**

while *n has children* **do**

$P(legal(a) = True|b) = \mu \sum_{s \in b} P(s|b) \prod(s,a)$ # for each $a \in G.possible\_actions(b)$

$a \leftarrow \arg\max_a P(legal(a) = True|b)[V(b') + C\sqrt{\frac{\log N(b)}{N(b')}}]$ where $b' \leftarrow \tau(b,a)$

$b \leftarrow \tau(b,a)$

$n \leftarrow Tree.get\_node(b))$

end

return n,b

Function `Rollout` (*s, G*) **:**

while *s is non-terminal* **do**

$t \sim \pi_t(G.action\_types(s))$

$a \sim \pi_a(G.actions\_of\_type(s,t))$

$s \leftarrow G.step(s,a)$

end

return $G.reward(s)$

Function `Expand` (*b, n, G*) **:**

foreach $a \in G.possible\_actions(b)$ **do**

$Tree.new\_node(b')$ where $b' \leftarrow \tau(b,a)$

end

$a \sim G.possible\_actions(b)$ # uniform random choice

return $\tau(b,a)$

Algorithm 7: The TBMCTSOR algorithm

However, it is unclear *why* rollouts in the belief space should perform so much worse than rollouts with the fully observable states that are sampled from belief states. The cause could be that when rollouts are done in the belief space, one must reason about action legality; or it could be due to having to reason about the win/continue chance nodes (which, as described earlier, are required because you cannot know with complete certainty when an action ends the game, because whether opponents hold VP cards is hidden).

Agent	10k	20k	30k	40k
TPOMCP	32.14%	42.23%	44.5%	47.94%
TISMCTS	31.31%	41.66%	44.16%	48.49%
TBMCTS	23.52%	31.5%	38.08%	40.33%
TBMCTSOR	32.05%	41.63%	45.09%	48.7%

Table 6.1: Win rates of the modified agents against 3 Stac agents, while varying the number of iterations. Each result is collected over 2000 games.

We have also timed how long the agents take to plan and present the statistics in Table 6.2. As before these are measured by running 10 games versus 3 Stac agents while fixing the number of iterations to 10k and the number of threads to 4. The times presented in this table can be tolerated by human players. TISMCTS has comparable values to TPOMCP so we didn't include it in the table. As expected, TBMCTS is much more expensive than the sampling based agents. It may initially appear surprising that TPOMCP takes longer than TBMCTSOR, however it must also compute the action masks for every decision in the tree phase. It is quite likely that TPOMCP with approximate belief updates and an abstract belief model would be quicker. These models have not been extensively optimised, e.g. caching previous computations such as the mask in POMCP and ISMCTS, so we do not perform experiments while limiting the allocated time for planning. After optimisations, it is very likely that these models would have very similar time requirements given that the current planning times are already comparable.

6.7.1 Rollouts with the Human Type Distribution

Since there is not a large difference between the performance of the 3 agents that perform sampling, we chose TPOMCP to evaluate if we can observe the same benefits of integrating preferences extracted from the human corpus, where these preferences are

Agent	Min	Max	Mean	Median
TPOMCP	456	7035	2113	2086
TBMCTS	270	19559	8161	8449
TBMCTSOR	757	5476	1963	1916

Table 6.2: The agents' planning time in milliseconds, each value measured over 10 games.

either conditioned on the other type legality or unconditioned (see Section 5.4.3 for details on how these are learned). Table 6.3 contains the results where we modified the rollout distribution of a TPOMCP agent. Similar to the experiment in the observable version of the game (see Table 5.9), the unconditioned is worse than the uniform distribution and conditioning on type legality addresses results in a better performance. The benefits over the uniform distribution do not seem as noticeable as before, but this may be due to the increased difficulty of the game when considering imperfect information. Interestingly, an agent that uses a uniform distribution is able to win over 25% of the games versus 3 agents that sample the action types from the conditioned distribution. Even though the 27.19% win rate is not significantly better, it illustrates why we also need to evaluate the performance of baseline agents versus the modified agents. Despite this result, an agent using the conditioned distribution wins more than 27.19% (i.e. it wins 32.10% of the games) versus versus 3 agents using the uniform distribution, so we can conclude that sampling types from the conditioned distribution during rollouts is better. As before, the effect on the planning time is quite small as seen in Table 6.4.

Modified	Baseline			
	Stac	uniform	unconditioned	conditioned
uniform	32.14%	–	22.25%	27.19%
unconditioned	25.63%	23.55%	–	17.25%
conditioned	34.75%	32.10%	34.80%	–

Table 6.3: Win rate of the TPOMCP agents while varying the distribution over types used in the rollout phase. Each result is measured over 2000 games.

Table 6.5 contains the statistics of the modified agents when these are playing versus Stac agents (i.e. the first column of results from Table 6.3). The effects of the two policies are very similar to the effects in the observable game (see Table 5.10). The

Agent	Min	Max	Mean	Median
uniform	456	7035	2113	2086
conditioned	857	25939	2178	2077

Table 6.4: TPOMCP's planning time in milliseconds while varying the type distribution, measured over 10 games.

conditioned policy biases towards building more settlements and cities while the unconditioned policy biases towards building more roads. The resource production of the conditioned biased agent is also higher than the agent with the unconditioned rollout policy. However, it is lower than the uniform agent's production which indicates that the conditioned agent is either more efficient in its use of these resources or targets building pieces that would generate the resources it requires.

It is interesting that the effects of the conditioned and unconditioned policies are comparable to the effects in the observable game, despite these being two completely different games. For example, the agents execute fewer actions on average (around 13 instead of 14 in the observable game) and focus more on playing development cards instead of building roads. These differences can be observed even in the uniform player that is unaffected by our modifications. The main reason is that development cards are kept hidden until are needed in the partially observable version of the game. The result is that the other players execute actions that force the owners of the development cards to play one of them. In the observable game, if you know a player has a knight you are less likely to attack them with the robber. Given that development cards provide reassurance, the planning-based agents focus less on achieving the longest road award and buy more development cards.

6.7.2 Experiments with Action Legality Probability

Given that planning in the belief space has not provided any additional benefits, we want to confirm which part of the model is detrimental: the action legality probabilities or the chance nodes. We focus here on the action legality function and modifications to it. First we look at its effects on the tree phase of the algorithm only. The first experiment that we run is to ignore this probability altogether and make each action equally likely $P(legal(a) = True|b) = 1$. Table 6.6 contains the results of TBMCTS and TBMCTSOR. We expected the performance to be slightly weaker, however there is no major difference between using and not using the legality probability.

Statistic	Agent		
	uniform	unconditioned	conditioned
Avg number of actions	13.4	12.96	13.43
Percentage build road	45.05%	46.30%	43.60%
Percentage build sett.	10.89%	11.89%	12.20%
Percentage build city	12.03%	11.56%	12.90%
Percentage play card	32.04%	30.26%	31.30%
Avg LA count	0.30	0.27	0.30
Avg LR count	0.97	0.92	0.95
Avg rss. from dice	55.28	52.15	54.65

Table 6.5: Statistics for the modified agents in Table 6.3 when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

Agent	10k	20k
TBMCTS	22.99%	31.29%
TBMCTSOR	30.88%	42.39%

Table 6.6: Win rates of the two TBMCTS agents against 3 Stac agents when the action legality probability is ignored. Each result is collected over 2000 games.

Another hypothesis is that some of these probabilities are so low that they cause UCT to never explore the corresponding actions. On the other hand, when ignoring the probabilities, UCT would over-explore unlikely actions. Therefore we smoothed these independent probabilities by increasing their temperature as following: $P(legal(a) = True|b) = \sqrt[\tau]{P(legal(a) = True|b)}$ and we have varied τ. Table6.7 contains the results of the two agents with the best tuned parameter τ. The TBMCTS agent has improved slightly but not enough to be comparable to the other agents. Smoothing the probabilities has most likely modified how the agent performs exploration fixing some of the issues created by the win/continue chance nodes. We suspect this based on the results we present in the following two subsections.

Another hypothesis that could explain the lack of improvement of the belief agents over sampling methods is that the belief planning algorithms still overestimate certain actions despite the action legality probabilities. This behaviour is caused by backprop-

Agent	10k	20k
TBMCTS	24.34%	33.04%
TBMCTSOR	31.96%	42.19%

Table 6.7: Win rates of the two TBMCTS agents against 3 Stac agents, while smoothing the action legality probability. Each result is collected over 2000 games.

agating simple returns of 1 for a win or 0 for a loss, and averaging them to estimate the value of children nodes $V(b') = \frac{W(b')}{N(b')}$, where $W(b')$ is the sum of rewards r received at that node. Given Equation 6.5 the action legality probability discourages selecting actions that are less likely to be legal given the current belief, but it does not take into account how likely the current estimation is given the whole sequence of actions. This means that the algorithm might select actions without taking into account the legality probability of the following actions. We can include this in the computation by replacing $W(b')$ with $G(b')$ in Equation 6.5, where $G(b')$ is computed as following:

$$G(b') = E[W(b')] = \sum_{i}^{N(b')} r_i \prod_{d=depth(b')}^{D_i} P(legal(a_d) = True|b_d) \tag{6.6}$$

In this equation, $depth(b')$ represents the depth of node with belief b' and D_i represents the maximum depth of ith iteration through node with belief b'. So, this equation computes the expected return with respect to the legality of the actions encountered during trajectory i through the tree. Unfortunately, this weakened the TBMCTSOR agent to a win rate of 24.50% versus 3 Stac agents. We believe that expected returns have an unwanted effect on the way UCT balances exploration and exploitation, since the expected returns diminish as the tree depth increases. Perhaps decaying the exploration parameter as a function of tree depth or using a smaller portion of the current iteration depth instead of the maximum depth D_i to compute the expectation could help alleviate this effect. Unfortunately, we were unable to explore this path further due to time restriction.

In TBMCTS, the action legality probability affects the rollouts also (see Section 6.6 for a description of the rollout algorithm). We attempted a final experiment, where we enforced the distribution over action types to be uniform instead of using the $\max_a P(legal(a) = True|b)$ values for each type to construct it. This modification had no impact on the performance of the TBMCTS agent. Since this is the only agent that is affected by this modification, we did not include this result in a table.

6.7.3 Experiments with Belief Chance Nodes

Since modifying the action legality probability function did not have any major impact on the performance of the belief agent, we focused our attention on the win/continue chance node, which decides when the game is won by an opponent versus when it continues. One of the initial motivations for planning in the belief space is that it will reduce the number of chance nodes: most transitions become deterministic modifications of the belief distribution. We discussed earlier the need to introduce the win/continue chance node. This is the one chance node that *is not* needed by the alternative 'sampling' approaches to adapting MCTS to games of imperfect information. We thought that the introduction of this chance node would be a price worth paying, however, because in practice this node should not be encountered too often in the rollouts. But as we will show in the next subsection, this node is actually very common for the methods that plan in the belief space.

Since we want to quantify the effect of the win/continue chance node on the performance of TBMCTS and TBMCSTOR, we make a modification to the game rules that results in the exclusion of this chance node. As mentioned in Section 6.3, keeping the victory point cards hidden are the only reason for its existence. We modify the game rules such that whenever a drawn development card is a victory point card the server reveals this information to the other players. All the other cards are kept hidden. Since this game modification provides valuable information to the players, we have decided to rerun the experiments for the TPOMCP agent also.

Table 6.8 contains the performance of TBMCTS or TPOMCP agents versus 3 Stac agents. Surprisingly, revealing victory point development cards has weakened the performance of the TPOMCP agent slightly. We believe this may be due to a possible improvement of the heuristic agent which no longer needs to estimate the score of each opponent. On the other hand, the number of games the TBMCTS agent wins did not change from the experiments presented in Table 6.1. Given these results we can conclude that the new chance nodes have an impact on the agent's performance. However this effect is not enough to explain the weak performance of TBMCTS versus the other agents. We believe that the weakness of this agent must also be due to the relaxations of the factored belief model described in Section 6.2. During lengthy rollouts, this error accumulates, as there is no feedback from the server to reduce the set of possible states. Unfortunately, we do not have a solution to this problem since the independence assumptions we made are very reasonable (see Appendix D).

Agent	10k	20k
TPOMCP	29.63%	40.46%
TBMCTS	22.85%	31.51%

Table 6.8: Win rates of TPOMCP and TBMCTS agents against 3 Stac agents, while varying the number of iterations. Each result is collected over 2000 games in which the VP development cards are observable.

6.8 Error Analysis

Following the experiments above that were designed to evaluate the effect of action legality probabilities and the win/continue chance node, we have decided to do a further analysis of the effects of the partial observability on the game complexity. We have gathered statistics for the resource model for one of our agents playing versus 3 Stac agents. TPOMCP and TBMCTSOR encounter very similar scenarios so we will only present these gathered when the modified agent is a TBMCTSOR. Table 6.9 includes these statistics over 100 games and we clarify here what each field means. First of all, this agent faces a total of 9211 decisions spread over the 100 games, which means that the agent makes 92 moves per game on average.[2] Out of these, our agent is uncertain of another player hand in only a little over half of them and in only a quarter of them it is uncertain of the hand of the next player in turn. Due to how MCTS builds the game tree following the order of play, we considered the uncertainty of the next player's state to carry more weight than that over the other players' states. When the true state of the resource hands is partially-observable, there are on average 1.62 players whose hands our agent is uncertain of. This is an expected value since there are 3 opponents. As a reminder, the resource belief model tracks each player's hand independently of the other hands and our agent's belief is represented as distributions over the set of possible hands for each opponent. When an opponent's hand is partially-observable, there are on average 5.39 possible hands that opponent could have. The median is of only 3, which indicates that in the majority of cases there is a small set of possible worlds the player needs to reason over even though the maximum is quite high (113). Furthermore, the statistics describing the entropy of the distribution over the set of possible hands show that the entropy is generally low. This means either there are few options or the distribution is quite peaked.

[2]Multiple consecutive offers are counted as a single move due to how the replans only after a successful trade or a different action type is executed.

Property	Value
Total decisions	9211
Decisions when at least one player's hand is po	5023
Next player's hand is po	2518
Number of players with po hands on avg	1.62
Number of hands when po mean	5.39
Max number of hands when po	113
Number of hands when po median	3.0
Number of hands when po variance	38.58
Max entropy	3.97
Min entropy	0.2
Entropy mean	1.16
Entropy variance	0.36
Entropy median	1.04

Table 6.9: Statistics for the players' resource hands over 100 games. po stands for partially-observable and avg for average.

We decided to gather the same statistics only when the number of possible hands is over 10 (see Table 6.10), to see the upper bound of the game complexity. These were collected from a different run, hence the slightly different values. As expected there are far fewer cases when there are a large set of possible hands ($\approx$13% of the game) and the agent is uncertain of the next player's hand in only half these cases. Again it seems that there are very few extreme cases where there are a large set of possibilities (up to 170) since the mean is low (20.32) and the median even lower (16). These cases would present a challenge to methods that sample fully-observable states since the min, mean and median of the entropy are significantly higher. However, these cases are quite rare.

These statistics indicate that the imperfect information elements of the game do not increase the complexity of the game as much as we expected. It is worth keeping in mind that the set of possible worlds that describe a complete game state is much larger because a possible game state is created via combinations of all the possible resource hands for each opponent plus combinations of the development card hands for each player. By assuming the players' hands can be tracked independently we have reduced the difficulty of tracking the game's imperfect information. Furthermore, the method that samples a fully-observable state samples each player resource hand first and com-

Property	Value
Total decisions	9740
Decisions when at least one player's hand is po	1224
Next player's hand is po	659
Number of players with po hands on avg	1.19
Number of hands when po mean	20.32
Max number of hands when po	170
Number of hands when po median	16
Number of hands when po variance	125.63
Max entropy	4.06
Min entropy	1.33
Entropy mean	2.34
Entropy variance	0.16
Entropy median	2.33

Table 6.10: Statistics for the players' resource hands over 100 games when the player hand must be at least 10. po stands for partially-observable and avg for average.

bines them with the information on the development cards afterwards. This bucketing or hierarchical approach ensures that the sampler is more likely to cover all combinations a player could have, even if it may not cover the whole set of possible complete state descriptions. Finally, action requirements are either resources or development cards but never both. So, it may also be that the action values doesn't differ very much between sampled fully-observable states because these vary in the description that is irrelevant for the action requirements.

We further explore our hypothesis about whether the TBMCTS' weak performance could be (at least in part) caused by many extra win/continue chance nodes during rollouts—in other words, is it the case that a player often encounters, during rollouts, a situation where it is unsure as to whether the game has ended? We counted the number of chance nodes over 100 games for TBMCTS and TBMCTSOR. The latter encounters on average 6 nodes per planning phase (i.e. over 10k iterations), while the former encounters a staggering 4407 nodes per planning phase. The majority encountered by TBMCTS are in the rollouts as there is no difference between the two algorithms in the tree level. These results in combination with the ones in Sections 6.7.2 and 6.7.3 are sufficient to illustrate that the new chance nodes are problematic to the performance of

the model. However, these show that belief rollouts are also affecting the performance at least in the game of Settlers of Catan and the cause could be the assumptions we made when creating the belief model. Unfortunately, the game rules make it very challenging to do further error analysis, such as saving or loading games with certain characteristics; it is very hard to guarantee that the characteristics we look for have a large impact on the game outcome or only a temporary effect. In addition we require a large number of samples to ensure the results are significant and it would be extremely time consuming to generate sufficient situations.

6.9 Conclusions

We implemented two well-known extensions to MCTS for handling imperfect information: POMCP and ISMCTS. In order to benefit from sharing statistics in the tree, we have defined a factored belief model that allows us to perform quick belief updates in the tree and represent the nodes using an abstract representation of this model rather than the history made of action-observation tuples. We also defined a belief transition function for the stochastic effects of the partially-observable moves following the game dynamics. These two permit us to traverse the Belief MDP directly without sampling fully-observable states. We introduced a MCTS algorithm that plans in the belief space of the planning agent and an approximation that samples observable states at leaf nodes only.

We evaluated these agents versus the state of the art rule-based agent in Settlers of Catan. All agents are able to defeat this baseline agent given sufficient computational resources. Our game analysis indicates that there are fewer situations where the game state is unknown than we originally anticipated. Furthermore, this game is known as a "Euro" game, where the interaction between agents is minimal and there is very little conflict in the game. In combination with the inability to perform complex negotiation moves, players can be successful even if they focus entirely on their own plan. Given these characteristics, planning in the belief in the tree phase of the algorithm may not provide additional benefits. Surprisingly, the weakest agent is the one that also performs rollouts in the belief space. Our evaluation indicates that the uncertainty over the conditions required for the game to end and the assumptions we made when constructing the factored belief model are weakening this method. Perhaps, our proposed methods could be successful if applied to games where the agents can always observe when the next action finishes the game. Other characteristics of such games should be

the fact that it is useful to reason over what the opponent can do, and taking the hidden stochastic effects into account can help to narrow the space of possibilities. One such game, is the game of Cheat, where the game ends when a player's hand is empty and drawing a card from the deck is a stochastic action. Also, in a game like Cluedo, sampling from the belief space essentially solves the game (i.e., if you can observe who performed the crime etc, then you can win by making the accusation). Observable rollouts in such a case would be really odd, because the sampled state always has a clear winning strategy and then the game ends! You would therefore never explore alternative moves, other than making the accusation, resulting in poor planning.

Chapter 7

Supervised Learning of Policies from Game Play

In the previous chapters, we extracted policies from the human corpus in an abstract version of the game. There is a concern that such an approach may ignore important information due to the performed abstraction. As a result we turn our attention to approximating the human policies as a state to action mapping and we implement several Neural Network models trained in a supervised manner. The thesis focus is on extracting useful information from a very small data set—useful in the sense that it can enhance a learning agent's ability to search the space effectively for strategies that help it win the game. As before, the objective is to show how both the ESH and MAH hypotheses described in Chapter 1 on page 5 can provide guidance in designing the Neural Network for the low-resource scenario. In this chapter, we will only use standard evaluation methods to measure the performance of the networks, e.g. cross-validation. The benefits of combining the final models with planning will be compared to that of the preference extraction methods in Chapter 8.

Deep Neural Networks (Hinton, 2007) have been successfully applied to many tasks such as visual processing (Lecun et al., 1998; Krizhevsky et al., 2012) and speech (Deng et al., 2013). They are particularly suited to tasks where abstract formalisations of the domain are not available, but large amounts of training data are available. As mentioned before there is a large literature showing that a DNN can be successfully combined with various reinforcement learning methods (Tesauro, 1995; Riedmiller, 2005; Mnih et al., 2015; Silver et al., 2016). All these models require large amounts of data, many iterations and a large computational budget to converge to a reasonable result. Furthermore, large amounts of human (expert) data, unlike the data of humans

playing Go (Silver et al., 2016), may simply not exist to train via supervised learning. When the available data is insufficient, reinforcement learning or Monte-Carlo sampling can be used to fine-tune the policy captured during the supervised step. The objective of this research is to assess the utility of very sparse and noisy datasets. To achieve our goal, we compare the performance of the models trained on our corpus versus those trained on a large dataset of synthetic play generated by the state of the art rule-based agent. Since the goal of this thesis is not to create the strongest Settlers of Catan player but to evaluate the benefits of a sparse and noisy human corpus, we have not combined the two datasets or performed any other data augmentation. Another well-known enhancement that is likely to improve any agent is that of bootstrapping simulated play with an existing database of expert play, e.g. Nair et al. (2018); Hester et al. (2017). However, the training period of such approaches is considerably more expensive which is in contradiction with our goal of learning in a low-resource scenario. Furthermore, it would not only highlight the utility of our corpus but also if it can be combined with other offline learning methods, so we leave this for future work on improving the agent.

Another limitation of prior research is that the optimal policies required to handle each phase of the game may be orthogonal or too complex to be represented by a single model. The amount or quality of the training data may also not permit learning to play all phases well. Very long complex games further aggravate this problem by requiring the optimal policy for one task to shift throughout the game. In Settlers of Catan for example, it may be risky to trade with a player that is close to winning the game, but it is generally useful to trade at the beginning of the game. For the similar reasons, the best results in shooter games are achieved by modularising the models into a *mixture of experts*, where each expert specialises on one of the portions of the game (Tastan and Sukthankar, 2011; Lample and Chaplot, 2016). These approaches reduce the number of actions an agent has to consider, making the Q-function simpler to learn (Gaskett et al., 1999). The result is a much shorter training period and improved performance of the final agent. This approach dovetails with our ESH: it is, after all, a method for limiting search by exploiting the structure of the game that is defined by the game rules. Accordingly, we also explore how specialising parts of the model to specific phases in the game affects policy learning.

In the following sections, we present an integrated suite of techniques for enabling deep learning to cope with very small amounts of (human) training data. The models in previous work on complex games are not easily extended to games where the action

space is massive. To address this gap, we will design our network so that it gains efficiencies from the fact that only a subset of the possible actions are legal in any one state of the game. Following the MAH, we use a model of the environment to inform the neural network of which actions are legal in a given state. In addition, we present a mixture of experts model, where each network specialises on a specific phase of the game. We then take advantage of the larger amounts of data available for certain phases and use transfer learning in order to improve performance on the phases for which very little training data is available. We show the benefits of using a mixture of experts on synthetic data gathered from agent simulations in *Settlers of Catan* (where the players in the simulations were Stac agents), followed by applying transfer learning via pre-training on data gathered from human participants playing the game.

7.1 Datasets

We are evaluating the models on two datasets made of games of Settlers of Catan: a synthetic and a human one. Table 7.1 presents a comparison of the two datasets in terms of the amount of data available. The synthetic dataset was created using the JSettlers framework by running 4-player game simulations between the best heuristic based agent (what we have been calling the Stac agent) (Guhe and Lascarides, 2014a). The environment allows us to collect as much data as we want and permits evaluating how the model scales with the amount of data. The training set was collected from 5000 games and the test set from another 1000 games. The parameters of the models were fit on an evaluation set taken from another 1000, which we later discarded.

On the other hand, the human play has been extracted from the corpus described in Section 3.4 (Afantenos et al., 2012). In addition to containing a total of only 60 games, this dataset presents a completely different learning problem to that of the synthetic one. First of all, it contains many games that have less than 4 players. The optimal strategy in such games may differ entirely to that of 4 player games that we aim to test our final agent in. Secondly, these players use different policies and the policy of an individual player may be mixed (so what they do now in a given state may be different from what they would have done in that same state a while ago). Finally, the participants are not necessarily experts in playing the game, so the policy learned by our models cannot be assumed to be even a good approximation of the optimal policy (let alone truly optimal). Given the size of the dataset, we did not split it into training, evaluation and test sets, instead we only performed 10-fold cross-validation to evaluate

Task	Synthetic	Human
Free Road (0)	54k	446
Free Settlement (1)	48k	390
Normal (2)	60k	7588
Before Roll (3)	60k	461
Discard (4)	21k	192
Robber (5)	60k	858

Table 7.1: Total number of states in both datasets split according to the task. See Description 3.2.1 for a description of the 6 tasks.

the models.

The two datasets do not contain the negotiation actions preceding a trade. We make the assumption that the exchange executed in the real game is acceptable for both players and that it is the best action these players could possibly execute in the current game state. This may diverge from the player's preferred trade option in some cases—it remains an interesting path to explore in future work.

7.2 Representation

Many of the existing approaches in games use the raw data (Mnih et al., 2015; Silver et al., 2017), thereby enhancing the model's appetite for very large data sets to learn useful abstractions. We diverge from this trend and form the input as a 1D vector of numerical features, taken directly from the game state representation plus a set of features that abstract over the coordinates of pieces and board configuration. The abstraction is necessary for effective learning from the very small training set of human data that we have access to. Information on the number of pieces on the board of each type, the robber, longest roads, award owners and the visible information on development cards and player's hands is taken directly from the game state. The exact location of each piece is represented via their significance with regards to the board description by including the information on resource production, the expansion possibility (i.e. if the player is blocked) and future production of the closest legal locations. Since we are abstracting state descriptions from the board coordinates, using exact action coordinates is nonsensical. We chose to also represent the actions as feature vectors similar to how we represented them in Chapter 4. We achieved this by subtracting the state

vector in which the action was executed from the outcome state vector (following the representation of actions as transitions between states). Only the features that can be modified by actions are included in the feature vector describing an action.

The imperfect information is handled by assuming that the players have no information aside from the observable portion of the game since the human games do not include information on the players' beliefs. However, the vector contains information on the player that is next to move which is not observable by the other players. We will show in Section 7.7 how we modify this when building a model of an opponent whose exact hand our agent cannot observe. For non-deterministic actions, specific features are computed using the expected outcome following the game rules (e.g. resource production is decided based on the chances of rolling two six-sided dice). The action of stealing is the only exception, since the player executing the action doesn't know the victim's hand. There are a total of 157 state features and 73 action features. The complete list is included in Appendix G. Since our features are numerical, the features are normalised such that the distribution over their values in the training dataset has 0 mean and unit variance.

None of the features presented above inform the model of the benefits of certain actions. However, there is no doubt that this abstraction aids the learning process, especially given the complexity of the game and the lack of human data. We do not have a baseline model that learns from the raw description of the game, since we strongly believe the performance of such a model will be very weak. The abstraction considerably reduces the number of parameters required. A model that does not benefit from such a dimensionality reduction would overfit our human dataset. Therefore, the input of all the models presented in this section is created with the abstraction described above.

7.3 Frequency Baseline

Before we continue describing the performance of the baseline model, we will briefly discuss how we measure the performance of the models. We report our results as accuracy defined as the number of times the model selects the correct action (i.e. the action chosen by the player from the set of legal actions) over the total number of samples. A sample is formed of the state description, the descriptions of each legal action and the description of the action chosen by the player in the game (i.e. the label action). This performance measure reflects the goal of learning to imitate the policy

that generated the data and not predicting the action's class.

We created a frequency-based baseline that counts the frequency of the action descriptions that were also the positive label actions in the training set. It then selects the action with the highest count from the set of legal ones for each sample during evaluation. Table 7.2 contains the accuracy of the frequency-based model on the two datasets on each of the 6 game tasks described on page 38. We "trained" one baseline model per task. For this experiment only, the human data was split in two sets: 90% for training and 10% for evaluation. All other models can easily outperform this baseline. The inability of the baseline to select the correct option on the initial placement task is most likely due to the large number of initial board positions (i.e. $\approx 1.2 * 10^{15}$). It is very unlikely that the same action will be available and also chosen by the rule-based agents across multiple games. On the other hand, task 3 contains the roll action which is very often chosen over the alternative of playing a knight card. It seems the human players prefer ending their turn over other actions in the normal task (i.e. task 2). Perhaps, this is due to a preference for waiting to roll the needed resources over trading with the opponents.

Task	Synthetic		Human	
	Eval	Train	Eval	Train
0. Free road building	29.09%	28.82%	39.77%	33.52%
1. Free settlement building	0%	0%	0%	0%
2. Normal	6.58%	6.42%	10.91%	11.29%
3. Before roll	75.09%	75.22%	77.17%	74.25%
4. Discard	11.18%	10.36%	13.16%	4.55%
5. Move robber	8.33%	8.06%	4.71%	4.8%

Table 7.2: Frequency-based baseline accuracy on the two datasets.

7.4 Single Model

In general, the number of classes is hardcoded in the output layer when training a classification model. Previous research afforded such an approach since their training data was not limited and the number of actions that can be executed in the game is relatively small compared to our setting. We have performed an abstraction as shown in Section 7.2 over certain state features which aids considerably in reducing the space

of the problem. This reduces the number of parameters required to define the model, but it creates a disconnection between the new state definition and the actions that are defined using coordinates on the board. To overcome this, we have also parametrised the actions in the same space. The effect of this parametrisation is that the total number of actions has increased slightly, but sufficiently to make training with the amount of data we have difficult again (see Appendix G). Our approach of parametrising the actions is similar to that of He et al. (2015). However they have applied their method to a reinforcement learning task and have not evaluated the benefits of normalising the network's output before the backpropagation step.

Fortunately, we have access to a game model which we previously used to perform game simulations. Following the Model-based Abstraction Hypothesis described in Chapter 1 (see page 5), we show now how we can use the game model to reduce the complexity of the learning problem. The game model, which we refer to as M, can present the list of legal actions $A_l \subset A$ given the current state of the game $M : S \mapsto A_l$. So, the network only needs to evaluate this set of options A_l which represents the actions that can be executed in the current game state. This set, made of 65 actions on average (see Table 3.1), is much smaller than the full set of actions A (i.e. 1882, see Table 1.1). A smaller set of target classes also means that the network has a reduced number of parameters. As a result, the model's capacity is reduced and it is less likely to overfit. Using a game model to reduce the output layer is inspired by the fact that a player does not need to reason over the illegal actions; e.g. a player never weighs building a settlement versus which opponent to steal from because these two actions are never legal at the same time. By limiting the number of options we greatly simplify the problem: the network does not need to learn which action is legal. Prior work assumes a fixed set of classes in the output layer and the cost of this is the models need to also unnecessarily learn which actions are legal.

Since the cardinality of the set of legal actions $|A_l| = n$ varies from one state to another, we need to make our network adaptable to this number. We inform it of this number n via the shape of the input that is created from the output of the model M. The input is a matrix, where each row is a vector of features that represent the current state and one of the legal actions. This is achieved by replicating the state vector n times, concatenating each of them with the vector describing one of the legal actions and stacking the resulting vectors. The resulting matrix has n rows and 231 columns (including the bias), where n is the number of legal actions and 230 (157+73) is the size of the feature vector resulting from concatenating the state and action representations.

This stacked input can be seen as a minibatch. However, the size of this minibatch varies from one sample to another since the number n of legal actions also varies.

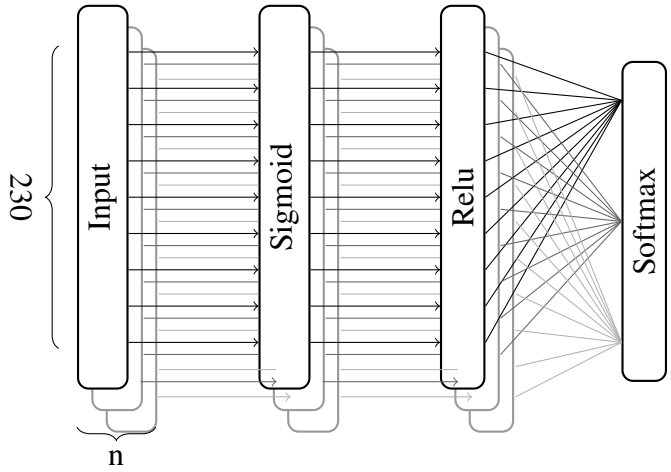


Figure 7.1: A diagram of the DNN which shows the computation flow to illustrate how the input is minibatched, and not the way the nodes are connected. The hidden layers are fully-connected in reality.

The network is a feedforward neural network with two hidden layers, as shown in Figure 7.1. Both hidden layers are fully-connected with 256 nodes. The first layer has a sigmoid activation function while the second one uses a rectifier linear function. The output layer implements a softmax function that outputs a distribution over the n actions. The network doesn't predict which is the correct class given the input, but rather what is the probability of each input pair being the correct one taking into account the other pairs. This comparison between the legal state–action pairs is achieved by sharing the weights that are used to evaluate each of them followed by turning the output of the network into a distribution with the softmax layer. Another view of this approach is that the network learns a representation for a state–action pair which allows it to separate the correct pair from the rest by a large margin. The separation is imposed via the cross-entropy loss with hard labels of $[0, 1]$, where 1 is assigned to the correct pair and 0 to the rest.

The standard softmax formula is shown in equation 7.2, where $\mathbf{z}$ is typically computed as $\mathbf{z} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$ ($\mathbf{h}$ is the previous layer's output and $\mathbf{b}$ is a vector containing the biases). The softmax equation shown here is following the standard activation function equation shown in Equation 2.8 (see page 23) but using linear algebra to compute the function's input. $\mathbf{W}$ is the matrix that contains the weights and its size is usually fixed according to the number of classes. However, we do not know the number of classes

beforehand and defining a matrix that would encode all possible actions is not feasible. Instead we use a 1D vector of weights $\mathbf{w}$ to compute the unnormalised probabilities of each action i separately as shown in equation 7.1. The results are stacked to form the full $\mathbf{z}$ which is then normalised using the softmax function. Since this vector of weights is shared among the inputs, the number of total parameters that need to be learned is dramatically reduced (from $n_{classes} \times n_h$ to only n_h, where n_h is the number of nodes in the previous hidden layer). The partial derivative of the softmax function with respect to its input has not changed so the derivative of the error with respect to the network's weights is the same and the training can be done using the standard backpropagation algorithm.

$$z_i = \mathbf{w}^T \mathbf{h_i} + b \tag{7.1}$$

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{7.2}$$

The network can handle any number of rows of the input matrix and therefore any number of classes. In fact, the architecture is equivalent to evaluating each state–action pair iteratively and turning the resulting vector into a probability distribution. On the other hand, our approach assumes a mutual exclusivity between the available (i.e. legal) state–action pairs, forcing the network to rank the options in the order of preference according to their features. It also normalises the output, making the training more stable. This normalisation is required since the large branching factor of the game causes the data to be very skewed towards the negative case. A simple alternative to our approach is to consider each state–action pair separately and perform a binary classification while iterating over each sample. This means that each pair is classified as true if the action is the one performed or false otherwise. This is incorrect, because a positive value on one pair entails negative values for all the other legal pairs, and yet in a binary classification task the pairs are considered independently.

Another advantage to our architecture is the ability to evaluate all possible actions in a given state with only a single forward pass through the network as it is done in a standard network using minibatch training. The proposed approach can be seen as an adaptive minibatch training, where the size of the minibatch varies with the number of legal actions in the current state. It is also straightforward to extend to the minibatch (of states) case. Most of the forward pass would be computed in parallel, except for the softmax operation which needs to be computed iteratively on each portion of the

input corresponding to a set of legal state–action pairs.

During training we feed the network one sample at a time, where each sample is a set of the legal state–action pairs stacked in a matrix format as described earlier. The updates are done using RMSProp:[1] they are performed using minibatches of size equal to the number of legal actions. We noticed a large improvement when using RMSProp compared to the simple update rule. We also tried using minibatches of states, but these gave no improvements. This could be caused by the lack of data or by the fact that the number of legal actions has a huge variance throughout the game. The network's weights are initialised using the Xavier algorithm (Glorot and Bengio, 2010).

Due to the abstraction described in the previous section, the target action that the network tries to learn may be encountered twice in the list of legal actions. We handle this rare case by giving equal weight to each of the corresponding indices in the target vector, such that the sum of the vector is still equal to 1. During evaluation either option is considered correct. As regularisation, we perform label smoothing (Goodfellow et al., 2016), by multiplying the target labels with 10 and normalising with a softmax function with temperature 1. This has an effect of reducing the hard 1 with ε and increasing the 0 with $\frac{\varepsilon}{n-1}$. We observed that uniformly softening the labels has resulted in 5% absolute increase in the validation accuracy on some tasks, but we do not include these fine-tuning results in the thesis. In combination with early stopping, this has been enough to prevent the models from overfitting. Other regularisation techniques such as parameter norm penalties or dropout reduced both the training and evaluation performance significantly. In order to keep everything clear, we will not provide results to show improvements due to parameter tuning.

The size of the network, activation functions, initialisation and update methods were chosen based on the performance on a different evaluation set made of synthetic data. Since this model is eventually combined with a sampling method to create an agent that takes decisions in an online manner, the prediction time was another reason for keeping the size of the model relatively small compared to the state of the art in other domains.

[1] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

7.4.1 Evaluation on Synthetic Dataset

Table 7.3 contains the results of this model trained on the synthetic data. Even though the model doesn't differentiate between the tasks in any way, there was a noticeable improvement when the model is trained by iteratively feeding it samples from each task, instead of randomly shuffling the order of the tasks or performing full epochs on each task. The results presented here are the best on each task separately, and are not taken from a specific epoch where the best overall performance was noticed. Therefore, the overall performance across the tasks as shown in the table is slightly better than if we stopped training the single model based on the average performance across the 6 tasks.

Task	Evaluation	Train
0. Free road building	67.01%	66.88%
1. Free settlement building	65.98%	65.75%
2. Normal	48.35%	47.30%
3. Before roll	80.07%	80.38%
4. Discard	38.87%	40.36%
5. Move robber	23.16%	24.51%

Table 7.3: Single model accuracy on synthetic data.

7.4.2 Logistic Regression Baseline

We also wanted to measure the effect of evaluating the set of actions together using the proposed softmax output layer with the multi-class cross-entropy loss function. We created a second baseline which is a DNN that differs to our model in the output layer only. Instead of the approach proposed in Section 7.4, it has a sigmoid function and we used the binary cross-entropy loss function. Each state–action feature vector is evaluated independently:

$$y = \sigma(\mathbf{w}^T \mathbf{h} + b) \tag{7.3}$$

Apart from this difference, the baseline network benefits from the same optimisation and regularisation techniques as the proposed network. We noticed it is very hard to decide on the threshold between the two classes due to how skewed the data is. To overcome this issue during evaluation, the pair that the network assigned the highest value is considered as labelled 1 and the rest are 0 for each sample.

This baseline can be trained with minibatches of different size then the size of the set of legal actions. We do not include the fine-tuning process of the minibatch size, since on average these are weaker compared to the case where the minibatch size is the same size as the legal set of actions given the game state. The latter is the same to how the proposed model is trained, using the game model. The adaptive minibatch aids the training process as it guarantees that each minibatch contains at least one positive class. It also ensures that all actions that are legal in one state are used at once during training.

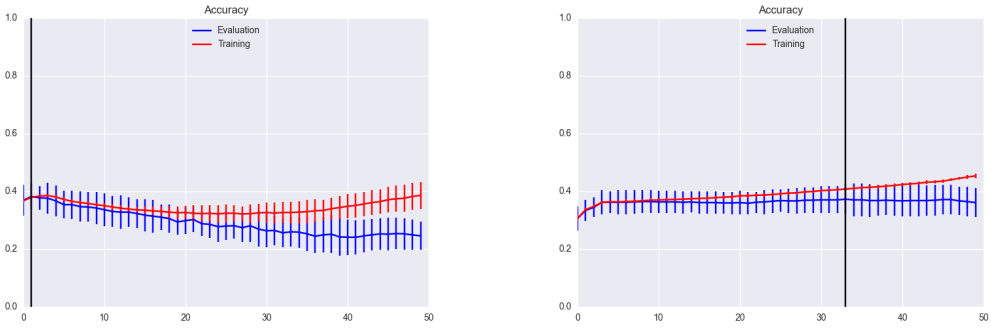
Overall, the results are considerably weaker than that of the proposed model on most tasks on the synthetic data as shown in Table 7.4. Similar to the other single model, these results are the best on each tasks individually. The only task where the two approaches have the same performance is task 3 which, as mentioned before, it seems to be a straightforward decision. Finally, the simple model is prone to numerical underflow due to the skewness of the data. It would require additional regularisation compared to the proposed network. To have a fair comparison between the two, the results presented here are without any additional regularisation, before the network encounters these issues.

Task	Eval	Train
0. Free road building	63.77%	64.21%
1. Free settlement building	54.4%	54.08%
2. Normal	45.06%	44.63%
3. Before roll	80.06%	80.11%
4. Discard	37.38%	37.93%
5. Move robber	22.44%	23.45%

Table 7.4: Baseline DNN accuracy on the synthetic dataset.

7.5 Mixture of Experts Model

The single model reaches the best performance on each task after a different number of iterations. Also, the single model displays a very unstable learning curve (see Figure 7.2 for a comparison between the single model and the mixture of experts model on one task of the human data). Our initial solution on the synthetic dataset was to cap the number of samples for each task to 50K, despite having access to consider-



(a) Single (b) Mixture of Experts

Figure 7.2: Learning curves of the two models on task 5 on the human data. Black vertical line indicates the highest mean value on the validation sets and vertical lines indicate standard deviation.

ably more samples on some of the tasks, such as the normal task. In combination with iterating over the tasks when performing updates, this has helped significantly to stabilise the training. On the human dataset, we cannot afford to throw away any samples. Furthermore, when iterating over the tasks, we cannot resample the ones that have a little amount of data as we risk further overfitting on them. Table 7.1 illustrates the lack of data and the skewness towards the normal task. As a result, we need a way of separating the learning process on each task to control it better.

Ensemble learning is a popular approach for supervised learning that uses multiple models to increase performance. One example that seems fit for our needs is a mixture of experts model that is based on the divide and conquer principle (Masoudnia and Ebrahimpour, 2014). Typically, a set of networks are trained alongside a gating network. The gating network chooses which expert to use for each sample (Jacobs et al., 1991). Another approach is to explicitly partition the space beforehand. Here we do the latter, by taking advantage of the existing game structure as shown in section 3.2.1 and predefine a function that chooses the expert deterministically. Each expert has the same architecture as the single model presented earlier. Hinton et al. (2015) have implemented a very related mixture of experts model, which they refer to as a set of specialists. However, the authors have separated their dataset using the predictions of a general model and clustering the set of classes based on how often these are predicted together. In our case, we have access to the game model that separates the data in a natural way that arises from the game rules.

7.5.1 Evaluation on Synthetic Dataset

Despite the high performance of the frequency-based baseline and the single models, the mixture of experts approach is significantly better on almost all tasks (see Table 7.2 for the performance of the frequency-based baseline and Table 7.3 for the performance of the single model). Table 7.5 shows these results and how the model improves with the increase of data. Furthermore, the learning curves no longer presented the unusual instability that we observed in the case of the single model. These results suggest that the performance is due to an increased capacity. In their setting, Hinton et al. (2015) have mentioned the specialist networks have a higher risk of overfitting. In our synthetic dataset we have not observed signs of overfitting. On the contrary, the generalisation error has increased noticeably only on two tasks, for both the baseline and the proposed output layer. We believe this is due to using a deterministic policy to generate the data since we have sufficient samples in the training set. Overfitting is slightly more visible on the human dataset for both models as shown in the next set of results.

Task	Size of training set							
	5k		10k		25k		50k	
	Eval	Train	Eval	Train	Eval	Train	Eval	Train
0	60.23%	59.32%	61.67%	62.45%	66.45%	71.50%	68.51%	70.90%
1	72.23%	76.74%	71.86%	73.9%	73.21%	73.96%	73.83%	73.45%
2	55.81%	55.97%	56.86%	57.90%	58.35%	58.39%	58.65%	59.33%
3	80.72%	85.1%	81.28%	85.31%	81.16%	86.11%	81.49%	84.76%
4	36.26%	54.34%	39.50%	46.48%	42.24%	55.42%	42.24%	55.42%
5	24.78%	33.24%	27.57%	39.34%	29.9%	42.08%	29.67%	36.92%

Table 7.5: Accuracy on synthetic data of the mixture of experts model while varying the amount of data available for training.

Table 7.6 shows that the baseline DNN is still weaker on average than the proposed network (Table 7.5), even though the mixture of experts settings has reduced the gap between the two considerably. There is however one task where the baseline outperforms the proposed approach: the discard task. It may be that considering the alternatives does not provide additional information since the players evaluate each action individually when presented with the decision problem in this task. Another possibility is that there is not an option that is clearly better than the others from the

list of legal actions because the options differ only slightly in this task. This result indicates that the proposed model performs best when applied to specific supervised problems, where a clear separation between the policy that generated the data and the other possible policies exists.

Task	Eval	Train
0. Free road building	67.04%	66.57%
1. Free settlement building	68.69%	68.02%
2. Normal	56.5%	56.71%
3. Before roll	81.67%	82.25%
4. Discard	45.51%	56.55%
5. Move robber	27.27%	30.69%

Table 7.6: Accuracy of mixture of baseline DNN on the full synthetic dataset.

The goal is to have a decent policy estimation that can predict the stored game play, so we can bias the search during online planning. Therefore, we are also interested in how the network ranks all the legal actions and how far the correct action is from the network's preferred option. Figure 7.3 shows the accuracy of the mixture of expert model as we allow the correct option to be in the first n options in the output (the value of n is given by the x axis). The accuracy for task 0 has an unusual curve due to most of the samples only containing 3 options, while the other tasks have many more than 10 legal actions (given the branching factor of the game tree). Overall, the correct option (i.e. the action observed in the evaluation dataset) is within the model's top 10 options 70% of the time. This is a promising result, suggesting that the network's output would be a good approximation of the synthetic policy and we could use this information to inform the tree search.

7.5.2 Evaluation on Human Dataset

We evaluated our models with 10-fold cross-validation over the whole set of human samples. In addition to displaying the evaluation and training accuracy, we included the 90% confidence intervals on the evaluation accuracy in the tables. The results of the single model are included in Table 7.7. This model presented slightly better performance compared to the standard mixture of experts on the tasks with fewer samples, indicating the benefits of sharing the knowledge between the phases. However, we observed that the training was very unstable and the accuracy on each task fluctuated

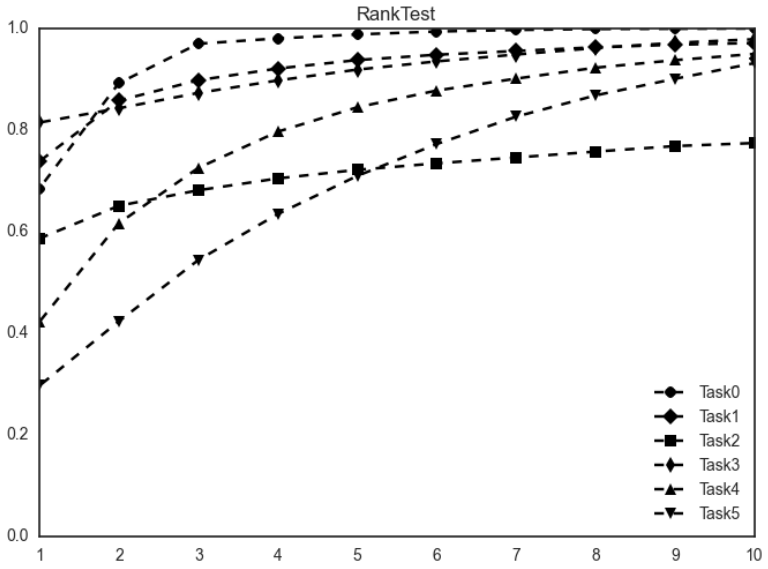


Figure 7.3: Percentage of correct classifications up to the first 10 choices of the Mixture of Experts model trained on synthetic data.

rather than having a steady increase with the epochs. Also, when the best performance on one of the tasks was observed, the performance on others was relatively weak, making it impossible to decide when to stop the training. The most likely cause is the lack of data in combination with the large difference in the size of the training sets. This is a characteristic of complex games: the mixture of experts model addresses this.

Task	Evaluation	Eval CI.	Train
0. Free road building	44.77%	±5.37	47.31%
1. Free settlement building	25.38%	±4.8	28.46%
2. Normal	62.14%	±1.55	63.53%
3. Before roll	79.35%	±3.07	79.3%
4. Discard	24.21%	±4.23	24.57%
5. Move robber	37.05%	±2.76	37.12%

Table 7.7: Accuracy of the single model on human data.

Table 7.8 contains the accuracy of the mixture of experts model. Due to the lack of data and the multitude of policies responsible for generating the human data (Afantenos et al., 2012), a weaker performance compared to the results on the synthetic data is expected as observed on tasks 0, 1 and 4. The results for the remaining tasks are very

similar. However, the performance on the move robber task is better. We believe this may be due to the features chosen, which may enable the learning algorithm to learn to explain a human behaviour much better than that of a complex heuristic agent that performs obscure computations over and above those including the visible features.

Task	Evaluation	Eval CI.	Train
0. Free road building	47.04%	±4.42	50.82%
1. Free settlement building	22.56%	±3.79	27.63%
2. Normal	62.71%	±1.39	63.22%
3. Before roll	81.3%	±3.77	82.89%
4. Discard	22.1%	±5.68	25.02%
5. Move robber	37.76%	±3.59	47.59%

Table 7.8: Accuracy of the mixture of experts model on human data.

We have only evaluated the baseline sigmoid output layer in the mixture of experts setting. Table 7.9 shows that the differences between the baseline and the proposed output layer are not as clear as in the synthetic data. However, the proposed approach performs better on the normal task (2) which is the most important task of the game. Another reason for the smaller differences could be that the human data is generated from many different policies, and encouraging a clear separation via the proposed output layer may not aid the learning process that much. As future work, it would be interesting to evaluate the two on a larger dataset of (human) games, while varying the number of policies (players) used to generate the data.

Task	Evaluation	Eval CI.	Train
0. Free road building	46.36%	±3.51	49.38%
1. Free settlement building	21.53%	±4.21	26.23%
2. Normal	59.92%	±1.37	60.60%
3. Before roll	78.7%	±3.27	81.10%
4. Discard	20.53%	±3.45	24.62%
5. Move robber	36%	±2.99	36.95%

Table 7.9: Performance of the mixture of baseline DNNs on human data.

7.6 Introducing Knowledge via Label Smoothing

In Chapter 4 we showed that a simple distance metric (e.g. cosine similarity) can be used to approximate the relevance of the actions from the corpus to those encountered in a new game. The relevance value has proven useful in designing a simple non-parametric method that would initialise a subset of current legal actions as well as rank them according to the average resemblance to the ones in the corpus. The performance of the model was surprisingly good and it would be interesting to use the similarity metric to provide additional information when fitting the Neural Network. Instead of adding noise to the labels, we have decided to create an interpolation between the real hard labels and the similarity of each action to the action labelled as 1:

$$z = w^l \times \mathbf{y} + w^s \times \mathbf{s} \tag{7.4}$$

w^l and w^s are weights that can be tuned, $\mathbf{y}$ are the true labels, $\mathbf{s}$ are the distances of each action to the action chosen by the human in the data computed using the cosine similarity metric as shown in Equation 4.2. We only used the feature vectors describing the actions to compute the similarity since the state features were the same for every action in the legal set. Using a similarity metric to rank the legal actions makes sense in Settlers of Catan given the representation we have chosen. As an example, a player may have to choose between two locations to place a settlement that differ only by a neighbouring hex and one location to place a road. If the best option is one of the settlements then it is obvious that choosing the other settlement should be punished less compared to building a road. Finally, the result of this interpolation is turned into a distribution using a softmax function with temperature 1 as before. The other parameters and the training procedure were the same as when we were training with the true labels only.

Table 7.10 contains the results of the Mixture of Experts model trained with the new labels. We have noticed that the best result was achieved by setting the weights $w^l = w^s = 3$. Unfortunately, we were unable to even achieve the same performance in some of the tasks compared to just adding random noise to the labels (see Table 7.8 for comparison). We also noticed that this method had no further benefits in reducing the generalisation error or in increasing the percentage of the time the correct label was found in the top 10 options of the model's output. It seems that reasoning about action relevance is more important when transferring to different states as we have shown in Chapter 4 then for ranking the legal action set in a given state as we did here.

Task	Evaluation	Eval CI.	Train
0. Free road building	47.27%	±5.05	50.35%
1. Free settlement building	20.77%	±4.4	25.01%
2. Normal	62.4%	±1.39	65.72%
3. Before roll	81.74%	±3.52	84.17%
4. Discard	22.63%	±4.68	37.1%
5. Move robber	38%	±2.41	39.7%

Table 7.10: Accuracy of the mixture of experts model with modified labels on human data.

We further hypothesise that the interpolation smooths the labels too much without providing a large amount of information to differentiate the actions between them. Actions only slightly modify the vector representation and the similarity values would be very close to each other. During the normal phase of the game where trading options dominate the action space for example, if the target action is trade ore for wheat then trading ore for clay and trading wood for wheat would be equally distant from the target action. As a result the similarity values resemble random noise and we just increased the amount of noise. Another possibility that could cause training to become unstable is that the similarity values may increase the differences between the already varied play encountered in our corpus. The output of the model may never get close to the saturation region of the output function and will therefore fluctuate a lot. Due to time limitations we have not insisted on finding out which of these two hypotheses is true. So we do not introduce knowledge via the labels when training any other models presented in the rest of the thesis.

7.7 Training an Opponent Policy for the Partially Observable Game

We aim to use the models presented here to bias the search in MCTS, by suggesting what is the best policy for each node for both our player and the opponent. We handled the lack of information our player has on the opponents' hands by ignoring information and using the totals that are observable. We must also account for this lack of information when the models suggest what an opponent can do when it is their turn. Since our agent has a belief represented as a distribution over the opponent's possible

hands, we can implement several methods:

1. Use the current trained Neural Network and sample n observable states whenever the agent is uncertain of what the current player's hand is. Use a weighted average of the outputs as a value to seed into the tree.

2. Mask the exact details on the resource and development cards for the current player in both the state and action representation. Retraining the Neural Networks on the same data represented with the smaller, abstracted feature vector.

3. Implement a different network architecture that can handle a sequence of states that would represent the history of the game. Retrain the network on the same dataset and hopefully it will be able to infer the information required from the sequence.

We chose option 2 on the basis that it provides a quick implementation that does not require additional computation. The first option required processing n samples that would estimate the current belief distribution. This is a sound approach and would probably work very well given that there are a small set of situations where there are a large number of possible states. But even for this small set, it will be hard to find the best balance between the time required for running the Neural Networks and that required for planning. Furthermore, combining Neural Networks with MCTS already increases the decision time significantly, as we will show in the next chapter. We would therefore be in danger of breaching the limits in decision time that human opponents would find acceptable.

We have discarded option 3, due to the lack of data in our corpus. Recurrent neural networks require an even larger amount of data. There is an exponential increase of the space of possibilities because trajectories are sequences of state–action pairs and there could be many combinations. Tables 7.11 and 7.12 contain the performance of the single and mixture of experts models trained on the human data with the masked features. There is no discard task in this case, since all discard actions in the partially-observable game appear the same to an external observer. Surprisingly, this modification didn't really affect the overall performance as much as we expected and even increased the performance on a small set of tasks. For comparison, see Table 7.8 for the unmasked MOE and Table 7.7 for the unmasked single model. It is worth mentioning that we kept the list of legal actions the same as in the observable case since we do not have access to the players' beliefs of what the opponents can and cannot do. Perhaps the

state features representing resources and development cards are not important in this case because the model can infer the benefits of the action from its effects only. Modifying the game model to produce a different set of legal actions based on a player's belief could provide more insight into this, but we have not explored this option due to time limitations.

Task	Evaluation	Eval CI.	Train
0. Free road building	44.32%	±5.19	49.33%
1. Free settlement building	23.08%	±4.38	26.23%
2. Normal	62.51%	±1.46	63.28%
3. Before roll	79.57%	±1.52	80.17%
5. Move robber	38.24%	±2.67	37.97%

Table 7.11: Accuracy of the single model on the masked human data.

Task	Evaluation	Eval CI.	Train
0. Free road building	46.82%	±5.27	49.98%
1. Free settlement building	22.31%	±3.6	25.5%
2. Normal	63.01%	±1.42	65.52%
3. Before roll	82.39%	±3.42	83.2%
5. Move robber	37.29%	±3.1	40.82%

Table 7.12: Accuracy of the mixture of experts model on the masked human data.

7.8 Transfer Learning between Experts via Pre-training

When little amount of data is available, splitting the data in order to specialise the models on specific tasks further reduces the amount of data each model encounters during training. We referred to the six game phases as tasks suggesting multi-task learning (Caruana, 1997) as a good method to regularise and improve the performance of the model further. Multi-task learning improves generalisation by learning multiple tasks in parallel while using the same representation. It would also allow a softer separation of the data since it allows transferring knowledge between the tasks. However, multi-task learning is performed by matching the same input to multiple labels, where each label is the target for each of the task the network needs to learn. Our setting differs

because we do not have multiple labels for the same input. In addition, one should not try to learn tasks that are impossible given a certain input. In Settlers of Catan all tasks are clearly separated, so the agent will never have to learn to discard in the initial phase of the game for example.

Nevertheless, we still want to transfer what was learned about one task to another, given that the corpus is made of only 60 games. This domain is suited to this technique because some of the tasks share similar dilemmas (e.g., a decision on where to build a road factors into initial road building and in the normal phase), and so it may help to use the larger data set for one task to inform learning on another for which the data set is much smaller. The training sets for some tasks are significantly smaller than those for others due to the rules of the game (e.g. 200 samples for task 4 and 6k training samples for task 2). Since part of the representation is the same, we can transfer the learned features from one task to another to increase the performance at least in the tasks with insufficient data. In computer vision, for instance, features learned on one task are general and transferable to other tasks (Yosinski et al., 2014), even if the samples do not come from the same distribution (Bengio et al., 2011). More generally, transfer learning has proved useful in both unsupervised (Mesnil et al., 2012) and supervised learning (Thrun, 1996), but it matters how transferable the learned features are. Furthermore, transfer learning should keep the experts consistent and the resulting combined policy will be closer to the policy used to play the games from which the samples were collected.

We perform the simplest form of transfer learning: we initialise the network's weights by pre-training for several epochs on the data available for the other tasks. The number of epochs was finetuned to the specific task, e.g. the best performance on task 4 was achieved after 10 pre-training epochs, while task 5 required only 5 epochs. Previous work has evaluated the performance of the network after transferring only part of it, while randomly reinitialising the weights of the rest. We achieved the best performance by transferring the weights of the full network, since our chosen architecture permits transferring the weights in the output layer also. The positive result enforces our belief that both the representation and the policy captured during pre-training contributed to the increase in performance.

7.8.1 Evaluation on Human Dataset

We have only focused on the unmasked Neural Networks for this experiment. As expected, transfer learning works best for fitting the tasks where less data is available, i.e. the initial placement(1) and discard tasks(4) (see Table 7.13 compared to Table 7.8). Even though the effect of the actions is not exactly the same across different tasks (e.g. placing the initial settlements yields an *immediate* production, in contrast to building a settlement in the normal part of the game), the weights learned over the other tasks help the neural network explain why certain features are important for the target task also. We did not see any improvements for the normal (2) and free road (0) tasks. The training data for task 2 is sufficient, while task 0 bares very little resemblance to any other task. The feature abstraction is also causing issues in task 0, as most of the cases where actions are confused with each other are included in this dataset. Unfortunately, pre-training also resulted in the model overfitting more on the training data in some of the tasks.

Task	Evaluation	Eval. CI.	Train
0. Free road building	46.59%	±4.65	51.94%
1. Free settlement building	27.44%	±4.83	32.11%
2. Normal	63.01%	±	63.47%
3. Before roll	83.69%	±2.14	87.25%
4. Discard	32.11%	±4.29	49.71%
5. Move robber	38.71%	±2.73	40.23%

Table 7.13: Transfer learning between experts on human data.

Similarly to the mixture of experts on synthetic data, Figure 7.4 shows the accuracy of the mixture of expert model with transfer learning on human data as we allow the correct option to be in the first n options in the output (the value of n is given by the x axis). Overall, the correct option is within the model's top 10 options 80% of the time, which surprisingly is higher than that on the synthetic data. Therefore this model could also be used to perform soft pruning of a search tree, despite training on a much smaller dataset.

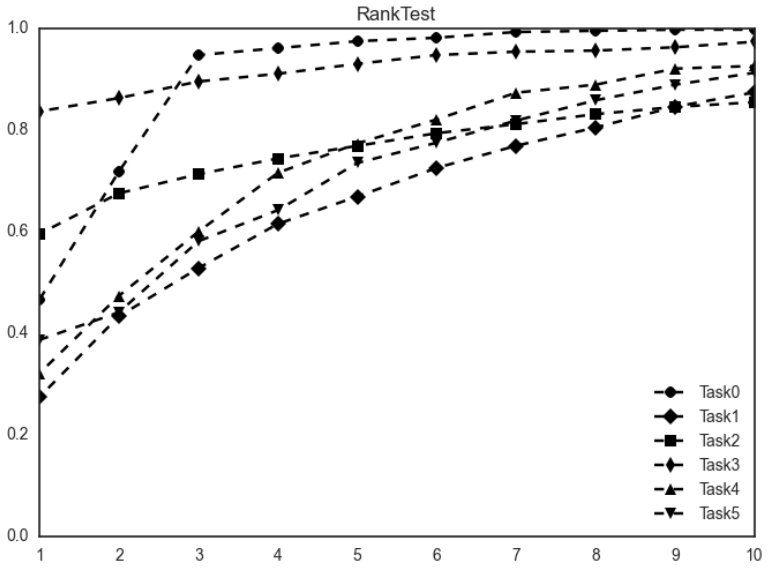


Figure 7.4: Percentage of correct classifications up to the first 10 choices of the best model trained on human data.

7.9 Conclusion

In this chapter, we have focused on extracting a policy from existing game play and increasing the accuracy of the models in the low-resource setting. Feature selection and extraction was required in our setting due to the very little amount of data. Despite this we have shown how we can use a model of the game to separate learning action legality from learning the correct action in order to keep the models small. Due to this modification, the networks had to learn in a setting similar to that of a binary classification, causing instability and weak performance due to data skewness. We took advantage of our game model to create minibatches of state–action pairs based on what actions were legal given a state and normalise the network output, essentially informing the model that the actions are mutually-exclusive. Both modifications were useful and improved upon a simple binary logistic regression approach, however batching has also proven useful in increasing the performance of the standard approach. Finally, we have explored introducing information via the labels without additional benefits to those of regularisation observed even when using uniform random noise.

We managed to achieve a good performance on both datasets and illustrated that the extracted policies could be useful in biasing tree search. The overall performance on the human data is slightly weaker, but this is expected given the simplicity of the

synthetic agent and the amount of synthetic data available. Also, a mixture of experts approach has proven more useful when there is sufficient data. On the human dataset we have showed how we can improve the performance by pre-training experts on the other tasks. The mixture of experts approaches are more prone to overfitting and we want to evaluate how these models behave on unseen game settings, i.e. different initial board configurations and versus a variety of opponents. In the next chapter, we evaluate which of the two datasets and models (single versus mixture of experts) are more appropriate for introducing knowledge in the tree phase of MCTS.

Chapter 8

Combining Extracted Policies with Monte Carlo Tree Search

In the previous chapter, we performed an intrinsic evaluation of a suite of models for predicting decisions about action in Settlers of Catan from both a synthetic data set and a much smaller corpus of human decisions. Our models were a vast improvement over the baselines. However, this is not necessarily an indicator on how the models will perform as part of an algorithm for learning decent strategies. This chapter is concerned with conducting an *extrinsic* evaluation of the models of the previous chapter. Our aim is to prove our main hypothesis detailed in Chapter 1: that even very small amounts of evidence of human play can be exploited to improve the performance of an agent that is learning a highly complex task. This chapter is therefore dedicated to combining some of our best models for predicting human decision making with our TMCTS algorithm from Chapter 5 (for the observable version of the game) and Chapter 6 (for the true, imperfect version of the game). Specifically, we will use the predictions to guide search on which actions to choose in the selection phase of TMCTS.

By combining TMCTS with the predictions about human decision making that are produced by the neural networks, trained on very small amounts of human data, we can evaluate the performance of the resulting agents as discussed in Section 3.5 to conclude which of the prediction models and datasets are more appropriate for learning policies in Settlers of Catan. Due to the normalisation effects of the proposed output layer over a binary classification, its resilience to numerical underflows and its slightly better overall performance on both datasets, we only compare models that implement this approach. We compare the benefits of having a single model versus a mixture of experts learning on both datasets (i.e., the synthetic data set produced by 4 Stac agents

playing each other in game simulations, and the much smaller data set of human play). We also evaluate if there are any benefits to transfer learning on the human dataset. Finally, we pitch the best agents from the two datasets against each other to evaluate the utility of learning from human data versus synthetic data.

In addition we compare the best approach to informing the tree level of the TMCTS algorithm: a Neural Network that provides a policy as a mapping from states to actions or a high-level policy as a mapping from states to action types learned via MLE. As a last experiment, we create an agent by combining the best improvements presented in this thesis and pitch it against the current state of the art agent, Stac. This chapter thus constitutes an empirical evaluation: the best combination of models for data mining, exploiting and exploring the complex task emerges after fine-tuning the parameters and comparing its performance to that of the alternatives.

8.1 Algorithms for Informing the Search

The literature contains many examples of combining UCT with domain knowledge, but there is very little comparison between the available options. Throughout this section, we will refer to the action of initialising the value of the node in the game tree to be explored via MCTS as *seeding* and the value used as a *seed*. We have considered the following existing approaches of informing MCTS:

- Naive seeding;
- Progressive bias;
- Rapid Action Value Estimation (RAVE);
- Bayesian UCT;
- Predictor + UCT (PUCT);
- Pruning;

Naive seeding has been implemented before in Settlers of Catan by Szita et al. (2010). This approach simply seeds the nodes with a value to initialise the number of wins. The authors have chosen the seed using a set of heuristics: they have observed that the performance of their agent versus the JSettlers agent from Thomas (2004) has increased significantly when biasing the search towards preferring building settlements

and cities. This simple form of seeding doesn't scale well to different opponent types and it is also dangerous since preferring building settlements or cities over every other option may not be good in every situation, e.g. when only poor locations are left buying development cards is a better alternative to achieving the last victory points. Naive seeding also relies on the software developer's own expertise on the task, since the heuristics are manually chosen.

One simple improvement would be to decay the effect of the seed. There are essentially 4 methods for implementing a decay effect on the (initial) seed so that information learned from MCTS iterations can (eventually) take over. Firstly, Chaslot et al. (2008b) introduce progressive bias, in which the UCT computation becomes: $UCT_{pb}(s,a) = UCT(s,a) + \frac{P(s,a)}{N(s,a)+1}$, where $N(s,a)$ is the number of times action a was selected from state s. $P(s,a)$ is the seed value that is calculated using domain knowledge.

UCT Rapid Action Value Estimation (RAVE) (Gelly and Silver, 2007) similarly decays the effect of the seed by combining the prior value with the UCT value as such: $UCT_{RAVE}(s,a) = \alpha P(s,a) + (1-\alpha)UCT(s,a)$, where $\alpha = \max\{0, \frac{V-N(s,a)}{V}\}$. V is the parameter that defines the amount of decay. The effect of the seed decays over time and the results from previous MCTS iterations take over. The benefit of RAVE is twofold: it handles both the uncertainty of the sampling method when the sample set is small (i.e., in the first few MCTS iterations), as well as taking into account that the seed may only be a general suggestion of what is a good action to perform. Usually, RAVE is combined with the all-moves-as-first heuristic, which reuses the information stored in the tree if the action was previously executed. However, it can be used with other forms of knowledge, such as a neural network that evaluates the utility of the move. Progressive bias and RAVE could also be combined together as shown by Graf and Platzner (2016).

On the other hand, we can view the seed as a means of performing an informed exploration of the space. Even though it is not clear if the information provided by the seed is good or bad, it is an initial guess. Therefore, it makes sense to combine it with the exploration term in UCT as it is done in the variant of PUCT that was used in AlphaGo (Silver et al., 2016): $PUCT(s,a) = Q(s,a) + CP(s,a)\sqrt{\frac{\sum_{a' \neq a} N(s,a')}{1+N(s,a)}}$, where $Q(s,a)$ is the action value, n the number of times its parent was visited and C the exploration parameter. As before, we can easily combine with afterstates to compute the value of action a executed in state s with the outcome state s': $PUCT(s,a) =$

$V(s') + CP(s,a)\sqrt{\frac{\sum_{x \neq s'} N(x)}{1+N(s')}}$, where x are the outcome states of the other legal actions that are not a.

Finally, we could use the distribution given by the predictor as a prior in Bayesian UCT (Tesauro et al., 2012). Unfortunately, their experiments show that Bayesian UCT is much slower than other approaches and we would like to keep the agent's planning period under a threshold that would be tolerated by human opponents.

There are also many pruning methods developed for MCTS (Browne et al., 2012). These generally require a careful design to make sure the branches that are pruned do not contain the optimal solution. Since we cannot trust the seeding method entirely, pruning is not appropriate. Other methods and variations are available, but we focused our attention on RAVE and PUCT given their performance in similarly complex games. The first experiment we performed with each of the models was to fine-tune the parameters. During this phase we observed RAVE to have a reduced overall performance compared to PUCT. The extra V parameter is very difficult to tune due to the large space of values that can be explored. RAVE was significantly weaker across many runs, therefore we chose PUCT, and we will only report the results achieved with this seeding strategy. Every TMCTS or TPOMCP agent in this section uses the PUCT selection policy, even if these are not seeded (which is equivalent to the seed being a uniform distribution over all legal or possible actions). We observed an improvement when using PUCT with a uniform distribution over the UCT policy we used before, so we will also use PUCT in the baseline TMCTS or TPOMCP agents.

8.2 Implementation Details

Previous work assumes access to unlimited resources (Silver et al., 2016) and the possibility to run the predictor on separate GPU threads. The models we present here, while they could be extended to such a scenario, are designed to be run on a standard desktop machine. The positive results presented later underline that our methods can be implemented even if one doesn't have a large computational budget. Our experiments in which we use the Stac agent as the baseline are run on CPU only machines with a maximum of 4 cores and 8Gb of RAM, where each game simulation uses a single CPU (i.e. the agents share the resources). In the partially-observable case or when multiple TMCTS combined with Neural Networks agents are playing the same game we allocate 16Gb of RAM.

Given the hardware limitation, the 4 threads that we used in Chapter 5 to conduct the TMCTS planning need to be shared with the prediction step. One could simply allocate n threads to the planning method and the remaining to the seeding. A quick evaluation showed that this approach results in either resources not being used, or insufficient resources being allocated to the prediction step. In the latter, we cannot properly evaluate the utility of the seeding method because it is not keeping up with the planning method. Our solution is to have a common thread pool and a priority queue, where the seeding has a higher priority. We also limited the seeding to use a maximum of two threads at one time to prevent starving the planning method. On average, over 95% of the tree is seeded at the end of the planning phase.

The evaluations performed in the seeding step are triggered during the expansion step. we used 6 queues when informing MCTS with the Neural Networks, one for each task (recall the definitions of the six tasks from Chapter 3, page 38). In the mixture of experts case, each queue is allocated to the corresponding expert, while the same model processes every queue in the single model case. To further take advantage of the parallelisation performed by linear algebra programs, we use a minibatch of 10 sets of state–action pairs which is equivalent to ten newly expanded nodes. Seeding the tree asynchronously permits us to wait until the batch is formed and perform fewer forward passes through the networks. We have not performed extensive tuning of the batch size, but we noticed that this number and the priority queue are sufficient to perform incremental evaluations that keep up with the tree expansion. The statistics and seeding values for the new nodes are used to compute their PUCT value as soon as these are available.

A detail worth clarifying is that PUCT requires $\sum_a P(s,a) = 1$, so we ensure this is true in our implementation. Given how a state can be visited following multiple trajectories in the tree, this aspect is very important.

8.3 Observable Version of Settlers of Catan

We first focus on the observable version of the game, where we use the best parallel TMCTS model presented in Chapter 5. That is, the model that:

- uses afterstates and transpositions;
- can execute unlimited trades during a turn;
- exploits a typed rollout, using a uniform distribution over action types.

Due to time and computational restrictions, we have focused on seeding TMCTS agents that run 10k iterations only.

The networks were trained via supervised learning while trying to enforce a maximum margin between the positive class and the rest. So these models assign the majority of the mass to one of the label classes similar to how they have been trained. Such a peaked distribution would reduce the amount of exploration done in the tree level and limit it to a small set of options. The effect would be a very aggressive pruning of some of the branches, and we already discussed that this is not appropriate as we cannot trust the data we have. A similar result was observed in AlphaGo, where the authors have smoothed the output of the neural network Silver et al. (2016). We employ the same approach and tune the value of the temperature τ in the softmax function:

$$softmax(z) = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}} \tag{8.1}$$

Game simulations with MCTS agents with unlimited trades are very long, and so given the eagerness of these agents to trade, it is impossible to find the best set of parameters when pitching the agents against another MCTS baseline. Therefore, we have chosen the Stac agent as our baseline for tuning the parameters. Random is too weak of a baseline to be informative of the best parameters; any reasonable agent has a 99% win rate against 3 random agents. Each combination of TMCTS with one of the networks is first tuned against Stac by performing a grid search in the parameter space, then evaluated against a TMCTS agent without seeding, a TMCTS agent with uniform seeding, and finally against the other combinations.

The results presented in Chapter 7 for the Single models are the best accuracy the model achieves for each task at different stages during the training. In general, the decision to stop the training for a Single model is taken based on the overall best performance without taking into account if the performance for a specific task is or not the best across all the epochs. We therefore retrain the Single networks on the human and synthetic datasets, while stopping based on the best overall performance. On the human data, we also retrain the other models (including the mixture of experts) on the full dataset and stop the training at the best epoch indicated by crossvalidation. This is a standard approach when insufficient data is available (Goodfellow et al., 2016).

8.3.1 Seeding with Neural Networks Trained on the Synthetic Data

We will first show the performance of the combination of TMCTS with the networks trained on the synthetic data. Table 8.1 contains the evaluation of all the combination agents, where the rows show the performance of the modified agent versus the baseline agents of the type shown on the columns of the second row. The first column shows the name of the modified agents and the second column shows the best performance of the agents against the Stac baseline after tuning the parameters. TMCTS with the single model (TMCTS-Single) has an exploration parameter $C = 6$ and temperature $\tau = 10$, while the combination with the mixture of experts model (TMCTS-MOE) has $C = 6$ and $\tau = 2$. Surprisingly, the combination with the single model seems to be a better choice against the Stac agent. However, considering that both models have been tuned against this agent, this result does not indicate how the combination generalises to other policies. The second column shows the performance against the TMCTS agent without any seeding (i.e. with PUCT and a uniform distribution over the legal actions). The combination with a mixture of experts is better than the combination with the single model, but neither are able to defeat TMCTS with uniform seeding. The synthetic data is generated by a single deterministic policy which visits similar situations across games, and we believe that the Neural Network overfitted to these situations. As future work, we could generate the data with an ε-greedy strategy to overcome this problem. The final two columns compare the agents against each other. Seeding with the mixture model is significantly better than seeding with the single model, when the two agents are pitched against each other. The last columns also illustrate that the base TMCTS has a harder time against 3 TMCTS agents combined with a mixture of experts than with the single model.

Modified	Baseline			
	Stac	TMCTS	TMCTS-Single	TMCTS-MOE
TMCTS	38.89%	–	30.47%	26.69%
TMCTS-Single	37.56%	18.25%	–	22.35%
TMCTS-MOE	36.8%	23.35%	28.88%	–

Table 8.1: Win rates of the TMCTS agents combined with the networks trained on synthetic data. Each result is measured over 2000 games.

8.3.2 Seeding with Neural Networks Trained on the Human Data

Table 8.2 displays the same evaluation of the models trained on the human dataset with the addition of the combination with transfer learning (TMCTS-TL). The best parameters versus the Stac baseline are $C = 4$ and $\tau = 4$ for all combinations. The results of these combinations are very different to the ones with the models trained on synthetic data. First of all, every combination is able to significantly defeat both baselines: Stac and TMCTS with a uniform prior. Secondly, the combination with the single model appears to be slightly better than TMCTS combined with a mixture of experts, even though these results are not significant. Also, transfer learning does not improve the agent; the performance is weaker, but not significantly weaker. Due to time limitations, we did not run further experiments to confirm these differences. It is possible that transfer learning causes the output of the network to be even more peaked over the option the network considers the best. The three combination on the small human corpus have comparable performances, and, different to the results during crossvalidation, there are no clear benefits of using a mixture of experts on the human data when combining with TMCTS. It is possible that a mixture of expert only increases the capacity of the model since the mixture of experts combination significantly outperforms the single combination on the large synthetic dataset.

Modified	Baseline				
	Stac	TMCTS	TMCTS-Single	TMCTS-MOE	TMCTS-TL
TMCTS	38.89%	–	19.60%	20.15%	20.40%
TMCTS-Single	42.55%	31.05%	–	26.02%	–
TMCTS-MOE	41.40%	28%	23.50%	–	24.95%
TMCTS-TL	41.45%	27.8%	–	24.29%	–

Table 8.2: Win rates of the TMCTS agents combined with the networks trained on human data. Each result is measured over 2000 games.

Table 8.3 contains the decision time of a TMCTS agent combined with the Single model versus that of an unseeded TMCTS agent. As before these are measured over 10 games versus 3 Stac agents. There is a large increase in the time required when adding seeding with the Neural Network. The increase in performance versus Stac agents is slightly less than doubling the number of iterations which yielded a 45.75% win rate versus 3 Stac agents (see Table 5.3 on page 68). However, running 20k iterations would double the average decision time, while seeding with the Neural network only

increased it by approximately 30%.

Agent	Min	Max	Mean	Median
TMCTS	45	3852	1433	1445
TMCTS-NN	567	7427	2078	2118

Table 8.3: The planning time in milliseconds of the TMCTS agent seeded with the single model versus a standard TMCTS agent. These are measured over 10 games.

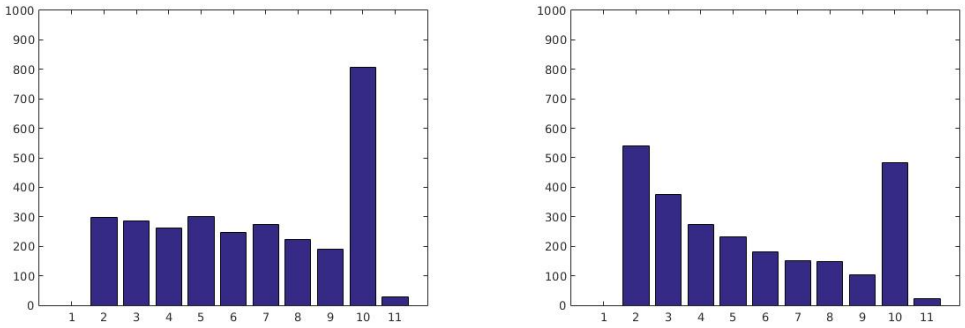
8.3.3 Comparison between Datasets

Previous results show that the combination with networks trained on human data have a higher win rate against the two baselines: Stac and TMCTS. The goal of this set of experiment is to directly compare the usefulness of the two datasets. The synthetic dataset is a very clean dataset generated by a single deterministic policy and it contains a large number of samples compared to the human dataset. The human dataset contains gameplay generated by players with a variety of strategies and of different level of skill. When the corpus was gathered, the human players were given the chance to play a few practice turns. However, it is still possible that the data is noisy: these players may prefer the physical game or did not have enough time to get used to the interface. To evaluate the two datasets, we compared the best models trained on each of them (i.e. TMCTS combined with the mixture of experts trained on the synthetic dataset, and TMCTS combined with the single model trained on the human dataset). The results presented in Table 8.4 and the victory points shown in Figure 8.1 indicate that, despite these limitations, training on human data is considerably better than training on the synthetic dataset generated by the Stac agent. This synthetic agent is a strong baseline that is comparable to human strength (Keizer et al., 2017), and it required many years of effort to develop (since the original JSettlers agent of Thomas (2004)). Despite this, the result highlights the clear benefits in using a human corpus, even with its clear limitations in terms of size and quality. It is possible that the variety of the gameplay is enough to overcome the extreme differences in the amount of data. Another possibility is that the noise contained in the human dataset has a regularising effect.

Table 8.5 provides more insight as to why biasing the planning agents with the offline policies trained on human data is so succesful. The synthetic trained network biases the agent towards building more pieces and to disregard playing development

Modified	Baseline	
	TMCTS-Synth	TMCTS-Human
TMCTS-Synth	–	20.14%
TMCTS-Human	28.69%	–

Table 8.4: Win rates of the TMCTS agents combined with the best models trained on each of the two datasets. Each result is measured over 2000 games.



(a) Human dataset

(b) Synthetic dataset

Figure 8.1: The two best combined agents' victory points, where x axis represents number of victory points and the y axis the number of games.

cards including aiming for the largest army award. Development cards provide countless benefits such as additional VP, knights to defend against the robber, free resources and roads. These benefits can turn the tide of a game, especially when the opponents ignore them completely. It is very likely that training on a single type of agent is the culprit for having such an unbalanced strategy. Another downside of combining with the synthetic trained network is the inefficiency of the agent in terms of resource usage. The TMCTS-Synth agent produces far more resources but it executes fewer actions. However, this could also be due to building settlements and cities being more expensive than buying development cards.

8.3.4 Seeding with the Action Type Distribution

Given the benefits observed in sampling actions from the action type distribution during rollouts, an obvious simple seeding method is to initialise the nodes in the tree based on this distribution. We used the same PUCT action selection policy but instead of using the output from the Neural Network, we computed a probability distribu-

Statistic	Agent	
	TMCTS-Synth	TMCTS-Human
Avg number of actions	12.66	14.63
Percentage build road	54.07%	48.17%
Percentage build sett.	16.05%	12.92%
Percentage build city	16.83%	10.53%
Percentage play card	13.05%	28.38%
Avg LA count	0.09	0.4
Avg LR count	1.06	1.04
Avg rss. from dice	57.55	52.72

Table 8.5: Statistics for the two combination agents TMCTS-Synth and TMCTS-Human when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

tion over the legal actions given their type $\pi_{type}(s) = P(a|type(a) = t)$ using one of the stochastic policies over types $\pi_t(T) = P(T = t)$ learned via MLE as described in Section 5.4.3. It is straightforward to compute this probability by setting each $P(a|type(a) = t) = \frac{P(T=t)}{N(t)}$, where $N(t)$ is the number of legal actions of type t. We evaluated both type policies: the unconditioned one and the one conditioned on what other action types are legal. We noticed that these distributions can get very peaked and we want to avoid aggressive prunning. As a result we have also tuned a temperature parameter versus Stac agents in order to smooth these distributions. The best parameter for the conditioned policy is 7, while the best for the unconditioned one is 5.

Table 8.6 contains the performance of these agents versus the Stac baseline and versus a TMCTS agent with uniform seeding. Surprisingly, these agents achieve similar or better performance than the agents seeded with the Neural Network versus the two baselines. It may be that the Neural Network provides information that is too specific due to how it maps state description to actions. The action type distributions are quite general and as a result a better fit for influencing the exploration term of UCT. Perhaps the amount of data we have is not sufficient and the Neural Network has overfit to the parts of the abstracted space encountered in our corpus. The agent seeded with the

conditioned policy seems to be slightly weaker when playing versus the agent using the unconditioned policy when these two play versus each other. However, the differences are not significant to the baseline performance of 25%. Also, the conditioned seeded agent performs better versus both baselines (Stac and uniform seeded TMCTS), while the unconditioned seeded agent's performance is not significantly better than the baseline performance when playing versus the TMCTS agent with uniform seeding.

Modified	Baseline			
	Stac	uniform	unconditioned	conditioned
uniform	38.89%	–	23.79%	22.25%
unconditioned	42.30%	26.55%	–	25.40%
conditioned	44.36%	30.47%	24.10%	–

Table 8.6: Win rates of the TMCTS agent with PUCT while varying the action type distribution used for seeding. Each result is measured over 2000 games.

The statistics for these modified agents are included in Table 8.7. The effects of the two distributions is comparable to that observed when it was used in the rollout phase (see Table 5.10).[1] The two agents build more settlements and cities over roads compared to the uniform agent, and are also more efficient in their use of resources. However, these effects are less pronounced compared to when the distributions were used in the rollout phase of the planning algorithm. This can be explained by the smoothing performed with the increase of the temperature parameter. Smoothing the distributions maintained the exploration ability of PUCT but reduces the effect of the type distributions. Another interesting observation is that the differences between the two agents is neglijable which explains the comparable performance observed in Table 8.6.

There seems to be a difference between biasing the tree phase and biasing the rollout phase with the same policy. When the tree phase is informed, the agent prefers playing development cards. On the other hand, playing development cards is less preferred when the rollout phase is modified. This is true for both conditioned and unconditioned distributions, and in both observable and partially observable versions of the game. Learning how to play development cards requires reasoning about opponent strategy more than the other actions. It is possible that combining these distributions

[1]The statistics of the uniform agent are different in this chapter compared to those in previous chapters because the uniform agent uses PUCT with a uniform prior in this chapter. UCT was used in the previous chapters.

with UCT in a multi-agent planning method results in the agent observing the large benefits of having answers to the possible opponents' moves. This is only a hypothesis and further investigation is required.

Statistic	Agent		
	uniform	unconditioned	conditioned
Avg number of actions	14.28	13.80	13.99
Percentage build road	51.56%	50.03%	49.54%
Percentage build sett.	13.11%	14.29%	14.58%
Percentage build city	10.97%	10.92%	11.02%
Percentage play card	24.36%	24.76%	24.86%
Avg LA count	0.30	0.29	0.32
Avg LR count	1.05	1.06	1.06
Avg rss. from dice	53.30	51.17	51.72

Table 8.7: Statistics for the modified agents in Table 8.6 when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

Table 8.8 shows that seeding with the type distribution has a negligible effect compared to seeding with a Neural Network. This is expected, since the former is a simple lookup operation in comparison to the computations performed by the network which require dedicated threads. Despite this, the performance versus 3 Stac agents is better. Since the two combinations also have comparable performance versus the TMCTS agents, simpler and quick methods are more appropriate in the low-resource case even in complex games as Settlers of Catan.

Agent	Min	Max	Mean	Median
TMCTS	45	3852	1433	1445
TMCTS-typeSeed	213	4053	1602	1658

Table 8.8: The planning time in milliseconds of the TMCTS agent seeded with the type distribution versus a standard TMCTS agent. These are measured over 10 games.

8.4 Partially-observable Version of Settlers of Catan

There are only a few minor modifications required to combine the masked Neural Network (described in Section 7.7 on page 139) with a MCTS algorithm that can handle the partially observable version of the game. During planning, we use the masked neural network to compute a distribution over the possible actions in the current belief, i.e. $P(b,a)$, rather than the legal actions given a state, which is the $P(s,a)$ previously used. There is a slight mismatch to how we trained the masked networks on sets of legal actions given the masked true state, but we do not have access to the true game state during planning. Our adaptive network design allows us to easily compute the probability over a larger set of actions by modifying the input. Regarding the planning algorithm, we have chosen to use TPOMCP since there was no major performance or computation complexity difference to TISMCTS or TBMCTSOR algorithms (see Section 6.7 for the comparison). As selection algorithm we adapted PUCT with afterstates to use statistics stored in belief nodes instead of state nodes:

$$PUCT(b,a) = V(b') + CP(b,a)\sqrt{\frac{\sum_{x \neq b'} N(x)}{1+N(b')}} \tag{8.2}$$

We used the unmasked seeding when our agent is next to move during the planning phase, and we added the masked implementation to seed the nodes that correspond to an opponent move. The first required modification for the opponent case is to reduce the state representation s by masking the features representing resources and development cards that our agent cannot observe. As explained in Section 7.7, this results in a pesimistic approximation of the belief state b as it assumes that our agent cannot infer anything about the hidden resources or development cards. Secondly, we need to handle how illegal actions given the current observable sampled state s are masked in TPOMCP. The masked Neural Network introduces a distribution over the possible actions, but only a subset of these are legal in each iteration of TPOMCP. So, the illegal actions given the sampled state are ignored during the selection phase (see masking description in Section 6.4.1). In this case the probability distribution used in PUCT may be less than 1, reducing the amount of exploration. To overcome this, we must normalise the distribution by multiplying each probability with a $\mu = \frac{1}{\sum_a \prod_{(s,a)P(b,a)}}$, where $s \sim b$ is the state sampled by TPOMCP at the root node and updated as the

game progressed:

$$PUCT(b,a) = V(b') + C\mu P(b,a)\sqrt{\frac{\sum_{x \neq b'} N(x)}{1+N(b')}} \tag{8.3}$$

8.4.1 Seeding with Models Trained on Human Dataset

We chose to only evaluate the TPOMCP algorithm combined with the Neural Networks that are trained on the masked human data, since training on human data yielded better results than training on synthetic data in the fully-observable scenario. Also, we have focused on evaluating the combination with the Single model only, because it had a slightly better performance than the other agents in the observable version of the game. Due to time restrictions, we leave the comparison of different combinations in the partially observable game to future work. We have retuned the temperature and exploration, but observed that the same values of 4 yielded the best performance. We limited the number of offers to 3 for all experiments in the partially-observable version of the game. Table 8.9 contains the results. As expected the overall performance of the agents combined with seeding is slightly weaker than in the observable version of the game (see Table 8.2), since the imperfect information increases the complexity of the game and limiting the number of offers reduces the performance of the agent as shown before. At the same time, seeding was less beneficial relative to PUCT with a uniform distribution over actions, even though the seeded agent is still the best performing agent. One of the reasons for this could be that we train the models on the set of legal actions rather than the possible actions. We chose to do this since we do not have access to the human players' belief. Secondly, preparing the input for the neural network takes longer than in the observable case since we need to mask it. Even though seeding still keeps up with planning, it may be that the small delay in providing the seed is enough for its utility to be reduced. Unfortunately, we didn't afford to explore these two hypotheses further.

The statistics in Table 8.10 indicate that combining the planning agent with the Single model trained on human data has a minor effect on the behaviour of the agent. This could explain the minor difference to the uniform biased agent. The most noticeable differences are the reduced preference of building roads and the increased preference in playing development cards. This is in line with the effects of biasing the tree phase of the algorithm with the type distribution.

As before we also evaluated seeding with the action type distribution. However,

Modified	Baseline		
	Stac	TPOMCP	TPOMCP-Single
TPOMCP	33.45%	–	19.65%
TPOMCP-Single	34.89%	30.69%	–

Table 8.9: Win rates of the TPOMCP agent combined with the Single model trained on human data in the partially-observable game. Each result is measured over 2000 games.

Statistic	Agent	
	TPOMCP	TPOMCP-Single
Avg number of actions	13.55	13.57
Percentage build road	45.20%	43.80%
Percentage build sett.	11.01%	11.13%
Percentage build city	12.14%	12.15%
Percentage play card	31.65%	32.92%
Avg LA count	0.29	0.32
Avg LR count	0.99	1.00
Avg rss. from dice	56.00	55.32

Table 8.10: Statistics for the modified agents in Table 8.10 when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

instead of spreading the mass assigned to one type over all legal actions of that type, we now spread it equally over all possible actions of that type. As in the case of seeding with the Neural Network, we used the normalised PUCT algorithm in the selection phase of TPOMCP. We tuned the temperature parameter again and observed that the same values yielded the best results: 5 for the unconditioned distribution and 7 for the conditioned one. The results are included in Table 8.11. The differences between the conditioned and the unconditioned distributions are further reduced to the point where there is no difference between these two. It is possible that limiting the number of offers the agent can make is one of the reasons the unconditioned distribution performs so well. One of the aims of the conditioned distribution was to capture the fact that trading is a means to obtain the resources required for other action types, by

extracting the player's preferences over the action types conditioned on what types are legal. Furthermore, compared to the massive differences between the conditioned and the unconditioned distributions when these are used in the rollout phase, the differences when these are used in the seeding phase are very small in both observable and partially-observable cases. Perhaps smoothing also has an impact on reducing the differences between the two, however increasing the temperature is needed to combine these with PUCT.

Modified	Baseline			
	Stac	uniform	unconditioned	conditioned
uniform	33.45%	–	21.01%	22.05%
unconditioned	40.06%	29.75%	–	25.32%
conditioned	40.56%	29.53%	24.52%	–

Table 8.11: Win rates of the TPOMCP agent while varying the action type distribution used for seeding in the partially-observable game. Each result is measured over 2000 games.

The effect of the conditioned and unconditioned distributions over types on the overall behaviour of the agent is very similar to those in the observable version of the game (see Table 8.12 compared to Table 8.7). The only difference is that the unexpected bias towards playing development cards is further increased. This version of the game is partially observable which supports the hypothesis presented earlier: learning how to play development cards requires reasoning about player interactions. As future work, additional experiments such as making some of the agents omniscient may aid in explaining this effect.

We included the statistics regarding the decision time depending on what seeding method was used in Table 8.13. *TPOMCP-TS* is the TPOMCP agent seeded with the conditioned distribution. Similar to the observable case (Tables 8.3 and 8.8), seeding with the type distribution has a minor effect on the decision time while the Neural Network is very expensive. Unfortunately, the decision time combined with only a minor improvement in performance indicates that using a Neural Network is not appropriate for our setting. On the other hand, using the typed distribution yielded a performance close to that of doubling the number of iterations and at a fraction of the cost. A TPOMCP agent running 20k iterations wins 42.23% of the games versus 3 Stac agents and a TPOMCP agent seeded with the conditioned type distribution wins 40.56% of

Statistic	Agent		
	uniform	unconditioned	conditioned
Avg number of actions	13.55	13.59	13.57
Percentage build road	45.20%	42.47%	42.19%
Percentage build sett.	11.01%	11.84%	12.02%
Percentage build city	12.14%	12.05%	11.97%
Percentage play card	31.65%	33.64%	33.81%
Avg LA count	0.29	0.34	0.34
Avg LR count	0.99	1.02	0.98
Avg rss. from dice	56.00	54.69	54.32

Table 8.12: Statistics for the modified agents in Table 8.11 when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

the games.

Agent	Min	Max	Mean	Median
TPOMCP	456	7035	2113	2086
TPOMCP-NN	793	19160	3372	3170
TPOMCP-TS	411	10096	2201	2165

Table 8.13: The planning time in milliseconds of the TPOMCP agent seeded with various methods versus a standard TPOMCP agent. These are measured over 10 games.

8.5 Final Agent

We dedicate this final small section to a small set of results with our final best agent. This agent is the best TPOMCP agent described in Chapter 6 with the conditioned typed rollout and the conditioned type seeding. We refer to this agent as *TPOMCP-TS-CR* in this section. We evaluate this agent versus Stac, versus TPOMCP that uses the conditioned type seeding but uniform typed rollouts, referred to as TPOMCP-TS, and versus TPOMCP that doesn't use any additional information in any of the two stages of the algorithm, referred to as TPOMCP. The MCTS-based agents run 10k iterations only. The results are in Table 8.14. This agent easily defeats the Stac agent,

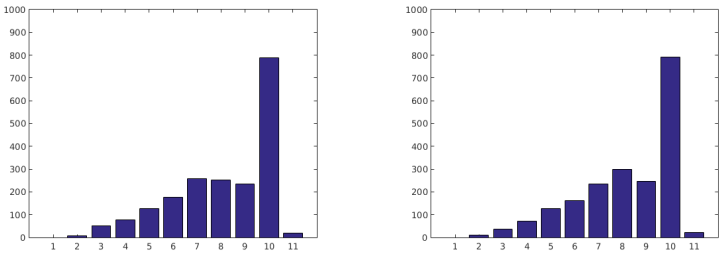
Modified	Baseline				
	Stac	TPOMCP	TPOMCP-NN	TPOMCP-TS	TPOMCP-TS-CR
TPOMCP	33.45%	–	19.65%	22.05%	31.50%
TPOMCP-NN	34.89%	30.69%	–	24.80%	24.70%
TPOMCP-TS	40.56%	29.53%	24.82%	–	24.71%
TPOMCP-TS-CR	40.70%	47.50%	33.30%	30.73%	–

Table 8.14: Win rates of our strongest agent TPOMCP-TS-CR versus Stac and other combinations of planning with offline methods over 2000 games.

the state of the art rule-based agent. However, the improvement brought by the conditioned typed rollouts does not seem noticeable (40.70% compared to 40.56%). We have performed further analysis of the results and observed that the conditioned typed rollouts increased the average number of victory points to 8.21 from 8.12 with the uniform typed rollouts. As shown in Figure 8.2, this is possibly due to a better performance in the early and mid game, since the agent with the conditioned typed rollouts finishes more games with 8 or 9 victory points when it does not win.

The benefits of the conditioned typed rollouts are more noticeable versus TPOMCP agents. The TPOMCP-TS-CR agent wins 30.73% of the games versus 3 TPOMCP-typeSeed, while the ablated agent does not achieve a performance significantly different to the baseline performance (24.71%). Against 3 TPOMCP agents without any seeding and with uniform typed rollouts, our best agent wins 47.5% of the games (from 29.53% as shown in Table 8.11). However, the TPOMCP agent is also able to defeat 3 of our best agents, even though it is winning fewer games in total (31.50%). This is an interesting result, and we can only hypothesise that either our improved agents are very competitive with the other 2 versions of themselves or that the uniform TPOMCP agent plays very different to our best agent. Taking into account opponent types might help with this behaviour, but we leave this and further analysis to future work.

Comparing with the statistics in Tables 6.5 and 8.12, the statistics in Table 8.15 show that biasing both the rollout and the tree phases at the same time results in an agent that takes the best from both. It is the most balanced agent in terms of playing development cards, and it focuses even more on building settlements and cities over roads since the two biases agreed in this regard. An additional benefit that can be observed is that the agent is faster in achieving the victory condition. This is illustrated by the fewer number of actions executed on average and fewer resources produced on average.



(a) TPOMCP-TS (b) TPOMCP-TS-CR

Figure 8.2: Victory points for the TPOMCP-TS-CR agent versus 3 Stac agents compared to the Victory points for the TPOMCP-TS agent.

Statistic	Agent TPOMCP-TS-CR
Avg number of actions	13.33
Percentage build road	40.88%
Percentage build sett.	13.40%
Percentage build city	13.13%
Percentage play card	32.60%
Avg LA count	0.30
Avg LR count	0.92
Avg rss. from dice	53.06

Table 8.15: Statistics for our strongest agent (see Table 8.14) when playing versus 3 Stac agents. Sett stands for settlement, LA for largest army award, LR for longest road award and rss for resources. Averages are computed over the number of games, while the percentages are computed out of the average number of actions (i.e. the first statistic).

Finally, we compare the performances of the TPOMCP and TPOMCP-TS-CR agents versus 3 Stac agents. This experiment shows how the algorithm scales with the number of planning iterations. The results in Table 8.16 indicate that the bias from the MLE trained policies is useful even if the algorithm is allowed to perform a large number of iterations. Our TPOMCP-TS-CR agent performs better than an uninformed TPOMCP agent that runs 2 times more iterations.

Agent	10k	20k	30k	40k
TPOMCP	33.45%	42.23%	44.5%	47.94%
TPOMCP-TS-CR	40.70%	49.00%	52.65%	53.65%

Table 8.16: Win rates of TPOMCP and TPOMCP-TS-CR agents against 3 Stac agents, while varying the number of iterations.

8.6 Conclusions

We presented an empirical evaluation of the improvements we have developed in this thesis by comparing the resulting agents in a tournament setting. Our best agent is a combination of all the methods we developed. This agent can easily defeat the current state of the art rule-based agent under a very tight computational budget. We expect large increases in performance if more resources are allocated since the planning agent could explore more of the game tree. There are several conclusions we can draw from the current results. First of all, the human data has not only proven sufficient but also essential in creating the best agent. This agent heavily relies on the various extracted policies. Secondly, human data is far better than synthetic data even if the latter is in large quantities. We attribute this fact to the diverse play that is usually present in a corpus generated by a large number of different human players. Thirdly, our ensemble approach was only useful when large quantity of data was available. The large increase in the model capacity as well as combining with pre-training seemed promising during standard cross-validation. However, these methods show signs of overfitting in comparison to a single model and have weaker performance when applied on unseen data during planning. Finally, a Maximum Likelihood Estimation approach that learns a high-level policy over types has proven more successful than deep learning techniques in our setting. This emphasizes the need for simpler and more robust methods in complex games when limited resources are available.

Chapter 9

Conclusion and Future Work

The aim of this thesis was to evaluate the requirements for creating a strong agent for a highly complex board game. Furthermore, we set out to explore different ways of extracting additional information when there is a lack of data and the available samples are very noisy. One of the main hypotheses was that we can exploit the environment's structure to overcome limitations in both data and computational resources and still improve the performance of our agent over the current state of the art.

Our main contributions include several key extensions to standard planning and learning models which have significantly increased their computational and data efficiency. Furthermore we have shown that access to enormous amounts of example game play is not a key requirement in creating a competitive agent. A highly sparse human corpus containing non-expert play can still be mined to increase the performance of any unbiased planning method. An important conclusion is that integrating high-level preferences extracted from the corpus that describe a human player's overall behaviour has been essential in creating our strongest agent. This highlights the importance of robustness in noisy and low-resource settings.

Finally, we have thoroughly analysed the game domain and the characteristics that make it a very difficult environment for both humans and machine learning techniques. In doing so, we have also shown that planning in the belief space is not appropriate and presented a detailed error analysis to justify our conclusion.

9.1 Main Findings

Before providing a summary of the main contributions of the thesis, we give a short reminder of the challenging aspects of the game:

- massive state space;
- massive action space;
- imperfect (and incomplete) information;
- the game depth is large (and typically larger than complex games tackled to date, such as GO);
- action space dominated by type of actions that are beneficial but not sufficient for winning the game.

First of all, the game has a massive state space which is further aggravated by the large diversity of the starting states. Secondly, a tabular representation of the action space without considering action legality provides a large hypothesis space for learning policies represented as mapping states to probabilities over the action space. The branching factor is large, but the huge depth compared to other domains increases the already large strategy space. Another important characteristic is that the action space is dominated by actions that are generally only needed to satisfy the requirements of other actions (e.g. trading which is needed to get resources and execute other actions) without a huge immediate benefit. The game requires planning a long sequence of actions while avoiding cycles that are unnatural in expert games. Finally, the imperfect information increases the state space and makes it challenging to reason over opponents' intentions.

Exploiting game structure during planning and learning. Our best performing models take advantage of the game structure as defined by the game rules. In Chapter 5 we showed how we can incorporate a prior distribution over action types during the rollout phase and increase the expressiveness of the policy. This simple modification combined with a uniform prior is the single most important change required to reduce the negative impact of having an action space dominated by one action type. We further improved the performance of the agent by extracting preferences over action types from our corpus. We attribute the increase in performance to the capability of the new rollout policy to generate trajectories that resemble trajectories generated by a standard player. In Chapter 7 we specialised models for certain tasks of the game. This has increased the accuracy on each of the task for both synthetic and human datasets during offline evaluation.

Model-based supervised learning. In Chapter 7 aimed to use training data to estimate which action from those that are legal in the given state are sufficiently optimal

to be worth exploring. This approach permitted keeping the output layer significantly smaller thus reducing the model's number of parameters compared to using the tabular representation of the action space to define the output layer. As a result the model generalisation error is reduced and we can learn from a very small dataset in a game with massive state–action space. We used a game model to form the minibatches of legal actions given a game state. Training with these minibatches has proven key in achieving a better performance over training with fixed size minibatches since even binary logistic regression models benefited from this batching policy. Introducing a mutual exclusivity via the softmax output layer punished representations that did not permit easy separation of the correct action, resulting in higher performance over a standard binary classification task.

Combining preferences with action legality. In Chapter 5 we showed how a simple MLE model can be used to extract preferences that provide a high-level description of what a standard human player would do. We showed how a richer description of these preferences by conditioning them on what other action types are legal can dramatically improve the rollout policy. This conditioning incorporates strong preferences over types that help differentiate actions required for satisfying requirements of other actions from those that are important milestones for winning the game. We observed the same effect in the partially-observable case (Chapter ch:poplanning). When seeding the tree policy (Chapter 8), the benefits of conditioning on action types legality was not as pronounced. Nonetheless, this simple baseline has proven both more efficient and more effective than training a Deep Neural Network to estimate a policy that maps states to actions. The conclusion is that combining methods that extract high-level strategies or meta-information about policies with planning at decision time can be very effective in learning to play complex games when faced with a lack of computation power and limited noisy data.

Belief Monte Carlo Tree Search. We have evaluated several known MCTS extensions for planning in partially-observable games in Chapter 6. Following on the positive results in Chapter 5, we have shown how a belief abstraction method can be used to integrate transpositions and afterstates into POMCP such that node statistics are shared. The benefits outweigh the increase in computation required to do several belief update steps in the tree level of MCTS (we suggested a cheap alternative that only updates the abstract representation for environments where even a small set of exact updates would be too expensive). We compared POMCP to a version of MCTS in the Belief MDP that does not sample fully observable states but rather retains the

uncertainty about the game state during the tree search and rollouts towards the game's end state(s). Surprisingly the exact Belief MCTS algorithm had a weak performance, while an approximation that samples observable states at the leaf nodes is comparable to POMCP's performance. Our results indicate that sampling is sufficient if statistics are shared and that there are no further benefits to reasoning in the belief space. The error analysis indicates that there are several reasons why POMCP is sufficient (given our factored belief representation):

1 The entropy of the distribution over the belief factors that are relevant to the current player's actions is usually low.

2 A smaller portion of the game than we initially believed is spent in partially-observable states.

3 Settlers of Catan requires chance nodes for handling the uncertainty of the game outcome (win or continue game) and these directly influence the reward function. Planning in the belief encounters many of these chance node and the uncertainty in the rollouts results is increased.

4 The observation model is deterministic so the trajectories in the belief space strongly resemble those in the state space. Planning in the belief should help further to aggregate the trajectories in the state space and share results, but we did not observe this benefit.

5 Settlers of Catan is a Eurogame where there are very few interactions between players. This means agents do not need to reason over opponents' hands other than when trading. Being able to accurately track what an opponent has and can do may be very important in human games but not in synthetic agent simulations.

9.2 Future Work

We have evaluated the proposed methods on a single game: Settlers of Catan. An obvious path to explore is to evaluate the proposed models in similar complex games, that are highly structured and present similar characteristics (e.g. action types that dominate the action space or generate cyclic behaviour that would not be tolerated by human players in reality). We are confident that some of our techniques will prove beneficial,

given positive initial results with a similar typed MCTS algorithm on Monopoly (Sammul, 2018). Other board games and video games that present similar structure are: the Civilisation series, Diplomacy, Battlestar Galactica, etc.

Another alternative is to further evaluate the benefit of planning in the belief space given our negative results for Settlers of Catan. In other games, key information on who is winning the game is either observable or players can be certain of it. Furthermore, we believe planning in the belief space can be beneficial if players have more control on what to hide and what to show to opponents instead of chance events deciding the outcome as in Settlers of Catan. Therefore, we believe planning in the belief space can be beneficial in games such as Cheat or Cluedo.

Settlers of Catan is a very popular game and there is a large community of players. There are many versions of the game available for different platforms, e.g. Microsoft, Android etc. Unfortunately, most of these games have not released their source code and it would be time consuming to receive access to their database of collected game play. In addition, we would need to create an interface to connect our game model to theirs as well as to parse the data. Creating our experimental framework and a fast forward sampling model for MCTS was already very time-consuming (see a brief description of the software in Section 3.3). One would need to dedicate an even larger amount of resources to connect all the platforms. Nonetheless, it would be interesting to compare the methods developed in this thesis to other supervised learning approaches that require a large number of resources, e.g. Silver et al. (2016). The game's official website (`www.catan.com`) may be a good starting place for future work.

A related future path that may generate stronger agents is to augment the human corpus with synthetic or self-play data. As discussed in several parts of the thesis, we have not explored this option for several reasons. First of all, we aimed to directly compare synthetic data versus human data. We did not aim to create the strongest agent for Settlers of Catan by combining the two data sources. Secondly, bootstrapping learning methods doesn't only highlight if the advice provided by the data is good, but also if the extracted information allows the learning method to correct or adapt it in case the advice is sub-optimal. To this end, we have already evaluated the agents resulted from combining the extracted policies with a planning method. Finally, bootstrapping deep reinforcement learning is still a very expensive approach and the combination only partly speeds up learning. There are existing methods that could be attempted as future work, e.g. Nair et al. (2018); Hester et al. (2017).

As mentioned before, this thesis does not contain an exhaustive search of the best

set of parameters or features. The search we performed is sufficient to be highly confident of the performance of each proposed model. There is a possibility for minor improvements of our *final* agent (evaluated in Section 8.5) if one wishes to perform an even more fine-grained exploration of the parameter space. Instead, we aimed to evaluate if learning from a highly sparse and noisy dataset of human play is possible in such complex games. We achieved this goal and as a by-product we also created the current best open-source Settlers of Catan player which is purely based on learning methods. Over the next few subsection we go in more detail over what we believe are the most interesting and possibly fruitful options for future work.

9.2.1 Human Evaluation

We evaluated the methods implemented in this thesis versus multiple baseline agents, where one of them (Stac) is known to have comparable performance to human players (Keizer et al., 2017). Since our best agent can easily defeat this baseline (40.70% win rate versus 3 Stac when running only 10k iterations), we can extrapolate that our agent will also prove a challenging opponent for human players. In order to be certain, we would need to run a tournament between our agent, a baseline agent and human (expert) players as in Keizer et al. (2017). The current JSettlers framework and the experiment setting as described by Keizer et al. (2017) (e.g. games of 1 human player versus 3 agents, the payment done via Amazon vouchers, etc) can be followed. Our agent requires considerably more time to take decisions (approximately 3 seconds on average compared to the almost instant decision making of Stac), so an increased incentive may be required to pre-empt participants from losing interest. A difficulty that needs addressing is the additional computation resources our agent requires. The agents may need to run on a cluster of machines rather than a single one. To reduce this burden further, estimating the time required for games to finish and allocating time slots for each participant may be an option. The planning, as well as the engineering behind setting up and running this experiment made it impossible to perform it as part of this thesis.

There is also a major concern that requires addressing before this experiment can be run: our agent's trading behaviour. We have observed that it is very eager to make trade offers, possibly in board situations where it does not have access to the resources required to achieve its plan. Our current solution is to limit the number of offers the agent can make per turn, but the effect is that the performance of the agent is slightly

reduced. Even so, the number of offers is still large (45 per game) compared to that of the Stac agent (12 per game). One of the reason the Stac agent may be able to make fewer trades is its significantly more sophisticated trading algorithm. Its heuristics allow it to make offers to multiple opponents at once, while our agent is evaluating the benefits of trading with *each opponent* during planning. It then makes the offers in the real game by following through the list of legal trades in their descending order according to their value estimated during planning. An approach which aggregates the offers that are the same but have different recipients in order to make a single offer in the real game may improve the experience of human players. Further engineering to rank these aggregated offers and decide how to reply if multiple opponents accept the offer will be required. Finally, our agent cannot react or make partial offers, where either what resources are given, or requested are mentioned in the offer. Due to time restrictions, we were unable to create these extensions and analyse which would be the best performing one.

9.2.2 Hierarchical Approaches

Our main results, particularly integrating a prior distribution over action types to inform different phases of the MCTS algorithm, indicate the need for hierarchical and mixture methods to learn in highly complex games. The simple MLE approach described in this thesis learns action type preferences that describe an overview of how a standard player would act. However, these preferences are reactive, in the sense that these do not take into account what the player has done in the past. Conditioning on the history that describe what the player has done in the past is an interesting path to explore. This is particularly appropriate in learning to play games that require following a consistent plan, such as Settlers of Catan where well-known general strategies are advised (see description of ore-wheat or clay-wood strategies in Section 3.2). The main challenge in this case is the sparsity of the data that is further aggravated by conditioning on sequences. Methods to generate approximative representations, such as function approximations similar to the approach of Christiano et al. (2017), are required. A relevant approach is that of hierarchical reinforcement learning Dietterich (2000), where macro actions or options are defined to reduce the space the algorithm needs to explore. Automatic extraction of such options (over actions or action types) from a sparse corpus of human play could also be an interesting path to explore in future work.

We have implemented a version of Mixture of Experts model for offline learning of a policy which specialises models to specific tasks in the game. However, there is an alternative offline learning approach that is similar to the strategy of sampling action types followed by the action specifics done in MCTS rollouts. One could specialise models to learn action specifics, e.g. a model learns where is the best coordinate for roads while another learns to place settlements, and train a different model to select the best action type to execute next. The latter can be a simple MLE approach to learn a density function over the set of legal (or possible) action types, while the specialist models can be DNNs. But, it would also be interesting to compare this approach to end-to-end training of a standard Mixture of Experts: a gating function that provides the distribution over which expert is more likely to answer correctly (this is equivalent to the distribution over action types), and a set of experts that each provide distributions over portions of the action space that include only legal actions of the type each is assigned to learn.

9.2.3 Multi-agent Perspective

One of the most interesting aspects of Settlers of Catan is the *negotiation game* that is played in parallel with the game addressed in this thesis (defined as a finite POMDP). Here, we refer to negotiations as the very diverse chat moves a player can make: from the simple exchanges, such as introduce themselves to the other people, to highly complex persuasion moves required to ensure a good trade. This negotiation game is very complex since it has an unbounded action and state space. But, it also requires reasoning about opponent types or preferences. In fact, Settlers of Catan is a social game and you cannot risk making certain moves without taking into consideration the effects these may have on how the others will perceive you. For example, being a nice player may help you form coalitions that in turn will prove decisive to winning the game. Negotiation utterances are interesting from a natural language generation perspective because one may need to phrase these with a specific goal in mind (an example goal would be to get a clay resource from the red player). This goal is not a static objective as in most domain-based language generation tasks as it depends on the current game state, on the other player types etc. But, it must be connected to the main objective of winning the game, which, on its own, is too generic to help inform how these negotiation utterances should be phrased.

A Bayesian formulation should also help in learning to take decisions in the finite

POMDP we addressed in this thesis. Settlers of Catan is a multi-player game and, in reality, it is highly unlikely that one would play in a 1 versus 3 of the same type of agents setting as in our experiments. Such a controlled setting was required for empirically evaluating synthetic agents one versus another. A Bayesian formulation of such a complex environment presents many challenges. One of these is defining these player types and their behaviours. Automatic methods to hypothesise these behaviours based on the problem description or example play may be required. Reasoning over player types could also be combined with learning macro actions or extracting policies from a corpus of human data. However, including player types may not be successful in the low-resource setting such as ours.

9.2.4 Reinforcement Learning

One of the main reason we chose planning at decision time over other reinforcement learning methods, such as deep reinforcement learning or inverse reinforcement learning, is the sheer size of the problem space. MCTS bypasses this by estimating a locally optimal policy, in the sense that this policy is sensible for the current game state and board configuration only. Deep reinforcement learning methods are known for the amount of data and computational resources they require, as well as their need to reach reward rich areas of the state space in order to bootstrap. The large branching factor and massive depth factor of Settlers of Catan combined with a highly sparse reward function, would have caused these methods to be extremely slow or even unsuccessful. Inverse reinforcement learning methods would also struggle given these characteristics. In addition, inverse reinforcement learning would have to deal with the added issue of very ambiguous trajectories generated in our environment, especially due to the action space being dominated by trading actions and the very varied game scenarios (which may require apparent contradictory play). The hierarchical solution we presented (sampling action types followed by action description) has significantly reduced the depth of rollouts and increased the efficiency of running them (see Table 5.6 on page 76). If such a hierarchical policy is used, the algorithm is more likely to encounter reward rich areas and this will speed up convergence. It would be interesting to explore hierarchical versions of deep reinforcement learning or inverse reinforcement learning that could learn both the policies (over actions and over action types) or just one.

Another possibility to speed up convergence of reinforcement learning algorithms

is specialising policies to certain tasks of the game, and training them iteratively while fixing the other policies. For example, Cuayáhuitl et al. (2015) have used the heuristics based agent to handle all actions in the game except deciding what resources to exchange when trading. The authors trained a neural policy to select these resources whenever the rule-based agent decided to trade. Using the rule-based agent helped to reduce the learning problem to the situations that are more likely to be encountered in real games, as well as aided in learning a trading policy that is in line with the game play strategy of the rule-based agent. Cuayáhuitl et al. (2015) have focused only on learning what resources to select, so the action space is small ($\approx$70) compared to the full game (1882, see Appendix A for details). Nonetheless, an iterative divide and conquer approach might also prove successful in applying reinforcement learning to the full game of Settlers of Catan.

Appendix A

Settlers of Catan Action Space

Table A.1 contains the maximum number of options for each action type in the game Settlers of Catan given our minor modification of the trade action space. The set of options of building or placing a piece contains the legal locations on the board for the specific type, move robber with stealing contains the number of hexes times the number of opponents (plus the case where there is no adjacent piece), bank trades include the 3 options of 1 for 2 resources, 1 for 3 resources and 1 for 4 resources for each combination of resources (excluding exchanging the same resource type), trades between players include the 1 for 1 resource, 1 for 2 of the same type, 2 of the same type for 1 resource, 1 for 2 of different type and 2 of different type for 1 with each of the three opponents, discards include all possible unique combinations of resource types up to 10 resources which is the maximum limit implemented in the planning method.

Build settlement	54
Build city	54
Build road	72
Move robber	$19 * (3 + 1)$
Buy development card	1
Play knight	1
Play free road	1
Play monopoly	5
Play discovery	15
Roll dice	1
End turn	1
Bank trades	$5 * 4 * 3$
Player trades	$(20 + 60 * 2) * 3$
Discard	1001
Total	1882

Table A.1: Game actions

Appendix B

Features for the Non-parametric Method

This feature template was used to represent the states in the nearest neighbours experiment. The full vector length is 110 and is created from two sets of features: a set describing a general board description and a set describing each players' state starting from the current player and going round based on the order of play. The features describing the general game state are:

- State number taken from JSettlers;
- Number of players;
- Total number of roads built;
- Total number of settlements built;
- Total number of cities built;
- The position of the current player based on play order;

The set of features describing a player's state:

- Current victory points;
- If the player owns the largest army;
- If the player owns the longest road;
- The number of roads on the board;

- The number of settlements on the board;
- The number of cities on the board;
- The sum of numbers (these are on the hexes this player's settlements and cities are touching; in the case of a city the value is duplicated);
- Maximum production of each resource type computed as a sum of settlements and cities touching that resource (where a settlement = 1 and a city = 2);
- The port types this player has access to;
- A small set of heuristics:
  - Is the player's territory connected;
  - Does the player have an isolated settlement;
  - The current longest road;
  - The longest possible road;
  - Distance to opponent's pieces;
  - Distance to a port;
  - Distance to the next legal location for a settlement;

Appendix C

Monte Carlo Tree Search Parameter Tuning

This appendix includes the results from tuning the parameters of the MCTS algorithm in the fully-observable game. We focus our attention on the exploration parameter C, the effects of the trading limit, on the effects of the selection policy for the BEST_ACTION(n) function from Algorithm 1, and compare the sequential agent versus the parallel agent. We first show the effect of increasing the exploration parameter value. Table C.1 shows that increasing C has a negative effect on performance. Since the performance as a function of C is a monotonic function decreasing with the increase of C, we limited our exploration to the values included in the table.

0.5	1	2	3	4
36.20%	33.95%	30.10%	29.50%	29.00%

Table C.1: Win rates of the MCTS agent versus 3 Stac agents in the fully-observable domain while varying the value of the exploration parameter C. The MCTS agent runs 10k iterations and can make unlimited offers.

In Section 5.5, we observed that including an offer limit affects the performance of the agent. While tuning the exploration parameter C and the number of iterations, it became clear that this limit affects the algorithm more as the number of iterations is increased (see Table C.2). This further highlights the need for future work on developing a better approach to reduce the trading eagerness of the planning agents.

Selecting the action with the maximum value, i.e. $\max Q(s,a)$, as the next action to be executed after planning is controversial. This approach is considered risky since

Iterations	C				
	0.5	1	2	3	4
10k	31.20%	29.90%	27.20%	27.45%	26.55%
20k	40.85%	40.90%	40.8%	37.55%	37.65%
30k	46.75%	45.90%	44.40%	44.55%	43.50%
40k	47.35%	48.15%	46.69%	46.85%	46.83%
50k	50.85%	51.45%	49.60%	48.32%	50.76%

Table C.2: Win rate of the MCTS agent with max value policy versus 3 Stac agents as we vary the number of iterations and the exploration parameter C. Trade offers are limited to 3 per turn.

an action with few visits may have received some lucky wins before the planning ends. Choosing the most visited action is much less error-prone, therefore the most successful agents in Go (Silver et al., 2016, 2017) use the robust child policy, i.e. select the action with the most visits ($\max N(s,a)$). Table C.3 shows the performance of the MCTS agent with the robust child policy. Due to time constraints, the MCTS agent was limited to 3 offers per turn, so these results can be compared to those included in Table C.2 where the MCTS agent uses the max value policy. The results indicate that the performance of the two policies is comparable. In the end, we chose the max value policy following Szita et al. (2010), the authors of SmartSettlers.

Iterations	C				
	0.5	1	2	3	4
10k	31.50%	30.45%	27.25%	27.85%	27.00%
20k	40.80%	40.85%	40.00%	38.80%	38.95%
30k	45.15%	46.80%	44.30%	43.40%	44.15%
40k	47.90%	48.40%	46.35%	47.10%	46.55%
50k	49.05%	50.05%	51.60%	48.55%	47.05%

Table C.3: Win rate of the MCTS agent with robust child policy versus 3 Stac agents as we vary the number of iterations and the exploration parameter C. Trade offers are limited to 3 per turn.

Finally, we have compared the performance of the sequential agent versus the parallel agent while limiting the number of iterations to 10k. We used our strongest MCTS

agent, i.e. TMCTS, and we compared its performance when the number of threads is 1 (Sequential) versus 4 (Parallel). Table C.4 contains the results. As expected Sequential performs slightly better than Parallel since it doesn't lose any results due to concurrent modifications in the tree. Another reason for this improved performance is that threads may explore the same branch in the parallel version, while in the Sequential agent an update is always performed before starting a new iteration. So the parallel version duplicates some of the effort. Virtual loss punishes duplication, however it is not sufficient. Despite these results, the parallel agent provides almost 3 times the speed up compared to the sequential agent (see Table 5.1), so we use the parallel version in the thesis.

Modified	Baseline		
	Stac	Sequential	Parallel
Sequential	33.15%	–	26.35%
Parallel	31.20%	23.53%	–

Table C.4: Win rates of the two TMCTS agents: Sequential and Parallel. Each result is measured over 2000 games.

Appendix D

Factored Belief Model

In Settlers of Catan, the players' resources and development cards are the only parts of the game that opponents can be uncertain of. Fortunately, it is obvious from the game rules that these two aspects are independent and can be tracked by separate models, referred from now on as the *resource model* and the *development model*. Further, since this is a 4-player game, tracking a belief over a complete description of the state would result in an explosion of possible states given the large number of combinations between each possible resource hand and development hand for each player. But, we only require the information for the current player to reason over the next legal moves. Therefore we track each player's resource hands individually and we make the assumption that these are independent. Given that players steal resources from each other, this is clearly a relaxed implementation. This means that sampling complete state descriptions from this model will create a small number of additional impossible states, but it will always include the true state and the whole set of possible states. Unfortunately, this relaxation is the best we can do without developing a very complex model. It may be very challenging if not impossible to create a better model; none of the opponents observe what a player discards and there could be numerous stealing actions where players repeatedly steal from each other. Given these minor assumptions, the belief is represented using the following product:

$$b_i^t = \prod_{j \neq i} P(X_c^j, \ldots, X_{wd}^j | H) P(X_k^j, \ldots, X_v^j | H) \tag{D.1}$$

In this equation we show how we represent the belief b_i^t of player i at time t, by enumerating over the opponents j to compute the joint distribution over their resources: clay (c), ore(o), sheep(s), wheat (wh) and wood (wd), and the joint distribution over

their development cards: knights (k), monopoly (m), discovery (d), free road (r) and victory points (v). For completeness we include the dependence on the history H which is usually a list of actions and observations. However, this is a Belief MDP, so we replace H with a tuple $(b_i^{t-1}, a_{t-1}, o_{t-1})$. We do not want to delve on implementation details, so we only mention that the joint probability distribution over resources is computed by keeping track of complete set of resources that describe all possible hands mapped to their probability mass, while the one over the development card is computed via simple combinatorics, and by keeping track of totals for played and unplayed cards.

One minor point we want to make is that the belief distribution is highly sparse, so there is no point in keeping track of all possible states (i.e., possible resource hands and development hands in our case). One reason is that the game is not ergodic (i.e. only a subset of possible states can be encountered in a given game due to the large variety of initial board settings). Another cause is that only a subset of the states from those reachable in the current game are possible at the current time step. Even though we show that we enumerate over all states $s \in S$, in fact we only keep track of a small subset $s \in S^-$ such that $b(s) > 0$.

The belief transition function $b' = \tau(b, a, o)$ is straightforward for the development card model: we just need to update the totals following the actions of buying or playing a development card. In the resource model case, we built an update method that is inspired from Situation Calculus (Russell and Norvig, 2009). A possible state (or situation) is described with fluents for the quantities of each resource type that form the players' hands. For each action in the game that requires an update to the resource model, we define a possibility axiom and an effect axiom. For example, building a road possibility axiom requires the description of the possible world (i.e. resource set) to include at least one clay and one wood. The effect axiom will modify the resource set by decrementing one clay and one wood. We solve the representational frame problem via a mixture of successor-state axioms for the fluents describing a player's hand and explicit frame axioms that state which players' hands are modified. The main difference to Situation Calculus is that we need to keep track of multiple situations at the same time since we don't know the true state prior to the action being modelled. We handle this by attaching probabilities to each situation. We use this model to track the belief of our agent as actions are executed. So an update is performed when an action is observed in the real game or in the game model used by MCTS. After an update, sets of situations may be inconsistent with the relevant possibility axiom. Therefore, we also normalise the ones resulting from the effect axiom such that their probability

mass sums to 1.

Appendix E

Settlers of Catan Partially-observable Game Model

The following list contains the set of action effects and observations in the game model according to the game rules:

1. **Build action**: this is a fully-observable action and every player observes the full effect. Depending on the type of piece built, the possibility axiom requires the initial resource set to contain the required resources, and the effect is to subtract the requirements from the initial set.

2. **Trade action**: all players observe what is exchanged. The possibility and effect axiom are the same as for the build action. The update is performed for two players in a normal trade, or for a single player in the trade with bank or port case.

3. **End turn**: observable action and no update is required to the belief model;

4. **Roll dice**: observable action followed by a chance event whose consequences (in terms of which players get which resources) are observable. The effect is to add the relevant resources to each possible situation.

5. **Move robber**: observable action and no update is performed to the belief. This is followed by a steal action (as described in more detail in point 10);

6. **Play development card**: observable action and its effect is to update the totals in the development model. Depending on the development card, this action can be followed by other actions such as move robber, build two free roads, receive

two resources of a chosen type or take all resources of a chosen type from all the other players (monopoly). All these actions are fully-observable and the effects are straightforward, except for the monopoly action.

7. **Monopoly action**: observable action and effect, but the axioms are not straightforward to implement. In the partially-observable setting, the current player doesn't know the true state so it may not know the true totals for specific resource types for each player. The game rules dictate that all players must give away *all* resources of the chosen type to the player that played the monopoly card. In the planning phase, there is no server that can enforce this so we must create a chance event where the totals are sampled for each victim. The effect axiom updates the fluents by setting the quantity of the chosen type to 0.

8. **Buy development card**: partially observable action, where each opponent observes that the current player bought a development card but only the current player observes its type. The effect is to increase the total of unplayed development cards for the current player in the probabilistic development model.

9. **Discard**: partially observable action, where the opponents observe the total number of cards discarded but not their resource types. If the victim is the player whose belief we are tracking, then the effect axiom is similar to that of the build actions. When an opponent discards, the effect axiom computes all possible combinations that could have been discarded and generates new situations for each.

10. **Stealing**: this action follows every move robber action and the participants observe what type of resource was stolen while the other two players only observe that a card was stolen but its type remains hidden to them. Therefore, the update is done depending on the perspective of the agent whose belief we are modelling: is it a participant or just an observer? If it is a participant, then the effects are observed following a chance node. If it is an observer, then the effect axiom is the same as that of the discard action, but now we must update the situations for both the victim and the perpetrator.

Staying in the belief space turns the effects of the discard action and stealing action into deterministic modifications to the belief. Unfortunately, we had to add one new chance node for the monopoly action (see above for motivation). This is due to the game rules which clearly state that the exact quantities are revealed when this action is

performed. The game rules also dictate that the observation model is deterministic i.e. there is a single observation that has a mass equal to 1: $P(O = o|s,a) = 1$. As a result, the observations received after monopoly actions have a strong revealing effect (i.e. sets $b(s) = 0$ for a large number of $s \in S$) which make it pointless to attempt building a belief transition function for this action.

Appendix F

Abstract Belief Representation

The belief is represented in the tree as a vector of features that represent the belief of what each player holds in their hand. Therefore this is constructed by concatenating the following information for each player:

- minimum number of resources for each type of resource;
- maximum number of resources for each type of resource
- total number of unplayed and unknown development cards;
- how many of the above are definitely not victory point cards;
- total number of played or revealed development cards for each type;

followed by a set of features describing what is remaining in play overall:

- total number of unplayed development cards (including what is still in the deck);
- total number of unplayed development cards of each type;

The vector size contains 82 features in total.

Appendix G

Neural Network Input Features

The set of features used for describing the state is made of a set of features describing the current general game state:

- current turn number;
- dice result;
- number of settlements;
- number of roads;
- number of cities;
- has the current player played a development card;
- the board position of the current player;
- the resource type blocked by the robber;
- the number on the hex blocked by the robber;
- number of players affected by the robber;
- number of pieces affected by the robber;
- are there any development cards left in the deck;

followed by a set of player specific features for the current player:

- score;
- longest road label;

- largest army label;
- number of road pieces available to build;
- number of settlement pieces available to build;
- number of city pieces available to build;
- the resources the player has in its hand;
- access to port types;
- access to resource types, where a city counts as double access;
- production of each resource type by summing the numbers on the hexes this player pieces are touching while taking into account the robber effect;
- is this player affected by the robber;
- number of played knights;
- length of longest road;
- number of development cards (new, old and played);
- what type of actions are legal given the resources the player has in his hand;
- does this player have over 7 resources;
- is this player blocked (if it cannot build any new pieces on the board);
- expansion possibility represented as the production and access to ports of the closest available empty position;

continuing with the features for the opponents, iterating over them in the order of play. Each set of features for each opponent contains only the observable features from the above list. Therefore, the exact description of resources in hand and development cards in hand are replaced with their total which is always observable. If the game had less than 4 players, the feature vector is padded with 0s such that it is equal to the ones extracted from 4 player games. The length of the vector is 157.

The set of features describing the actions is made of only the features from the above list that any action could modify. For example the features regarding resources is contained in this list, while the features describing the opponents' total number of

development card is not included since a player cannot steal development cards from opponents. The set of features are computed via vector difference between the resulting state s' and the state s from which the action was executed. In the case of the stealing action, the features do not contain the stolen resource and only the change to the totals of the players involved in the action. The length of the feature vector describing the action is 73.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press.

Abramson, B. (1990). Expected-outcome: A general model of static evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(2):182–193.

Afantenos, S., Asher, N., Benamara, F., Cadilhac, A., Degremont, C., Denis, P., Guhe, M., Keizer, S., Lascarides, A., Oliver Lemon, P. M., Paul, S., Rieser, V., and Vieu, L. (2012). Developing a corpus of strategic conversation in the settlers of catan. In *Proceedings of the 1st Workshop on Games and NLP*, Kanazawa, Japan.

Albrecht, S. V. (2015). *Utilising policy types for effective ad hoc coordination in multiagent systems*. PhD thesis, University of Edinburgh, UK.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483.

Asmuth, J. and Littman, M. S. (2011). Approaching bayes-optimalilty using monte-carlo tree search. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, page 1926.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235–256.

Babeş-Vroman, M., Marivate, V., Subramanian, K., and Littman, M. (2011). Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 897–904, USA. Omnipress.

Bai, A., Srivastava, S., and Russell, S. (2016). Markovian state and action abstractions for mdps via hierarchical mcts. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 3029–3037. AAAI Press.

Balla, R.-K. and Fern, A. (2009). Uct for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 40–45, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bengio, Y., Bastien, F., Bergeron, A., Boulanger-lewandowski, N., Breuel, T. M., Chherawala, Y., Cisse, M., Ct, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Lebeuf, S. P., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In Gordon, G. J. and Dunson, D. B., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 164–172. Journal of Machine Learning Research - Workshop and Conference Proceedings.

Beygelzimer, A., Langford, J., Li, L., Reyzin, L., and Schapire, R. E. (2011). Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 19–26, Fort Lauderdale, USA.

Bigot, D., Zanuttini, B., Fargier, H., and Mengin, J. (2013). Probabilistic conditional preference networks. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA.

Bitan, M. and Kraus, S. (2017). Combining prediction of human decisions with ISMCTS in imperfect information games. *CoRR*, abs/1709.09451.

Bjarnason, R., Fern, A., and Tadepalli, P. (2009). Lower bounding klondike solitaire with monte-carlo planning. In Gerevini, A., Howe, A. E., Cesta, A., and Refanidis, I., editors, *ICAPS*. AAAI.

Boularias, A., Kober, J., and Peters, J. (2011). Relative entropy inverse reinforcement learning. In Gordon, G., Dunson, D., and Dudk, M., editors, *Proceedings of the*

Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 of *Proceedings of Machine Learning Research*, pages 182–189, Fort Lauderdale, FL, USA. PMLR.

Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., and Poole, D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191.

Branavan, S., Silver, D., and Barzilay, R. (2012). Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704.

Branca, L. and Johansson, S. J. (2007). Using multi-agent system technologies in settlers of catan bots. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence,*.

Brown, G. W. (1951). Iterative solution of games by fictitious play. In Koopmans, T. C., editor, *Activity Analysis of Production and Allocation*. Wiley, New York.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43.

Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122.

Caron, S., Kveton, B., Lelarge, M., and Bhagat, S. (2012). Leveraging side observations in stochastic bandits. In de Freitas, N. and Murphy, K. P., editors, *UAI*, pages 142–151. AUAI Press.

Caruana, R. (1997). Multitask learning. In *Machine Learning*, pages 41–75.

Castro, P. S. and Precup, D. (2010). Smarter sampling in model-based bayesian reinforcement learning. In *Proceedings of the 2010th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECMLPKDD'10, pages 200–214, Berlin, Heidelberg. Springer-Verlag.

Chaganty, A. T., Gaur, P., and Ravindran, B. (2012). Learning in a small world. In *AAMAS*, pages 391–397. IFAAMAS.

Chaslot, G., Winands, M. H. M., and van den Herik, H. J. (2008a). Parallel monte-carlo tree search. In van den Herik, H. J., Xu, X., Ma, Z., and Winands, M. H. M., editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 60–71. Springer.

Chaslot, G. M.-B., Hoock, J.-B., Perez, J., Rimmel, A., Teytaud, O., and Winands, M. H. (2009). Meta monte-carlo tree search for automatic opening book generation. In *Proceedings of the IJCAI'09 Workshop on General Intelligence in Game Playing Agents*, pages 7—12.

Chaslot, G. M. J., Winands, M. H., HERIK, H. J. V. D., Uiterwijk, J. W., and Bouzy, B. (2008b). Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357.

Childs, B. E., Brodeur, J. H., and Kocsis, L. (2008). Transpositions and move groups in monte carlo tree search. In Hingston, P. and Barone, L., editors, *CIG*, pages 389–395. IEEE.

Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, NIPS'17, pages 4302–4310, Long Beach, CA, USA.

Clark, C. and Storkey, A. (2015). Training deep convolutional neural networks to play go. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, volume 37 of *ICML'15*, pages 1766–1774, Lille, France. JMLR.org.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *In: Proceedings Computers and Games 2006*. Springer-Verlag.

Coulom, R. (2011). Clop: Confident local optimization for noisy black-box parameter tuning. In *Advances in Computer Games - 13th International Conference*.

Cowling, P. I., Powley, E. J., and Whitehouse, D. (2012a). Information set monte carlo tree search. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(2):120–143.

Cowling, P. I., Ward, C. D., and Powley, E. J. (2012b). Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(4):241–257.

Cuayáhuitl, H., Keizer, S., and Lemon, O. (2015). Strategic dialogue management via deep reinforcement learning. In *NIPS Workshop on Deep Reinforcement Learning*, Montreal, Canada.

Daumé, Iii, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Mach. Learn.*, 75(3):297–325.

Deeplearning4j, D. T. (2016). Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0. http://deeplearning4j.org.

Deng, J., Russakovsky, O., Krause, J., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2014). Scalable Multi-Label Annotation. In *CHI*.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *in Proc. Int. Conf. Acoust., Speech, Signal Process*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13:227303.

Dobre, M. and Lascarides, A. (2015). Online learning and mining human play in complex games. In *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, Tainan, Taiwan.

Dobre, M. and Lascarides, A. (2017). Combining a mixture of experts with transfer learning in complex games. In *Proceedings of the AAAI Spring Symposium: Learning from Observation of Humans*, Stanford, USA.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley, New York, 2 edition.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). Knowledge discovery and data mining : Towards a unifying framework. In *Proceedings of the 2nd international conference on Knowledge Discovery and Data mining (KDD'96)*, pages 82–88. AAAI Press.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fox, R., Krishnan, S., Stoica, I., and Goldberg, K. (2017). Multi-level discovery of deep options. *CoRR*, abs/1703.08294.

Frank, I. and Basin, D. (1998). Search in games with incomplete information: a case study using bridge card play. *Artificial Intelligence*, 100(12):87 – 123.

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. (2012). Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89(1):123–156.

Gal, Y., Grosz, B., Kraus, S., Pfeffer, A., and Shieber, S. (2005). Colored trails: a formalism for investigating decision-making in strategic environments. In *In IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*.

Gal, Y., Kasturirangan, R., Pfeffer, A., and Richards, W. (2008). How tacit knowledge guides action. In *AAAI Fall Symposium: Naturally-Inspired Artificial Intelligence*, volume FS-08-06 of *AAAI Technical Report*, pages 61–64. AAAI.

Gaskett, C., Wettergreen, D., and Zelinsky, A. (1999). Q-learning in continuous state and action spaces. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence: Advanced Topics in Artificial Intelligence*, AI '99, pages 417–428, London, UK, UK. Springer-Verlag.

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 273–280, New York, NY, USA. ACM.

Gelly, S. and Silver, D. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artif. Intell.*, 175(11):1856–1875.

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. (2006). Modification of uct with patterns in monte-carlo go. Technical report, INRIA, Paris.

Gigerenzer, G. and Selten, R. (2001). *Bounded Rationality: The adaptive toolbox*. Cambridge: The MIT Press.

Ginsberg, M. L. (2001). Gib: Imperfect information in a computationally challenging game. *J. Artif. Intell. Res. (JAIR)*, 14:303–358.

Gläscher, J., Daw, N., Dayan, P., and O'Doherty, J. P. (2010). States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*.

Gmytrasiewicz, P. J. and Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.*, 24:49–79.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Graf, T. and Platzner, M. (2016). *Using Deep Convolutional Neural Networks in Monte Carlo Tree Search*, pages 11–21. Springer International Publishing, Leiden, The Netherlands.

Guez, A., Silver, D., and Dayan, P. (2013). Scalable and efficient bayes-adaptive reinforcement learning based on monte-carlo tree search. *J. Artif. Int. Res.*, 48(1):841–883.

Guhe, M. and Lascarides, A. (2014a). Game strategies in *the settlers of catan*. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, Dortmund, Germany.

Guhe, M. and Lascarides, A. (2014b). Persuasion in complex games. In *Proceedings of the 18th Workshop on the Semantics and Pragmatics of Dialogue (SEMDIAL)*, pages 62–70, Edinburgh.

Guhe, M., Lascarides, A., O'Connor, K., and Rieser, V. (2013). Effects of belief and memory on strategic negotiation. In *Proceedings of the 17th Workshop on the Semantics and Pragmatics of Dialogue (DialDam)*, Amsterdam.

Harsanyi, J. (1967). Games with incomplete information played by 'bayesian' players. part i. the basic model. *Management Science*, 14(3):159182.

Harsanyi, J. (1968). Games with incomplete information played by 'bayesian' players. part ii. bayesian equilibrium points. *Management Science*, 14(5):320334.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. (2015). Deep reinforcement learning with a natural language action space. *CoRR*.

He, R., Brunskill, E., and Roy, N. (2010). Puma: Planning under uncertainty with macro-actions. In Fox, M. and Poole, D., editors, *AAAI*. AAAI Press.

Hegarty, M. (2004). Mechanical reasoning by mental simulation. *Trends in Cognitive Sciences*, 8(6):280–285.

Heinrich, J. and Silver, D. (2015). Smooth uct search in computer poker. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 554–560. AAAI Press.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. (2017). Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732.

Hindriks, K. and Tykhonov, D. (2008). Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, pages 331–338, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Hinton, G. E. (2007). Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434.

Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Irani, A. J. (2015). *Utilizing Negative Policy Information To Accelerate Reinforcement Learning*. PhD thesis, Georgia Institute of Technology.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87.

James, S., Konidaris, G., and Rosman, B. (2017). An analysis of monte carlo tree search. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3576–3582, San Francisco, California, USA.

Jouandeau, N. and Cazenave, T. (2014a). Monte-carlo tree reductions for stochastic games. In *Technologies and Applications of Artificial Intelligence, 19th International Conference, TAAI 2014, Taipei, Taiwan, November 21-23, 2014. Proceedings*, pages 228–238.

Jouandeau, N. and Cazenave, T. (2014b). Small and large MCTS playouts applied to chinese dark chess stochastic game. In *Computer Games - Third Workshop on Computer Games, CGW 2014, Held in Conjunction with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18, 2014, Revised Selected Papers*, pages 78–89.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134.

Kaelbling, L. P. and Lozano-Prez, T. (2013). Integrated task and motion planning in belief space. *I. J. Robotic Res.*, 32(9-10):1194–1227.

Kearns, M., Mansour, Y., and Ng, A. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3):193–208.

Keizer, S., Guhe, M., Cuayahuitl, H., Efstathiou, I., Engelbrecht, K.-P., Dobre, M., Lascarides, A., and Lemon, O. (2017). Evaluating persuasion strategies and deep reinforcement learning methods for negotiation dialogue agents. In *EACL*, Valencia, Spain.

Kloetzer, J., Iida, H., and Bouzy, B. (2007). The monte-carlo approach in amazons. In *Proc. Comput. Games Workshop,Amsterdam, Netherlands*, pages 113–124.

Knox, W. B. and Stone, P. (2008). TAMER: Training an Agent Manually via Evaluative Reinforcement. In *IEEE 7th International Conference on Development and Learning*.

Kocsis, L. and Szepesvari, C. (2006). Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems*.

Lample, G. and Chaplot, D. S. (2016). Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

Lelis, L. H., Zilles, S., and Holte, R. C. (2013). Stratified tree search: A novel suboptimal heuristic search algorithm. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 555–562, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 297–306, New York, NY, USA. ACM.

Maddison, C. J., Huang, A., Sutskever, I., and Silver, D. (2015). Move evaluation in go using deep convolutional neural networks. In *International Conference on Learning Representations*.

Masoudnia, S. and Ebrahimpour, R. (2014). Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2):275–293.

Menashe, J. and Stone, P. (2015). Monte carlo hierarchical model learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multia-*

gent Systems, AAMAS '15, pages 771–779, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I. J., Lavoie, E., Muller, X., Desjardins, G., Warde-Farley, D., Vincent, P., Courville, A. C., and Bergstra, J. (2012). Unsupervised and transfer learning challenge: a deep learning approach. In Guyon, I., Dror, G., Lemaire, V., Taylor, G. W., and Silver, D. L., editors, *ICML Unsupervised and Transfer Learning*, volume 27 of *JMLR Proceedings*, pages 97–110. JMLR.org.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Moraes, R. O., Mariño, J. R. H., and Lelis, L. H. S. (2018). Nested-greedy search for adversarial real-time games. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation, Brisbane, Australia*, pages 6292–6299.

Nash, J. F. (1950). Equilibrium points in n-Person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49.

Neto, H. and Julian, R. (2007). Ls-draughts - a draughts learning system based on genetic algorithms, neural network and temporal differences. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 2523–2529.

Neto, H. C., da Silva Julia, R. M., Caexeta, G. S., and Barcelos, A. R. A. (2014). Ls-visiondraughts: improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method. *Applied Intelligence*, 41(2):525–550.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the*

Sixteenth International Conference on Machine Learning, ICML '99, pages 278–287, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann.

Paquet, S., Tobin, L., and Chaib-draa, B. (2005). An online pomdp algorithm for complex multiagent environments. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 970–977, New York, NY, USA. ACM.

Pednault, E. P. D. (1987). Formulating Multi-Agent Dynamic-World Problems in the Classical Planning Framework. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82, San Mateo, CA. Morgan Kaufmann Publishers.

Pfeiffer, M. (2003). Machine learning applications in computer games. Master's thesis, Institute for Theoretical Computer Science, Graz University of Technology.

Quinn, A. J. and Bederson, B. B. (2011). Human computation: A survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA. ACM.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 463–471, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 729–736, New York, NY, USA. ACM.

Riedmiller, M. (2005). Neural fitted q iteration first experiences with a data efficient neural reinforcement learning method. In *In 16th European Conference on Machine Learning*, pages 317–328. Springer.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Roelofs, G. (2012). Monte carlo tree search in a modern board game framework. research paper available at umimaas.nl.

Rosch, E. (1978). Principles of categorization. In Rosch, E. and Lloyd, B. B., editors, *Cognition and Categorization*, pages 27–48. Erlbaum, Hillsdale, NJ.

Rosenfeld, A., Taylor, M. E., and Kraus, S. (2017). Speeding up tabular reinforcement learning using state-action similarities. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1722–1724, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Rosman, B. and Ramamoorthy, S. (2012). What good are actions? accelerating learning using learned action priors. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In Gordon, G., Dunson, D., and Dudk, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA. PMLR.

Russell, S. J. and Norvig, P. (2009). *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall.

Sammul, S. (2018). Learning to play monopoly with monte carlo tree search. Computer science bachelor thesis, School of Informatics, University of Edinburgh.

Savage, L. J. (1954). *The Foundations of Statistics*. Wiley Publications in Statistics.

Sawyer, B. and Rejeski, D. (2002). Serious games: Improving public policy through game-based learning and simulation.

Schaeffer, J. and van den Herik, H. (2002). Games, computers, and artificial intelligence. *Artificial Intelligence*, 134(12):1 – 7.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Sheppard, B. (2002). World-championship-caliber scrabble. *Artificial Intelligence*, 134(12):241 – 275.

Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354–.

Silver, D., Sutton, R. S., and 0003, M. M. (2007). Reinforcement learning of local shape in the game of go. In Veloso, M. M., editor, *IJCAI*, pages 1053–1058.

Silver, D., Sutton, R. S., and Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 968–975, New York, NY, USA. ACM.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large pomdps. In *In Advances in Neural Information Processing Systems 23*, pages 2164–2172.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). Despot: Online pomdp planning with regularization. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *NIPS*, pages 1772–1780.

Spelke, E. S., Breinlinger, K., Macomber, J., and Jacobson, K. (1992). Origins of knowledge. *Psychological review*, 99(4):605.

STAC (2018). Stac: Strategic conversation. https://www.irit.fr/STAC/.

Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In Koenig, S. and Holte, R. C., editors, *Abstraction, Reformulation, and Approximation*, pages 212–223, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sturtevant, N. R. (2008). An analysis of uct in multi-player games. *ICGA Journal*, 31(4):195–208.

Suay, H. B., Brys, T., Taylor, M. E., and Chernova, S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pages 429–437, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Subramanian, K., Isbell, C. L., and Thomaz, A. (2011). Learning Options through Human Interaction. In *Workshop on Agents Learning Interactively from Human Teachers at IJCAI*.

Subramanian, K., Scholz, J., Isbell, C. L., and Thomaz, A. L. (2016). Efficient exploration in monte carlo tree search using human action abstractions. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16.

Sutskever, I. and Nair, V. (2008). Mimicking go experts with convolutional neural networks. In *Artificial Neural Networks - ICANN 2008, 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part II*, pages 101–110.

Sutton, R. and Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge University Press, 1 edition.

Sutton, R. and Barto, A. (2018). *Reinforcement learning: An introduction.* Cambridge University Press, 2 edition.

Szepesvari, C. (2010). *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Szita, I., Chaslot, G., and Spronck, P. (2010). Monte-carlo tree search in settlers of catan. In van den Herik, H. and Spronck, P., editors, *Advances in Computer Games*, pages 21–32. Springer.

Tastan, B. and Sukthankar, G. R. (2011). Learning policies for first person shooter games using inverse reinforcement learning. In Bulitko, V. and Riedl, M. O., editors, *AIIDE*. The AAAI Press.

Tesauro, G. (1990). Neurogammon : a neural-network backgammon program. Technical Report RC 15619, IBM US Research Centers (Yorktown,San Jose,Almaden, US).

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68.

Tesauro, G., Rajan, V. T., and Segal, R. (2012). Bayesian inference in monte-carlo tree search. *CoRR*, abs/1203.3519.

Thomas, R. (2004). *Real-time Decision Making for Adversarial Environments Using a Plan-based Heuristic*. PhD thesis, Department of Computer Science, Northwestern University.

Thrun, S. (1995). Learning to play the game of chess. In *Advances in Neural Information Processing Systems 7*, pages 1069–1076. The MIT Press.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., and Kavukcuoglu, K. (2016). Strategic attentive writer for learning macro-actions. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3486–3494.

Vien, N. A. and Toussaint, M. (2015). Hierarchical monte-carlo planning. In *Proc. of The Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 15)*.

von Neumann, J. and Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton.

Wang, C., Member, S., Kulkarni, S. R., and Poor, H. V. (2005). Bandit problems with side observations. *IEEE Transactions on Automatic Control*, 50:338–355.

Wang, J., Zhu, T., Li, H., Hsueh, C. H., and Wu, I. C. (2015). Belief-state monte-carlo tree search for phantom games. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 267–274.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA. PMLR.

Whitehouse, D., Cowling, P. I., Powley, E. J., and Rollason, J. (2013). Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game. In Sukthankar, G. and Horswill, I., editors, *AIIDE*. AAAI.

Williams, J. D. (2005). Factored partially observable markov decision processes for dialogue management. In *In 4th Workshop on Knowledge and Reasoning in Practical Dialog Systems*, pages 76–82.

Xiao, C. and Müller, M. (2017). Integrating factorization ranked features in mcts: An experimental study. In *Computer Games*, pages 34–43. Springer International Publishing.

Xie, F., Müller, M., Holte, R., and Imai, T. (2014). Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2395–2402.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc.

Young, H. P. (2004). *Strategic Learning and Its Limits*. Oxford University Press.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press.