Artificial Intelligence Techniques Towards Adaptive Digital Games

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Marco Tamassia

Bachelor of Computer Science
University of Verona, Italy

Master of Computer Science
University of Verona, Italy

School of Science
College of Science, Engineering and Health
RMIT University

September 2017

I would like to dedicate this thesis to my loving partner Misuzu

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Marco Tamassia
September 2017

Acknowledgements

I would like to thank my supervisors, Fabio Zambetta, Xiaodong Li and Floyd Mueller, who always provided invaluable guidance and feedback during my doctoral training. Although not officially a supervisor, I would like to thank William Raffe for helping as if he were one.

I would also like to thank all the members of the Evolutionary Computation and Machine Learning (ECML) group at RMIT University for all the feedback and useful discussions, and for exposing me to a wide variety of research topics. In particular, I would like to thank Simon, who has been a great collaborator, friend and housemate.

I would like to thank Ruck Thawonmas and all the members of the Intelligent Computer Entertainment (ICE) lab. at Ritsumeikan University for welcoming, supporting and helping me and my research.

Finally, I would like to thank my family and friends for always supporting me. To my partner Misuzu, who showed unconditioned love and warmed every day with her smile. To my parents Adriana and Gianni for always wanting the best for me and my brothers. To Ali, Jessie and all the fellow Ph.D. students for all the coffee time; to my flatmates for the fun they added to my day-to-day life far from home; and to Lannari for making me feel there even being on the other side of the planet.

Abstract

Digital games rely on suspension of disbelief and challenge to immerse players in the game experience. Artificial Intelligence (AI) has a significant role in this task, both to provide adequate challenges to the player and to generate believable behaviours. An emerging area of application for digital games is augmented reality. Theme parks are expressing interest to augment the user experience via digital games. Bringing together cyber- and physical- aspects in theme parks will provide new avenues of entertainment to customers.

This thesis contributes to the field of AI in games, in particular by proposing techniques aimed at improving players' experience. The technical contributions are in the field of learning from demonstration, abstraction in learning and dynamic difficulty adjustment. In particular, we propose a novel approach to learn options for the Options framework from demonstrations; a novel approach to handle progressively refined state abstractions for the Reinforcement Learning framework; a novel approach to dynamic difficulty adjustment based on state-action values, which in our experiments we compute via Monte Carlo Tree Search. All proposed techniques are tested in video games.

The final contribution is an analysis of real-world data, collected in a theme park queuing area where we deployed an augmented reality mobile game; the data we collected suggests that digital games in such environments can benefit from AI techniques, which can improve time perception in players. Time perception, in fact, is altered when players enter the state of "flow", which can only happen if suspension of disbelief is maintained and if the level of challenge is adequate. The conclusion suggests this is a promising direction for investigation in future work.

Contents

List of Figures **xv**

List of Tables **xix**

1 Introduction **1**
1.1 Research Goals . . . 3
1.1.1 Research questions . . . 4
1.1.2 Contributions . . . 5
1.1.3 Publications . . . 6
1.2 Thesis structure . . . 7

2 Background and Related Work **9**
2.1 Reinforcement Learning . . . 10
2.1.1 Markov Decision Processes . . . 11
2.1.2 Value Functions . . . 12
2.1.3 Optimal-Value Functions . . . 13
2.1.4 Model-free Learning and Control . . . 14
2.1.5 Exploration-Exploitation . . . 15
2.2 Abstraction . . . 15
2.2.1 Temporal abstraction . . . 16
2.2.2 State abstraction . . . 18
2.3 Monte Carlo Tree Search . . . 19
2.3.1 Combinatorial games . . . 20
2.3.2 Tree search techniques . . . 21
2.3.3 The algorithm . . . 22
2.3.4 UCT . . . 25
2.3.5 Dynamic Difficulty Adjustment . . . 26
2.4 Summary . . . 27

3 Learning Options from Demonstrations **29**
3.1 Options construction . . . 29
3.2 Identifying Useful Sub-goals . . . 30
3.3 Reduce the Number of Sub-goals . . . 32
3.3.1 Graph-based Clustering . . . 32
3.3.2 Features-abstraction Aggregation . . . 33
3.4 Experiments . . . 34
3.4.1 Grid-world experimental setup . . . 34
3.4.2 Grid-world experiments results . . . 37
3.4.3 Pac-Man experimental setup . . . 43
3.5 Pac-Man experiments results . . . 49
3.6 Summary . . . 53

4 Dynamic Choice of State Abstraction **57**
4.1 Dynamic Abstraction Choice . . . 58
4.2 Experiments . . . 63
4.3 Results . . . 66
4.4 Summary . . . 71

5 Monte Carlo Tree Search for Dynamic Difficulty Adjustment **77**
5.1 Challenge Sensitive Action Selection . . . 78
5.2 Targeting outcomes . . . 79
5.2.1 Reactive Outcome Sensitive Action Selection . . . 80
5.2.2 Proactive Outcome Sensitive Action Selection . . . 81
5.2.3 *True ROSAS* and *True POSAS* . . . 82
5.2.4 Steeper variations . . . 84
5.2.5 Adaptive variations . . . 85
5.3 Experimental setup . . . 88
5.3.1 Implementation details . . . 90
5.4 Results and Discussion . . . 91
5.4.1 Discussion . . . 94
5.5 Summary . . . 102

6 Opportunities for AI Techniques in Theme Parks **103**
6.1 Attention and Flow . . . 104
6.2 Game design . . . 105
6.3 Experimental setup . . . 107

6.3.1 Study Site and Demographics of Participants 107
6.3.2 Experimental Design 108
6.3.3 Data Collection and Analysis 109
6.4 Results and Discussion 110
6.4.1 Statistical Significance and Effect Size 110
6.4.2 Correlations 110
6.4.3 Linear Regression 113
6.4.4 Survey analysis 113
6.5 Limitations and Future Work 115
6.6 Summary 116

7 Conclusions **119**
7.1 Future Works 121

Bibliography **125**

List of Figures

2.1 A representation of the Reinforcement Learning paradigm (image from [92]). 11
2.2 A representation of an asymmetric tree. 23
2.3 A representation of the Monte Carlo Tree Search algorithm (image from [14]). 24

3.1 The grid-world used in the experiments. The yellow squares represent the destinations used in the different experiments: the one in the upper corner is labeled as [0], the one in the hallway is labeled [45] and the other one is labeled [47]. These labels will be used to present the results of the experiments. The black squares represent unreachable states in the wall setting and low rewards states in the ponds setting. 35
3.2 Results of experiments in the "pond" settings with destination as the top-left corner (state [0]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options. 38
3.3 Results of experiments in the "wall" settings with destination as the top-left corner (state [0]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options. 39
3.4 Results of experiments in the "pond" settings with destination as the top hallway (state [45]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options. 40
3.5 Results of experiments in the "wall" settings with destination as the top hallway (state [45]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options. 41

3.6 This figure shows the results obtained in the "pond" setting with destination [47]. For details, see Figure 3.2. 42
3.7 This figure shows the results obtained in the "wall" setting with destination [47]. For details, see Figure 3.2. 43
3.8 Annealing schedule for exploration parameter ϵ. 44
3.9 The video game used in the experiments, Pac-Man. Pac-Man, the yellow entity, is about to eat a capsule; the smaller white dots are pills and the coloured entities with eyes are ghosts. 46
3.10 Agents performance 51
3.11 Histograms of the average reward per episode. 52
3.12 Exploration rate of the agent. The X axis shows the number of the episode, the Y axis shows the number of distinct states visited since the beginning of the experiment. Results are averaged over 200 runs of Q-learning. 54
3.13 Exploration rate of the agent. The X axis shows the number of the episode, the Y axis shows the number of distinct states visited since the beginning of the experiment. Results are averaged over 200 runs of Q-learning. 55

4.1 A visual qualitative representation of the possible situations in which confidence intervals overlap. Here, a^* is the action with the highest mean and a is another action. When confidence intervals overlap, the confidence level cannot be guaranteed. On top of each subfigure, the decision made by Algorithm 4 at lines 11–12 is indicated. 61
4.2 A screenshot of the video game used in the experiments, Pac-Man. . 64
4.3 Scores of the games (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter. 66
4.4 Decisions made with each abstraction (y axis) for each successive episode (x axis), using $\sigma = 0.2$ in Multi-Abstraction Q-learning with ϵ_{CI}-Greedy. 67
4.5 Decisions made with each abstraction (y axis) for each successive episode (x axis), using $\sigma = 0.5$ in Multi-Abstraction Q-learning with ϵ_{CI}-Greedy. 68
4.6 Victories to total games ratio (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter. 69

4.7 Eaten ghosts (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter. 70
4.8 Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy versus standard Q-learning with different sets of features. 71
4.9 Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ-Greedy versus ϵ_{CI}-Greedy. 72
4.10 Histograms of the average reward per episode. 72

5.1 Action selection strategy used by *ROSAS*. 80
5.2 Action selection strategy used by *POSAS*. 81
5.3 Score metric used by *True ROSAS*. 82
5.4 Score metric used by *True POSAS*. 83
5.5 Score metric used by *Steeper True ROSAS*. 85
5.6 Score metric used by *Steeper True POSAS*. 86
5.7 Score metric used by *Adaptive True ROSAS* visualised in 2D (a) and 3D (b). 86
5.8 Score metric used by *Adaptive True POSAS* visualised in 2D (a) and 3D (b). 87
5.9 Agents and characters used in the experiment: in (a) the agents we are testing, in (b) the participants of the 2016 competition in order of final rank when playing the *ZEN* character, in (c) the characters available in the game. 89
5.10 Agents and characters used in the experiment: in (a) the agents we are testing, in (b) the participants of the 2016 competition in order of final rank when playing the ZEN character, in (c) the characters available in the game. 89
5.11 Boxplot of the final HP difference distribution for each tested agent. . 95
5.11 The evolution of HP of the two players and their difference during fights. The data is aggregated across fights. 97
5.12 The histogram of the rank of selected actions for different Adaptive AIs. Since more actions can have the same value, the rank is computed as an interval: in the histogram, for every action we computed the minimum and maximum rank a and b and increased all the points between a and b by $\frac{1}{b-a}$. The data is aggregated across fights. 98
5.13 The histogram of the score of selected actions for different Adaptive AIs. The data is aggregated across fights. 99

5.14 Boxplot of the final HP difference for each tested agent against specific opponents. The data is aggregated across fights. 100
5.15 Boxplot of the results of our surveys. The blue rectangles include the inner quartiles, the green line indicates the median and the dashed purple line indicates the mean. 101

6.1 Distribution of waited time in our sample 108
6.2 Box-plot of perception error for players and non-players 111
6.3 Correlation between some variables and perception error. 112
6.4 The figure shows the distribution of answers for selected questions of the survey administered to participants. 115

List of Tables

3.1 Mean and standard error of the average reward per episode across the 200 runs of the experiments 50
3.2 Most often extracted goals. #C is the number of capsules, #WG is the number of weak ghost, ΔG is the distance of the closest non-weak ghosts and #Det. is the number of detections. 50

4.1 Mean and standard error of the average reward per episode across the 50 runs of the experiments in Figure 4.8. Columns report values, respectively, for Multi-Abstraction Q-learning and for Q-Learning with features Fd,Gh,ScGh,Cp, Fd,Gh,ScGh and Fd,Gh. 75

5.1 The victories percentage of the fights in the first experiment, against bots. 91
5.2 The victories percentage of the fights in the second experiment, against humans. 92
5.3 Final HP differences of the first experiment. Initial HP is 500. 92
5.4 Final HP differences of the second experiment. Initial HP is 500. . . . 92

6.1 Statistical metrics of perception error for players and non-players. The two groups both count 100 participants, but these round numbers were not planned and are a coincidence. 111
6.2 Results of statistical significance tests and effect size test. 111
6.3 Correlations found significant, along with their r coefficient and the p-value. 112
6.4 Results of the linear regressions conducted. A and B are, respectively, coefficients for non-playing time and playing time; C is the intercept. R^2 is a statistical measure $0 \leq R^2 \leq 1$ indicating how closely the data matches the formula A * non-playing time + B * playing time + C = perceived time. 114

6.5 Statistics of the survey responses that we collected. Responses, in a Likert scale, are assigned values 1 to 5 which represent, respectively, strong disagreement or strong agreement. 114

Chapter 1

Introduction

"Smart" items, such as smart appliances, smart phones etc often lack intelligence in most of the meanings of the word. Usually, the intelligence exhibited by these items comes from human expertise that was transferred into the software. This, however, is unfeasible in complex domains, such as games, robotics and agent control, where unforeseen situations may arise. In these cases, human expertise needs to be complemented with or replaced by some form of artificial intelligence able to extrapolate sensible choices given the circumstances.

Among the many real problems that Artificial Intelligence (AI) systems could solve, agent control fits many of the challenges presented in video games applications. Engagement and suspension of disbelief can greatly benefit by having agents that show non-trivial behaviours. Unfortunately, current state-of-the-art AI systems struggle to produce such behaviours due to the long time required to compute an answer that is optimal. This limitation is particularly important in the context of video games, which have tight computational time constraints [17]. This limited budget is partly due to other parts of a game requiring computation time, and partly to the fact that players expect a real-time response. Additionally, video games cannot afford to alienate the player with errors produced by an AI that did not have enough time to "get to know" the player. Nevertheless, high quality video game titles (a.k.a. AAA video games[1]) are now more than ever focusing on AI techniques.

AI techniques for these purposes need to analyse the consequences of their actions projected in the future. However, in most cases, this is not practical because the further into the future the agent looks, and the more accurate the agent knowledge, the more computations need to be completed; and this growth is exponential. Modern video games collect a large number of statistics about players, which make

[1] video games with multi-million dollars budget

for a huge player description space. With the processing time increasing rapidly with the future outlook and the volume of data, the time required for computations becomes impractical.

Another problem in systems that need to learn their environment, either from scratch or from an initial rough description is the sparsity of data. In the initial stages of the learning process, the agent has little experience, which prevents the system from generalising well. This phenomenon is called the *cold start* problem, a term first introduced in recommender system research, which also deals with tailoring output to the single user [81].

To address the large state space and the sparsity problems, researchers have turned their attention to the ways in which human and animal brains deal with such complexity. One of the methods present in nature is that of *selective attention* [13], which is based on the notion of *saliency*; that is, an item that is salient stands out relatively to its neighbours. The idea behind this is that, in conditions of limited computational power, one can ignore part of the available information and still obtain acceptable performance.

This concept of focusing only on part of a problem has already been largely investigated in the machine learning and in the data analysis fields. The practice of *feature selection* [39], which consists in explicitly ignoring irrelevant part of the information, is ubiquitous in such fields. This is done during the model creation and, once a feature is ignored, it will never be used during the learning process. However, often, the relevance of the same piece of information depends on the context. As a consequence, feature selection has to choose between ignoring sometimes-relevant information or preserving sometimes-irrelevant information.

A second method used by the human brain to tackle complex problems is that of *chaining* [15]. By putting together a sequence of simple behaviours, one agent can give rise to a more complex behaviour. After a new behaviour is learned, its execution is activated sub-consciously by a different region of the brain to reduce cognitive load.

This concept has already been formalised in the AI literature in the *options* framework [93]. Options are indeed routines that can be executed by an agent; such agent needs only to decide to commit to an option and, by doing so, it gives up control to the option itself. This decreases the computational burden on the agent, which only needs to learn how to chain options and can delegate lower level complexity to the options themselves. Work has been done in the context of learning options through the concept of novelty [76], which is inspired by the natural curiosity

observable in toddlers and kids. By means of this principle, researchers were able to let a learning agent develop incrementally complex options [83].

These two concepts can be seen as two different forms of abstraction, the first being an abstraction over the state of the environment (i.e. from complete information to only relevant information) and the second being an abstraction over the actions that the agent can perform (i.e. from low-level actions to high-level plans). Such concepts can be implemented in actual AI systems that optimise an outcome. In video games, this often is that of providing as strong an opponent as possible. However, industry has an even superior need for agents that play at the level of difficulty of their opponent, while maintaining suspension of disbelief; i.e., such agents need to feel realistic.

The area of research that focuses on such problem is that of Dynamic Difficulty Adjustment (DDA), often called also Game Balancing (GB) [46, 84]. Notice that, while the terms suggest DDA being dynamic and GB not being so, the two terms are used interchangeably in the literature. In fact, both terms are applicable to any technique that adapts the difficulty of a game to that of the player, regardless on whether this is done once at the start of a game or continuously throughout the game. Despite the great potential in terms of industry applications, DDA has not been investigated extensively; this thesis offers further study in the area.

1.1 Research Goals

This thesis focuses on Artificial Intelligence (AI) techniques applicable to games. In particular, we focus first on learning systems based on the Reinforcement Learning (RL) framework and explore novel techniques based on the concepts of selective attention and chaining. These novel methods provide new and effective angles of approach to the problems of learning behaviours in large and complex environments.

In the second part of the thesis, we focus on AI techniques aimed at supporting player experience in video games. In particular, we propose novel techniques for Dynamic Difficulty Adjustment (DDA) based on Monte Carlo Tree-Search (MCTS). These novel techniques apply ideas known in the psychology (i.e., flow, a mental state of deep focus and complete immersion) and video games AI (i.e., balanced matches have a 50% victory/defeat outcome) communities to the emerging field of DDA, suggesting a new approach that can effectively provide better experiences to players.

This research is part of an ARC Grant in collaboration with Village Roadshow. The purpose of the grant is to augment theme park clients experience using digital technologies. Since these can include games, we see this as a perfect fit for application of the proposed techniques, and of DDA in particular. Having challenging but balanced games will make it easier for theme park customers to become players and it will keep them interested for longer, keeping their attention away from the fact they may be waiting in line. In the context of the grant, a Augmented Reality (AR) mobile game was developed to augment the environment of queues in a theme park. The app consists of a simple, collaborative game that superimposes virtual elements on the reality filmed by the mobile camera. We show a data-grounded analysis that suggests that these techniques could have a positive impact on time perception if applied to games in waiting areas.

The game was played by a large variety of theme park visitors, including parents, kids and teenagers. Because of the large variety of users, the pre-defined level of challenge of the game cannot possibly fit everyone. In the final part of this thesis we analyse the data collected in a field experiment and suggest that integrating AI techniques such as those proposed to adjust the game difficulty could benefit time perception of queuing players.

When a game provides the right level of challenge, the player mind can enter a state of "flow" [18, 26], a state where the mind is sharp-focused on the task at hand and loses track of time. This is ideal in the context of a queue that can potentially last for hours, as it is normal in busy days in theme parks. Works in psychology verified that individuals whose mind is busy perceive time pass faster [36, 16].

1.1.1 Research questions

This research project investigates new approaches to Artificial Intelligence techniques, with a specific interest for video games. The project aims at answering the following research questions:

- Can goal-oriented options be learned from expert demonstration by using the concept of surprise? Will such options accelerate the learning process? *Goal-oriented options are the most common, and recent research investigated methods to learn them from demonstration. However, the concept of surprise has not been studied in this context. Resulting techniques will be inspired by the human brain and can produce high-quality options. Options introduce bias in the exploration and learning process, and good options can improve early performance through such bias. Options*

can help improve the performance of learning systems, which, currently, struggle in large and complex environments.

- Can an agent refine its abstraction on the knowledge of the environment over time? *State engineering needs to find a balance between short and long term advantages. A fine-grained state space produces sparse knowledge in the short-term. A coarse-grained state space hinders long-term performance. The ability to dynamically change the level of detail achieves the best of both worlds. An effective method of setting the degree of abstraction to be used depending on the knowledge currently held can help an agent achieving better performance at any step in its learning process.*

- Can an agent playing in a game target a 50% chance of victory while maintaining believability? *A 50% chance of victory means that the skills of the player and the opponents are evenly matched. When skill levels are matched, players can enter the "flow" state of mind, where they enjoy their time and lose track of it, while maximising their performance. Various methods have been proposed for difficulty adjustments, but only one [4] targets the outcome of 50% chance of victory, and its approach presents several limitations. A more effective approach can improve user experience and advance the quality of video games.*

- Is there evidence to suggest that adjusting the difficulty of a game played in a queuing environment can improve time perception? *Games allow players to enter a focused state where time perception is altered, the "flow" state. A faster time perception would be particularly beneficial in waiting situations. However, to allow for a flow state, a game must not be too easy or too difficult. Dynamic difficulty adjustment tailors the game to the player to this end. Achieving this is of great value to the theme parks industry where people spend most of their time waiting.*

1.1.2 Contributions

This research focuses on novel Artificial Intelligence (AI) techniques based on concepts from psychology. We first investigated how to use the concept of surprise to learn goals that can be embedded in options. To improve options performance we looked into state abstraction. We then focused on the problem of Dynamic Difficulty Adjustment (DDA). Finally, we analyse real-world data and suggest that DDA used in games could benefit time perception of players in theme parks queuing areas. The contributions of this research work are:

- An algorithmic solution to learn options from expert demonstrations based on the notion of "surprise". This algorithm detects unexpected expert decisions and infers the expert motivating goal. Extracted goals are used to create goal-oriented options that bias learning in a positive way, increasing performance during the learning process. The use of the concept of surprise has not been investigated before in the problem of learning options.

- An algorithmic solution to perform online state abstraction in Markov Decision Processes using standard Student's t test. This is coupled with an ad-hoc heuristic exploration strategy to further improve agent performance. These techniques increase performance in the learning process by avoiding the problem of data sparsity. This approach of progressively using finer-grained abstractions by considering more features has not been investigated before in RL.

- Algorithmic solutions to target a 50% chance of victory in games. These algorithms use Monte Carlo Tree Search to choose the action most likely to produce the desired outcome, while maintaining a believable behaviour. These approaches build on well-established ideas in the research communities of psychology and video games AI and improve on the state of the art.

- Analysis and discussion of real-world data that suggests that applying DDA to games played in a theme park queuing area could benefit time perception of players, making them feel like they had been waiting a shorter time span than they actually had. A quantitative analysis of data collected in a theme park queuing environment has not been performed before. This can inform theme parks managers on how to improve the experience of their customers.

1.1.3 Publications

This research project has produced the following papers:

- Tamassia, M., Zambetta, F., Raffe, W., and Li, X. (2015). Learning options for an MDP from demonstrations. In Chalup, S., Blair, A., and Randall, M., editors, *Artificial Life and Computational Intelligence*, volume 8955 of *Lecture Notes in Computer Science*, pages 226–242. Springer International Publishing

- Raffe, W. L., Tamassia, M., Zambetta, F., Li, X., Pell, S. J., and Mueller, F. F. (2015b). Player-computer interaction features for designing digital play expe-

riences across six degrees of water contact. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pages 295–305. ACM

- Raffe, W. L., Tamassia, M., Zambetta, F., Li, X., and Mueller, F. F. (2015a). Enhancing theme park experiences through adaptive cyber-physical play. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 503–510. IEEE

- Tamassia, M., Zambetta, F., Raffe, W., Mueller, F. F., and Li, X. (2016b). Learning options from demonstrations: A pac-man case study. *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*

- Tamassia, M., Zambetta, F., Raffe, W., Mueller, F. F., and Li, X. (2016a). Dynamic choice of state abstraction in q-learning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press

- Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., and Mueller, F. F. (2017b). Monte carlo tree search based algorithms for dynamic difficulty adjustment. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE

- Suguru, I., Ishihara, M., Tamassia, M., Tomohiro, H., Thawonmas, R., and Zambetta, F. (2017). Procedural play generation according to play arcs using monte-carlo tree search. In *Game-ON'2017: 18th International Conference on Intelligent Games and Simulation*

- Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., and Mueller, F. F. (2017a). Measuring player skill using dynamic difficulty adjustment. In *Proceedings of the Australasian Computer Science Week Multiconference*. ACM

- Zambetta, F., Raffe, W. L., Tamassia, M., Mueller, F. F., Li, X., Dang, D., Quinten, N., Patibanda, R., and Satterley, J. (2017). Reducing perceived waiting time in theme park queues via an augmented reality game. *ACM Transactions on Computer-Human Interaction (TOCHI)* **[Under review]**

1.2 Thesis structure

This thesis is structured as follows. In chapter 2 we introduce background information covering the concepts of Reinforcement Learning, Markov Decision Process,

options framework and abstraction. In chapter 3 we present a novel algorithm to learn options from expert demonstrations based on the notion of surprise. Experiments to test the algorithm in a grid-world and in the video game Pac-Man are also reported and discussed. In chapter 4 we introduce a novel algorithm to perform dynamic state-abstraction in Markov Decision Processes using standard Student's t test; a novel ad-hoc exploration strategy is also introduced to further improve the algorithm performance. Experiments to test the algorithm in the video game Pac-Man are also reported and discussed. In chapter 5 we present novel techniques based on Monte Carlo Tree Search to achieve Dynamic Difficulty Adjustment while maintaining believability. Experiments to test the algorithm in a 2D fighting video game are also reported and discussed. In chapter 6 we analyse data collected during a field study in a theme park queuing area and show how the data suggest the suitability of AI techniques in such environments. Finally, in chapter 7 we draw the conclusions of this work and we look at the potential for future work.

Chapter 2

Background and Related Work

The overarching topic of this thesis is Artificial Intelligence applied to video games. In particular, the focus is on Decision Theory techniques: Reinforcement Learning (RL) techniques are used in the first part, and techniques based on Monte Carlo Tree Search (MCTS) are used in the second part.

Decision theory combines probability theory with utility theory to provide a formal and complete framework for decisions made under uncertainty [79]. RL and MCTS provide two different answers to decision problems. The former takes the learning approach, where a system builds a model of the environment and of the value of actions that can be performed therein. The latter, assumes a perfect simulator of the environment is provided and produces an answer on-line by using the simulator to perform statistical sampling.

Both these techniques have upsides and downsides. RL needs time to train and, for non-trivial problems, requires an ad-hoc representation of the states of the world that compromises between accuracy and learning time, but then is very fast at execution time. MCTS assumes that a simulator of the environment is given and needs enough time on-line to perform its statistical sampling, but does not need to compromise on the accuracy of the state representation.

This chapter gives a background in Reinforcement Learning and in Monte Carlo Tree Search, necessary to understand the contents of this thesis. The chapter also covers related work to that presented in this thesis. In particular, Section 2.1 gives an introduction to Reinforcement Learning and the techniques used in this thesis, covering Markov Decision Processes (MDPs) (2.1.1), value functions (2.1.2), optimal value functions (2.1.3), model-free learning and control (2.1.4) and exploration-exploitation (2.1.5). Section 2.2 surveys works related to abstraction in Reinforcement Learning, including temporal abstraction (2.2.1) and state abstraction (2.2.2). Section 2.3 gives

an introduction to Monte Carlo Tree Search (MCTS), including combinatorial games (2.3.1, tree-search techniques (2.3.2), Monte Carlo methods for tree-search (2.3.3) and UCT (2.3.4).

2.1 Reinforcement Learning

Machine Learning (ML) [67] is a discipline in the area of Artificial Intelligence that deals with systems that learn from data. This is opposed to, for example, expert systems where details on how to complete a task are implemented in a system by engineers. In Machine Learning, general algorithms exist that can solve different tasks by being fed with different data.

The historically most famous paradigms in ML are Supervised Learning and Unsupervised Learning. In Supervised Learning (SL), a learning system is presented with labelled data and must learn to associate data with labels. A good SL system is also able to generalise its knowledge and predict labels for previously unseen inputs. In Unsupervised Learning (UL), data is presented to the learning system which then extracts patterns or other useful information from the data. The most common application of UL is clustering, where the algorithm divides data into groups.

Reinforcement Learning [92] (RL) is another paradigm in ML which gained media attention in recent years when, associated with Deep Neural Networks, solved problems of unprecedented complexity. The most famous such achievements are in the domains of Atari 2600 and Go. In the former, the system learned to play games based on the pixels of the screen [68]; in the latter, the system AlphaGo was able to defeat the European Go champion [82]. More recently, RL was used to learn locomotive behaviours in rich simulated environments [42] and it even beat a professional DotA 2 (a popular Multiplayer Online Battle Arena video game) player in a 1-vs-1 match [70].

From a certain point of view, RL sits in between Supervised and Unsupervised: data is not labelled, but also the system is not left completely in the dark about the data. RL is usually an interactive process, where the system receives (unlabelled) data in input and produces an answer; the system is then fed back with an evaluation on how good the answer was and uses this information to update its internal knowledge to produce better answers in the future.

Reinforcement Learning settings are better understood as Agent-Environment systems, as shown in Figure 2.1. An agent observes the state of the environment and influences it by performing actions; when an action is performed, the environment

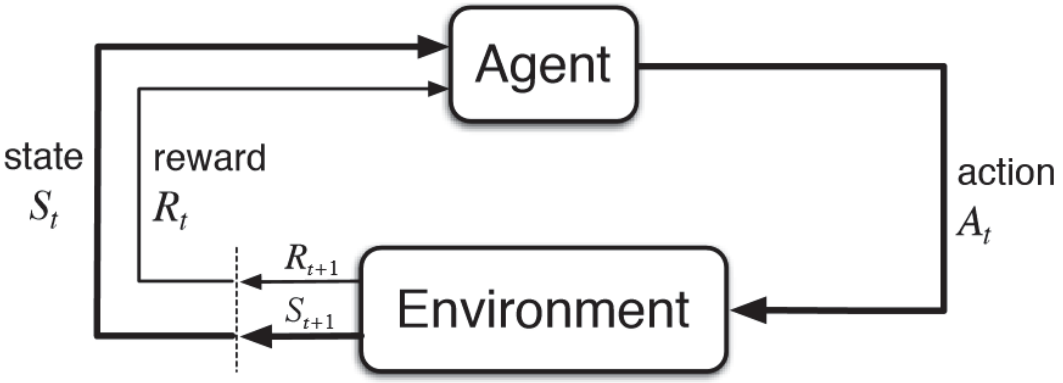


Figure 2.1 A representation of the Reinforcement Learning paradigm (image from [92]).

changes its state and sends a reward to the agent that signals how good the outcome is. Environment state transitions are, in general, stochastic; that is, the action performed by the agent does not uniquely determine the new state but, rather, it determines the random distribution from which the new state is sampled. The same holds for rewards, in the most general definition.

2.1.1 Markov Decision Processes

The definition of RL encompasses all algorithms that learn through an evaluation signal. However, the vast majority of works in RL assume that the environment has the Markov property. The Markov property says that all the relevant information for state transitions and rewards are contained in the state of the environment. This allows the environment to be modelled as a Markov Decision Process (MDP).

An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the set of all possible *states* of the environment; $\mathcal{A}$ is the set of *actions* the agent can perform; $T : \mathcal{S} \times \mathcal{A} \to \mathrm{PD}(\mathcal{S})$ associates state and action to a *next state probability distribution*; $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathrm{PD}(\mathbb{R})$ associates state, action and next state to a *reward distribution*; $\gamma < 1$ is a *discount factor* used to decrease the value of future rewards [73]. Notice that this is the most general definition for the reward, but other definitions are often used where the reward is deterministic or does not depend on the next state. In the following, we also interchangeably use slightly a different definition for T: $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$.

An agent-environment interaction develops in sequential time steps. At every time step t, the agent senses the state of the environment s_t, decides an action a_t to perform, receives a reward r_t and senses the new state of the environment s_{t+1}. The

purpose of the agent is to maximize the expected discounted reward collected over time, often called the *return*:

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots = \sum_{k=0}^{\infty} \gamma^k r_k.$$

In order to do so, it must learn a policy $\pi^* : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{A})$ that associates states to a probability distribution over actions. At time step t, then, after sensing the state s_t, the agent will draw action a_t from the distribution $\pi^*(s_t)$. In cases where the environment is stationary, the optimal policy can be deterministic: $\pi^* : \mathcal{S} \to \mathcal{A}$; in general, to account for adversarial behaviour, the optimal policy may need to be stochastic [58]. In this thesis we focus on stationary environments; in this introductory chapter, however, we will use the most general definition.

2.1.2 Value Functions

It is fundamental to notice that the optimal policy does not maximise the immediate reward but, rather, the long-term return. In other words, the optimal policy strategises so as to strike the best compromise between immediate reward and the degree of how promising the next state is expected to be. The difficulty in this is that the second objective is hard to define: it needs to consider rewards collected from the current time step on-wards and it must take in account that state transitions, as well as rewards, are stochastic. *Value functions* are elegant formalisms that capture this concept.

The state value function $V^\pi(s)$ associates state s with the expected future return if following policy π from s. As Bellman discovered, the state value function can be defined recursively [9]:

$$V^\pi(s) = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \mid s_0 = s, \pi \right\} \tag{2.1}$$

$$= \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^\pi(s') \big], \tag{2.2}$$

where s' is the state of the environment after in the following time step. In fact, repeatedly applying Equation 2.2 from right to left, the state value function can be

computed in a finite number of iterations:

$$V^{\pi}(s)_{k+1} = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^{\pi}(s')_k \big]. \tag{2.3}$$

This recursive equation captures the notion that the value of a state(-action) depends on the expected value of the next state which, in turn depends on the expected value of its next state and so on. Equation 2.2 is an instance of a *Bellman equation*. A Bellman equation writes the value of a decision problem at a certain point in time in terms of the payoff from some initial choices and the value of the remaining decision problem that results from those initial choices. Bellman equations are at the foundation of dynamic programming, an optimisation approach that transforms a complex problem into a sequence of simpler problems [12].

2.1.3 Optimal-Value Functions

If a policy π_k is sub-optimal, its state value function V^{π_k} may have a wrong estimate for some states. However, V^{π_k} can be used to define a better policy π_{k+1} as follows:

$$\pi_{k+1}(s) = \arg\max_{a} \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^{\pi_k}(s') \big]. \tag{2.4}$$

This process can be iteratively repeated and is guaranteed to converge to the optimal value function [73]:

$$V^{*}(s) = \max_{a} \mathbb{E} \left\{ \sum_{k=0}^{\infty} \gamma^k r_k \mid s_0 = s, \pi^{*} \right\} \tag{2.5}$$

$$= \max_{a} \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^{*}(s') \big]. \tag{2.6}$$

The resulting algorithm is called *policy iteration* and is an instance of a dynamic programming technique.

Notice that policy iteration needs to compute the value function at every iteration, by repeatedly applying Equation 2.3. An alternative method for improving over a sub-optimal value function that avoids this is *value iteration*. Value iteration iteratively applies Equation 2.6 from left to right, repeatedly:

$$V(s)_{k+1} = \max_{a} \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^{(}s')_k \big]. \tag{2.7}$$

Value iteration is also guaranteed to converge to the optimal value function [73].

Under the assumption that the environment can be modelled as a Markov Decision Process, optimal value functions provide information that allows to trivially compute the optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s') \big[\mathbb{E}\{R(s,a,s')\} + \gamma V^*(s')\big].$$

2.1.4 Model-free Learning and Control

Dynamic programming approaches need the model of the environment to be given in input. This can be seen by the presence of both T and R in Equations 2.4 and 2.7. In order to learn without a model, a RL agent needs to make use of its experience in the environment in order to approximate an optimal value function. Experience in the environment can be formalised as a sequence of tuples in the form of (s_t, a_t, r_t, s_{t+1}). Q-learning achieves this by updating an estimate of the state-action value function at every step using the following update rule [107]:

$$Q(s_t, a_t) \overset{\alpha}{\leftarrow} r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t), \tag{2.8}$$

where $x \overset{\alpha}{\leftarrow} y$ is short for $x \leftarrow x + \alpha y$ and $0 < \alpha \leq 1$ is a learning factor. Under the condition that every state is visited infinitely often, Q-learning converges to the optimal state-action value function [29]. An important property of Q-learning is that it learns *off-line*, meaning that it can learn the value function associated to the optimal policy no matter what the policy being used while interacting with the environment is [92].

The state-action value function allows to compute the optimal policy without the need of the model of the environment:

$$\pi^*(s) = \arg\max_a Q^*(s,a).$$

A state-action value function, however, does not need to be perfect to be used for action selection. In fact, it is common to use the Q-learning estimate at every time step to choose the next action [38].

2.1.5 Exploration-Exploitation

When the state-action value function is used to choose the next action, the choice is called a *greedy* choice. An agent performing exclusively greedy choices is not guaranteed to choose every action infinitely often, in the limit. Recall that this is a necessary condition for guaranteeing Q-learning convergence. This is confirmed in empirical tests: a greedy agent will often not converge to the optimal value function [92].

Strategies aimed at avoiding this are called *exploration* strategies, because they push the agent to explore (seemingly) sub-optimal regions of the state space. However, in practical cases, exploration must not come too much at the cost of exploitation, that is, actual collection of rewards. Good exploration strategies strike a good compromise in the so-called exploration-exploitation dilemma.

The most popular exploration strategy is the ϵ-*greedy* strategy. This simple strategy chooses a random action with probability ϵ, and the greedy one with probability $1 - \epsilon$ [92]. Formally:

$$\pi(s) = \begin{cases} \text{random}(\mathcal{S}) & \text{with prob. } \epsilon \\ \arg\max_a Q^*(s,a) & \text{with prob. } 1-\epsilon. \end{cases}$$

In real cases, however, it is desired that the agent converges to the optimal policy. ϵ-greedy prevents this from happening because some of the actions are randomly (therefore potentially sub-optimally) chosen. To guarantee convergence to the optimal policy, the parameter ϵ is annealed (i.e., reduced) over time, so as to smoothly transition from an exploration policy to a greedy policy. This strategy is sometimes called *annealing* ϵ-*greedy*; an example of this strategy is ϵ_n-greedy [6]. Another common exploration strategy is Softmax [92], but we do not present it here as it is not relevant to understand the work in this thesis.

2.2 Abstraction

Abstraction is often used in Markov Decision Processes when dealing with real world problems. In the most general sense, abstracting means letting go of some details. For example, it is common in MDP to assume that the environment dynamics can be described by a linear model, thereby leaving out non-linear details. The types of abstraction that are used in this thesis are temporal and state abstraction. In

the former case, introduced in Section 2.2.1, the notion that actions are limited to one time-step is generalised to allow for longer courses of action. In the latter case, introduced in Section 2.2.2, state specific information are ignored to allow for knowledge transfer between states.

2.2.1 Temporal abstraction

One problem with MDPs is that an action only affects the state and reward immediately following in time. With no notion of a course of action persisting for longer than a time-step, conventional MDP methods are unable to take advantage of the simplicities and efficiencies sometimes available at higher levels of temporal abstraction [93]. To address this issue, Sutton et al. introduced the *options* framework, which elegantly extends the MDP concept to include temporal abstractions.

The central concept in this work is that of options, closed-loop policies for taking actions over time [93]. The framework extends an MDP to allow the use of actions (called *primitive actions*) and options interchangeably. The extension is natural enough that dynamic programming and model-free learning methods work with minimal modifications.

An option, like any other action, can be chosen by a policy. However, unlike what happens for primitive actions, when an option is chosen, its internal policy is followed for some period of time, until the option stochastically terminates. Formally, an option o is defined as a tuple $o = (\mathcal{I}, \pi, \beta))$, where:

- $\mathcal{I}$ is an *initiation set*, including the states from which option o can be chosen;
- π is the policy to be followed when option o is selected;
- $\beta : \mathcal{S} \to \mathbb{R}$ expresses the probability of termination of option o and depends on the state.

A slight variation of the Q-learning update rule can be used to update the state-option value upon option termination. The rule used is as follows [93]:

$$Q(s_t, o_t) \overset{\alpha}{\leftarrow} r + \gamma^k \max_{o \in \mathcal{O}} Q(s_{t+k}, o) - Q(s_t, o_t), \tag{2.9}$$

where $r = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{k-1} r_{t+k}$ is the discounted cumulative reward obtained during the option execution. This rule leads Q to converge to the optimal state-option value function under the same conditions of Q-learning, that all states are visited infinitely many times in the limit.

Learning options

Options are useful tools to achieve good performance in Reinforcement Learning. However, their core components are not trivial to define. As Jong et al. argue, when options are used during the learning phase, they introduce bias in how the state-action space is explored [50]. While this implies that options are a valuable tool to direct exploration, it also means that they must be used with care, at the risk of worsening performance.

Because hand-crafting options is expensive and error-prone, research has been directed to automatic learning. Existing literature on this subject can be divided in three different main areas: approaches based on state spaces structure, on intrinsic motivation and on learning from demonstration. Most of the works use a variety of approaches to identify goal states, which are then used to build options whose objective is to reach such states.

Earlier work is focused on using the state space structure to identify useful goals. McGovern and Barto infer goals online, based on commonalities across multiple paths to a solution [65]. Stolle and Precup detect "bottleneck" states, which are states frequently visited, during a variety of tasks with different goals [86]. Menache et al. also use "bottleneck" states as sub-goals, but they use graph theory to compute them [66]. Mannor et al. learn a model of the environment online and then use a clustering algorithm to partition the state space; then, they create options for reaching each of the regions [61]. Şimşek et al., similarly, split the state transition graph in clusters, but they build graphs online and only using recent experience [27]. Jonsson and Barto use a Bayesian network to model actions in a factored state space and introduce options that make changing the value of some state features easier, thereby decomposing a complex task in simpler ones [52].

Other works focus on the notion of intrinsic motivation; that is, driving exploration and deciding goals based on some notion independent on the reward. Şimşek and Barto use the concept of relative novelty to identify states that connect areas of the state space with different degrees of exploration; they use these states as goals for options, assuming that such options are useful since they allow to transition to different parts of the state space [83]. Bonarini et al. score states according to how "interesting" they are; their notion of interest is related to how difficult it is to get to a state and to remain in that state [11]. Vigorito and Barto work in a factored MDP where they use the ability to change the value of a state variable as the driver for intrinsic motivation; such notion both drives the exploration process and is used to create options [104].

A third area of research focuses on learning options from expert information, such as solution policies or demonstrations trajectories (i.e., traces interactions between an expert agent and the environment). Pickett and Barto find commonly occurring sub-policies in the solutions to a set of tasks [72]. Zang et al. break trajectories to identify the best sub-problem in terms of size, frequency and abstraction; they then build an option to solve the sub-problem and repeat the process [116]. Konidaris et al. segment trajectories into a chain by detecting a change of appropriate abstraction (or when the segment is too complex) and build an option to solve each segment [54]. Subramanian et al. ask humans to give high level goals and build options to solve them [88].

One of the key contributions of this thesis is a method to learn options from demonstrations, based one the intrinsically motivating concept of "surprise". Our approach is to find "surprising" decisions made by experts in demonstrations, infer their unpredicted intentions and use them as goals. This idea reflects studies in cognitive research where infants were observed to get excited by unexpected events and motivated to explore them further [85].

2.2.2 State abstraction

The biggest obstacle preventing the application of MDPs in practical scenarios is the "curse of dimensionality". This name is used to describe undesirable phenomena that occur when dealing with high-dimensional data. In RL this happens when the state space is too large; in such cases, dynamic programming methods take a long time to converge, and model-free learning methods require an unacceptably long learning time. This usually happens when states are described by many features, because the state space size increases exponentially with the number of features.

This obstacle has prompted research in the area of state abstraction. Most of the work in the area focuses on choosing a way to abstract the state space (i.e., partitioning it) prior to the learning phase. Jong and Stone use hypothesis testing to discover useful abstractions using Q-values learned in multiple runs [51]. Cobo et al. select the features that are useful to reproduce the behaviour of a given set of demonstrations [19]. Hallak et al. use time-series data to select among the models provided by a human expert [40]. Jiang et al. provide theoretical guarantees on the used abstraction, but their approach requires the use of value iteration, which is computationally expensive [49]. Džeroski et al. learn structural representations, which form an abstraction over the state space [33].

Function approximation techniques are a popular way to implicitly learn generalisations over the state space. Tesauro use a neural network to play Backgammon at a strong master level [101]. Sutton use sparse-coarse-coded function approximators to generalise over the state space [91]. Stone and Sutton use a linear tile-coding function approximator to play Robocup Soccer [87]. Ng et al. use Inverse Reinforcement Learning to learn the parameters of a linear approximator of the Q function that controls a small helicopter [69]. In recent years, deep neural networks have produced unprecedented results. Mnih et al. use Deep Q-learning to play Atari video games using pixels as input [68]. Silver et al. use a system based on deep neural networks and Monte Carlo Tree Search, AlphaGo, to defeat the European Go champion [82]. Heess et al. use deep neural networks to power a RL system that learns locomotive behaviours in rich simulated environments [42].

A smaller corpus of research focuses on learning state abstractions online. To the best of the candidate's knowledge, only the works of McCallum fall in this area. In [62], he expands temporal memory to distinguish variations in rewards, and does so via hypothesis testing; this approach, however, is slow because memory is expanded one step at a time. In [63] he proposes to store raw history, so when memory is expanded, history can be re-analysed to properly compute values; this approach, reportedly, does not handle noise very well. In [64], McCallum et al. proposes to use, along with stored history, a tree to know how deep (how far back in history) one needs to look to distinguish situations: branches are added when a statistical test says that samples come from two different distributions[1].

One of the key contributions of this thesis is a method to use state abstraction during the learning phase. The proposed approach is based on statistical tests, which are used to switch from coarse to more fine-grained abstraction as estimates become reliable enough.

2.3 Monte Carlo Tree Search

When a problem involves two or more agents, it is said to be in the realm of Game theory. Game theory is an area at the intersection of Artificial Intelligence and Mathematics that deals with situations in which multiple agents, each with its own utility function, interact with each other.

One such game can be described by the following components:

[1]The test used is Kolmogorov-Smirnov [22]

- S: the set of states, where s_0 is the initial state;
- $S_T \subset S$: the set of terminal states;
- $n \in \mathcal{N}$: the number of players;
- A: the set of actions;
- $f : S \times A \to S$: the state transition function;
- $R : S \to \mathcal{R}^k$: the utility function;
- $\rho : S \to (0, 1, \ldots, n)$: player about to act in each state.

A game starts in state s_0. At each time step t, the agent $\rho(s_t)$ is informed of the state s_t and performs an action $a_t \in A$; the state then transitions according to $s_{t+1} = f(s_t, a_t)$ and the agent receives a reward $R(s_{t+1})$. The game terminates if the current state is terminal; i.e., $s_t \in S_T$. Each agent i has its own policy $\pi_i : S \times A \to [0, 1]$; such that $0 \leq \pi_i(s, a) \leq 1$ and $\sum_{a \in A} \pi_i(s, a) = 1$ for all s, which determines the probability of performing action a in state s.

2.3.1 Combinatorial games

Games can be classified based on certain properties that they may or may not have. The most popular such properties are:

- *Zero-sum*: A game is called "zero-sum" if the reward to all players sums to zero. In the case of 2-player games, this means that the players are in direct competition: every advantage a player gains yields an equal disadvantage to the opponent, and vice versa.
- *Information*: If the game state is fully observable by players, the game is said to have "perfect information".
- *Determinism*: A game is called "deterministic" if chance plays a role in state transitions and/or rewards.
- *Sequential*: A game is called "sequential" if player actions are applied one after the other; the alternative is that actions are applied simultaneously. The subtle difference is that in non-sequential games, a player acting after another will have knowledge of the action chosen by the previous player and its consequences.

Games with two players that are zero-sum, perfect information, deterministic, discrete and sequential are described as *combinatorial games*. These include games such as Go, Chess and Tic Tac Toe, as well as many others. Solitaire puzzles may also be described as combinatorial games played between the puzzle designer and the puzzle solver, although games with more than two players are not considered combinatorial due to the social aspect of coalitions that may arise during play. Combinatorial games make excellent test beds for AI experiments as they are controlled environments defined by simple rules, but which typically exhibit deep and complex play that can present significant research challenges, as amply demonstrated by Go. [14]

2.3.2 Tree search techniques

In non-trivial games, the reward is often delayed. In particular, in many games the reward is non-zero only in terminal states. That means that, in order to figure out what the best action is, an agent needs to look far into the future. Since the evolution of the state is dependent on every agent's decisions, the most natural way to represent the problem is through a tree data structure.

The simplest tree-search technique is called *minimax* and is suitable to combinatorial games. Minimax attempts to minimise the maximum loss of an agent; in zero-sum games, this is the same as maximising the minimum gain of the agent. To this end, Minimax explores the virtual tree of the game, where nodes represent states and edges represent actions; the tree alternates player levels and opponent levels. Player levels represent states where the player is the one to execute the next action, opponent levels represent states where the opponent is.

Minimax associates a value to each node of the tree, and does so recursively. Starting from the root, and applying recursion, the algorithm calculates the value of a node (representing state s) as follows:

- *on a leaf (s is terminal)* the value is the utility of the state, $R(s)$;
- *on a non-leaf node in a player level* the value is the maximum value among those of the children plus the value of the current node, $R(s)$;
- *on a non-leaf node in a opponent level* the value is the minimum value among those of the children plus the value of the current node, $R(s)$;

In games like Chess and Go, the utility of all non-terminal states is zero, making the algorithm propagating the leaf values upstream without modification.

Often, the tree is not explored completely because most non-trivial games have a very large state space. In such cases, the exploration of the tree is stopped a a certain depth and a heuristic $h : s \to \mathcal{R}$ is used instead of the actual final outcome of the game. The algorithm then calculates the value of a node as follows:

- *on a leaf (s is terminal)* the value is the utility of the state, $R(s)$;
- *on a node at depth at least d* the value is calculated by the heuristic function $h(s)$;
- *on a non-leaf node in a player level* the value is the maximum value among those of the children plus the value of the current node, $R(s)$;
- *on a non-leaf node in a opponent level* the value is the minimum value among those of the children plus the value of the current node, $R(s)$;

Deep Blue, the algorithm that defeated the Chess world champion Gary Kasparov, used this algorithm: it explored the tree up to 12 moves and then used a heuristic to compute the value of the state [45].

Minimax can be extended by using further heuristics to prune the tree ($\alpha - \beta$ pruning), however, this is not relevant for understanding the content of this thesis and is therefore not covered here.

Minimax can also be extended to deal with a larger number of players. The resulting algorithm, $\max^n$, maximises the reward of each player at each step. This, however, is not relevant to this thesis and is therefore not covered.

Minimax can finally be extended to deal with non-deterministic state transitions. This is done by adding "chance" levels, where the recursion step chooses a random value from one of the children nodes. This algorithm, called "Expectimax" is also not relevant and therefore not covered.

2.3.3 The algorithm

Monte Carlo methods are a class of algorithms that rely on repeated random sampling to obtain numerical results. These methods have proven successful first in areas like numerical integration, and have since been used in a wide array of domains, including game research [14].

In fact, random sampling could be an alternative approach to compute the value of actions that could relieve the agent from the burden of having to explore the whole tree. This could be done, ideally, by sampling random games after each initial action. This, however, is only possible if the agent has, beside the simulator of the

environment, a simulator for every other agent. Since this is not realistic, the next two options are:

- To assume the other agents perform random actions; this, however, has serious limits because in reality opponents are likely to choose better-than-average actions (from their perspective).
- To assume the other agents are perfectly rational; this means falling back to Minimax/Expectimax, including the disadvantage of having to explore all of the large tree.

A hybrid approach could be to apply the same principle of Minimax until a certain depth, and then use the average outcome of random games to estimate the value of leaf nodes. This has the advantage of providing a heuristic that does not require domain expertise, but does not solve the problem of having to explore the full tree up to the preset depth limit.

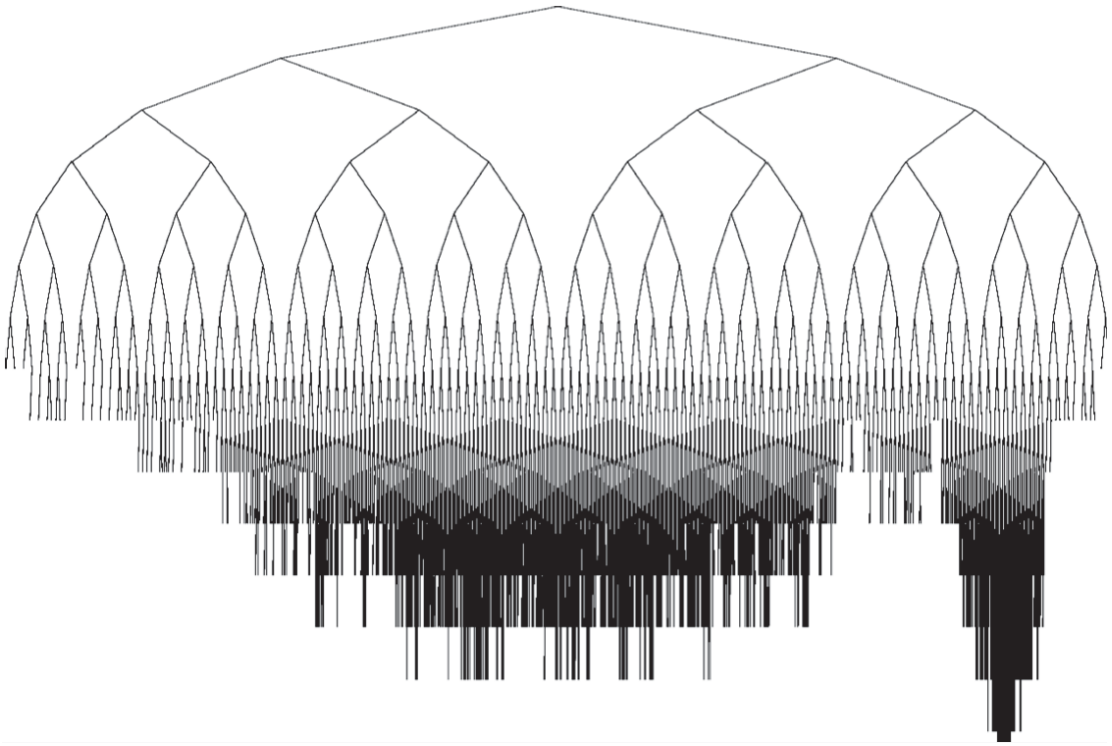


Figure 2.2 A representation of an asymmetric tree.

Monte Carlo Tree Search (MCTS) offers a solution to this problem by providing *asymmetrical exploration* of the search tree as illustrated in Figure 2.2. To do this,

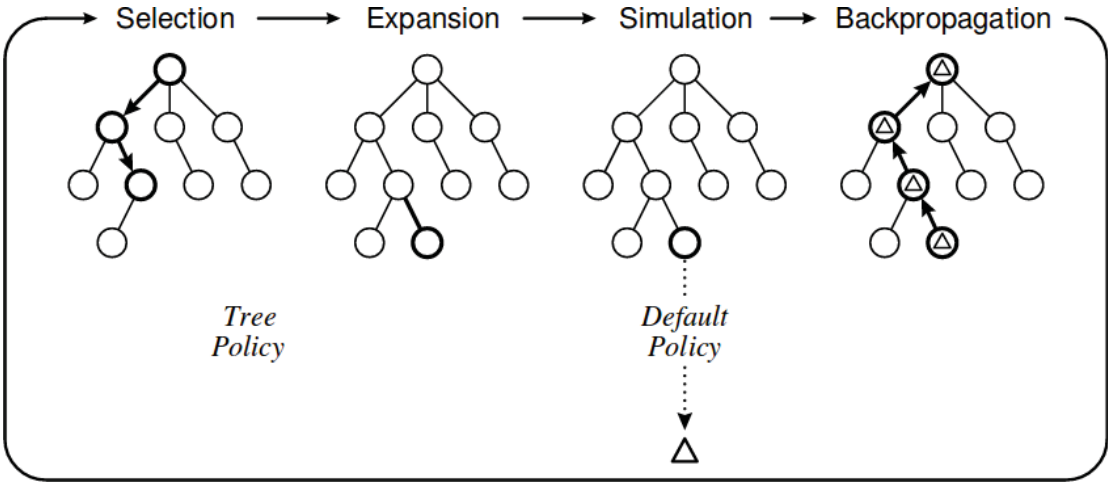


Figure 2.3 A representation of the Monte Carlo Tree Search algorithm (image from [14]).

MCTS iteratively builds progressively better value estimates by deepening its search in the tree only in branches that look promising according to its current estimates.

This is done by repeatedly executing four phases, one after the other. The four phases, illustrated in Figure 2.3, are:

- *Selection*: in this phase, the tree is descended along the path of the most promising actions according to the current estimates, until a leaf is reached.
- *Expansion*: in this phase, if the leaf reached is above a certain depth threshold, it is expanded by adding all possible actions as children.
- *Simulation*: in this phase, the actions indicated by the path root–leaf are run into the simulator, followed by a sequence of random actions (up until the game ends or up to a certain length).
- *Back-propagation*: in this phase, the value of the outcome of the simulation (or a heuristic, if the simulation stopped before the end of the game) is used to update the statistics of the nodes in the path root–leaf.

This gives MCTS another advantage over Minimax: the four phases can be repeated as many times as the time allotted for the computation permits, and can be stopped at any time. The pseudocode for MCTS is shown in algorithm 1.

Algorithm 1: Pseudocode for MCTS.

Input : State s
Input : Depth limit l
Input : Playout length p
Input : Heuristic function h
Input : Simulator sim

```
create root node r
r.visits ← 0
r.total_reward ← 0
repeat
    select a leaf node n
    if n.depth < l then
        create children nodes of n
    end
    actions ← actions from r to n + p random actions
    s' ← sim(s, actions)
    for every node n' in the path r–n do
        n'.visits ← n'. visits +1
        n'.total_reward ← n'.total_reward +h(s')
    end
until allotted time expires
```

2.3.4 UCT

The process illustrated above is missing an important detail: the definition of what a "promising node" is. The default policy is to choose an unexplored node and, if all nodes are explored, to choose the node with the highest value. This policy is not robust because it can cause a strong branch to never be explored because of an unlucky first visit; conversely, a weak branch could become the action of choice because of a lucky first visit. To address this problem, UCT is used to drive the tree exploration [53]. UCT stands for UCB1 applied to Trees, where UCB1 stands for Upper Confidence Bound [6].

UCB1 minimises a measure called "regret" in "bandits", which can be thought as depth-1 trees. Regret is the difference between the reward that could be earned by choosing optimally and the reward that was actually earned.

$$R_N = \mu^* n - \mu_j \sum_{i=1}^{K} E\left[T_j(n)\right], \tag{2.10}$$

where μ^* is the best possible expected reward, $E[T_j(n)]$ denotes the expected number of plays for action j and K is the number of actions.

This is implemented by adding to the value of a node a number that increases with the exploration of other actions and decreases with the exploration of the node action. The UCB1 formula is as follows:

$$\text{UCB1} = \overline{X}_j + \sqrt{\frac{2 \ln n}{n_j}}, \tag{2.11}$$

where $\overline{X}_j$ is the average reward from action j, n_j is the number of times action j was tried and n is the overall number of plays so far.

UCT generalises UCB1 to trees, where the behaviour of UCB1 in deeper, recently created nodes causes a drift in the probability distribution of rewards. UCT is proved to also minimise regret and minimally changes the UCB1 formula:

$$\text{UCT} = \overline{X}_j + 2C_p \sqrt{\frac{\ln n}{n_j}}, \tag{2.12}$$

where $\overline{X}_j$ is the average reward from action j, n_j is the number of times action j was tried at the node, n is the sum of n_j for all actions and $C_P > 0$ is a constant.

2.3.5 Dynamic Difficulty Adjustment

Video games present challenges to players. Some game genres, through careful level design, present an increasing level of challenge as the game progresses. Examples of this are *Super Mario Bros* and *Portal*. Other games, where game play is emergent from simple rules, as opposed to imposed by level design, have a harder time modulating the level of challenge to players. In many games, this is roughly addressed by letting the player select a level of difficulty: the menu with "Easy", "Medium" and "Hard" (and variations) is something every video game player has seen in their lives.

This approach has several limitations: first, it groups potentially millions of players in three (maybe up to five) groups in terms of skills. This is likely too coarse to reflect reality. Another limitation is that, once chosen, the difficulty selection cannot be changed through the game. This ignores the fact that players get better at the game, let alone that they have different learning curves. Finally, this approach to skill selection is uni-dimensional. Many games require a varied set of skills; shooting games, for example, require fast reflexes, precise aiming, team coordination,

awareness of surroundings and knowledge of the levels. The level of the player at each of these skills cannot possibly be conveyed by selecting a point on a unidimensional scale.

Addressing this problem represents a big challenge for video games. Balancing the difficulty of the game to the skills of the player is one necessary step to allow the player to enter the "flow" state. Flow is a mental state where a person is deeply focused on the task at hand and feels disconnected from reality to such an extent that the sense of time is altered [25]. Flow has been described as a state of extreme immersion and has been studied both qualitatively and quantitatively in the field of video games [48]. Sweetser and Wyeth translated the model of flow to the specific field of games and proposed GameFlow [94]. Cowley et al. translate the ideas of flow into video games from an information processing point of view [24].

While a balance of difficulty and skills is not necessary to achieve immersion [48], it is a requirement to reach the deeper state of flow [25, 94, 24]. Researchers in the Artificial Intelligence field have studied techniques to achieve difficulty adjustment or at least game customisation to increase satisfaction. Togelius et al. used evolutionary techniques to generate tracks for a racing game that were tailored to the individual player [102]. Leigh et al. used co-evolution to find dominant strategies in a game that were tuned away by humans in an iterative process, aimed at making all strategies as balanced as possible [56]. Hunicke proposed a method to balance the challenge of a shooting game by altering supply and demand of game items (e.g., by scaling the frequency spawn of health packs or the damage of opponent weapons) [46]. Hao et al. propose a technique that limits the computation time allotted to Monte Carlo Tree Search (MCTS), increasing or decreasing it, respectively, if the MCTS-controlled agent is faring too poorly or too well [41]. Andrade et al. use Reinforcement Learning to predict the outcome of different actions at a given time and program their AI controller to choose an action higher or lower in the ranking depending on how well their agent is doing against the human player.

In this thesis, we propose a novel technique inspired by the work of Andrade et al. and addresses some of its shortcomings. This research is detailed in Chapter 5.

2.4 Summary

This chapter introduced background information necessary to comprehend the rest of this thesis. First, concepts of Reinforcement Learning were introduced, which are necessary to understand the contributions in Chapters 3 and 4; the concept of

Abstraction in RL was also introduced, which is at the base of the contributions in Chapter 4. Then Monte Carlo Tree Search was presented, which is extensively used in Chapter 5. Finally, Dynamic Difficulty Adjustment was introduced, which is required to better understand Chapters 5 and 6.

Chapter 3

Learning Options from Demonstrations

The options framework, introduced in Section 2.2.1, opens the door to temporal abstraction in Markov Decision Processes. Temporal abstraction allows MDP methods to model the effects of longer courses of action, called "options", which span across several time-steps. While yielding these advantages, options have one significant drawback: it is difficult to define "good" options. Research has contributed several approaches to learning options, either using the structure of the state space [65, 86, 66, 61, 27, 52], by leveraging on the concept of intrinsic motivation [83, 11, 104] or by learning from demonstrations or solutions [72, 116, 54, 88].

In this chapter, we introduce a novel algorithm to learn options from expert demonstrations. In this context, an expert demonstration is a trace of the interaction between the expert and the environment, where by expert we mean any agent who is familiar with the environment. Our approach to learning behaviours from demonstrations is to find "surprising" decisions made by the experts, infer their intentions and use them as goals. This idea reflects studies in cognitive research where infants were observed to get excited by unexpected events and motivated to explore further [85].

3.1 Options construction

The proposed approach produces options whose policy is goal-directed. This is in line with previous research in this area, as discussed in Section 2.2.1. More

specifically, the policy of every option learned by the proposed approach is greedy on a reward function that is zero for every state except goal states.

Formally, the reward function of option o is defined as

$$R_o(s) = \begin{cases} c & \text{if } s = g_o \\ 0 & \text{otherwise} \end{cases}, \tag{3.1}$$

where $c > 0$ is a constant term and g_o is the goal of option o. This reward function induces a value function V_o via the Bellman operator described in Equation 2.6 which, in turn, defines a policy via greedy action selection:

$$\pi_o(s) = \arg\max_a R_o(s) + \gamma \sum_{s'} T(s, a, s') V_o(s'). \tag{3.2}$$

The initiation set for option o is the set of all states g_o^* from which it is possible to reach the goal state, except goal states themselves:

$$\mathcal{I}_o = g_o^* \setminus \{g_o\}. \tag{3.3}$$

The termination probabilities are defined as:

$$\beta_o(s) = \begin{cases} 0.2 & \text{if } s \in \mathcal{I}_o \\ 1 & \text{otherwise,} \end{cases} \tag{3.4}$$

where 0.2 is hand-tuned.

3.2 Identifying Useful Sub-goals

Our procedure analyses expert demonstrations. A demonstration is a sequence of of states and actions that describe the interaction of an expert with the environment. The first step is then to identify useful sub-goals. The algorithm we use assumes that if a sub-goal is useful, at least one of the demonstrations makes good use of it. In other words, the algorithm cannot identify sub-goals that do not appear as such in the demonstrations.

The intuition behind our method to identify sub-goals is to rewrite the demonstrations in terms of intermediate goals and steps taken to reach them. That is, we want to rewrite every step (s_i, a_i) of the demonstration as $(s_i, \pi_i(s_i))$. Such problem

Algorithm 2: Goals extraction; uses a multi-set structure, which is a set where information about the number of occurrences of each element is preserved.

Input : Environment model T
Input : Expert demonstration demo
last_s, _ $\leftarrow$ last(demo)
goals $\leftarrow$ multi-set $\varnothing$
prev_s $\leftarrow$ last_s
cur_goal $\leftarrow$ last_s
for $s, a \in$ *reverse(demo)* **do**
    $\hat{R} \leftarrow$ zero everywhere except in cur_goal
    $\pi \leftarrow$ Value_Iteration$(T, \hat{R})$
    if $\pi(s) \neq a$ **then**
        add prev_s to goals
        cur_goal $\leftarrow$ prev_s
    end
    prev_s $\leftarrow$ s
end
return goals

has a very large number of solutions; to constrain it, we impose that all policies π_i must be goal oriented, as defined in Equation 3.2. We then search for the smallest set $\{\pi_i\}_i$ of such policies that allows the rewriting process.

We propose a greedy algorithm to search for a solution. The algorithm walks through each demonstration backwards, assuming the current goal is the last state of the demonstration. At each step it computes the best action to take to reach what is believed to be the current goal; it then compares such action with the action taken by the expert. If there is a mismatch, the current goal is updated via a simple inference and it is added to the set of recognised goals. Algorithm 2 shows the pseudocode for such procedure. Notice that the procedure expects the model of the environment state transitions T to be given as input: this is necessary to perform Value iteration and compute the policy induced by the goal-oriented reward function.

Algorithm 2 finds an exact solution only in deterministic MDPs. In stochastic MDPs, a state and action pair does not uniquely determine the next state but rather it conditions the probability distribution of the next state. As a consequence, the next state in the demonstration is not necessarily the goal that the expert wanted to achieve. Not knowing the intent of the expert, we assume that the next state is what the expert wished to achieve. This is a heuristic that seems to work well in practice.

3.3 Reduce the Number of Sub-goals

Algorithm 2 outputs a multi-set of goals. A multi-set is a data structure similar to a normal set, but where information about the number of occurrences is preserved. These are all the goals necessary to equivalently rewrite all the demonstrations as discussed above. However, the output will likely include a large quantity of goals, and it is undesirable to add as many options to the MDP so as not to slow down learning too much.

Some of the goals detected by the algorithm may be similar to each other, and it therefore makes sense to aggregate them. We propose two approaches to this end. The first method uses a distance measure and, using graph theory and clustering, aggregates states that are similar in terms of the state transitions graph structure. The second method aggregates goals by abstracting some state features, assuming the MDP is factored.

3.3.1 Graph-based Clustering

The first approach we propose to perform goals aggregation is based on graph theory. In particular, aggregation is based on the state transitions graph. The similarity concept that we base our approach is captured by the likelihood of transitioning from one state to another state, assuming the best action to this end is chosen. In particular, we define distance such as two states are more distant the less likely it is to transition from one state to the other. We then define a distance measure between states s_i and s_j as follows:

$$\Delta(s_i, s_j) = \min_{a \in \mathcal{A}} \frac{1}{T(s_i, a, s_j)},$$

where $T(s_i, a, s_j)$ is the probability of transitioning from state s_i to state s_j when executing action a.

This distance measure is then used to build a probability matrix. We use the *all-pairs shortest path* algorithm to build the probability matrix [35]. This algorithm has a complexity $O(|S|^3)$, however, the idea is that this algorithm is run only once in a precomputation phase and, thus, does not affect learning time.

The distance matrix is fed to the *DBSCAN* clustering algorithm [34]. DBSCAN has been chosen because it does not require a specification of the number of clusters, unlike many other clustering algorithms. The clusters are sorted by size and a random representative from the top k clusters is selected, where k is arbitrary. The

so-chosen representatives form the set of learned sub-goals, and can then be enriched with an initiation set, a policy and a distribution of termination probabilities and so be transformed in options. Notice that, while k is known in advance, DBSCAN allows us not to bias the number of clusters; we then sample k of the clusters found by DBSCAN.

3.3.2 Features-abstraction Aggregation

One of the main drawbacks of the method described above is its computational complexity. Therefore, we propose a second method that is based on state abstraction. This method assumes the MDP is factored; that is, states consist of values assumed by a fixed set of features. In this case, aggregation can be performed by ignoring some features and considering as the same "state" all states which have the same values on the features that are not ignored.

This greatly reduces the number of goal states. Furthermore aggregated states can become more meaningful if ignored features happen to carry information that is only relevant for action decision but not to describe a situation in which the agent wishes to be (i.e., a goal). For example, consider an agent driving a car: while it is certainly useful to know if another car is coming from the left, the same information is an over-specification when describing the goal of "being at the destination".

In order to allow for options to use multiple goals, we need to slightly alter the definitions in Equations 3.1 and 3.3 as follows:

$$R_o(s) = \begin{cases} c & \text{if } s \in G_o \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

and

$$\mathcal{I}_o = G_o^* \setminus G_o, \tag{3.6}$$

where G_o is the set of goals of option o and G_o^* is the set of all states from which it is possible to reach any goal.

The aggregated goals are sorted by the number any of their internal states has been detected as a goal by the algorithm. Afterwards, the top k goals are selected, where k is arbitrary.

3.4 Experiments

We tested the proposed method for learning options from demonstration in two settings that are popular in Reinforcement Learning research. The first, simpler setting is a grid world, where every state is simply the agent location in the world. The second, more challenging setting is the popular arcade video game Pac-Man.

The testing environments have been implemented in Python3 using the Numpy library[1] [106] and the Scikit Learn toolkit[2] [71]. We used GNU Parallel[3] to parallelize the execution of experiments [99]

This section details the experimental setup of both settings and presents the results of the experiments.

3.4.1 Grid-world experimental setup

In the first experimental setup, we tested the Q-learning algorithm in a grid-world. A grid-world is an environment where a state is described only by a location in the world; the agent navigates such environment by choosing to move in one of the cardinal directions. The shape of such environment is very similar to that used in [93], where a square 13×13 world is divided in 4 areas by means of barriers. Figure 3.1 shows a representation of such environment. Barriers are interrupted to allow traveling from one "room" to another; these holes are called "hallways". The environment is non-deterministic: the probability of transitioning in the desired direction is $\frac{2}{3}$; the agent moves in one of the other cardinal directions with probability of $\frac{1}{3}$, that is $\frac{1}{9}$ for each of them.

Flavors

We used the grid-world topology described above in two different flavors.

- The black states separating the four areas are walls, meaning they are impenetrable. This is the same setting used in [93].
- The black states separating the four areas are ponds, which means they are not impassable, but have a low reward. The rationale behind this choice is to make the hallways part of the paths *chosen* by the experts rather than *unavoidable* steps. This is because our algorithm detects as subgoals only states that the

[1] `http://www.numpy.org/`

[2] `http://scikit-learn.org/`

[3] `http://www.gnu.org/software/parallel/`

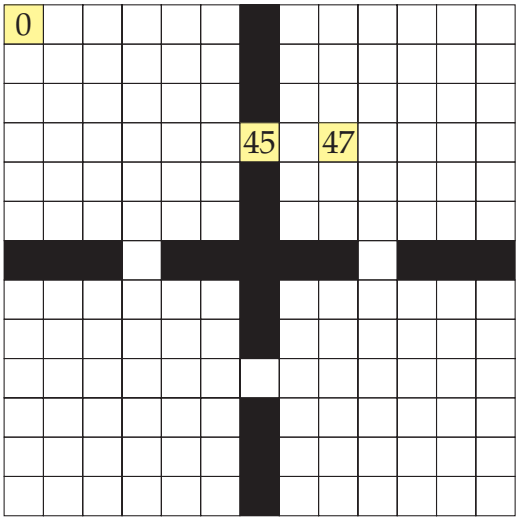


Figure 3.1 The grid-world used in the experiments. The yellow squares represent the destinations used in the different experiments: the one in the upper corner is labeled as [0], the one in the hallway is labeled [45] and the other one is labeled [47]. These labels will be used to present the results of the experiments. The black squares represent unreachable states in the wall setting and low rewards states in the ponds setting.

experts explicitly chose among the others - as opposed to states that were simply not avoidable.

Specifically, while normal states have a reward of 0, ponds have a reward of -10. The final state has reward 1000. When the final state is reached, the execution terminates.

The key difference between the two flavors lies in where the obstacles are encoded. In the "walls" case, the unreachability of the blocks is encoded in the reward function, which our algorithm knows. As a consequence, the expert decision of passing through the hallways is not surprising. In the "ponds" scenario, the penalty of these states is encoded in the reward function, which is unknown by our algorithm. As a consequence, the decision of the expert of passing through a hallway rather than directly over a wall is unpredictable and therefore picked up by our algorithm.

In each of the flavors, different sets of experiments are run: each set of experiments uses a different destination for the agent. The states we chose as destinations are shown in Figure 3.1. We selected these destinations for specific reasons: one of them is in a corner, away from most common paths; another one is in a hallway, which is one of the hand-crafted options in [93]; the third one is close to a hallway and, as such, is part of many common paths.

By choosing different destinations, it is possible to compare the performance of agents using different sets of options. If the options are close to the destination, the agent is advantaged because when it chooses a random, exploratory action, it is more likely that it will select an option the will lead it close to the destination. On the other hand, if the options are all far from the destination, they will put the agent at disadvantage.

Q-learning details

The value of the discount factor of the MDP was set to $\gamma = 0.97$. The options are generated according to Equation 3.1 using $c = 50$. The exploration strategy used in the experiments is ϵ-greedy, with $\epsilon = 0.1$.

Baseline

The purpose of the experiments is to compare the quality of the hand-crafted options used in [93], aiming at the hallways, denoted by $\mathcal{H}$, with that of options learned from demonstrations by our algorithm, denoted $\mathcal{L}$. These sets of options are also compared against the set of primitive actions only, denoted $\mathcal{A}$. Sets $\mathcal{A} \cup \mathcal{H}$ and $\mathcal{A} \cup \mathcal{L}$ are also tested,

We selected the handcrafted options as the baseline for comparison, rather than a random selection of states, because the former are supposed to perform better. This is because, due to the structure of the environment, a hallway state is for sure in any path that travels between two different rooms. Given this, their contribution to the Q-learning algorithm is expected to be more useful than that of other random locations [93].

Goals number reduction

We reduced the number of detected goals using the graph-based technique described in Section 3.3.1. We then selected a representative from each of the top 4 clusters of goals. The number 4 is selected to provide a fair comparison with the baseline, which uses 4 options.

Experimental procedure

Each of the 5 setups ($\mathcal{A}$, $\mathcal{H}$, $\mathcal{L}$, $\mathcal{A}\cup\mathcal{H}$ and $\mathcal{A}\cup\mathcal{L}$) was repeated 200 times, each allowing Q-learning to run for 3000 episodes. For the setups including learned options $\mathcal{L}$, a new set of demonstrations was generated at every repetition as follows:

- a state was randomly selected as a goal;
- a reward function was generated as in Equation 3.1;
- Value Iteration was used to compute the optimal value function;
- the optimal policy was computed from the optimal value function;
- a random initial state was selected;
- the optimal policy was followed until the goal state was reached;
- every step of this interaction was recorded and made into a demonstration.

3.4.2 Grid-world experiments results

In this section we present the results of the experiments performed in the grid-world. We will often refer to "bad" options when referring to options that lead to an area in the state space that is far away from the destination state the agent is pursuing.

Figures 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7 show the result of the experiments that have been run. The X axis is the number of episodes, while the Y axis is the (average) number of steps per episode.

The plots show the average performance on 200 experiments run over 3000 episodes. Experiments have been repeated 200 times because the performance showed a high variance, which is due to the dependence on the random initial state. For each of environment, options set, destination and repetition, 4 demonstrations were generated and were used to learn options. All of the experiments have been run on both the "pond" and "wall" settings and on all of the destinations, [0], [45], [47] (Figure 3.1 visually shows where such states are located in the environment. Results are presented using a sliding window with size of 20 to average values and hence smooth the curves.

In the following, we will often mention that an option is close/far from a destination: this is actually a short-hand to say that the subgoal which is captured by an option is close/far from the destination. For the sake of conciseness, we will stick to the former, shorter version.

Top-left corner as destination

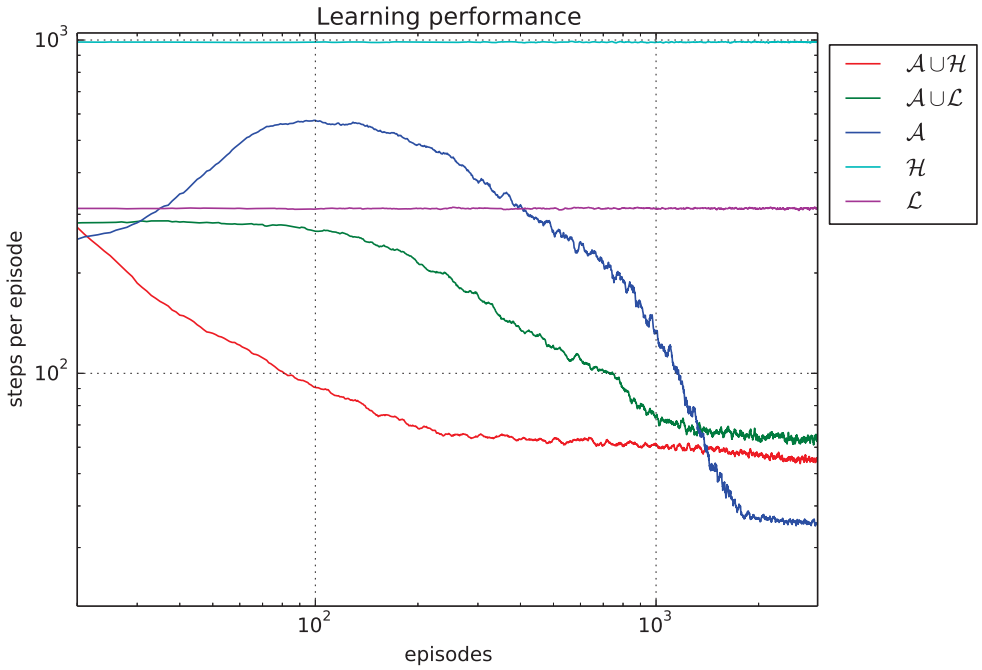


Figure 3.2 Results of experiments in the "pond" settings with destination as the top-left corner (state [0]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options.

Figures 3.2 and 3.3 show the results of the experiments where the agent's destination is the top-left corner, labeled as [0] in Figure 3.1.

In the pond setting, in Figure 3.2, the agents using primitive actions and hand-crafted options ($\mathcal{A} \cup \mathcal{H}$) or actions and learned options ($\mathcal{A} \cup \mathcal{L}$) both converge, but the former performs consistently better than the latter. The agent using primitive actions only ($\mathcal{A}$) performs worse at the beginning becomes the best performer at the end of the process. We interpret this difference as an effect of the ϵ-greedy strategy: when an agent randomly chooses a bad option, it ends up further away than it would by just choosing a bad action. Agents using only options, either hand-crafted ($\mathcal{H}$) or learned ($\mathcal{L}$) performs very poorly, because the distance of the options from the destination makes it unlikely that the destination is ever reached.

It is worth analysing the particular performance curve of the agent using only primitive actions, which decreases, then increases and finally decreases again in all the pond settings (figures 3.2, 3.4, 3.6). At first, this may seem strange, because

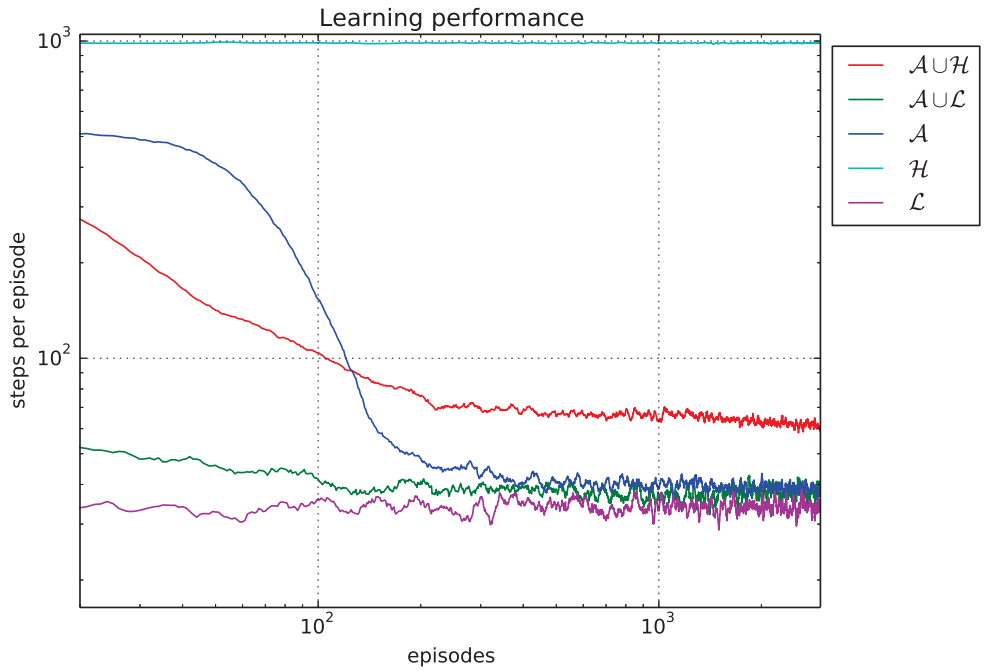


Figure 3.3 Results of experiments in the "wall" settings with destination as the top-left corner (state [0]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options.

Q-learning performance, on average, improves over time [92]. In this plot, however, the curve represents the number of steps taken for each episode rather than the cumulative reward, which is the metric that has the property of decreasing on average in Q-learning. Let us analyse what causes this particular performance curve. At first, the agent knows nothing about the environment and even about its destination; in fact, it will only learn about what the destination is when it accidentally walks on it. In this situation, it makes sense that the number of steps is high because the agent aimlessly wanders around the environment. Over time, the agent finds what the destination is and its wandering decreases as it learns the environment topology. When the agent figures out that walking through the pond is expensive, the number of steps increases again because the agent needs to figure out ways to get to the destination that do not pass through the pond. In the end, when the agent has good knowledge of the environment, the number of steps is minimised.

In the wall setting, in Figure 3.3, the agents using learned options, either with primitive actions ($\mathcal{A} \cup \mathcal{L}$) or without ($\mathcal{L}$) both outperform the agents using hand-

crafted options and primitive actions ($\mathcal{A} \cup \mathcal{H}$) or primitive actions only ($\mathcal{A}$). This may seem surprising because the wall setting should not allow the algorithm to produce meaningful options. This is actually the case, but with the twist that, often, the only option that is learned is the destination itself. This biases the exploration directly to the destination, therefore showing the agent the correct path very early on.

Top hallway as destination

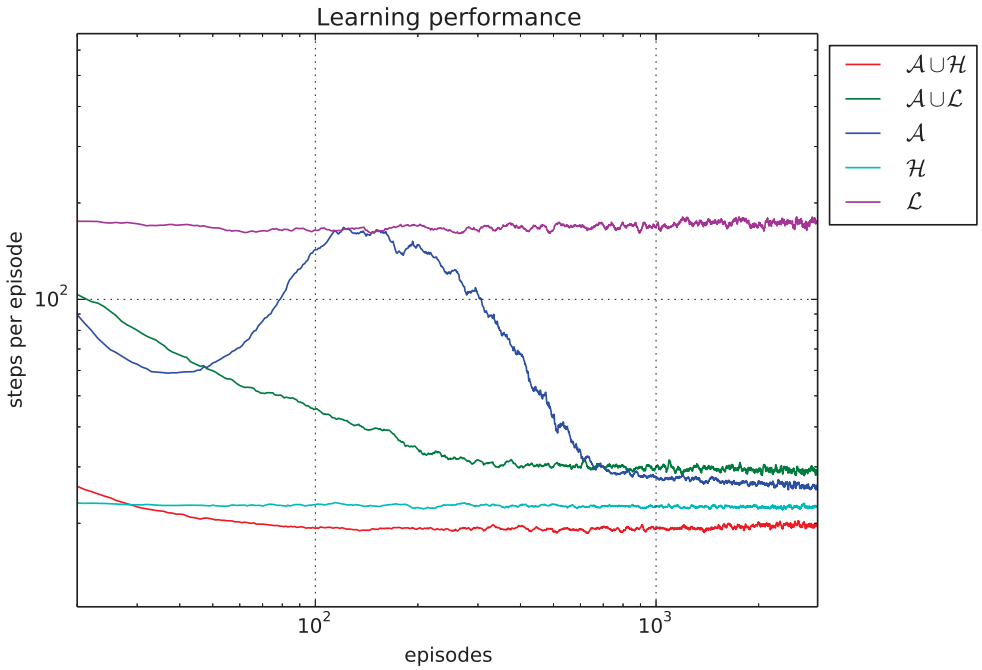


Figure 3.4 Results of experiments in the "pond" settings with destination as the top hallway (state [45]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options.

Figures 3.4 and 3.5 show the results of the experiments where the agent's destination is the top hallway, labeled as [45] in Figure 3.1.

In the pond setting, in Figure 3.4, the best performers are clearly the agents using hand-crafted options, either with primitive actions ($\mathcal{A} \cup \mathcal{H}$) or without ($\mathcal{H}$). This is because the destination is exactly one of the options, making it very easy for the agent to find a good path to the destination. The difference in performance of these two agents is an effect of the ϵ-greedy strategy: when a bad option is chosen, the agent is taken further away from the destination than it is when choosing a

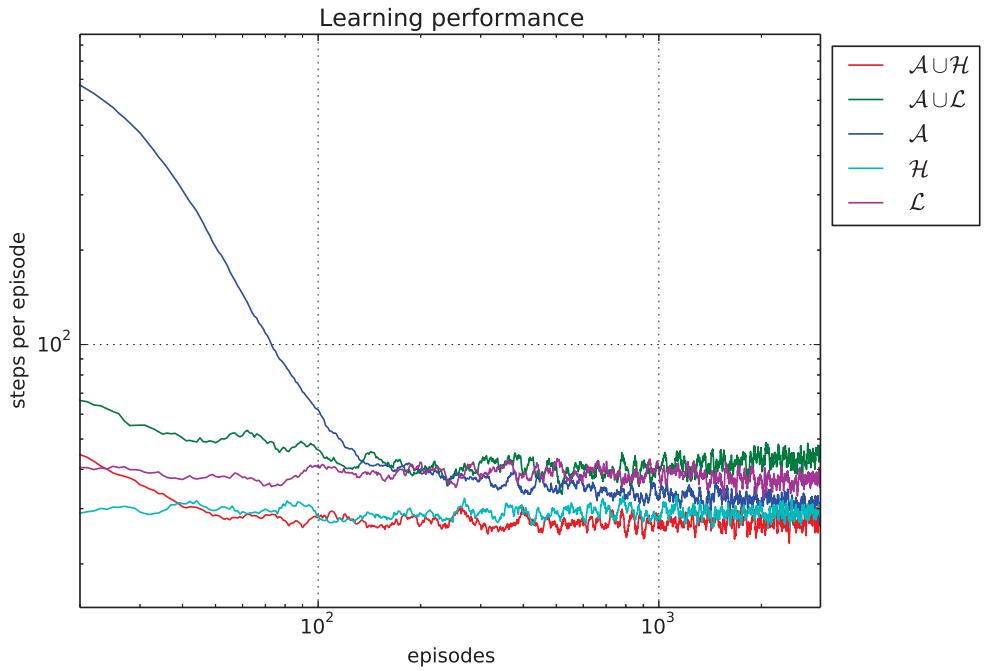


Figure 3.5 Results of experiments in the "wall" settings with destination as the top hallway (state [45]). $\mathcal{A}$ denotes the set of primitive actions (cardinal directions), $\mathcal{H}$ denotes handcrafted options (the hallways), and $\mathcal{L}$ denotes the set of learned options.

bad action. The agent using only primitive actions ($\mathcal{A}$) also performs well after an initial exploration phase. The agent using learned options and primitive actions ($\mathcal{A} \cup \mathcal{L}$) converges performs better earlier on, but is later surpassed by the agent using only primitive actions ($\mathcal{A}$). This is, once again, due to the ϵ-greedy strategy. Finally, the agent using only learned options ($\mathcal{L}$) performs the worst, showing that our algorithm did not learn the destination as one of the options. This is because movement towards the top hallway are not surprising given it is the destination.

In the wall setting, in Figure 3.5, the agents using hand-crafted options, either with primitive actions ($\mathcal{A} \cup \mathcal{H}$) or without ($\mathcal{H}$) both perform well from the beginning, once more, because the destination is one of the subgoals. In this case, also sets $\mathcal{L}$ and $\mathcal{A} \cup \mathcal{L}$ perform well, again because the topology makes it so that, most of the times, the only learned option is the destination. Also set $\mathcal{A}$ performs well, even though it takes longer to converge.

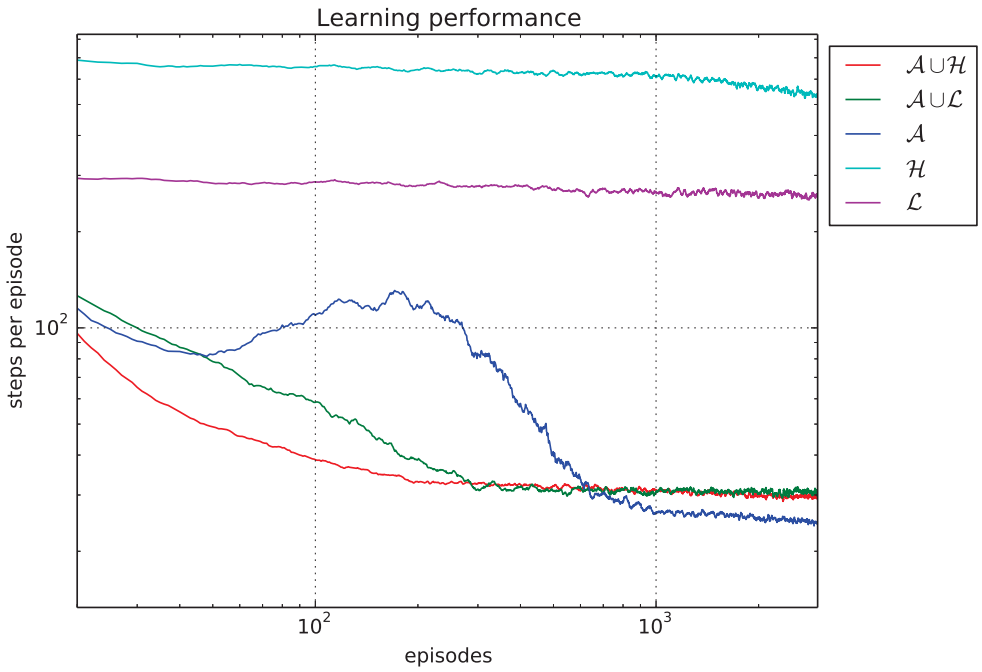


Figure 3.6 This figure shows the results obtained in the "pond" setting with destination [47]. For details, see Figure 3.2.

2 steps away from top hallway as destination

Figures 3.6 and 3.7 show the results of the experiments where the agent's destination is the state on the left of the hallway, labeled as [47] in Figure 3.1.

In the pond setting, in Figure 3.6, the agents using primitive actions with or without any set of options ($\mathcal{A}\cup\mathcal{L}$, $\mathcal{A}\cup\mathcal{H}$ and $\mathcal{A}$) perform very well. Agents using only options ($\mathcal{H}$ and $\mathcal{L}$) perform unexpectedly poorly. We hypothesize that, due to the low rewards received, the algorithm does not have sufficient information to tell which of the option is the best one to choose since they all appear equally bad.

In the wall setting, in Figure 3.7, the performance of the agents using learned options, either with primitive actions ($\mathcal{A}\cup\mathcal{L}$) or without ($\mathcal{L}$) are about the same as the pond setting. The agent using primitive actions only ($\mathcal{A}$) reaches convergence more quickly and the agent using hand-crafted options and primitive actions ($\mathcal{A}\cup\mathcal{H}$) performs better in general. The agent using hand-crafted options only ($\mathcal{H}$) also perform better than in the pond setting; however, since the destination is only close to the options but not coincident, reaching it takes some luck (i.e. time).

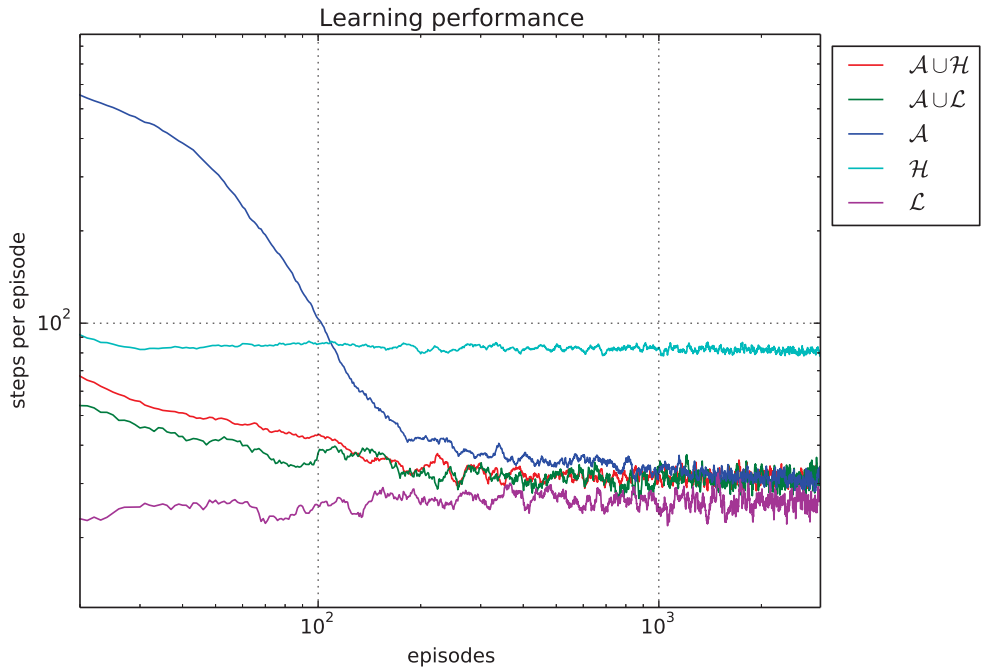


Figure 3.7 This figure shows the results obtained in the "wall" setting with destination [47]. For details, see Figure 3.2.

When used along with primitive actions, learned options perform about as well as hand-crafted options, with a slight advantage in most cases. Furthermore, it can be noticed that the bias introduced by the options in the random exploration provides a speed-up in the learning process, letting the agents reach a stable performance after only a few tens of episodes. This is one of the cases in which the introduction of a bias (as mentioned in [50]) is benefiting the learning process; since it is derived by the behavior of experts, the exploration tends to follow the footsteps of the experts.

3.4.3 Pac-Man experimental setup

In this section we describe the second experimental setup and associated results used to test the ideas proposed in this chapter. The testing environment used here is the famous arcade video game Pac-Man. The experiments are designed to show the benefits of using options learned from demonstration compared to standard Q-learning in controlling a Pac-Man agent. Notice that the purpose of this work is not to implement the best Pac-Man agent but, rather, to show that using options learned from experts gives learning agents an advantage. These are two different problems: engineering Pac-Man agents to achieve the best performance versus autonomously

learning how to effectively play. We choose to focus on the second problem, and use Pac-Man as a test-bed.

Q-learning details

To test our algorirthm, we used Q-learning enriched with some options learned from our algorithm. Here, we specify the parameters used in Q-learning, as well the specific exploration strategy associated. The discount factor of the MDP is set $\gamma = 0.97$ (a commonly chosen value [92]), while the reward peak for goal-driven policies was arbitrarily set to 50. The exploration strategy used in the experiments is Annealing ϵ-greedy, with $\epsilon = 0.1$ (also, a common choice [92]). The annealing schedule is based on the episode number k and is defined as $\epsilon(k) = 0.1/(1 + e^{\frac{3k}{1000}-3})$. These parameters have been chosen to create a sigmoid function where most of the descent happens between episodes 500 and 1500. Figure 3.8 shows the function described above.

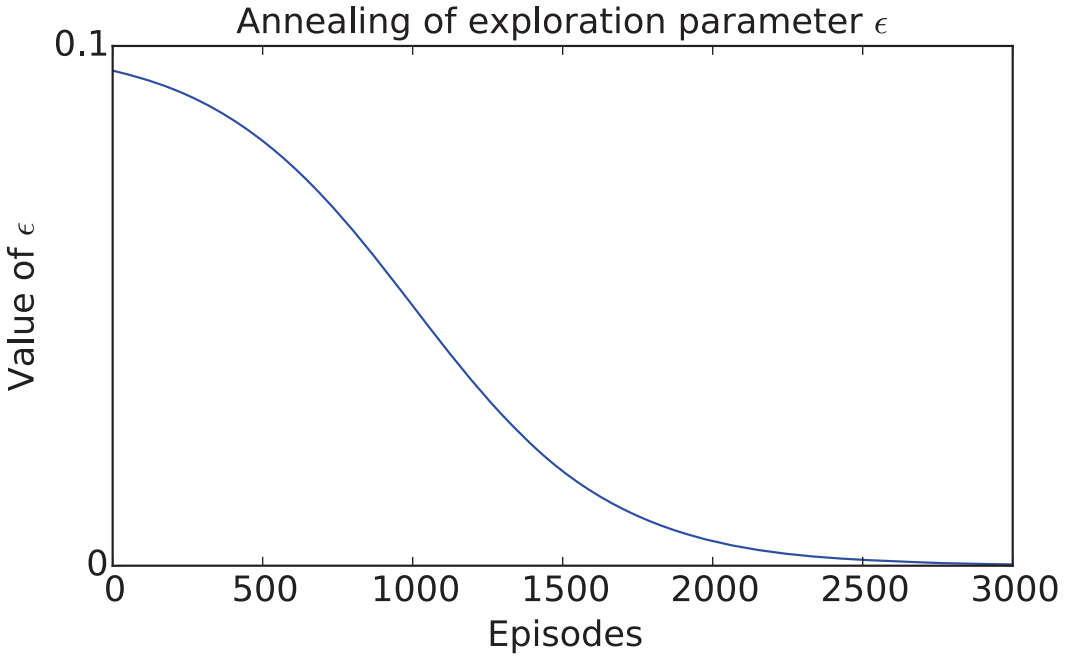


Figure 3.8 Annealing schedule for exploration parameter ϵ.

Pac-Man

The video game *Pac-Man*[4] is an arcade game that was popular in the 1980s.

In this game, the player controls Pac-Man, an agent moving in a two-dimensional environment whose purpose is to collect all the white pills. Pac-Man is chased by a number of ghosts, each of which will kill Pac-Man if he collides with them. There are also a number of special capsules that make all the ghosts weak for a limited period of time. If Pac-Man collides with a weak ghost, the ghost dies and reappears at the center of the game area in a non-weak state. The level used in the experiments is shown in figure 3.9.

We used the implementation of Pac-Man developed by University of California, Berkeley, for their Artificial Intelligence course[5].

A score is given to players depending on their performance:

- -1 at every time step;
- +10 for every eaten pill;
- +200 for every killed ghost;
- +500 upon victory (eating all pills);
- -500 upon defeat (being eaten by a ghost).

State space

The state space in Pac-Man is very large, including information about whether or not every single pill and food pellet has been eaten, the position and state of each ghost as well as the position of Pac-Man. This all induces a combinatorial state space that is un-treatable in standard Q-learning.

In order to reduce it, we extracted a set of features and used them as the MDP state features. The features we adopted are the following:

- **direction** of the closest pill;
- **direction** and **distance** of the closest non-weak ghost;
- **number** of capsules in the level;

[4]Additional information can be found at `http://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php?print=1` (checked on March 3rd, 2016).

[5]Available at `http://ai.berkeley.edu` (checked on March 3rd, 2016).

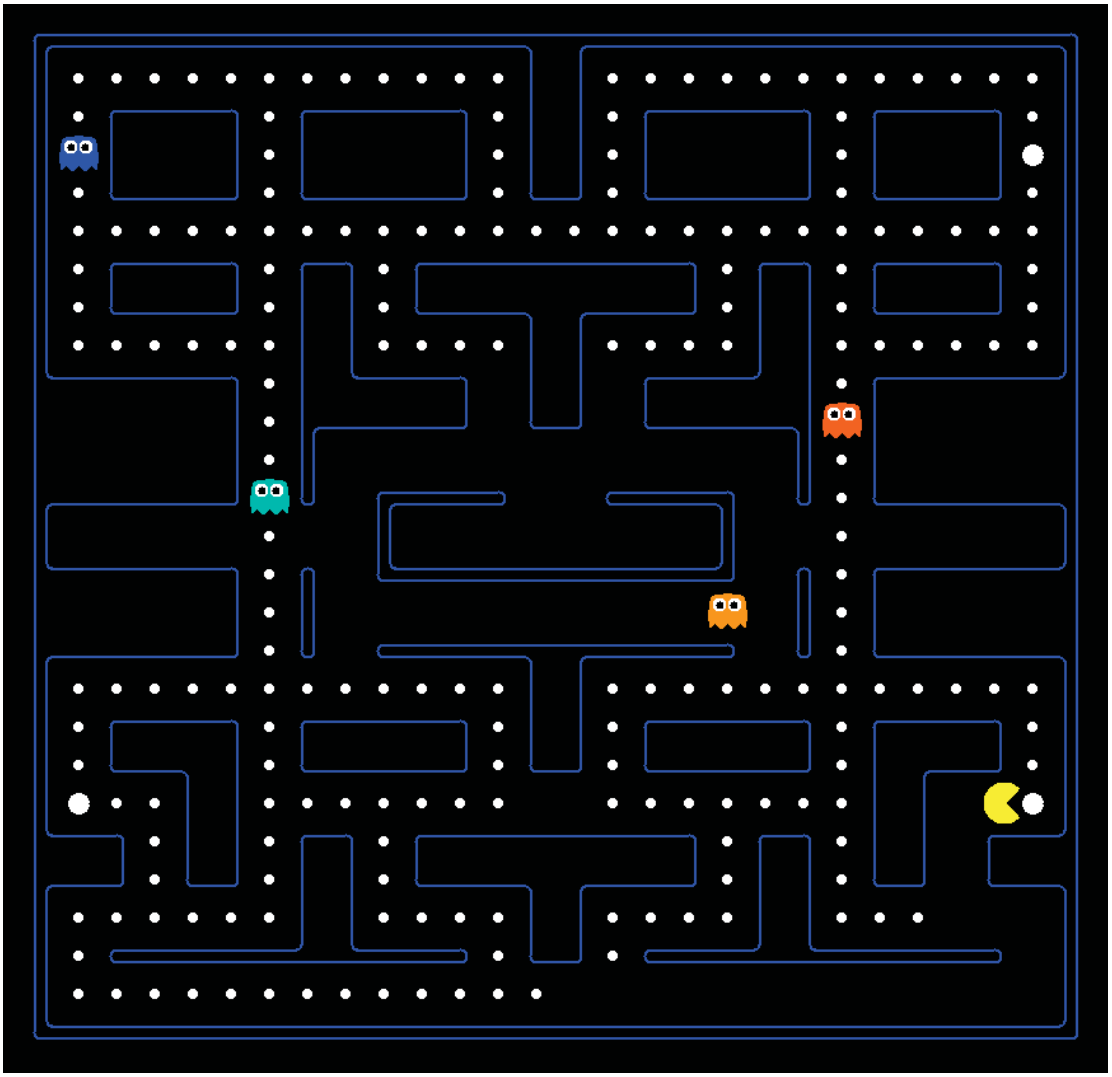


Figure 3.9 The video game used in the experiments, Pac-Man. Pac-Man, the yellow entity, is about to eat a capsule; the smaller white dots are pills and the coloured entities with eyes are ghosts.

- **number** of weak ghosts and the **direction** of the closest.

Here, distance information can assume values *close*, *midrange*, *far* and *very-far*, depending on the length of the shortest path as computed by the Dijkstra algorithm [23]. Directional information represents the direction that Pac-Man should follow to reach the object via the shortest path.

Automatic agents

The experiments compare the goals detected by the algorithm from two different automatic agents. That is, these automatic agents are the experts in our setting. In this section we present the mechanisms driving such automatic agents' behaviour.

Both automatic agents include a perfect model of the game and both compute a heuristic value for the state that would hypothetically follow each available action, and select the action that would yield the highest score. Both agents compute the score taking into account information about non-weak ghosts and pills according to the following formula:

$$score_c = \begin{cases} -\max(0, 7-g)^2 - \frac{p}{100} & \text{if } p > 0 \\ -\max(0, 7-g)^2 + 1000 & \text{otherwise} \end{cases}, \tag{3.7}$$

where g is the length of the shortest-path to the nearest non-weak ghost and p is the length of the shortest-path to the nearest pill.

Conservative agent

The first agent, which we call the *conservative* agent, uses Equation 3.7 to evaluate the states value. This formula encourages reaching the closest pill while staying 7 tiles away from the closest non-weak ghosts. The agent, thus, avoids taking risks by staying at a safe distance from threats. While this does not take into account the possibility of ambushes, it is effective in practice.

Aggressive agent

Meanwhile, the second agent, which we call *aggressive*, takes into account the same information as the conservative as well as that regarding weak ghosts and capsules. The formula it uses is as follows:

$$score_a = score_c - 1000 \cdot c - 100 \cdot n - 10 \cdot w, \tag{3.8}$$

where c is the number of capsules present on the level, n is the number of weak ghosts and w is the length of the shortest-path to the nearest weak ghost. The formula in Equation 3.8 encourages eating capsules and weak ghosts. The aggressive agent will therefore tend to earn more points. However, this behaviour is risky because it does not take into account for how long the ghosts are going to remain weak.

Finally, Ghosts are driven by a stochastic policy. A random action is chosen at every junction and repeated until another junction is encountered.

Goal number reduction

We reduced the number of detected goals using the features abstraction technique described in Section 3.3.2. We then select the top 4 clusters of goals and create a multi-goal option for each of them.

The features which we used to aggregate goals, are those that do not carry directional information. This choice has been driven by the need to address redundancy in the obtained goals and also by the uselessness of directional information. Since directional information is descriptive of specific details rather than high-level ideas, it is less useful when used to characterise a goal.

Experimental procedure

The procedure followed for the experiments with each agent $A \in \{\text{cons}, \text{aggr}\}$ (Section 3.4.3) is as follows:

1. A set D_A of 1000 Pac-Man games is recorded.
2. A model of the environment T_A is learned analysing games in D_A.
3. Algorithm 2 is used to extract goals from games (used as demonstrations):

   $G_A = \biguplus_{d \in D_A} \text{EXTRACT\_GOALS}(T_A, d)$, where $\uplus$ indicates a multi-set sum. A multi-set is a set where information about the number of occurrences of each element is preserved. Notice that Algorithm 2 returns a multi-set.
4. Aggregate states that have equal values for the heuristically selected features described above; let F be the set of such features:

   $G_{A,v} = \{g_i \mid \Pi_F(g_i) = v\}$, for all v, where Π_X indicates the relational algebra projection operator, which strips the input of all features except those in X. Notice that each $G_{A,v}$ is a multi-set, as opposed to a set.

5. Rank all multi-sets $G_{A,v}$ based on their cardinality in descending order.
6. For each of the first top 4 ranked multi-sets $\overline{G}$ of goals:
   (a) create a reward function that is zero everywhere except for goals in $\overline{G}$;
   (b) run Value Iteration on reward $\overline{R}$ and model T_A to find a policy $\overline{\pi}$;
   (c) compute the set $\overline{G}^*$ of all states from which it is possible to reach any state in $\overline{G}$;
   (d) create an option $o = (\mathcal{I}, \pi, \beta)$ as follows:
   - $\mathcal{I} = \overline{G}^* \setminus \overline{G}$;
   - $\pi = \overline{\pi}$;
   - $\beta(s) = \begin{cases} 0.2 & \text{if } s \in \overline{G}^* \setminus \overline{G} \\ 1 & \text{otherwise} \end{cases}$.

   (e) add o to $\mathcal{O}_A$ (remember $A \in \{\text{cons}, \text{aggr}\}$)

After computing the options, Q-learning was run 200 times for 3000 episodes (enough for the learning curve to qualitatively settle) with agents using one of $\mathcal{A}$, $\mathcal{A} \cup \mathcal{O}_{\text{aggr}}$ or $\mathcal{A} \cup \mathcal{O}_{\text{cons}}$ as the options set.

Notice that in this set of experiments, unlike in the grid-world ones, the model is learned directly from the expert demonstrations, as opposed to being given in input.

3.5 Pac-Man experiments results

In this section we report the results of our experiments. First, we quantitatively analyze the data, and then we discuss the top detected goals and the performance of Q-learning with and without options learned with our algorithm.

Quantitative analysis of Q-learning performance

The experiments showed that using options learned by the proposed algorithm leads to better performance when compared to not using options. To quantitatively measure the performance of the different agents, we computed the average reward per episode for each of the 200 runs of each of the 3 agents. This operation induces a probability distribution for each agent, all of which appear to be approximately normally distributed, as shown in Figure 3.11. Table 3.1 shows the means and standard errors of the data. Two two-samples t-tests (with unequal variance) have

	Cons. Options ■	Aggr. Options ■	No Options ■
Mean	1560.33	1513.39	1431.65
Std. err	109.72	58.84	120.70

Table 3.1 Mean and standard error of the average reward per episode across the 200 runs of the experiments

(a) Conservative agent

#C	#WG	ΔG	#Det.
3	4	-	177636
4	0	v. far	157646
2	4	-	155110
1	4	-	144402

(b) Aggressive agent

#C	#WG	ΔG	#Det.
4	0	v. far	156254
1	0	far	125638
2	0	far	120047
3	4	-	117520

Table 3.2 Most often extracted goals. #C is the number of capsules, #WG is the number of weak ghost, ΔG is the distance of the closest non-weak ghosts and #Det. is the number of detections.

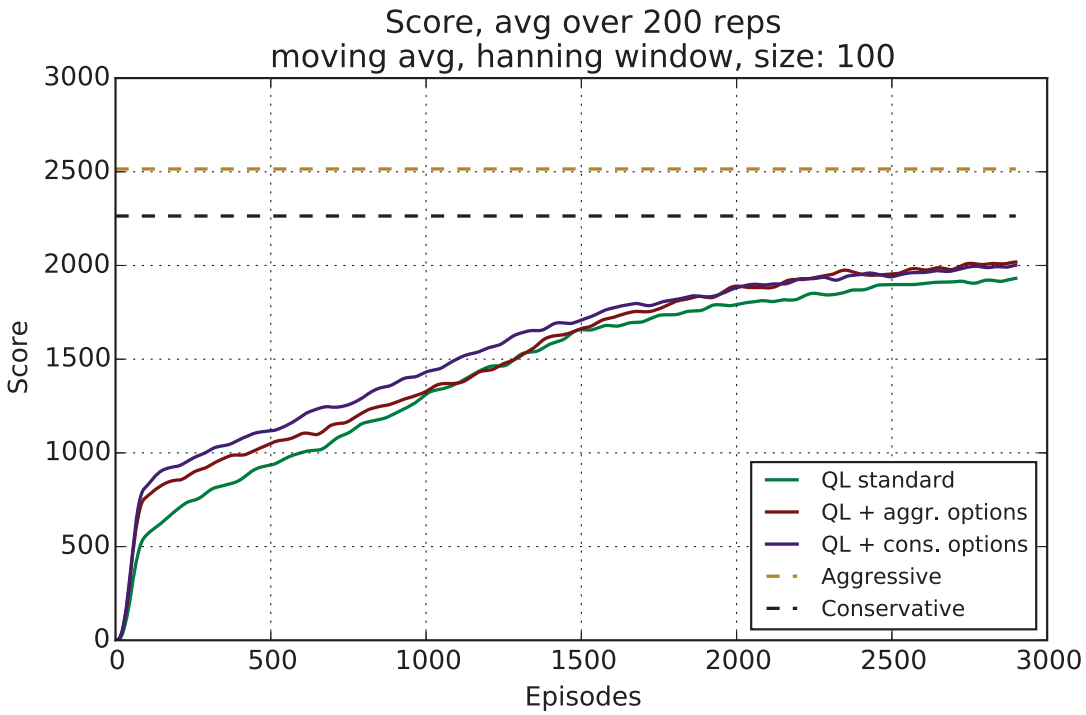
been executed to pairwise compare each of the agents using options with the agent not using options. Both t-tests determined that the distributions mean are different with p-value < 0.001, therefore confirming that both agents using learned options perform significantly better than the agent that does not use options.

Extracted goals

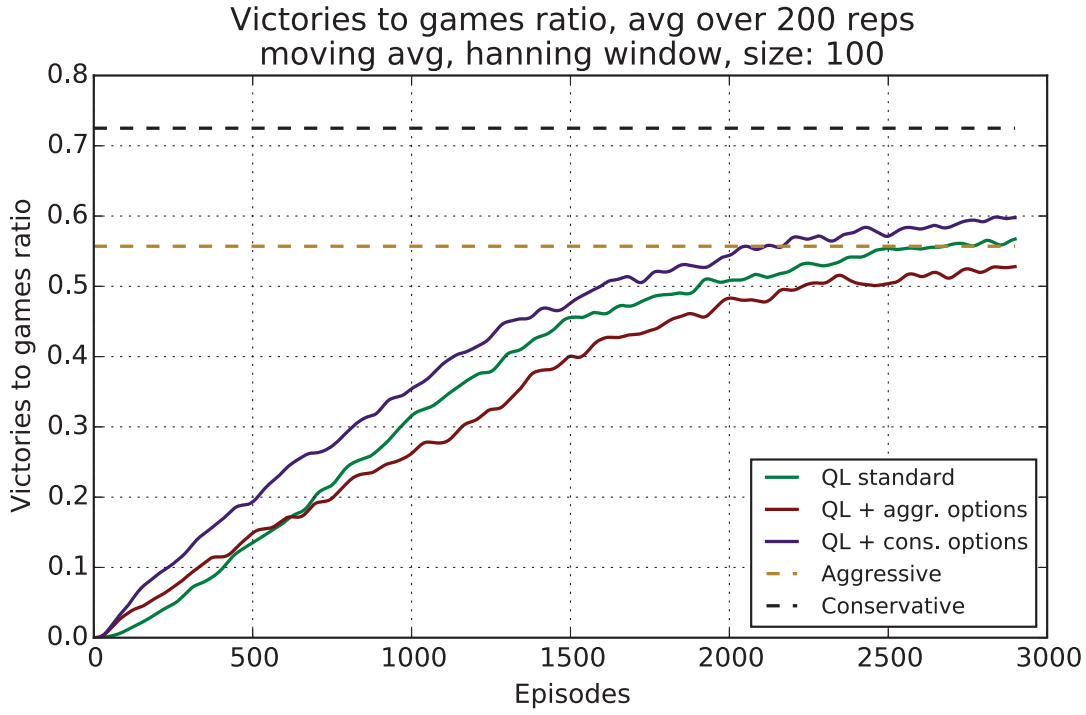
Analysing the extracted goals can give useful insights into how the proposed algorithm works. Table 3.2 shows the top 4 goals (those selected to become options) detected for both agents.

Table 3.2 reveals that the conservative agent has a preference for leaving weak ghosts alone. In fact, among the top goals, most of them have weak ghosts and they are all *alive*. The only state where no ghost is weak shows another priority of the agent: that of staying as far away as possible from ghosts.

On the other hand, the aggressive agent shows a strong preference for eating weak ghosts. Reaching this conclusion from the table is not intuitive because our approach is based on detection of goal states rather than actions. The table reports, as goals, the states following the action of eating. On the other hand, a human would intuitively consider the action of eating itself as a goal. While this is not possible to detect with our algorithm, it is a direction for future research to narrow the gap between human reasoning and the algorithm output.



(a) Score against episode number.



(b) Win/loss ratio against episode number.

Figure 3.10 Agents performance

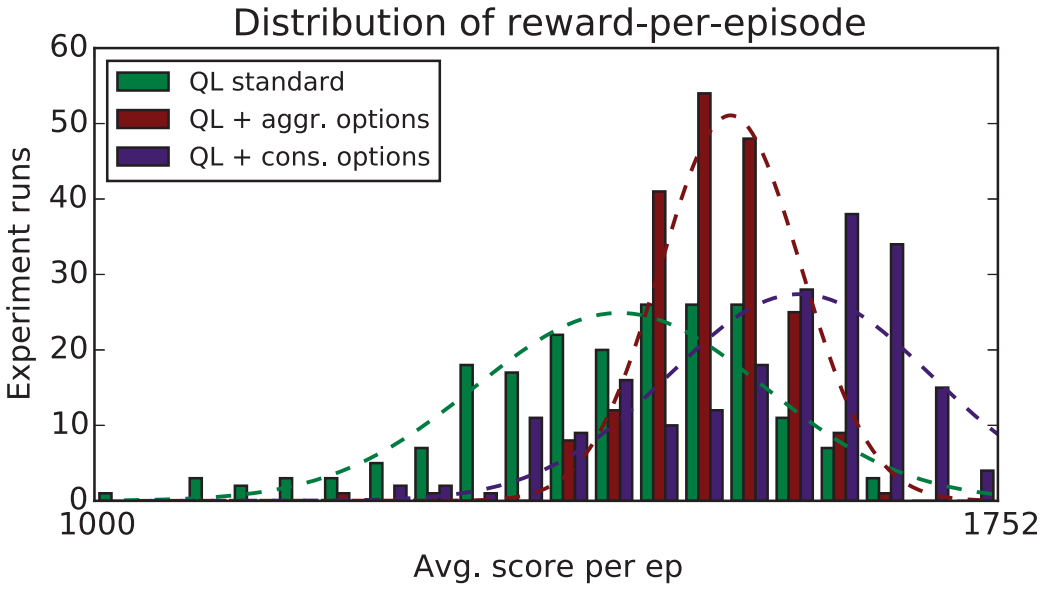


Figure 3.11 Histograms of the average reward per episode.

Q-learning performance

Performance of the agents is shown in Figures 3.10a and 3.10b, respectively in terms of cumulative reward and of average outcome. The two figures show qualitatively similar curves, which confirms that the score measure adopted is closely related to games outcome. The most notable difference is in the performance of the agent using aggressive options which, even while scoring similarly to the other agents, has a comparatively lower victories/games ratio. This is due to the nature of its algorithm: this agent pursues and eats weak ghosts, thus scoring well; however, getting close to weak ghosts is dangerous since they may switch back to a non-weak state, making Pac-Man an easy prey.

The fact that the automatic conservative agent still outperforms all the learning agents can be partially explained by the better granularity that it uses to observe the features. In fact, Equation 3.7 uses the raw value of the distance features rather than the aggregated one used by the learning agents.

The advantage of the agents using options, with respect to the agent not using them, can be explained by the bias introduced by options in the exploration phase; this is in line with the opinion expressed by Jong et al. [50]. These results provide supporting evidence that such bias can be positively harnessed when options are learned by experts using our algorithm. All the agents, ultimately, reach similar

performance. However, the bias introduced by the options brings better results in the early stages of the exploration.

Figures 3.12 and 3.13 show, respectively, the number of distinct explored states and state-action pairs since the beginning of the experiment up to every episode. It can be seen that the agent using aggressive options has explored the most, while being the worst performer (as showed in figures 3.10). This suggests that the difference in performance is not attributable to the *quantity* of explored states or state-action pairs but, rather, to their *quality*. We argue that such quality is biased towards *good quality* thanks to the experience extracted from expert demonstrations; here, the term "good quality" is used to indicate better performance achievement.

It is worth noting that the number of explored states is well below the number of possible states, which is given by:

$$\overbrace{5}^{\substack{\text{4 cardinal}\\\text{directions}}}_{\substack{\text{Dir.}\\\text{food}}} \cdot \overbrace{5}^{\substack{\text{4 cardinal}\\\text{directions}\\\text{+ None}}}_{\substack{\text{Dir.}\\\text{ghost}}} \cdot \overbrace{5}^{\substack{\text{1 of 4}\\\text{values}\\\text{(see Sec.}\\\text{3.4.3)}}}_{\substack{\text{Dist.}\\\text{ghost}}} \cdot \overbrace{5}^{\text{0 – 4}}_{\text{\#capsules}} \cdot \overbrace{5}^{\text{0 – 4}}_{\substack{\text{\#weak}\\\text{ghosts}}} \cdot \overbrace{5}^{\substack{\text{4 cardinal}\\\text{directions}\\\text{+ None}}}_{\substack{\text{Dir.}\\\text{weak}\\\text{ghost}}} = 12,500. \tag{3.9}$$

This is because the states where one or both of the ghosts are weak are relatively rare and thus represent a poorly explored region of the state space.

3.6 Summary

In this chapter, we investigated how options learned from expert demonstrations can improve the performance of Q-learning. Q-learning can be affected by the bias introduced by options, and options learned from experts are good candidates to deliver a good bias. We proposed one algorithm to achieve this goal and tested it in two very different environments. The first being a simple grid-world, typically used in Reinforcement Learning research to show a proof of concept. the second, more challenging, is the arcade video game of Pac-man. The experiments performed in these experiments support the thesis that options can be effectively learned from demonstration using the proposed algorithm and produce performance improvements.

The study in this chapter shows how expert knowledge can be extracted through demonstrations via the concept of surprise. This is useful to initially navigate, even if sub-optimally, a complex state-space if the agent has no prior knowledge. The

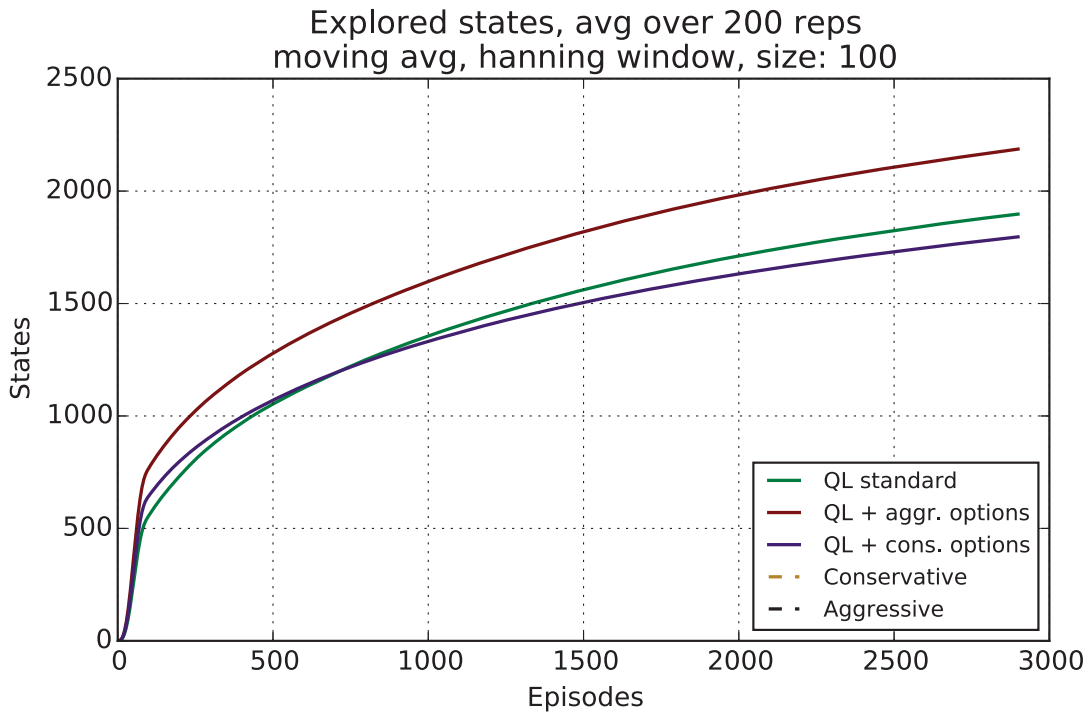


Figure 3.12 Exploration rate of the agent. The X axis shows the number of the episode, the Y axis shows the number of distinct states visited since the beginning of the experiment. Results are averaged over 200 runs of Q-learning.

next chapter shows another approach to tackle the problems arising from the lack of prior knowledge: starting the learning process by simplifying its state representation and only refining it over time, as more experience is collected.

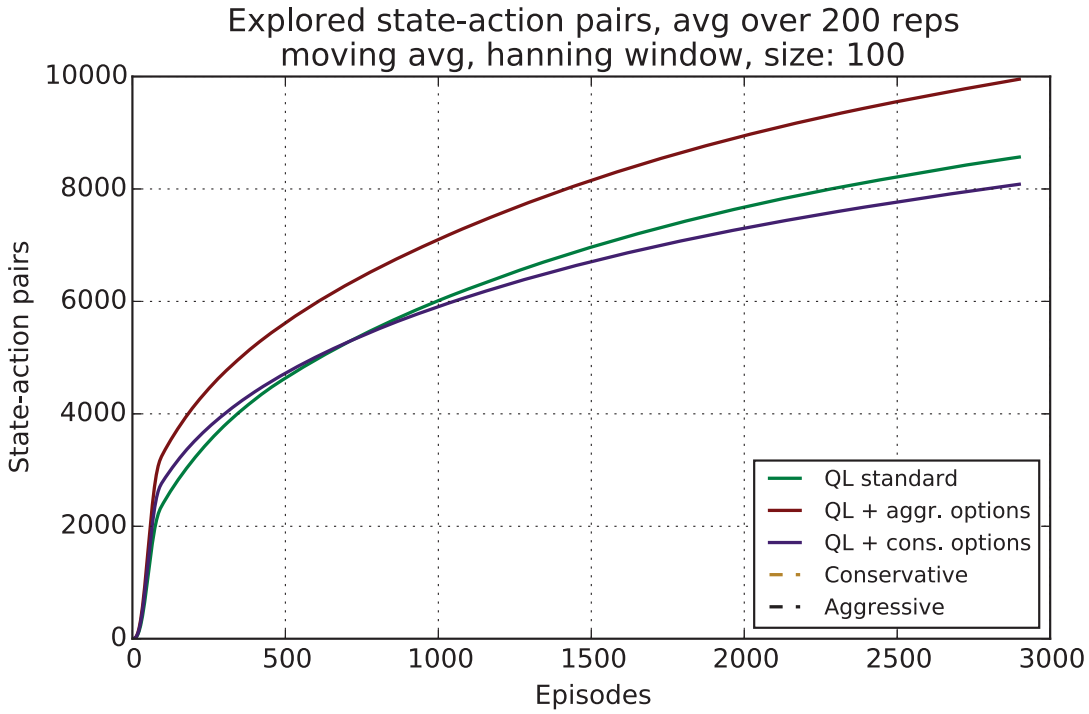


Figure 3.13 Exploration rate of the agent. The X axis shows the number of the episode, the Y axis shows the number of distinct states visited since the beginning of the experiment. Results are averaged over 200 runs of Q-learning.

Chapter 4

Dynamic Choice of State Abstraction

Markov Decision Processes are based on the Markov assumption which states that it is sufficient to know the current state of the environment to make predictions about the outcome of actions. One way for an agent to learn useful information about the environment dynamics is by interacting with it, in a sequence of observations of state and action. Based on the Markov assumption, Temporal Difference (TD) algorithms [90] encode useful information about the environment in the form of associations of states to utilities. For example, Q-learning, one of the most popular TD algorithms, associates state-action pairs to future rewards [107]. When a TD agent needs to make a decision, it will choose the action that is likely to yield the highest long-term utility value according to previous experience.

If states are rich in information, in the early stages of the learning process, the agent knowledge of the environment is sparse. If the agent considers every feature making up the state, it will take a considerable amount of time to learn associations for all of the many possible states, especially if outcomes are stochastic. To quote Andre and Russell, "Without state abstraction, every trip from A to B is a new trip" [5]. During this learning period, an agent may make blind decisions due to "details" in the state preventing an exact match with past experience. TD agents lack the ability to use knowledge of states similar to the current one. This research problem falls under the umbrella of Transfer learning [100] and, at a higher level, Generalisation [92].

The most common approach to address this issue is to use linear approximation [103, 95]. In this instance, an agent only has to learn the weights of the linear transformation mapping state features to utility. However, a linear approximation may not be sufficient if non-linear dynamics exist in the environment. In this case, sparsity issues are addressed by stripping states of "superfluous" details. This

process is called *state abstraction*, and it consists in aggregating states. By only considering the most important information and ignoring details, two states that are effectively different will appear the same, inducing a partitioning of the state space. The drawback of this, however, is that if the information used to encode states is not rich enough, the agent will not be able to make informed decisions. Examples of this approach are coarse coding and tile coding [92].

State abstraction has attracted attention in the reinforcement learning community in the past two decades. Most of the literature on the subject focuses on choosing an abstraction prior to the actual learning [51, 19, 40, 49]. McCallum's work, however, explored online state abstraction, which is also the focus of our work. [62–64].

In this chapter we propose an algorithm that shifts from coarse partitionings to more fine-grained ones through time. The choice of which partitioning to use is done at every step and can be different from state to state, allowing for more flexible learning. The criteria used by our algorithm to decide when to enrich state information is to compare the confidence interval of utility estimates. The idea is that, at the beginning of the process, most decisions are made using coarse partitionings while, in the long run, more choices are made with more informative partitionings. To our knowledge, this is the first attempt to combine both coarse and fine-grained partitionings online.

We evaluate our algorithm by comparing it with Q-learning in the context of the video game Pac-Man. Our experiments show that the proposed algorithm produces better performance than fixed state-size Q-learning during the learning phase. We also propose a strategy to direct exploration in a way that allows the algorithm to switch to fine-grained abstractions earlier. Experiments show that this strategy produces better performance than the standard ϵ-Greedy.

4.1 Dynamic Abstraction Choice

In reality, because environments are often stochastic, a number of trials are necessary for each state and action pair to evaluate reasonable estimates. In particular, in the early stages of an agent's life, its knowledge is rather sparse, often leading to blind decisions. To deal with this, a common approach is not to model the entire MDP, but a simplified one obtained by reducing the state space size via state aggregation [105, 100]. By means of carefully engineered state aggregations, Q-learning generalizes well over the little information it has.

However, it is desirable that in the long-run, the agent makes its decisions considering all the nuances of each state, rather than based on coarse aggregations. More information on the state allows the agent to make more informed decisions.

We propose a novel algorithm to achieve the advantages of both situations, at the cost of a slight increase in processing time. In the following, we refer to abstractions as functions mapping a state to an aggregation of states. Each abstraction induces an abstrated state space of smaller size than the original one and, consequently, a smaller Q-table. However, such Q-tables do not need to be memorized since they can be inferred by appropriately aggregating entries of the original Q-table.

The algorithm we propose, "Multi-Abstraction Q-learning", is presented in pseudocode in Algorithm 3. Multi-Abstraction Q-learning is given a list of abstractions of decreasing granularity, and maintains the Q-table associated with the original state representation. At decision time, the algorithm chooses the most granular abstraction whose Q-values are precise with sufficient confidence.

Formally, an abstraction is defined as $\beta_i : \mathcal{S} \to \mathcal{S}_i$. We also introduce an ordering for abstractions, based on their granularity. Formally, an abstraction β is more granular than a second abstraction β' (denoted $\beta > \beta'$) if both the following conditions hold:

- Any two states $s, s' \in \mathcal{S}$ mapped to the same abstracted state by (the more granular) abstraction β are also mapped to the same abstracted state by (the more coarse) abstraction β'. Formally:

$$\forall s, s' \in \mathcal{S} \,.\, \beta(s) = \beta(s') \Rightarrow \beta'(s) = \beta'(s')$$

- There is at least a pair of states $s, s' \in \mathcal{S}$ that are mapped to different abstracted states by (the more granular) abstraction β but that are mapped to the same abstracted state by (the more coarse) abstraction β'. Formally:

$$\exists s, s' \in \mathcal{S} \,.\, \beta(s) \neq \beta(s') \wedge \beta'(s) = \beta'(s')$$

Algorithm 3 is given a list of abstractions of decreasing granularity. The algorithm decides which action to take by choosing the most suitable abstraction at every time step. The chosen abstraction is the most granular one that provides a high confidence that the action with the highest estimated Q-value is actually the best one. Confidence of a state-action pair (s, a) is computed by running a t-test over the

history $H(s, a)$ of values that the Q-table entry $Q(s, a)$ has assumed. The steps used to test the confidence of an abstraction are as follows:

1. the action a^* with the highest estimated Q-value is found;
2. the boundaries $\bot_a, \top_a$ of the confidence intervals of all actions a are calculated;
3. for all $a \neq a^*$, test whether $\bot_{a^*} > \top_a$: each test is passed with a $1-\sigma$ confidence level;
4. if all the tests are passed, it is reasonable to assume that the true Q-value of a^* is actually the highest.

Optimization

The procedure described in Algorithm 3 has some significant inefficiencies. However, notice that they can be overcome and have been introduced in the listing for the purpose of clarity. In the follwing paragraphs, we briefly explain how to address these issues.

There are two main bottlenecks. The first is at line 12, where the union operation iterates over all the states to find the siblings. This adds a significant amount of computational time to compute the same information repeatedly. In fact, caching the state space partitioning (i.e. the sets of "siblings") for each abstraction is a better solution. By doing so, the time complexity of retrieving such information is constant at the cost of a linear (in the number of states and abstractions) increase in memory complexity.

The second bottleneck is the procedure at line 13 which computes the confidence intervals. The procedure iterates over the whole set X_a^j of samples at every invocation to compute their mean and variance. An alternative, more efficient solution is to store mean and variance of the Q-value for every state-action pair and update them online [108]. It is, then, possible to efficiently compute mean and variance of a virtual union set by aggregating the stored statistics [109]. This modification makes storing the history of Q-values unnecessary, significantly reducing the space requirements of the algorithm.

Confidence driven exploration

While ϵ-greedy is expected to shrink the confidence intervals in the long run through random exploration, it has no awareness of their existence. It is reasonable to sup-

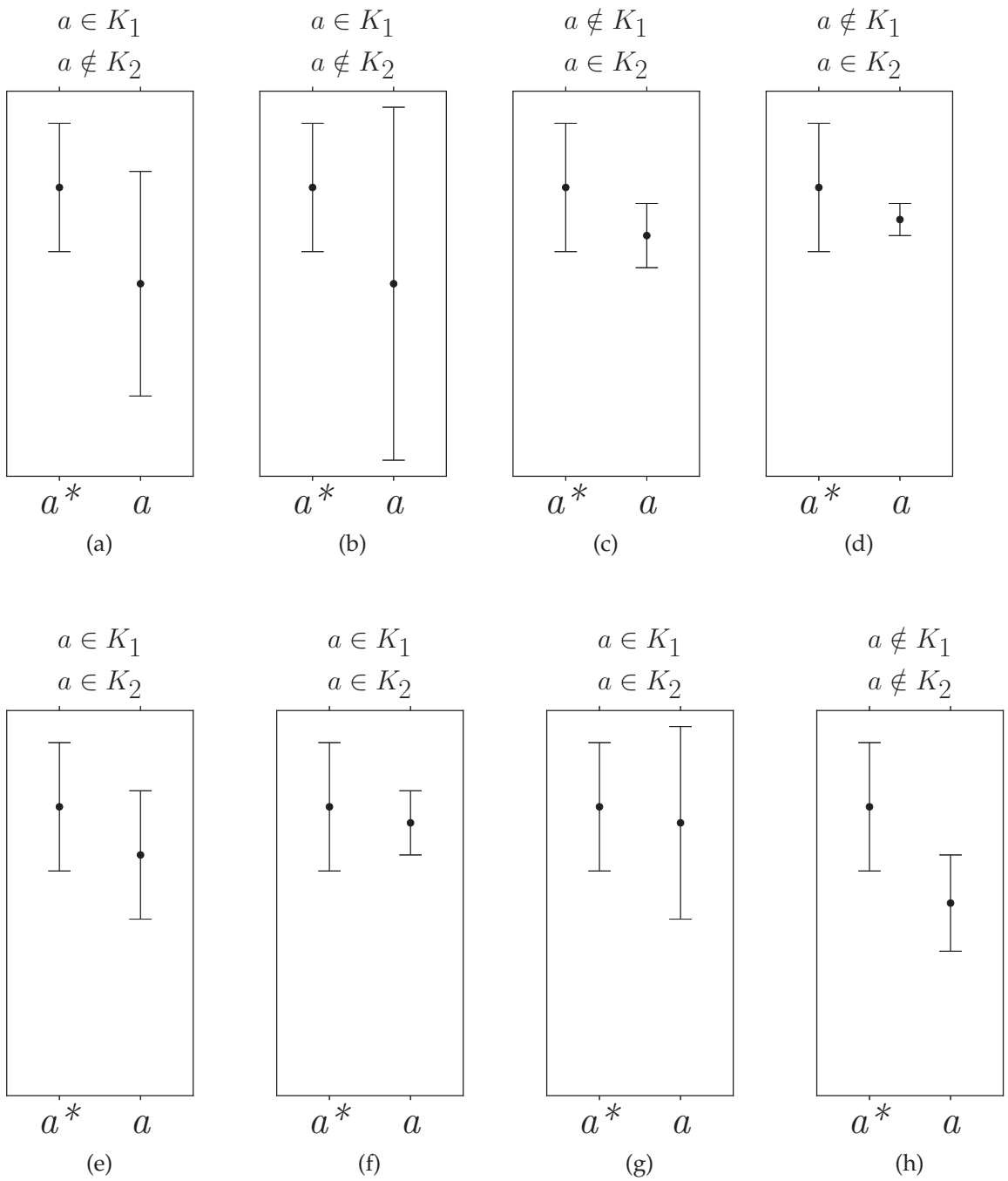


Figure 4.1 A visual qualitative representation of the possible situations in which confidence intervals overlap. Here, a^* is the action with the highest mean and a is another action. When confidence intervals overlap, the confidence level cannot be guaranteed. On top of each subfigure, the decision made by Algorithm 4 at lines 11–12 is indicated.

pose that using a different exploration strategy making use of this knowledge would produce better results. Such a strategy would bias the exploration so as to perform actions whose confidence intervals are preventing the use of the next abstraction. We propose a variation on the traditional ϵ-greedy strategy that integrates such bias. The close-form definition is slightly cumbersome; so, with clarity in mind, we provide the pseudocode instead. The pseudocode in Algorithm 4 describes such procedure and is meant to replace line 22 of Algorithm 3.

This procedure requires two parameters ϵ_{CI} and ϵ_{R}, which set the probability of exploring versus exploiting, similarly to ϵ-greedy. Unlike ϵ-greedy, however, this procedure performs a second type of exploration. That is, with probability ϵ_{CI}, it selects an action whose confidence interval needs to be reduced in order for the next abstraction to be usable.

Figure 4.1 shows a qualitative representation of the reasoning behind Algorithm 4. Depicted are the different possible situations in which confidence intervals of actions a^* (action with the highest mean) and a overlap. On top of each subfigure, the behavior of the algorithm in lines 11–12 is reported. Notice that in many cases there will be multiple actions matching the criteria of a: in such cases the choice is random among them. For the sake of concisenes, in the remainder of this section as well as in Algorithm 4 we will refer to the confidence interval of the estimate of the average Q-value of action a simpy as the confidence interval of a.

Line 11 of Algorithm 4 captures cases shown in figures 4.1a, 4.1b, 4.1e, 4.1f, 4.1g, where the upper bound of the confidence interval of some action a is higher than the average Q-value of the best action a^*. Such actions are stored in set K_1. Line 12 of Algorithm 4 captures cases shown in figures 4.1c, 4.1d, 4.1e, 4.1f, 4.1g, where the average Q-value of some action a is higher than the lower bound of the confidence interval of the best action a^*. Such actions are stored in set K_2. Algorithm 4 makes the simplifying assumption that further samples will shrink the confidence intervals without moving the average value. Notice that this does not introduce bias since the average value is an unbiased estimate of the mean value. With this assumption in mind, the procedure selects:

- a random action from K_1 if $K_1 \neq \emptyset$ and $K_2 = \emptyset$ because the only way to remove the overlaps of the confidence intervals is to shink those of the actions in K_1;
- a^* if $K_1 = \emptyset$ and $K_2 \neq \emptyset$ because, while choosing actions from K_2 would also be an effective way to remove the overlaps, choosing a^* is the choice with the highest expected future reward;

- a random action from $K_1 \cup \{a^*\}$ if $K_1 \neq \emptyset$ and $K_2 \neq \emptyset$ because of the same reasons explained above.

Notice that the case where $K_1 = \emptyset$ and $K_2 = \emptyset$ is only possible if the all the abstractions are usable, but this case is captured in lines 2–4. Following this strategy, confidence intervals that are overlapping will shink until they do not overlap anymore, therefore allowing the usage of the next abstraction, until the most fine-grained abstraction is usable.

4.2 Experiments

We evaluate the effectiveness of our algorithm using Pac-Man, a real-time arcade retro game[1]. Games of this type are of interest to the scientific Artificial Intelligence community due to the challenges of open-endedness and tight time-constraints they pose [78]. Pac-Man has been used as test-bed in a sizeable amount of literature, including [37, 77, 80]. We adopted the implementation currently used in UC Berkeley to teach AI, originally developed by DeNero and Klein [32][2]. Our algorithms were implemented in Python using the Numpy library[3] [106] and parallelized using GNU Parallel[4] [99]. We use the same Pac-man environment detailed in section 3.4.3, however we use a simpler level topology, as depicted in figure 4.2. This is not a standard Pac-Man level, but a simpler one, provided with the code-base. We chose this because experiments require less computational time.

Players receive a score based on their performance. In the implementation we used, Pac-Man receives 10 points for each eaten pill, and it loses 1 point at each time step, while receiving 200 points every time a ghost is killed. Furthermore, 500 points are earned upon victory (i.e. when all the pills have been eaten), while 500 points are lost upon death. These scores have been chosen by the developers of the framework, and we adopt them without changes in this work.

Tests

The first question we wanted to answer is what significance setting σ yields the highest performance in Multi-Abstraction Q-learning (Algorithm 3) with ϵ_{CI} strategy.

[1] Additional information can be found at `http://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php?print=1` (checked on March 3rd, 2016).

[2] Currently available for download at `http://ai.berkeley.edu/project_overview.html`

[3] `http://www.numpy.org/`

[4] `http://www.gnu.org/software/parallel/`

Figure 4.2 A screenshot of the video game used in the experiments, Pac-Man.

To answer this question, we tested the algorithm with $\sigma \in \{0.1, 0.2, 0.5, 0.9\}$. All the configurations used food and threatening ghosts information to describe states, successively adding edible ghosts information and, lastly, capsules information. In these tests, we set $\epsilon_{\text{CI}} = 0.05$ and $\epsilon_{\text{R}} = 0.05$.

Secondly, we wanted to evaluate whether shifting abstractions - from coarse to fine-grained - improves agents performance. To test this, we ran tests on Pac-Man using different agent algorithms:

- Q-learning where states included food and threatening ghosts information;
- Q-learning where states included food, threatening ghosts and edible ghosts information;
- Q-learning where states included food, threatening ghosts, edible ghosts and capsules information.

We compared the performance of these algorithms with those of the best configuration from the previous test, that with $\sigma = 0.9$. The exploration strategy used in these three configurations was ϵ-Greedy with $\epsilon = 0.1$.

We ran tests using each of these algorithms for 30000 consecutive episodes and we measured the reward collected during each of them. We repeated this 50 times

and averaged the results. Agent performance is expected to improve over time, as they gather information on the environment: however, improvement rate and final performance depend upon the agent algorithm and its state representation.

The final question we wanted to answer is to what degree the performance of the other experiments are due to Multi-Abstraction Q-learning versus to the ϵ_{CI}-Greedy strategy. To test this, we ran experiments using the following algorithms:

- Multi-Abstraction Q-learning with ϵ-Greedy with $\epsilon = 0.1$;
- Q-learning with ϵ_{CI} with $\epsilon_{\text{CI}} = 0.05$ and $\epsilon_{\text{R}} = 0.05$.

We compared the performance of these algorithms with the performance of the best configuration in the first experiment, that with $\sigma = 0.9$. The features used in these experiments are the same used in the first set of experiments; that is, all of them: food, threatening/edible ghosts and capsules.

State space

Performance of MDPs are heavily influenced by the shape of the state space. In our experiments, the state space is the Cartesian product of 8 features. Each of the features is related to objects in game; i.e., threatening/edible ghosts, pills and capsules. Features are either distance or direction information to such objects.

The "direction" feature specifies the direction that Pac-Man should follow to reach the closest object of the category. The direction information can assume five different values: one for each of the cardinal directions, plus an additional value used when there are no instances of the objects; e.g., if all the ghosts are edible, there is no threatening ghost. In the case of distance, the feature specifies $\lfloor \log_2 d \rfloor$, where d is the length of the shortest path to the closest object of the category. Shortest paths are computed by the Dijkstra algorithm for shortest path on graphs (see [23] for more information). Distance information can be null as well.

Annealing exploration

We compared different exploration strategies; i.e. ϵ-Greedy and our proposal, ϵ_{CI}-Greedy, showed in Algorithm 4. Even though we presented the naïve versions of the two strategies, in our experiments we used the simulated annealing version. This technique slowly decreases the amount of exploration as time progresses, so to gradually shift from an exploration policy to the greedy policy over time. In ϵ-Greedy policies this is done by decreasing the value of ϵ. In ϵ_{CI}-Greedy, we similarly

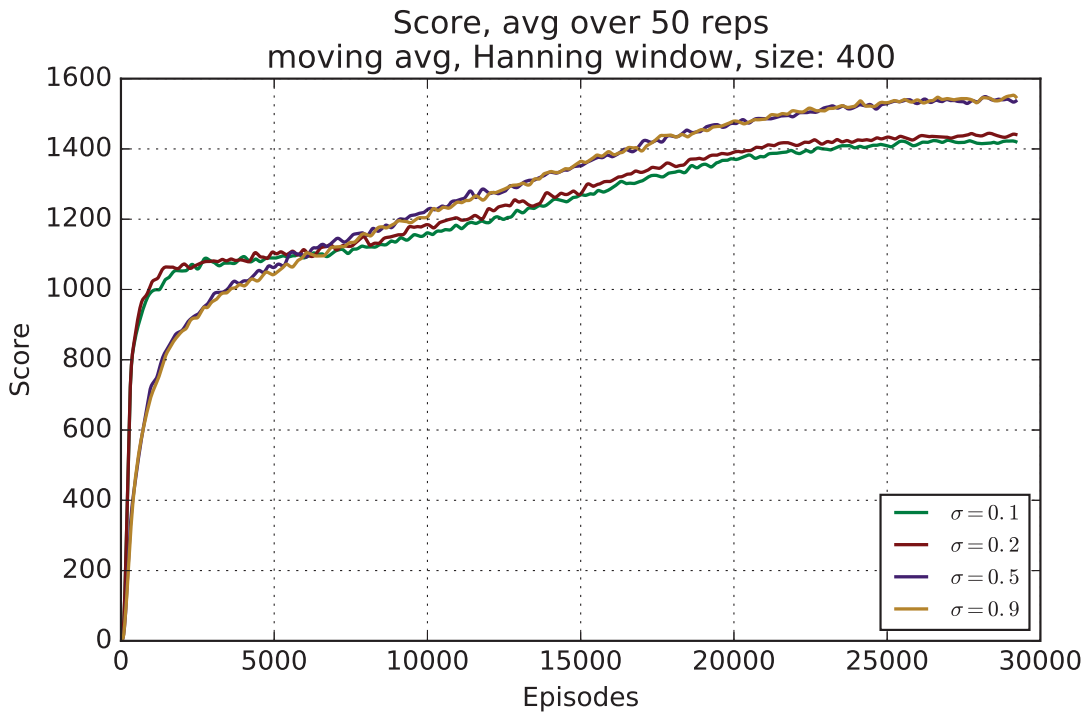


Figure 4.3 Scores of the games (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter.

decrease both ϵ_{R} and ϵ_{CI}. The annealing schedule we chose is based on the sigmoid function, $s(x) = \frac{1}{1+e^x}$. Our schedule is defined as follows:

$$\epsilon(t) = \hat{\epsilon} \cdot s\left(u \cdot (m - t)\right),$$

where $\hat{\epsilon}$ is the maximum value for the exploration parameter, t is the current episode number, m is the desired center for the schedule and u controls the width of the function.

4.3 Results

In this section we discuss the results of the experiments we performed. All the figures in this section are smoothed using a moving average weighted by a Hanning function. The Hanning function is bell-shaped and smoothly zeroes at the edges. Using it to weight contributions in a moving window gives greater importance to central elements while still taking the surrounding element in account.

In the first experiment, different σ-values are compared in Multi-Abstraction Q-learning (Algorithm 3) using ϵ_{CI}-Greedy. The scores achieved by the different

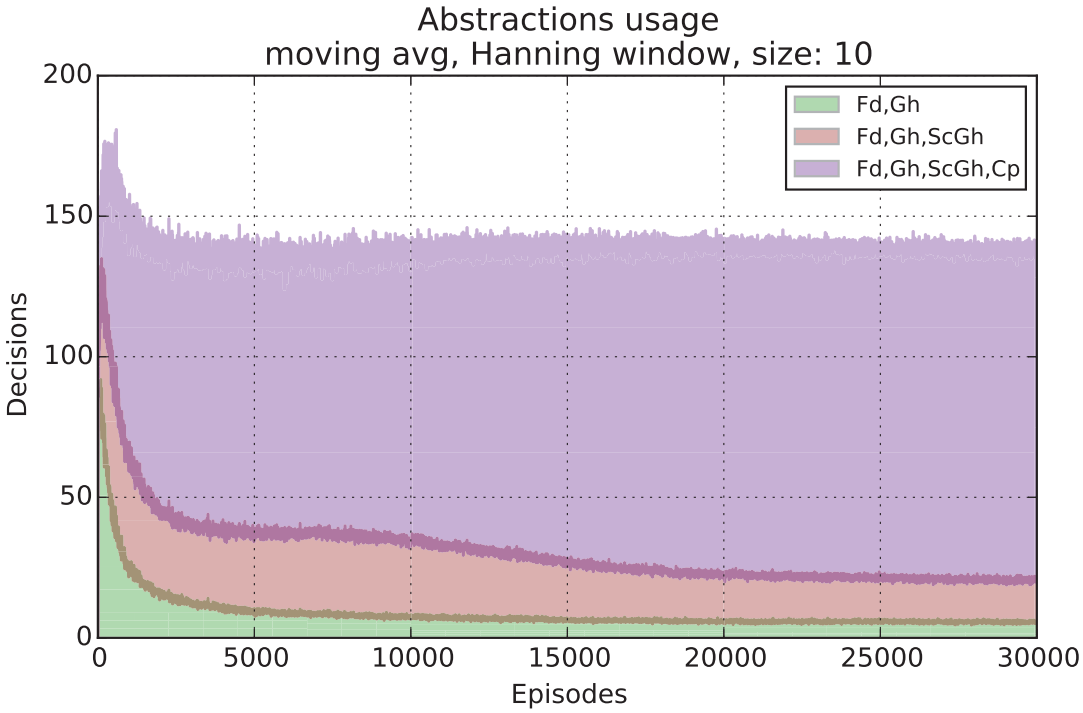


Figure 4.4 Decisions made with each abstraction (y axis) for each successive episode (x axis), using $\sigma = 0.2$ in Multi-Abstraction Q-learning with ϵ_{CI}-Greedy.

configurations are shown in Figure 4.3. It is surprising that the lines dominating the chart are those using 0.5 and 0.9 as σ-values.

The most likely explanation for this is that 0.1 and 0.2 are too conservative values. While in normal t-tests values of 0.1 are unacceptably high, the trend here is heavily shifted. In fact, orthodox t-tests assume that the distributions are static over time. Here, however, (expected) Q-values veer from the common initial value towards their true values. For this reason, seemingly "premature" Q-values, which have a "high variance" from a t-test perspective, reliably estimate the best action.

Figures 4.4 and 4.5 show the percentage of decisions that have been made with each abstraction in successive episodes for the two configurations $\sigma = 0.2$ and $\sigma = 0.5$. There is a remarkable difference in that the former keeps using coarse abstractions throughout the learning process, while the latter barely uses any, except at the very early stages.

Figure 4.6 shows the percentage of victories for the configurations in the first experiment. The configurations winning the most often are $\sigma = 0.1$ and $\sigma = 0.2$. Considering the scores shown in Figure 4.3 this seems counterintuitive, because one would expect that the configurations with the highest scores are also those winning

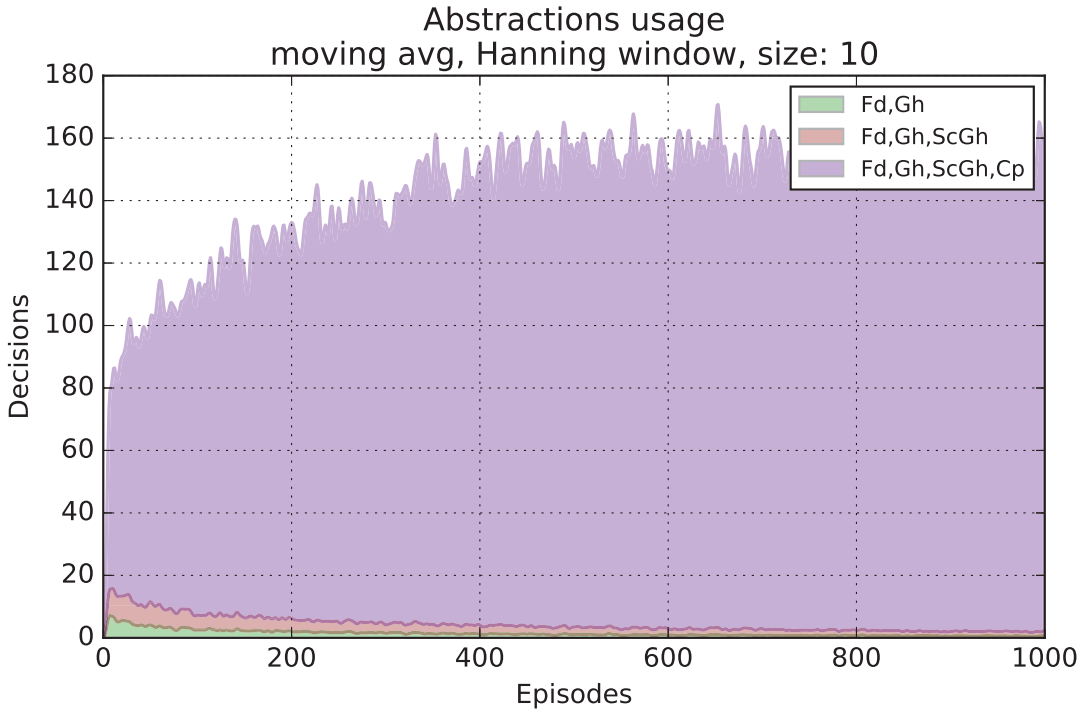


Figure 4.5 Decisions made with each abstraction (y axis) for each successive episode (x axis), using $\sigma = 0.5$ in Multi-Abstraction Q-learning with ϵ_{CI}-Greedy.

the most often. However, these numbers make sense when the structure of the game is considered: to maximize the score, Pac-Man needs to eat ghosts, but that poses an added risk in terms of winning/losing (i.e. if the ghost suddenly turns back to a threatening status). Figure 4.7 shows the average number of ghosts eaten in each successive episode: it can be observed that there is a significant difference between the configurations $\sigma = 0.1$ and $\sigma = 0.2$ and the configurations $\sigma = 0.5$ and $\sigma = 0.9$. The similarity of the trends showed in Figures 4.7 and 4.3, where dominant configurations are $\sigma = 0.5$ and $\sigma = 0.9$ in both cases, supports this theory.

In the second experiment, our technique is compared with three configurations of Q-learning, each using an increasing amount of features. Figure 4.8 compares them to the best configuration of the first experiment, Multi-Abstraction Q-learning with $\sigma = 0.9$.

The curves show that the Q-learning configuration with the least features, at first, performs the best, showing that using more features at the beginning worsens the performance. However, this configuration is later surpassed by the Q-learning configuration using the intermediate amount of features, showing that having more features pays off when sparsity fades out. Finally, the Q-learning configuration using

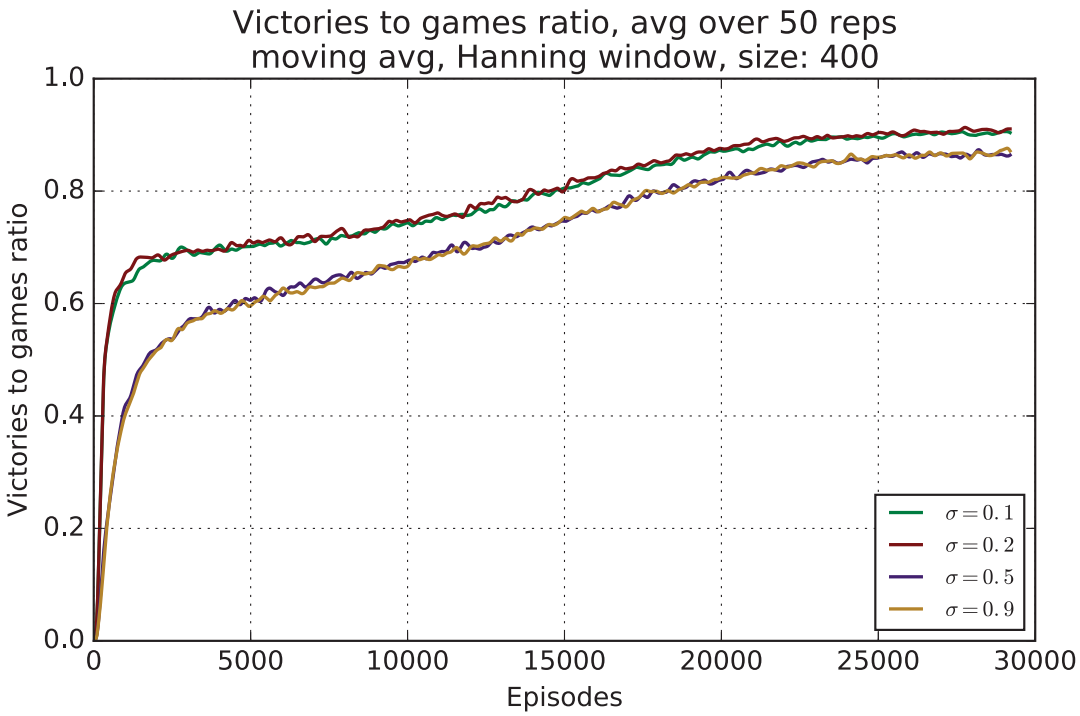


Figure 4.6 Victories to total games ratio (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter.

the most features surpasses both the other two Q-learning configurations. These trends show how, in normal Q-learning, more features produce better performance at later stages at the cost of performance in the early stages.

Except at the very beginning of the process, Multi-Abstraction Q-learning produces significantly better results than the other configurations. Importantly, it also converges to the same values as the Q-learning configuration using all of the features: this shows that the early improvement in performance does not come at the cost of later performance.

It could be argued that our approach can be replaced by predetermined rules. In fact, the intersection points of Q-Learning performance curves in Figure 4.8 provide a clear indication of when it is convenient to switch abstraction. This would produce better performance than any of the three Q-Learning agents. However, because our approach allows each state to be used at a different abstraction, it is more adaptive and produces far better performance during the learning phase, as shown in Figure 4.8.

It could also be argued that, since the final performance of Multi-Abstraction Q-learning is the same as that of standard Q-Learning, an agent might as well just

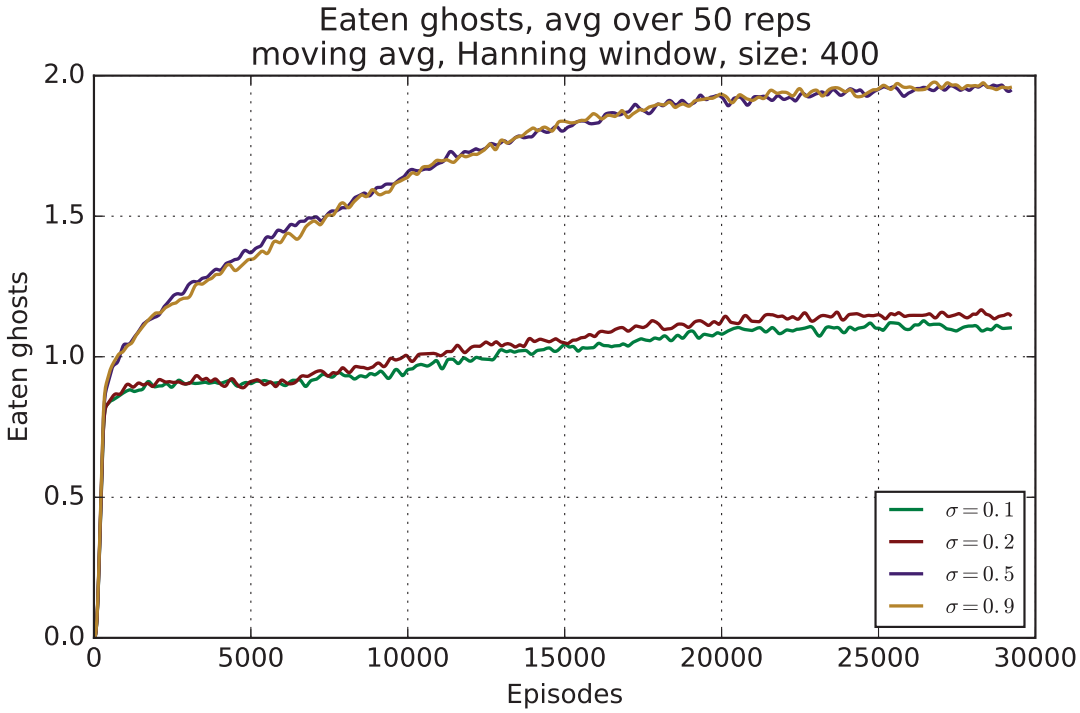


Figure 4.7 Eaten ghosts (y axis) per successive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy, varying significance parameter.

use standard Q-Learning. While this is true, the advantage of Multi-Abstraction Q-learning is an improvement in performance during the learning phase as opposed to an improvement in final performance.

The results of the third experiment are shown in Figure 4.9. The experiment shows that Multi-Abstraction Q-learning and ϵ_{CI}-Greedy do create a sinergy in performance. On one side, ϵ_{CI}-Greedy does not seem to affect the performance of Q-learning; on the other side, Multi-Abstraction Q-learning without ϵ_{CI}-Greedy performs worse than Q-learning. However, when Multi-Abstraction Q-learning is used in concert with ϵ_{CI}-Greedy, they produce better performance than all other combinations.

Statistical analysis of Q-learning performance

The experiments showed that Multi-Abstraction Q-learning produces better performance than Q-Learning with any set of features. To quantitatively measure the performance of the different configurations, we computed the average reward per episode for each of the 50 runs of each of the 3 agents. This operation induces a distributions for each agent, all of which appear to be approximately normally

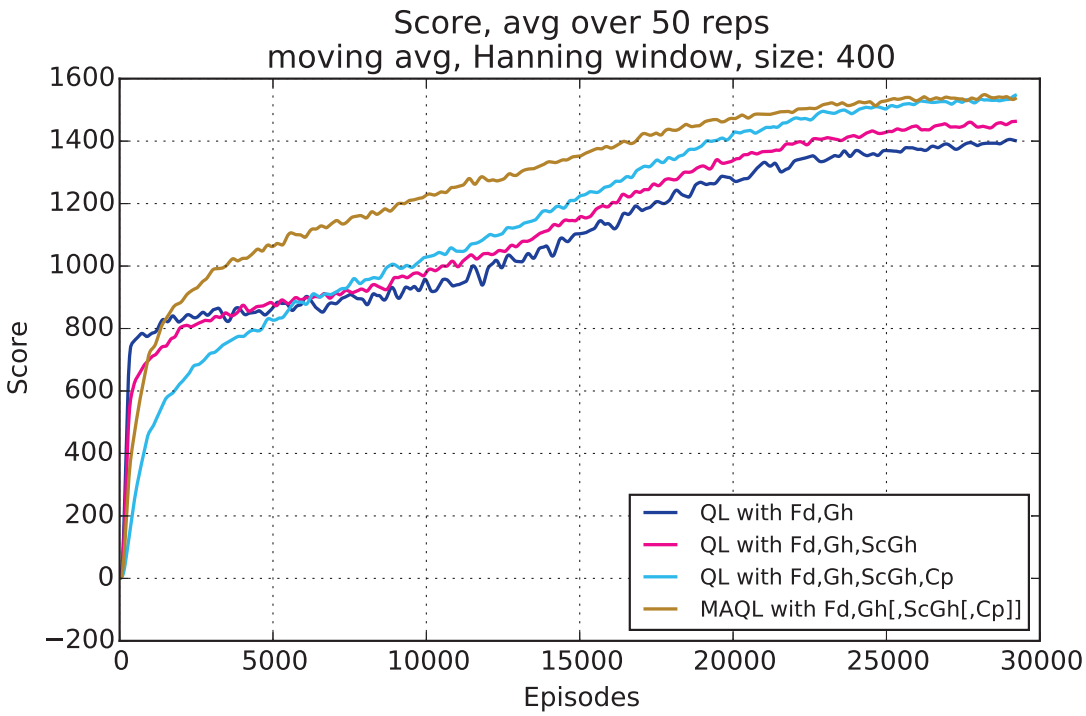


Figure 4.8 Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ_{CI}-Greedy versus standard Q-learning with different sets of features.

distributed, as shown in Figure 4.10. Table 4.1 shows the means and standard errors of the data. Three two-samples t-tests (with unequal variance) have been executed to pairwise compare the agents sorted by average per-episode reward. All t-tests determined that the distributions mean are different with p-value < 0.001, therefore confirming that both agents using learned options perform significantly better than the agent that does not use options.

4.4 Summary

In this chapter we presented a novel variation of Q-learning, which we name "Multi-Abstraction Q-learning". The algorithm we propose uses different state-abstractions for each state, increasing the level of detail over time. This allows the agent to overcome the initial sparsity in its utility estimates, typical of richer state representations. The agent can still make full use of the maximum level of detail later on in the learning process. Our experiments show that this algorithm produces better performance than standard Q-learning.

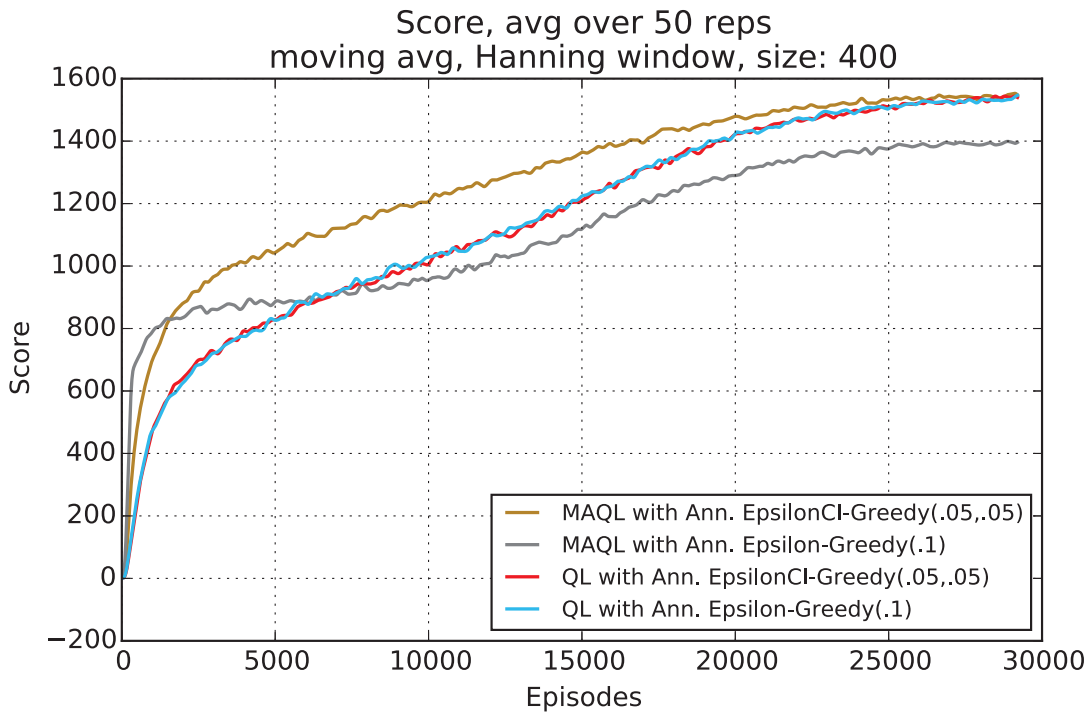


Figure 4.9 Score (y axis) per succesive episode (x axis) using Multi-Abstraction Q-learning with ϵ-Greedy versus ϵ_{CI}-Greedy.

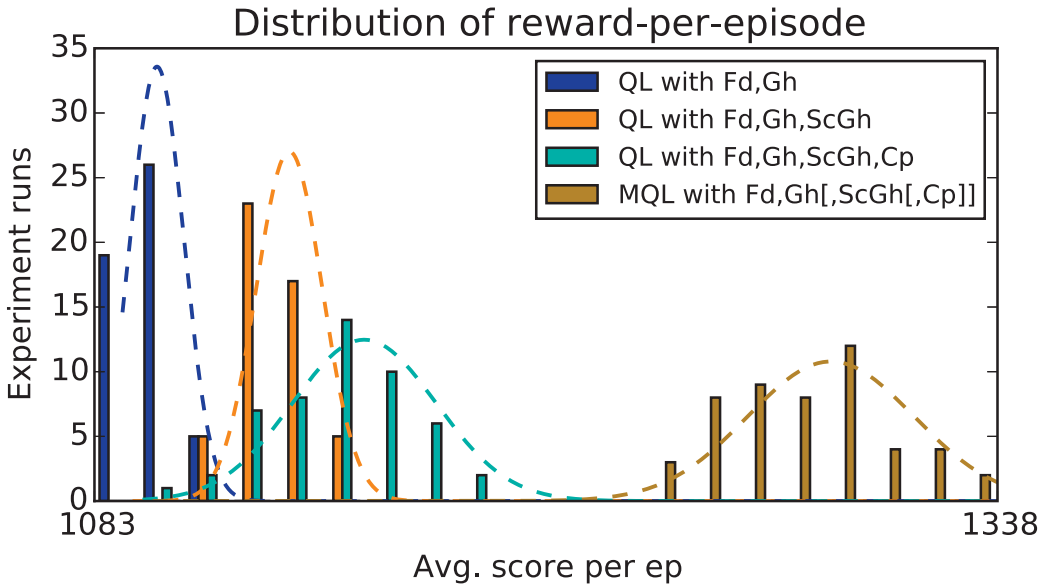


Figure 4.10 Histograms of the average reward per episode.

We also proposed a novel exploration strategy, ϵ_{CI}-Greedy. This strategy directs exploration to reduce sparsity of information, thereby allowing the agent to switch to

more detailed abstractions earlier. Our experiments show that this strategy produces better results than standard ϵ-Greedy.

Both Chapters 3 and 4 reported studies where the objective for an agent is to perform as well as possible. Chapter 5 focuses on a more specific scenario, that of a video game, and on different goal, that of making the agent perform at the same level of its opponent.

Algorithm 3: Multi-Abstraction Q-learning algorithm for abstraction shifting. $\biguplus$ indicates a multiset sum; a multiset is a set where information about the number of occurrences of each element is preserved. $\overline{X}$ indicates the sample average. Procedure CI computes the confidence interval of the mean of the given sample.

Input : Learning rate α
Input : Exploration parameter ϵ
Input : Abstractions $\beta_1 > \beta_2 > \ldots > \beta_m$
Input : Default Q-value, initQ
Input : Significance level for t-tests, σ

1 **for** $s, a \in \mathcal{S} \times \mathcal{A}$ **do**
2 $\quad Q(s,a) \leftarrow$ initQ
3 $\quad H(s,a) \leftarrow$ empty list // history of $Q(s,a)$
4 **end**
5 $s \leftarrow$ observe state
6 **repeat**
7 $\quad j^* \leftarrow m$
8 $\quad$ found $\leftarrow$ false
9 $\quad$ **for** $j \leftarrow 1 \ldots m$ **do**
10 $\quad\quad \xi \leftarrow \{\, s' \in \mathcal{S} \mid \beta_j(s') = \beta_j(s) \,\}$ // siblings
11 $\quad\quad$ **for** $a \in \mathcal{A}$ **do**
12 $\quad\quad\quad X_a^j \leftarrow \biguplus_{s' \in \xi} H(s',a)$ // samples of siblings
13 $\quad\quad\quad \bot_a^j, \top_a^j \leftarrow \mathrm{CI}(\sigma, X_a^j)$ // lower and upper bounds
14 $\quad\quad$ **end**
15 $\quad\quad a^* \leftarrow \arg\max_{a \in \mathcal{A}} \overline{X_a^j}$
16 $\quad\quad$ **if** $\bot_{a^*}^j > \top_a^j$ *for all* $a \in \mathcal{A}, a \neq a^*$ **then**
17 $\quad\quad\quad j^* \leftarrow j - 1$
18 $\quad\quad\quad$ found $\leftarrow$ true
19 $\quad\quad\quad$ go to line 22
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 $\quad a* \leftarrow \begin{cases} \arg\max_{a \in \mathcal{A}} \overline{X_a^{j^*}} & \text{with prob. } 1-\epsilon \\ \mathrm{random}(\mathcal{A}) & \text{with prob. } \epsilon \end{cases}$
23 $\quad$ perform action a^*
24 $\quad s' \leftarrow$ observe state
25 $\quad r \leftarrow$ receive reward
26 $\quad \hat{q} \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$
27 $\quad Q(s,a^*) \overset{\alpha}{\leftarrow} \hat{q} - Q(s,a^*)$
28 $\quad$ append $\hat{q}$ to $H(s,a^*)$
29 $\quad s \leftarrow s'$
30 **until** *apocalypse*

Algorithm 4: C.I. driven exploration.

1 $\begin{cases} \textbf{return } a^* & \text{with prob. } 1-\epsilon_{\text{CI}}-\epsilon_{\text{R}} \\ \textbf{return } \text{random}(\mathcal{A}) & \text{with prob. } \epsilon_{\text{R}} \\ \text{go to line 11} & \text{with prob. } \epsilon_{\text{CI}} \end{cases}$
2 **if** $j^* = 1$ *// most granular abstraction already in use* **then**
3 | **return** a^* // no exploration needed
4 **end**
5 **if** $found$ *// acceptable abstraction found* **then**
6 | $\hat{j} \leftarrow j^* - 1$ // use it
7 **else**
8 | $\hat{j} \leftarrow m$ // use the most coarse one
9 **end**
10 $a^* \leftarrow \arg\max_{a \in \mathcal{A}} \overline{X_a^{\hat{j}}}$
11 $K_1 = \left\{ a \in \mathcal{A} \;\middle|\; \top_a^{\hat{j}} > \overline{X_{a^*}^{\hat{j}}} \right\}$
12 $K_2 = \left\{ a \in \mathcal{A} \;\middle|\; \overline{X_a^{\hat{j}}} > \bot_{a^*}^{\hat{j}} \right\}$
13 **if** $K_1 \neq \emptyset$ and $K_2 \neq \emptyset$ **then**
14 | **return** $\text{random}(K_1 \cup \{ a^* \})$
15 **else if** $K_1 \neq \emptyset$ **then**
16 | **return** $\text{random}(K_1)$
17 **else**
18 | **return** a^*
19 **end**

	MQL ■	QL w/4 ■	QL w/3 ■	QL w/2 ■
Mean	1281.07	1152.11	1133.77	1099.25
Std. err	23.71	20.51	9.46	7.62

Table 4.1 Mean and standard error of the average reward per episode across the 50 runs of the experiments in Figure 4.8. Columns report values, respectively, for Multi-Abstraction Q-learning and for Q-Learning with features Fd,Gh,ScGh,Cp, Fd,Gh,ScGh and Fd,Gh.

Chapter 5

Monte Carlo Tree Search for Dynamic Difficulty Adjustment

Video games are computer driven simulations that allow players to experience emotions uncommon in their daily lives. Racing, fighting and shooting games, to name the most popular genres, can bring the player in tense, adrenaline-rich experiences. This immersing experience can reach an extreme called the state of "flow", where the player dissociates from reality to such an extent that they lose time perception. One essential prerequisite to reach flow, among others, is a balance between player skills and challenge offered by the game. When a game is too easy the player experiences boredom, while if the game is too difficult they experience anxiety. Both these situations disrupt the immersion and prevents it from reaching its deepest levels.

This has led researchers to investigate mechanisms to adapt the game difficulty to the player. This research is reviewed in section 2.3.5. Some researchers base their works on the assumption that a balanced game is one where the outcome is completely unpredictable (chance of victory is 50%) [4]. Inspired by the work of Andrade et al., we propose a technique based on theirs, approaching the problem from a different angle. We test our technique in a 2D fighting video game, the Fighting ICE platform, and compare it with the technique proposed by Andrade et al.. Experiments show our technique has better performance in terms of game balancing.

The work reported in this chapter is the output of a collaboration with fellow Ph.D. student Simon Demediuk. This chapter only reports the part of this collaboration that I focused on; that is, the Dynamic Difficulty Adjustment agents.

This chapter is structured as follows: section 5.1 describes the technique proposed in [4], section 5.2 details our proposed technique and some variations, section 5.3 reports the experiments we performed and section 5.4 presents the results and discusses the results of the experiments.

5.1 Challenge Sensitive Action Selection

Andrade et al. [4] propose using Reinforcement Learning to achieve Dynamic Difficulty Adjustment. Their objective is to achieve a zero health-points (HP) difference between the human player and their agent in a fighting game; this serves as a proxy for a game with an unpredictable outcome (a 50% chance of victory). To achieve this, they periodically test the current HP difference between players and increase/decrease the level of their agent depending on where the HP difference leans.

Their agent is driven by the Challenge Sensitive Action Selection (CSAS) algorithm. CSAS is based on Reinforcement Learning (RL). The agent first gather knowledge of the domain at hand and is then able to offer predictions of the consequences of choosing each action given a state of the environment/game. CSAS uses RL as an oracle and performs the action selection on top of it.

When in a state s, CSAS computes the Q-value $Q(s, a)$ of all actions a in that state. It is worth remembering that the Q-value $Q(s, a)$ indicates the expected long-term cumulative reward of performing action a in state s and performing all successive actions optimally (according to its own knowledge, encoded in the Q function itself). The reward used by Andrade et al. is the change in HP difference between the players: positive if the action causes more damage than it receives, negative otherwise. CSAS then ranks all actions according to their Q-value and chooses the action to perform based on the current level. In particular, the current level is a number between 0 and 1 and expresses a percentile.

Initially the level is set at 0.5, which makes CSAS choose the 50th percentile action (the action with the median value). The level can then increase leading to the selection of stronger actions, if the agent is losing the fight, or decrease if the agent is winning. The level is altered based on the HP difference periodically. In their experiments, Andrade et al. alter the level every 100 cycles, corresponding to roughly one every 9 seconds, for a total of 10 updates during a single fight.

This technique presents several limitations. The main limitation is the level update frequency: performing this too frequently can result in jittering behaviour,

while performing it too slowly can prevent the agent from adapting before the fight ends. Another limitation is that the Q-value assumes that subsequent moves are optimal, therefore preventing the mechanism from choosing a sub-optimal strategy and instead focusing on sub-optimal-in-the-short-term decisions. Finally, the position of an action with respect to the other actions in the ranking is not a good indicator of how strong an action is in absolute terms. In general, there can be more advantageous states where most of the actions are going to be stronger than desired; the opposite can also happen.

5.2 Targeting outcomes

In light of the limitations highlighted in the previous section, we propose techniques that try to address the shortcomings of CSAS. Our techniques, similarly to CSAS, rely on an oracle to provide values for given state-action pairs. However, we use Monte Carlo Tree Search (MCTS) as such oracle, because it does not require to choose compromises between state-representation richness and learning time - MCTS, in fact, does not require learning time and uses the complete state information to carry its computation. This comes at the cost of performing computations online; however, even in the short time allotted for decision (less that $\frac{1}{60}$ of a second), MCTS produced good performance.

It is worth remembering that MCTS builds a tree to search the best course of action. It expands the tree asymmetrically, going more in depth in more promising branches. The algorithm starts from the root node, representing the current state. It then repeats the following steps repeatedly, as many times as time permits:

1. descends the tree all the way to a leaf led by most-promising steps;
2. decides whether to expand the leaf by adding its children, one for each action (and descends into a child if it does);
3. performs a simulation of the evolution of the game given the actions encoded in the path root–leaf followed by some random actions;
4. computes some metric from the resulting state and updates the average score of all traversed nodes.

After building the tree, MCTS selects an action considering the average score or the number of visits of the children of the root node. A more in-depth introduction to

MCTS is given in section 2.3. Notice that different score metrics as well as different action selection strategies lead to very different results.

5.2.1 Reactive Outcome Sensitive Action Selection

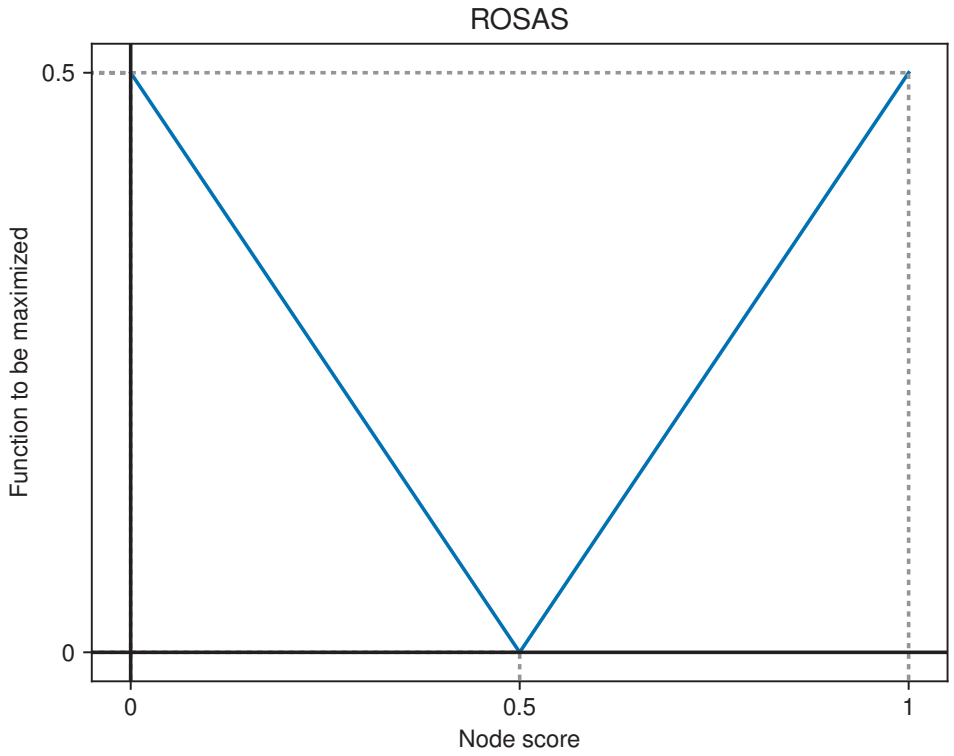


Figure 5.1 Action selection strategy used by *ROSAS*.

The first technique we propose is called *Reactive Outcome Sensitive Action Selection* (*ROSAS*). *ROSAS*, similarly to *CSAS*, uses value estimates that are higher the stronger an action is. In the situation of a fighting game, we set *ROSAS* to use MCTS with a score metric that is proportional to the HP difference (the score needs to be normalised to be between 0 and 1, as required by UCT).

ROSAS then chooses the action that leads to the score closest to 0.5. In the fighting game scenario, this means a HP difference of 0. Formally, *ROSAS* uses the following formula:

$$\text{action} = \arg\min \begin{cases} 0.5 - r[a].\text{score} & \text{if } r[a].\text{score} \leq 0.5 \\ r[a].\text{score} - 0.5 & \text{otherwise} \end{cases}, \tag{5.1}$$

where r is the root node of the tree and $r[a]$ is the child of r corresponding to action a. Basically, this is the distance from the score of 0.5. Figure 5.1 shows the function that *ROSAS* minimises.

5.2.2 Proactive Outcome Sensitive Action Selection

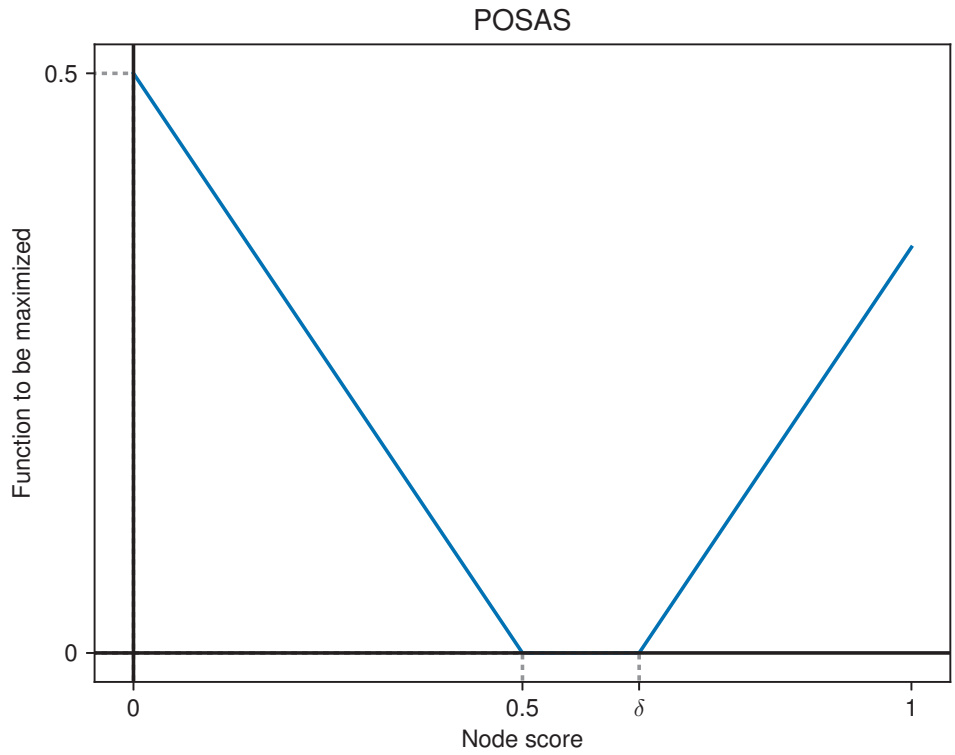


Figure 5.2 Action selection strategy used by *POSAS*.

A shortcoming of *ROSAS* is that the agent only fights back when it is losing. This can interrupt suspension of disbelief in the player, as it feels unnatural. To overcome this, we propose *Proactive Outcome Sensitive Action Selection* (*POSAS*), which attempts to solve this by keeping a more aggressive behaviour even if it is not losing. The idea is to treat scores slightly higher than neutral still as equally desirable. This way, *POSAS* will not wait to be put in a losing condition before fighting back.

The formula used by *POSAS* is the following:

$$\text{action} = \arg\min \begin{cases} 0.5 - r[a].\text{score} & \text{if } r[a].\text{score} \leq 0.5 \\ r[a].\text{score} - (0.5 + \delta) & \text{if } r[a].\text{score} > 0.5 \\ 0 & \text{otherwise} \end{cases}, \tag{5.2}$$

where δ regulates how much advantage the agent needs to have before becoming "docile" again, r is the root node of the tree and $r[a]$ is the child of r corresponding to action a. Basically, this is the distance from the interval $[0.5, 0.5 + \delta]$. Figure 5.2 shows the function that *POSAS* minimises.

5.2.3 *True ROSAS* and *True POSAS*

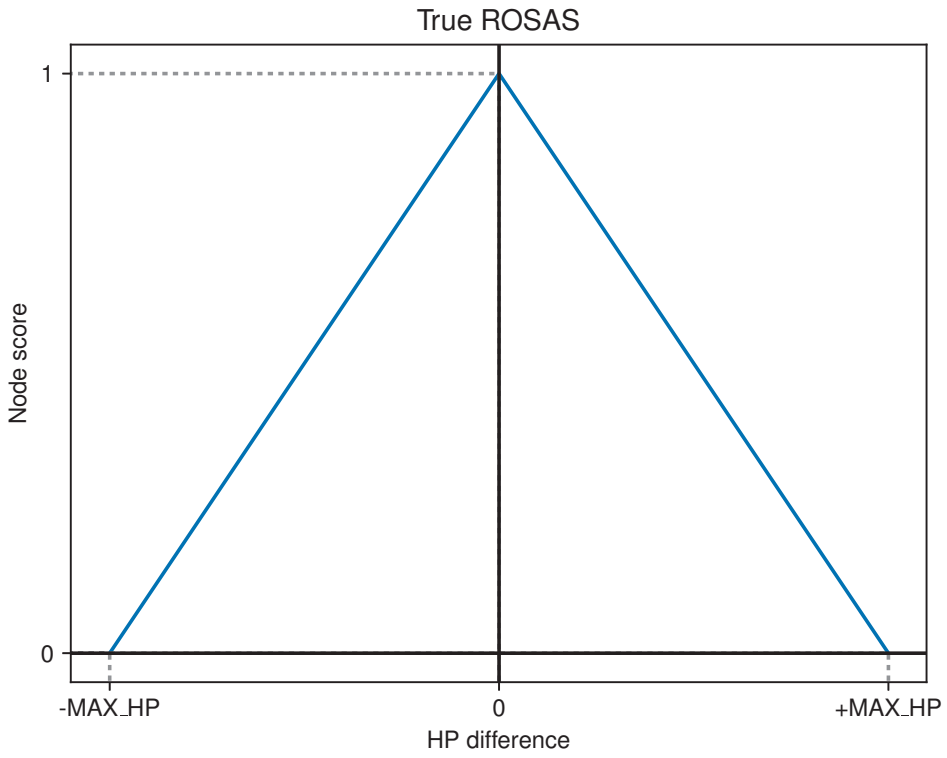


Figure 5.3 Score metric used by *True ROSAS*.

The main strength of MCTS when compared with other search techniques, such as minimax [79], is that it build the search tree asymmetrically, saving resources for

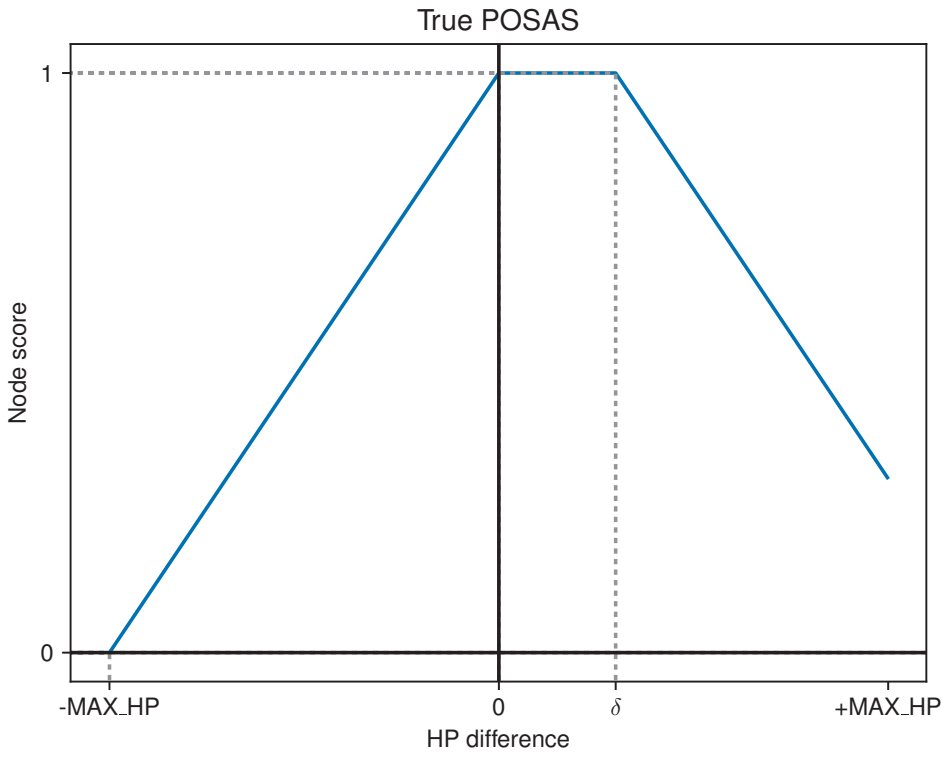


Figure 5.4 Score metric used by *True POSAS*.

branches that are more promising[1]. However, *ROSAS* and *POSAS* misuse this tool of asymmetry: since the score metric pushes exploration towards branches that lead to higher HP differences, the bulk of MCTS computation time is directed towards the wrong branches. The branches that we would like most time to be spent on are those leading to a low (in absolute value) HP difference.

To address this problem, we propose *True ROSAS* and *True POSAS*. These algorithms both use the default action selection strategy of MCTS: choosing the action with the highest score. However, they use different score metrics to both define what the score that should be maximised is and to drive the tree exploration towards branches maximising that score.

[1] $\alpha - \beta$ pruning can also avoid exploring some branches, but there is no guarantee it will be able to do so, because that depends on whether it can establish some bounds around the values nodes in that branch can assume.

True ROSAS defines the score metric as follows:

$$\text{node.score} = \begin{cases} \frac{\text{MAX\_HP}-x}{\text{MAX\_HP}} & \text{if HP difference} \geq 0 \\ \frac{\text{MAX\_HP}+x}{\text{MAX\_HP}} & \text{otherwise} \end{cases}, \tag{5.3}$$

where x is the HP difference after the action. Figure 5.3 shows the score metric function used by *True ROSAS*.

In the same spirit, *POSAS* defines the score metric as follows:

$$\text{node.score} = \begin{cases} 1 & \text{if } 0 \leq \text{HP difference} \leq \delta \\ \frac{\text{MAX\_HP}-x+\delta}{\text{MAX\_HP}} & \text{if HP difference} > \delta \\ \frac{\text{MAX\_HP}+x}{\text{MAX\_HP}} & \text{otherwise} \end{cases}, \tag{5.4}$$

where x is the HP difference after the action. Figure 5.4 shows the score metric function used by *True POSAS*.

5.2.4 Steeper variations

Assuming one of our adaptive agents is effective, it will maintain a small HP difference. In such situation, it may be more effective to use a score metric that is steeper around a zero HP difference and saturates at zero at larger HP difference values. To test this, we introduce steeper variations of *True ROSAS* and *True POSAS*. We define their score metrics as follows. Equation 5.5 shows the metric for *Steeper True ROSAS*, while 5.6 shows the metric for *Steeper True POSAS*. In both equations, the number 100 is an arbitrary choice, corresponding to 20% of the maximum players HP.

$$\text{node.score} = \begin{cases} \frac{100-x}{100} & \text{if } 0 \leq \text{HP difference} \leq 100 \\ \frac{100+x}{100} & \text{if } -100 \leq \text{HP difference} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.5}$$

where x is the HP difference after the action. Figure 5.5 shows the score metric function used by *Steeper True ROSAS*.

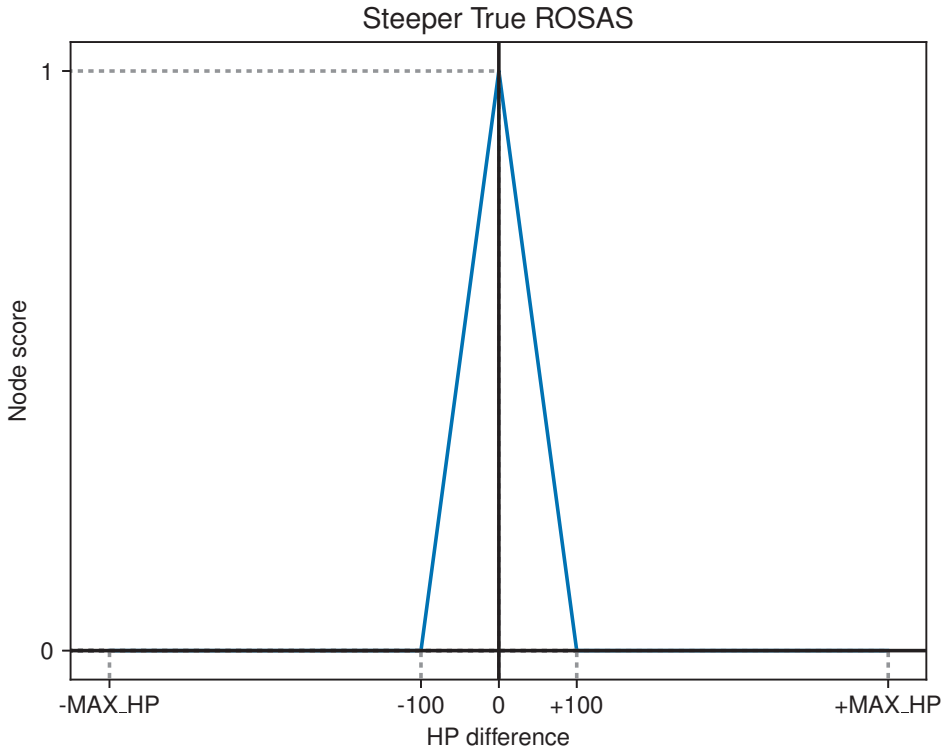


Figure 5.5 Score metric used by *Steeper True ROSAS*.

$$\text{node.score} = \begin{cases} 1 & \text{if } 0 \leq \text{HP difference} \leq \delta \\ \frac{100-x+\delta}{100} & \text{if } \delta < \text{HP difference} \leq \delta + 100 \\ \frac{100+x}{100} & \text{if } -100 < \text{HP difference} \leq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.6}$$

where x is the HP difference after the action. Figure 5.6 shows the score metric function used by *Steeper True POSAS*.

5.2.5 Adaptive variations

The steeper variations rely on the adaptive agent to perform well to begin with. However, this could not be the case. In such situations, if the HP difference grows too large, the score metric of the steeper variations will flatten and assume the value of zero, because the steepness is only within a range around 0. Ideally, an agent would adapt the steepness so as to be as steep as possible while making sure to

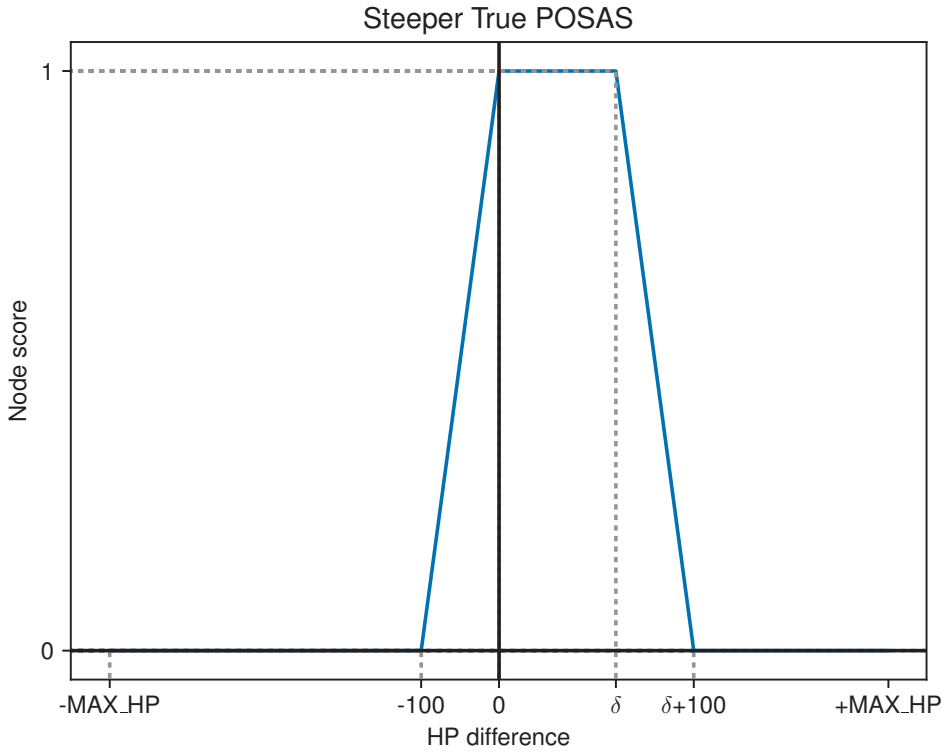
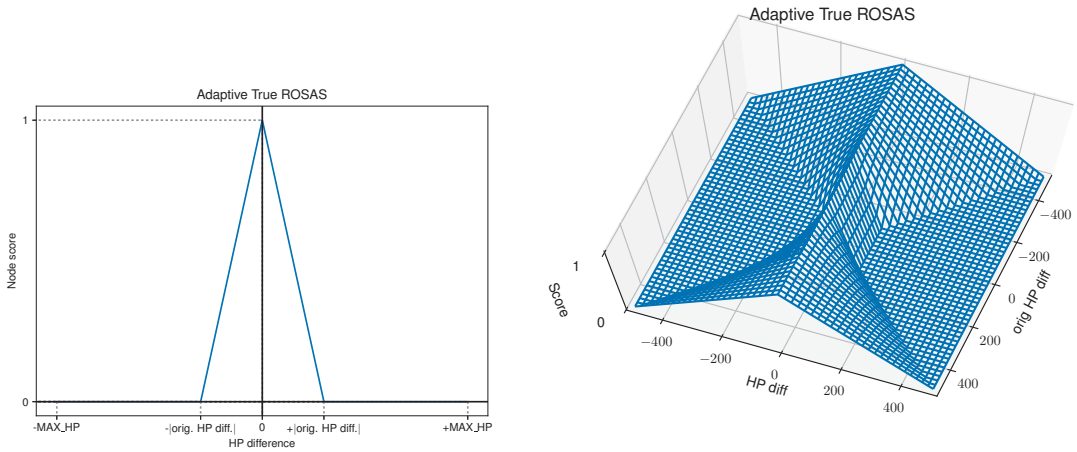


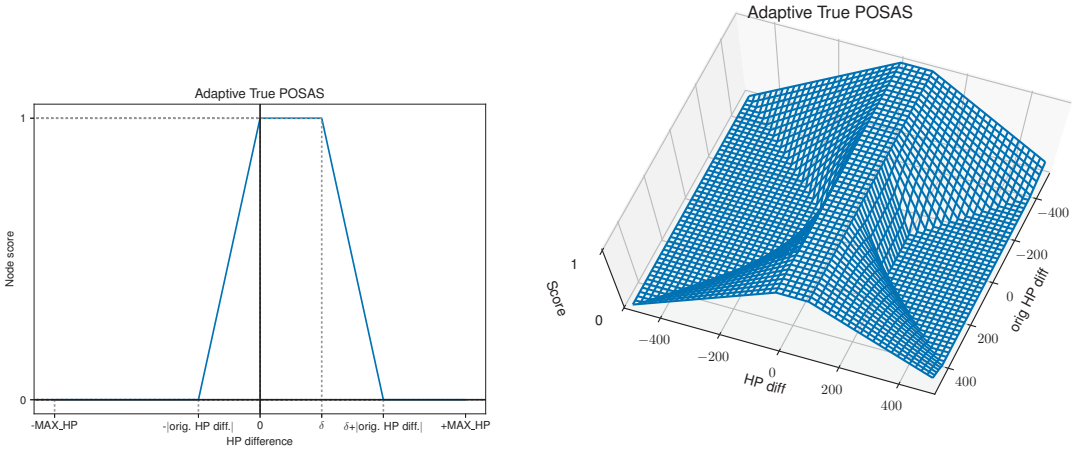
Figure 5.6 Score metric used by *Steeper True POSAS*.



(a) (b)

Figure 5.7 Score metric used by *Adaptive True ROSAS* visualised in 2D (a) and 3D (b).

always have a gradient towards the better actions. We propose adaptive variations of *True ROSAS* and *True POSAS* that achieve this. Notice that the score metric of



(a) (b)

Figure 5.8 Score metric used by *Adaptive True POSAS* visualised in 2D (a) and 3D (b).

such agents is bi-dimensional: the variables are the HP difference *after* the action and the HP difference *before* the action.

The formula for *Adaptive True ROSAS* is as follows:

$$\text{node.score} = \begin{cases} \frac{y-x}{y} & \text{if } 0 \leq \text{HP difference} \leq |\text{orig. HP diff.}| \\ \frac{y+x}{y} & \text{if } |\text{orig. HP diff.}| \leq \text{HP difference} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.7}$$

where x is the HP difference after the action and y is the HP difference before the action. Figures 5.7 (a) and (b) show the score metric function used by *Adaptive True ROSAS*.

In the same spirit, *POSAS* defines the score metric as follows:

$$\text{node.score} = \begin{cases} 1 & \text{if } 0 \leq \text{HP difference} \leq \delta \\ \frac{\max(100,y)-x+\delta}{\max(100,y)} & \text{if } \delta < \text{HP difference} \leq |\text{orig. HP diff.}+\delta| \\ \frac{\max(100,y)+x}{\max(100,y)} & \text{if } -|\text{orig. HP diff.}| \leq \text{HP difference} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{5.8}$$

where x is the HP difference after the action and y is the HP difference before the action. Figures 5.8 (a) and (b) show the score metric function used by *Adaptive True POSAS*.

5.3 Experimental setup

To assess if our approach can achieve a 50% win rate, we setup two experiments where we let each of our agents face a set of opponents. In the first experiment, our agents face a set of pre-programmed opponents. In the second experiment, the agents face a set of human players. We choose the Fighting ICE[2], a 2D fighting game, as a testing platform. The Fighting ICE has been used in the past as an environment for AI agent competitions.

Our agents are implemented in Java, so that they can interface with the game. In addition to *CSAS* and all our proposed agents, two MCTS variations are implemented. For a complete list, refer to Figure 5.9a. As opponents for the first experiment the entire set of participants to the Fighting ICE competition at the *Computer Intelligence in Games* (CIG) 2016[3] is used. The competition has received 14 submissions: for a complete list, refer to Figure 5.9b.

In the first experiment, we run 90 games for each combination of one of our agents and one CIG 2016 competition participant. Every other game we reverse P1 and P2[4]. The fights feature all the 3×3 combinations of characters available in the game (refer to Figure 5.9c).

For every agent we test, there are then

$$\underbrace{3}_{\text{AI characters}} \cdot \underbrace{14}_{\text{opponents}} \cdot \underbrace{3}_{\text{opp. characters}} \cdot \underbrace{2}_{\text{swap sides}} \cdot \underbrace{5}_{\text{repetitions}} = 1260$$

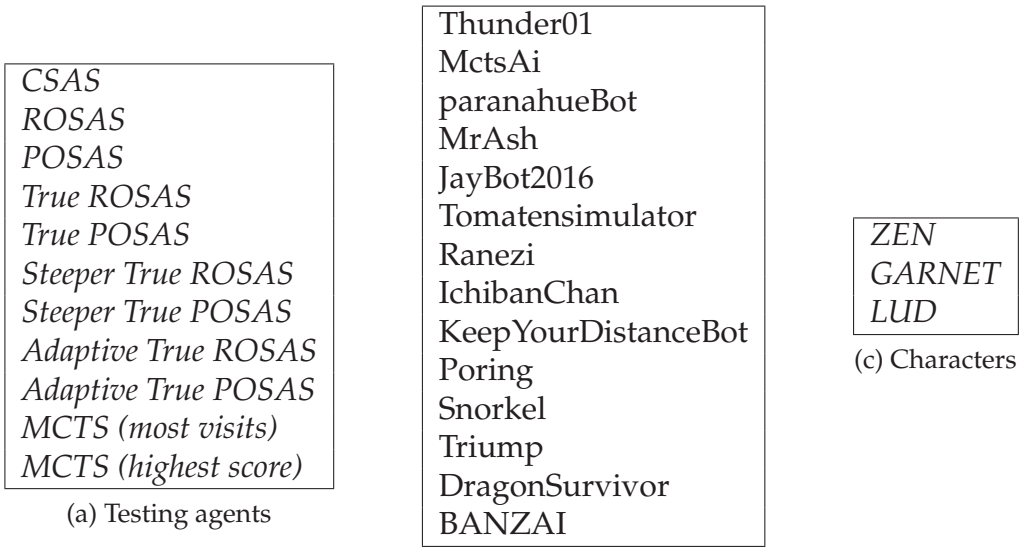
fights, and in every fight there are 3 rounds. We consider rounds separately, since they are unrelated from each other: this makes for a total of $1260 \cdot 3 = 3780$ rounds or data points for each agent. Hereafter, we refer to a "fight" as a single round.

In the second experiment, we face a selection of our agents with a set of 31 human players. The participants to this experiment are aged between 21 and 59, with an average of 26.8(standard deviation 10.7) and a median of 25. Of the participants, 25 are males and 6 are females. We asked the participants to fill a brief survey to gather information about familiarity and skill level with 2D fighting games prior the experiment. The average reported familiarity level, in a scale from 0 (never heard of the genre) to 3 (very familiar with the genre), 1.65 (equivalent to about 55%). The

[2]Developed by the Intelligent Computer Entertainment (ICE) lab. at Ritsumeikan University, Japan, and available at `http://www.ice.ci.ritsumei.ac.jp/~ftgaic/`

[3]Available at `http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/2016Competition.zip`

[4]The game is not exactly symmetric, and some bots behave better when placed on one side.

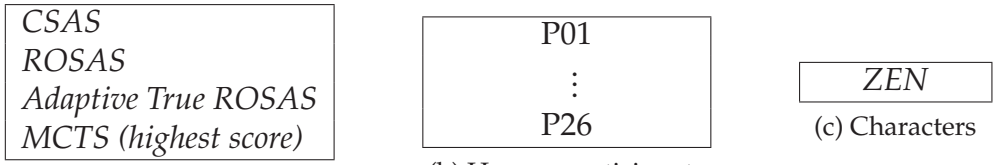


(a) Testing agents

(b) 2016 participants

(c) Characters

Figure 5.9 Agents and characters used in the experiment: in (a) the agents we are testing, in (b) the participants of the 2016 competition in order of final rank when playing the *ZEN* character, in (c) the characters available in the game.



(a) Testing agents

(b) Human participants

(c) Characters

Figure 5.10 Agents and characters used in the experiment: in (a) the agents we are testing, in (b) the participants of the 2016 competition in order of final rank when playing the ZEN character, in (c) the characters available in the game.

average reported skill level, in a scale from 0 (never played before) to 4 (very good), is 1.71 (equivalent to about 43%).

In the experiment with humans, we only use a subset of our agents in order to make experiments about 30 minutes long. We want to test a "non True" agent as well as a "True" one, along with *CSAS* and *MCTS* as baselines. We decide to use *ROSAS* and *Adaptive True ROSAS* as they do not have proactive behaviour and therefore represent the more basic versions of our AIs. The opponents are fought one after the other, with a short questionnaire before each fight. The order of the opponents is as follows: *ROSAS* first, then *CSAS* and *Adaptive True ROSAS* in random order, then *MCTS (highest score)* and finally *ROSAS* again. This specific protocol is chosen to use *ROSAS* to measure the players ability before and after fights. This data is used in a parallel study that is not reported in this thesis. Only the character *ZEN* is used in the second experiment.

Our purpose is to test whether the tested agents achieve a 50% win/loss rate and whether they achieve an average 0 HP difference at the end of the fights. Furthermore, we want to gather insights from human players regarding the perceived difficulty as well as realism and enjoyment for the various AIs.

5.3.1 Implementation details

All of the agents we test rely on MCTS, one way or another. The MCTS algorithm can be tweaked in many ways, and here we report the choices we made in our implementation.

Since the Fighting ICE is a real-time game, actions need to be computed at a high frequency: the game queries both participant bots at every frame. Since there are 60 frames per second, each frame lasts approximately 16.7 ms[5]. The engine also needs time to compute the effects of the previous frame actions on the game state, before handling it to the agents for the next decision. For this reason we allocate 40% of each frame to the engine and only allow MCTS to run for the remaining 60%. This is a conservative estimate, but we come to the conclusion that this is a better price to pay than being late. Being late, in fact, would mean executing the action in a different state than the state it was meant to; this, in a real-time game where pixels matter, can mean a great deal.

To allow MCTS more computational time when a quick response is unlikely necessary, we distinguish cases where the characters are at most 150 pixel away from

[5]The game actually relies on a graphic library that makes frames shorter if it is currently late on the schedule.

Agent	Mean	TOSB ($0.4 < p < 0.6$) p-value
CSAS	30.418	>0.999
ROSAS	29.242	>0.999
POSAS	34.220	>0.999
True ROSAS	29.674	>0.999
True POSAS	46.585	**<0.001**
Steeper True ROSAS	31.357	>0.999
Steeper True POSAS	47.233	**<0.001**
Adaptive True ROSAS	32.214	>0.999
Adaptive True POSAS	47.591	**<0.001**
MCTS (highest score)	78.312	>0.999
MCTS (most visits)	79.823	>0.999

Table 5.1 The victories percentage of the fights in the first experiment, against bots.

each other. When characters are close, we let MCTS run for 2 frames, we limit the tree depth to 2 and we limit simulations to 40 frames. When characters are more distant, we let MCTS run for 6 frames, we limit the tree depth at 8 and we limit simulations to 60 frames. In both cases, MCTS projects the current state ahead the proper number of frames (assuming no action from both characters) so as to make decisions that will be relevant when the time is up.

We only create children of a node after it has been visited 10 times. Each tree is adversarial, meaning that odd levels model actions of our agent, while even levels model actions of the opponent. We limit playouts to be 4 moves long, 2 for each character.

Finally, we set the constant for UCT to $C_p = 0.1$. This is the best value in a preliminary search we run.

5.4 Results and Discussion

In this section we report the results of the experiments. We also discuss in depth various aspects of the approaches of the agents.

Of all the fights we run, some failed for technical reasons. We present here the results of fights that did not fail and where some HP was successfully taken from either of the players. This selection step was necessary as sometimes the AIs get entangled in a loop of non-offensive behaviours (such as throwing kicks while out of range). This can happen to decision tree-based AIs when the rules do not predict

Agent	Mean	TOSB ($0.4 < p < 0.6$) p-value
CSAS	40.860	0.472
ROSAS	46.825	**0.037**
Adaptive True ROSAS	51.075	0.066
MCTS (highest score)	90.323	>0.999

Table 5.2 The victories percentage of the fights in the second experiment, against humans.

Agent	Mean	Std. err.	TOST ($\|HP\| \leq 125$) p-value
CSAS	−74.996	2.501	0.264
ROSAS	−35.442	1.454	0.041
POSAS	−33.394	1.560	0.049
True ROSAS	−35.974	1.373	0.033
True POSAS	−22.350	1.461	0.024
Steeper True ROSAS	−47.834	1.698	0.099
Steeper True POSAS	−34.258	1.765	0.074
Adaptive True ROSAS	−34.284	1.246	0.020
Adaptive True POSAS	−24.273	1.414	0.022
MCTS (highest score)	146.084	3.222	0.577
MCTS (most visits)	155.548	3.237	0.611

Table 5.3 Final HP differences of the first experiment. Initial HP is 500.

Agent	Mean	Std. err.	TOST ($\|HP\| \leq 125$) p-value
CSAS	−28.796	10.436	0.051
ROSAS	−12.304	3.842	<0.001
Adaptive True ROSAS	−13.194	4.321	<0.001
MCTS (highest score)	268.043	18.553	0.914

Table 5.4 Final HP differences of the second experiment. Initial HP is 500.

some situation or when they are designed to play defensively. In such situations, adaptive AIs are unlikely to initiate an attack, so in some cases the fight ends before they strike their first attack. Even MCTS-based AIs can get stuck; this is because of technical limitations: the trees generated by such AIs do not look far into the future due to the real-time constrains, and this can make all actions look equally valuable when the characters are too far away, because any damage would occur too far into the future for MCTS to take it into account.

Tables 5.1, 5.2, 5.3, and 5.4 report statistics of the data collected during the experiments. Let us focus first on tables 5.1 and 5.2. The tables show statistics of the percentage of victories for our agents in the fights, respectively for the experiment with bots and the one with humans. We can see that only three of the agents achieved an average percentage of victories close to 50% in the experiments with bots: *True POSAS*, *Steeper True POSAS* and *Adaptive True POSAS*. In the experiment with humans only two agents achieved an average percentage of victories close to 50%: *ROSAS* and *Adaptive True ROSAS*.

To quantitatively measure the confidence in these results, we model the distribution of the outcomes as a Binomial distribution, and we perform two one-sided Binomial tests, each of which measures the confidence that the actual parameter p of the Binomial distribution is, respectively, above 0.4 and below 0.6. In Tables 5.1 and 5.2, the highest of the two p-values resulting from the tests is reported. The tests on the data from the first experiment, with bots, place $> 99.9\%$ confidence that the parameter p of the Binomial distribution is indeed within $[0.4, 0.6]$ for the above mentioned agents. The tests on the data from the second experiment, with humans, place $> 95\%$ confidence for $0.4 < p < 0.6$ only in the case of *ROSAS*. This surprising result can be explained by the fact that, due to our peculiar protocol, we had twice as many data points for *ROSAS* than for any other agent, causing higher confidence in the statistical test.

Let us now focus on Tables 5.3 and 5.4. The tables show statistics of the final HP difference in the fights in the experiments with bots and with humans, respectively. Each player starts the battle with 500 HP. All of our agents achieve an average final HP difference closer to 0 than *CSAS*. To quantitatively measure the confidence that the average final HP difference is within an interval around 0, we run the two one-sided t-test (TOST), which is a statistical test for equivalence [57]. This test performs two one-sided t-tests having opposite orientations: one test has a null hypothesis assuming the mean of the population to be larger than $+\Delta$, the other assuming it is smaller than $-\Delta$, and reports the highest of the two p-values found.

In Tables 5.1 and 5.4, the value of Δ is reported in brackets, while each cell reports the p-values of the test. The tests place $> 95\%$ confidence that most of our agents (except the "Steeper" variations) achieve an average final HP difference within ± 125 HP against bots. An even higher level of confidence is achieved ($> 99.9\%$) in the experiment with humans. The better performance of the agents against humans compared to bots can be explained by the relatively low skill level reported by our participants. As will be discussed later, a fundamental limitation of adaptive agents is the maximum skill level they can play at: the lower the skill level of an opponent, the easier it is for the adaptive agent to match the skill level.

These results suggest that most our agents can track the HP of the opponent and adapt to it, and some of them reliably achieve a 50% victory rate. This is convincing evidence that our algorithms achieve their purpose. Furthermore, the numbers seem to support the idea that our agents outperform *CSAS*.

Finally, the results of the MCTS agents show that the near-0 average final HP difference achieved by the adaptive agents is not the natural result of MCTS, but a specific outcome of our algorithms.

5.4.1 Discussion

Our algorithms closely match the desired outcome of a 0 HP difference, suggested in previous work [4] as an appropriate level of difficulty. Some of the algorithms achieved a 50% victory rate, which has been suggested in the literature [110, 60, 43] as an appropriate level of difficulty. However, not all of them achieved this result; in fact, one can notice that only proactive agents succeeded in this. This is due to the inherent nature of the algorithms responding to the players action. Purely reactive agents, in a situation of low health for both players, will take damage before dealing any, and therefore lose more often than not. Proactive agents partially deal with this aspect. This can be seen in table 5.1, where the algorithms in the *ROSAS* family achieve a slightly lower mean compared to those in the *POSAS* family. It seems that further work in finding the exact balance between reactivity and proactivity could achieve even better results, but there is no guarantee on the robustness of such agent. For example, in a different environment, with different initial HP, or faster/slower or more/less powerful moves, performance could slightly improve or slightly worsen.

The added value of a 50% victory rate lies in the uncertainty of the outcome which keeps the player on the edge. Considering this, we suggest that another (possibly more accurate) metric for such uncertainty could be the HP difference between the players throughout the game. A HP difference close to 0 throughout the

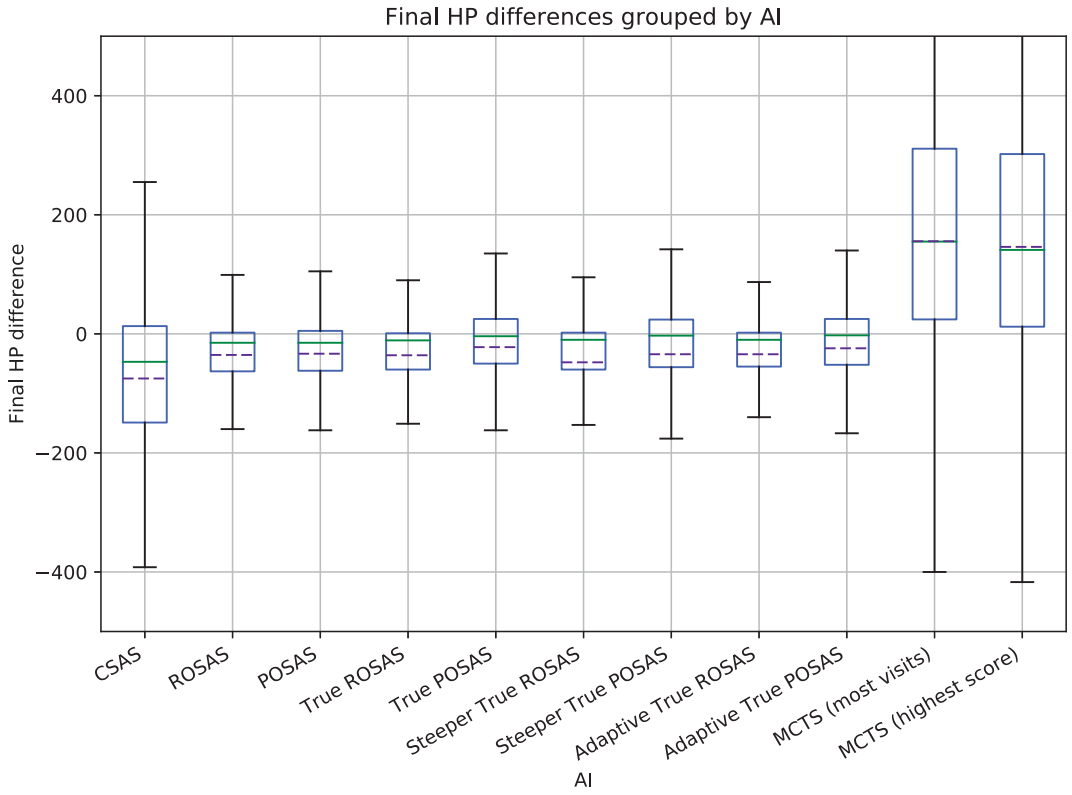
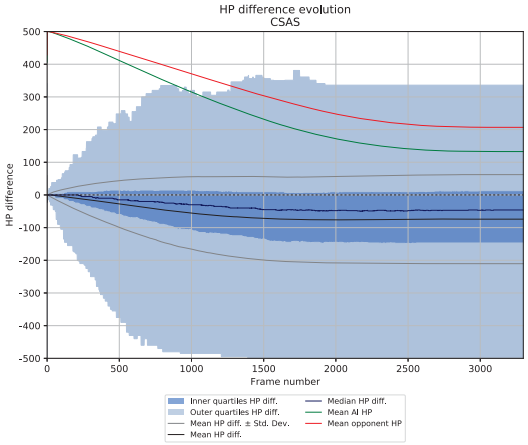


Figure 5.11 Boxplot of the final HP difference distribution for each tested agent.

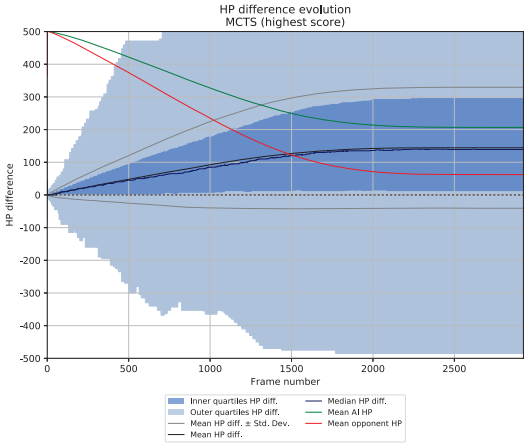
game is a symptom of an unpredictable game, which could keep the player on the edge for a longer span of time. For this reason we speculate that there could be more value in keeping the player on the edge of losing rather than achieving an exact 50% win rate. In this sense, our agents seem to perform well. This can be observed in Figure 5.11, which shows the evolution of the HP of both characters as well as their HP difference; in particular, the figure shows the distribution of the HP difference by representing quartiles using different colours. We can observe that our *OSAS agents track a 0 HP difference more accurately and precisely than *CSAS*. Notice that the HP of the players seems to never reach 0 because in some fights the AIs get stuck in a non-attacking cyclic pattern.

Rank vs Score

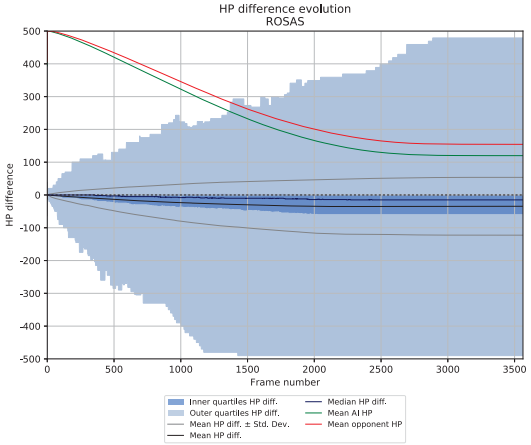
As explained in Section 5.1, one of the limitations of CSAS is in its strategy for selecting stronger or weaker actions. The rank, while being a proxy for actions strength, is not the best: the median action, for example, can be stronger or weaker



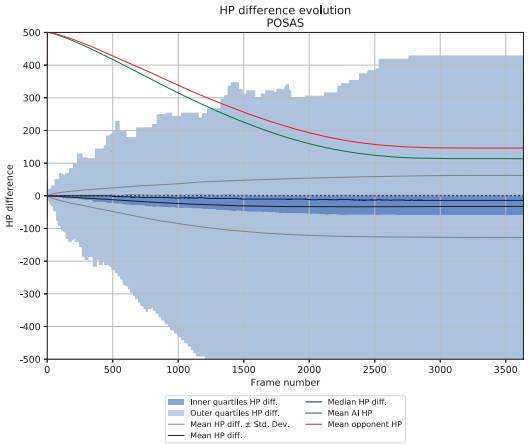
(a) CSAS



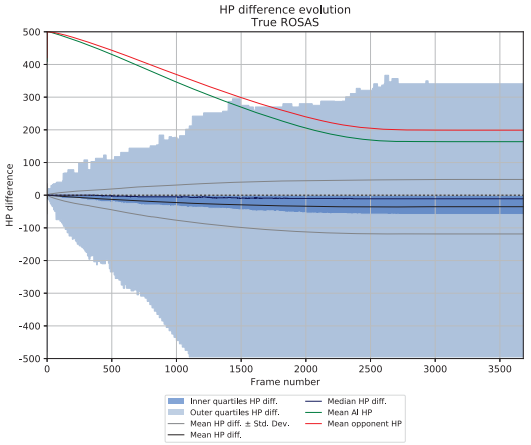
(b) MCTS (most visits)



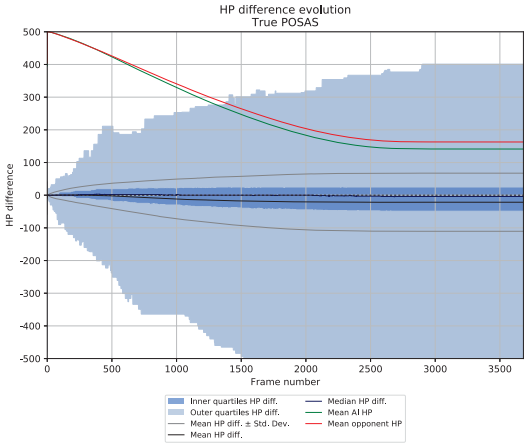
(c) ROSAS



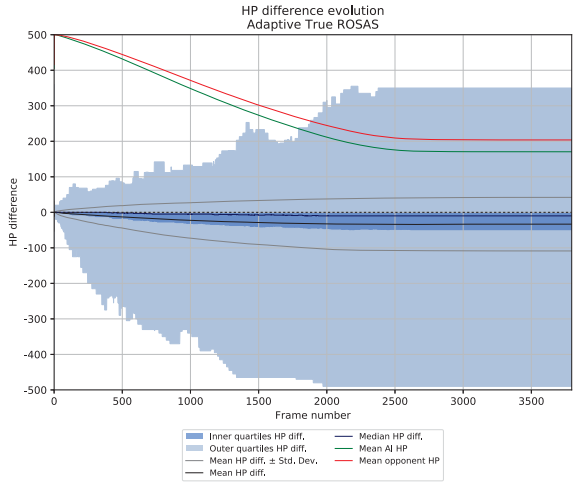
(d) POSAS



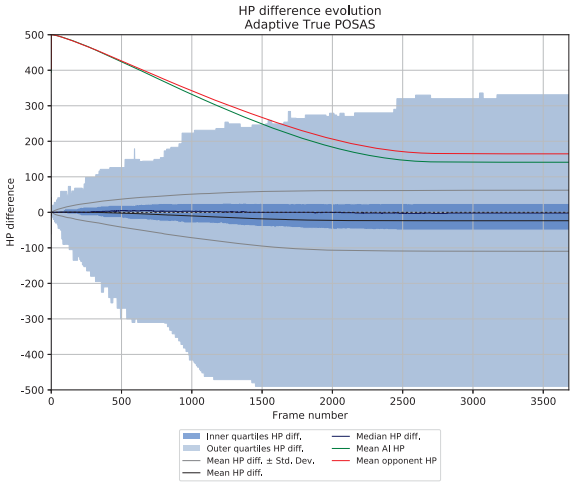
(e) True ROSAS



(f) True POSAS



(g) Adaptive True ROSAS



(h) Adaptive True POSAS

Figure 5.11 The evolution of HP of the two players and their difference during fights. The data is aggregated across fights.

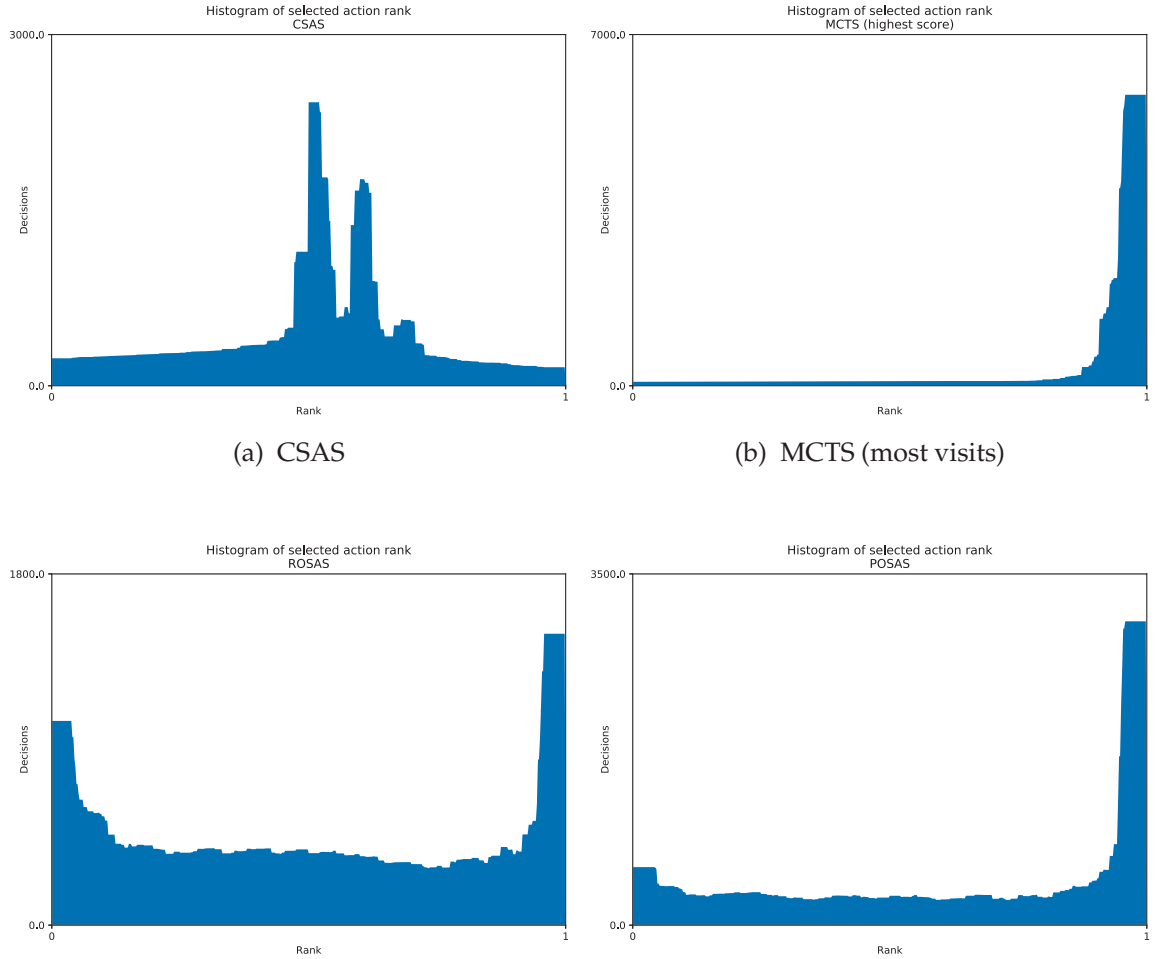
depending on the situation. We analysed the data generated by our experiments to observe this phenomenon directly. Figures 5.12 show the histograms of the rank of selected actions by different AIs. CSAS, in Figure 5.12b, tends to choose actions ranked in the middle. However, *ROSAS* and *POSAS*, in Figure 5.12c and 5.12d, tend to choose actions more at the extremes of the ranks spectrum.

Since *ROSAS* and *POSAS* choose actions more likely to yield a 0 HP difference and considering the heavy tails shown in these plots, it seems that choosing actions in the middle as often as CSAS does is not very effective to achieve 0 HP difference outcomes. This matches the results shown in Table 5.3.

In fact, we can look at the histograms of the score of selected actions, shown in Figure 5.13. It is evident that CSAS tends to choose its actions in a more restricted "score" region. This may also be due to the its delay in adapting to the opponent ability, which causes CSAS to choose stronger/weaker actions later, and therefore less frequently than desirable.

Limitations

As effective as it can be, an intrinsic limitation of all DDA agent systems is that the difficulty level provided by the agents is limited by the maximum level the AI can play. In this case, our DDA agents are all limited by the maximum strength of MCTS. Figure 5.11 also shows the results from an agent that uses MCTS with the aim of



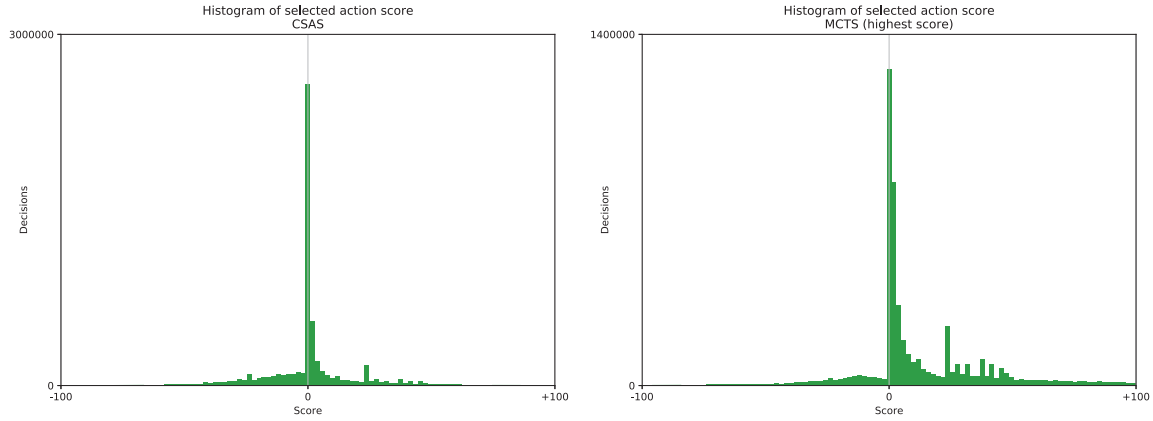
(a) CSAS

(b) MCTS (most visits)

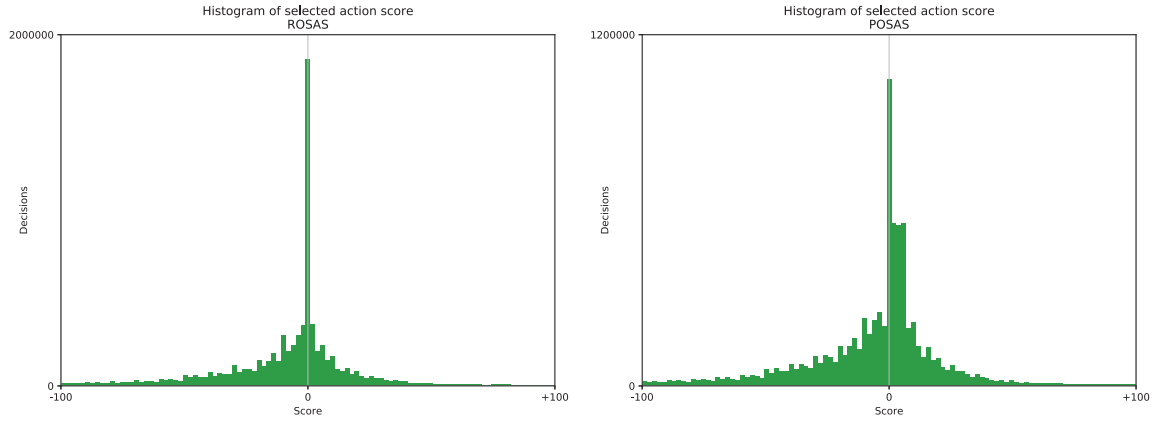
(c) ROSAS

(d) POSAS

Figure 5.12 The histogram of the rank of selected actions for different Adaptive AIs. Since more actions can have the same value, the rank is computed as an interval: in the histogram, for every action we computed the minimum and maximum rank a and b and increased all the points between a and b by $\frac{1}{b-a}$. The data is aggregated across fights.

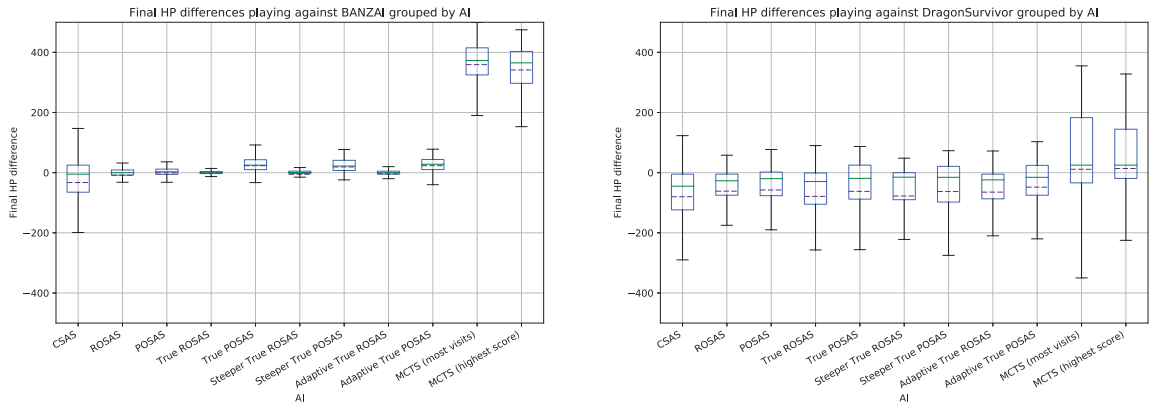


(a) CSAS (b) MCTS (most visits)



(c) ROSAS (d) POSAS

Figure 5.13 The histogram of the score of selected actions for different Adaptive AIs. The data is aggregated across fights.



(a) vs BANZAI (b) vs DragonSurvivor

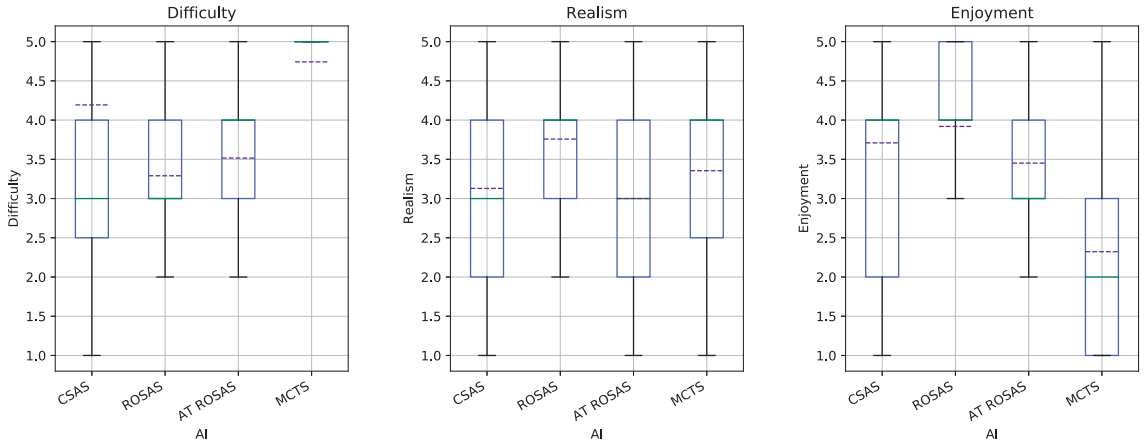
Figure 5.14 Boxplot of the final HP difference for each tested agent against specific opponents. The data is aggregated across fights.

winning rather than providing an adaptive level of difficulty. In these experiments the MCTS agent seemed able to deal with different opponents. However, it may be the case that a stronger bot or a professional human player can consistently beat this traditional MCTS agent. In this case our agents will not be provide a suitable level of difficulty for that bot or player.

This phenomenon can begin to be observable in our experiments themselves: not all of the opponents were at the same level. Figure 5.14 shows boxplots of the final HP difference of the tested AIs against two specific opponents. We chose these opponents specifically to highlight how performance of our agents change relative to the maximum performance they can achieve (which is similar to that achieved by MCTS). Specifically, Figure 5.14a shows the performance of the agents against *BANZAI*, quite a weak opponent: the performance of the adaptive agents is very good, targeting a 0 HP difference with small variance; notice how well MCTS performs against *BANZAI*. On the other hand, Figure 5.14b shows the performance of the adaptive agents against *DragonSurvivor*, quite a strong opponent: the performance of the adaptive agents is quite poor (in terms of targeting a 0 HP difference), as is that of MCTS (in terms of achieving a large positive HP difference).

Future Work

The agents tested in the experiments implement piece-wise linear functions, as defined in Section 5.2. However, smooth functions could be tested instead, such



(a) Difficulty (b) Realism (c) Enjoyment

Figure 5.15 Boxplot of the results of our surveys. The blue rectangles include the inner quartiles, the green line indicates the median and the dashed purple line indicates the mean.

as Gaussian-shaped ones. This could solve the problem discussed in Section 5.2.5, because such functions would never actually reach 0 and always provide small differences in action values that would help rank the actions even when the HP difference is far from 0.

Human perception

In our experiment involving humans, we asked our participants to complete a small survey after every fight, to assess the perceived enjoyment, realism and difficulty against every opponent. In particular, we asked every player to rate each fight on a scale from 1 (least) to 5 (most) in term of difficulty, realism and enjoyment. Figures 5.15 (a)–(c) show the results of the surveys in boxplots.

As expected, the most difficult agent was *MCTS (highest score)*. At the other end of the spectrum of difficulty is *ROSAS*, being perceived as fairly difficult, but not to an extreme extent. Notably, *MCTS (highest score)* was also the least enjoyable, while *ROSAS* was the most enjoyable. Finally, *ROSAS* is also the most realistic of the agents.

While some of these differences are small, the overall picture seems to be that the agent preferred by the participants is *ROSAS*. These results show a preference for a "non True" agent over a "True" one, while the results of the study on victory rates and final HP differences show that "True" agents have better performance.

A possible explanation is that "True" agents, while achieving quantitatively better performance, are too responsive in their matching their opponent HP, resulting in a higher perceived difficulty and possibly disrupting disbelief. Further research in this direction is needed to investigate these aspects.

5.5 Summary

In this chapter, we proposed and tested novel techniques in the area of Dynamic Difficulty Adjustment (DDA). Our techniques use Monte Carlo Tree Search (MCTS) with a variety of action selection policies and utility functions. Our approach aims at achieving a 50% win rate; however, due to practical constraints, we used a proxy instead, targeting a zero HP difference in a 2D fighting game.

The experiments conducted suggest that our technique performs better than a related state-of-the-art technique, Challenge Sensitive Action Selection (CSAS) [4], in that it achieves more reliably both a small final HP differences and a victory rate closer to 50%. The experiments show the robustness of our approach, testing our DDA agents against adversaries with a variety of skill levels. Finally, our techniques require less parameter tuning than CSAS.

This chapter showed a technique to adapt the skill level of an automatic agent in a video game to that of the opponent; that is, a technique to achieve DDA. The next chapter reports a study that shows the potential of DDA in theme parks scenario, where having visitors lose track of time (while waiting for an attraction) can have a positive impact on the memory of the experience and increase customers return rate.

One of the most sought-after experiences by gamers is immersion. Since DDA can ease the way into a state of immersion, more research in the area is desirable. This research contributes to the area by proposing a practical and well-grounded approach to the problem.

Chapter 6

Opportunities for AI Techniques in Theme Parks

If you have ever been to a theme park, especially during a peak season, you know the frustration of being stuck in queues. While a theme park inherently promises a day of novel experiences and family-friendly entertainment, the reality is that customers spend many hours of their day simply waiting for these experiences to begin. In fact, queuing times at popular theme parks can be hours long for a single attraction [44].

On-going research and industry developments aim at reducing queue times by carefully structuring the layout of the park to improve customer flow [1], providing queue skipping mechanisms to allow for structured load-management [21], or scheduling staff rosters and live show times to increase attraction turn-over rates or entice customers away from specific attractions at busy times [10].

However, as with any overloaded system, there is a limit to how much actual queuing time can be reduced. Thus, rather than (or in addition to) attempting to reduce the *actual* queuing time, theme park managers may instead aim to reduce the *perceived* waiting time. That is, to induce a sense of time passing quickly in waiting customers by improving the quality of the queuing experience [55].

The problem of long waits stems from boredom, a state of mind in which time seems to slow down. This is caused by a lack of stimuli provided by the queuing environment, which causes customers to redirect more attention to the passage of time [114]. The opposite of boredom, at the other end of the spectrum, is "flow", a state of peak enjoyment and high focus experienced by individuals engaged in play [25]. Theme park managers would much rather have their waiting customers in the latter rather than the former.

In the context of a research project with a theme park industry partner, we developed an Augmented Reality (AR) video game prototype and we deployed it in a theme park queue. Considering that video games are a popular avenue for people to reach the state of flow [94, 24], here we present the results of our in-the-wild experiment and argue that video games, in the specific context of theme park queues, can greatly benefit from the use of Dynamic Difficulty Adjustment (DDA) to 1) engage a diverse audience, made up by children as well as adults of different cultures, genders and gaming experiences, 2) overcome the lack of familiarity that may stop customers from quickly giving up with the game and 3) increase the longevity of the game by offering an increased challenge as players become more skilled.

While research in DDA exists [111, 2, 7, 8, 31, 112, 47], limited adoption in industry settings is an indicator for the limitations of the existing techniques, which can come short in effectiveness and realism. We hope this study will motivate more work in the area by showing the opportunities of DDA in a real world industry scenario. For example, the technique proposed in chapter 5 could be tested in an real-world queue environment.

This chapter is structured as follows: section 6.1 gives an introduction to the concept of attention and flow in time judgement psychology; section 6.2 details the design of the game we developed; section 6.3 describes the setup of our experiment; section 6.4 reports and discusses the results of the experiment; section 6.5 discusses the limitations of this work.

6.1 Attention and Flow

Time judgement, the process by which the brain perceives the passage of time, is sensitive to the division of mental attentional resources between temporal and non-temporal tasks [114]. In particular, the more demanding non-temporal information processing is, the more attentional resources are consumed by it, leaving fewer resources for timing [113]. When too much resources are dedicated to temporal tasks, the mind enters the state of boredom, an unpleasant state of mind that can cause anger and anxiety and has been linked to various types of misbehaviour [114]. At the other end of the spectrum, compared to boredom, is flow. Flow is a mental state of peak enjoyment and deep focus, popular among game researchers for its peculiarity that people experiencing flow lose track of time. When in a state of flow

[25], attentional resources are almost fully allocated for non-temporal information processing and, as a result, time judgement is minimised.

One more advantage of flow, from a game perspective, is that a player immersed in the game will continue to play the game [24]. To ensure that players remain immersed in a particular video game, the game must not cause frustration nor boredom, all too common emotions in games that are, respectively, too difficult or too easy. To avoid this, the level of challenge presented by the game to the player needs to be tuned to the player's skill level in that game. If the level of challenge is appropriate for the individual player, the player is more engaged; otherwise, the player may become frustrated or disengaged [18].

The process of adapting the difficulty of a game involves changing the strategies and behaviour of the AI opponent or characteristics of the environment to match the skill level of the player. This also produces a video game that is longer-lived, since the game can alter its level of challenge as the skill level of the player progresses. The area of Dynamic Difficulty Adjustment (DDA), a sub-area of Artificial Intelligence, studies techniques that aim at addressing these problems and is an active area of research. Existing DDA research is presented in Section 2.3.5.

Games not using DDA (the vast majority), usually ask the player what level of challenge they would like, in the form of a choice between options such as "easy", "medium" or "hard". These static difficulty settings are used by developers to group players into a limited number of skill ranges.

This approach has a fundamental limitation: it divides the world in a small number of categories, which is not guaranteed to capture the diversity of the players base. This problem is exacerbated in theme parks, where the customer base is quite heterogeneous, comprising of children and adults, boys and girls, gamers as well as non gamers, etc. Theme parks, therefore, are an excellent example of real-world scenario that will benefit from the introduction of DDA in its digital interactive technologies.

In this study, we do not develop a DDA system; rather, we conduct an experiment to motivate further work to investigate DDA in theme park contexts.

6.2 Game design

We develop a video game based on Augmented Reality. This choice of technology is driven by the potential for AR to build upon the theme of the ride and enhance the queuing experience; this can help to prevent the suspension of disbelief of the

player from being broken. Furthermore, it gives the player the feeling that the ride has already "started", which has been suggested as making longer waits acceptable to the waiting customer [59, 28].

The game is playable on mobile devices such as Android or Apple smartphones. With their device in hand, the queue members can point the camera towards special AR markers that are placed on the sides of the queue every few meters. While the game app normally shows a grey-scale representation of what the phone camera sees, when scanning a marker, an augmented reality portal opens up above the marker. This initiates the beginning of a level, after which enemies and collectables gradually spawn out of the portal. It is the task of the players to (individually or as a group) destroy the enemies by tapping with their fingers on them; at the same time, the must drag enough collectables to clear the level. The natural flow of the queue environment provides a time limit for the level: when there is an open space ahead of the players, social pressure makes them move on from the AR marker. To mitigate risk to expensive smartphones, the gameplay requires "turrets" to be pressed upon at all time: when a turret is not pressed, no projectiles will be generated from it; this should result in a player firmly holding the phone while the others tap on the screen. Once the level is completed, the player needs to find a new marker to play the next level. The game gradually intensifies with more and stronger enemies as the player progresses through levels, coupled with more powerful weapons and upgrades that can be bought (with points, not money) in a virtual shop between the levels.

Normal gameplay is periodically interrupted by a message that a "boss" enemy is appearing. Players are then required to locate a special boss AR marker with the smartphone camera. A separate but similar portal with another shield dome is augmented over the boss AR marker. The boss is represented by a large single enemy that slowly flies towards the shield at a slower rate compared to normal enemies. Unlike the standard enemy portals, all players in the queue will see and attack the same boss regardless of which boss AR marker they are viewing and advance a level if the boss has been successfully defeated by the collective effort. The boss is defeated if it receives sufficient laser taps from all players present in the queue before reaching the shield. The number of taps required to defeat the boss dynamically scales with the number of smartphones actively playing in the queue.

6.3 Experimental setup

In the following section we provide details of an in-the-wild user study that utilised the prototype in a real world theme park queue. A public study was chosen over a controlled user study because we are targeting perception of waiting time in the context of theme park queues and members of these queues embody specific emotions and expectations. Synthesising a controlled queue from recruited participants would impact the results of study due to the different state of mind of the participants.

6.3.1 Study Site and Demographics of Participants

The study was conducted at the Movie World theme park on the Gold Coast, Australia. The study was run over three days from Friday 6th to Sunday 8th of January, 2017. These dates were chosen to gain access to peak crowds during the Australian primary and secondary school summer holidays. The experiment was run during the entire 9am to 5pm operating hours of the theme park. However, due to queues being too small at some periods of the day, data was primarily acquired between 11am and 4pm on all three days.

While the prototype was designed for the queue of the sci-fi themed Justice League attraction, the experiment was instead conducted at the western themed Wild West Falls attraction. This was due to operational concerns during the chosen dates, including a larger expected queue at the Wild West Falls attraction. While feedback from a few participants and nonparticipants indicated that the clash in theme was noticed, it did not appear to significantly impact the experience of the participants.

In total, 100 groups of participants played the game and 25 of these groups conducted the post play interview. While we did not formally collect demographic data, we observed that most groups of players were young families with at least one member under the age of 18 and that almost all participants were fluent in English. This aligns with the family oriented nature of the Movie World theme park and general park admission statistics that identified roughly 60% of guests as being local Australian residents. On average, groups played for 15 minutes and 10 seconds, while the median value is 13 minutes. Figure 6.1 shows the distribution of the waiting times.

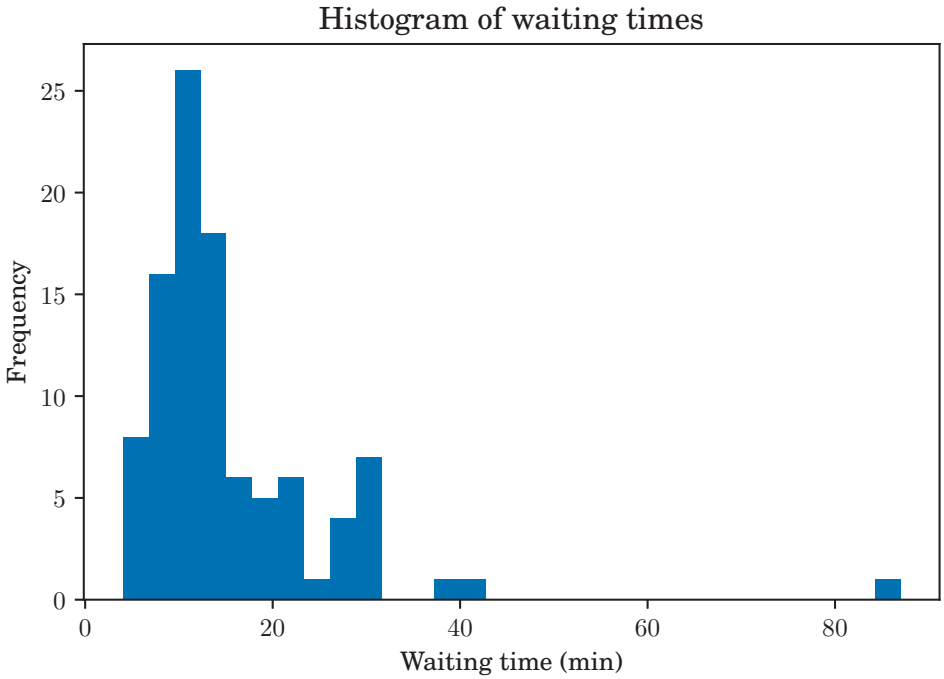


Figure 6.1 Distribution of waited time in our sample

6.3.2 Experimental Design

The prototype was installed onto eight Samsung Galaxy S5 phones that were reused throughout the three experimental days. The prototype included instructions on how to play the game as well as information for participants regarding the experiment. These devices were connected to the local server via WiFi to coordinate boss battles and to store survey results. The devices were locked such that participants could only play the game on the phone and not use it for any other purpose. Each device was given both a physical identifier (marked on the back of the phone) and a digital one.

Participants fell into one of two groups; players and non-players. Players were recruited by the investigators by occasionally approaching the back of the queue, where they would demonstrate the game and describe the experiment to the last group of guests in queue and ask whether they would like to participate. If surrounding groups enquired about game, they were also presented with the opportunity to participate. Signage was also posted around the entrance to the attraction to advise theme park guests of the experiment. Only one device was provided to each group of family or friends and they were encourage to play together on the one device.

Players were told that they could play the game as little or as much as they liked and were requested to return the device to another investigator located at the head of the queue just before they got on the attraction. The recruiting investigator then recorded the device identifier and the time that it was provided to the players.

Of note is that it was initially planned to recruit participants at the entrance to the attraction queue, out of sight of the actual queue. However, this proved ineffective as most potential participants were in a hurry to join the queue and therefore declined to hear more about the game. This relates to the queue design dimension of queue members wanting to get started; guests were eager to join the queue and reserve their spot in queue but once they were already waiting they were more receptive to being provided with additional entertainment.

At the head of the queue, when a device was returned to the investigator located there, a ten question Likert-style survey was loaded (see Section 6.4.4) on the device and the players were asked to complete it as a group, with the results being registered against the devices digital identifier and stored on the game server. After the survey was completed, the investigator asked the group to estimate, without looking at a clock, the amount of time that they had been in queue since they were provided with the device. Where estimations within the group varied and no consensus was reached, each estimation was recorded. These estimates, along with the device identifier and the time that the device was returned, were then recorded.

Non-playing participants were recruited in a similar fashion to playing participants but instead of being provided with a device they were simply provided with a token with a unique identifier on it and asked to return the token to the investigator at the head of the queue. Once at the head of the queue, the non-players were also asked to estimate their wait time since being given the token and these details were recorded similarly to playing participants. Non-players did not answer the survey or participate in the interviews as most of the questions in these were aimed at qualitatively analysing the queue experience in relation to the gameplay experience.

6.3.3 Data Collection and Analysis

Through this experimental design, we are able to track the progression of players and non-players through the queue. Actual wait time is calculated by taking the difference between the time that the participant is at the back of the queue and the time he or she is at the front of the queue. The degree of time misperception is then calculated as the difference between the actual wait time and the perceived wait time (participants' time estimations). The game also recorded how long players were

interacting with the game during their total time in the queue, giving a measure of engagement with the game.

Non-player participants provided a baseline and by comparing the degree of time misperception of the two groups we determine the impact that our prototype had on perceived waiting time. Additionally, survey data provided a quantifiable perspective on the player experience while the interviews allowed for more expressive and open ended qualitative data to be collected. The results of analysing the data are provided in the next section.

6.4 Results and Discussion

We validated our hypothesis using data collected during the experiment. In this section we provide a quantitative analysis of numerical data. Our hypothesis is that playing with our game while waiting in line makes players perceive time as passing more quickly. To test this, we computed the perception error for players and non-players, as described in Section 6.3.3. Table 6.1 reports statistics for the samples and Figure 6.2 shows a box-plot of their distribution. Notice that the mean/median perception error is more negative in players than in non-players. In the following sections we analyse the statistical significance of this difference and we present other evidence supporting our hypothesis.

6.4.1 Statistical Significance and Effect Size

We compared the perception error of players and non-players using a two independent samples t-test and a Mann-Whitney rank test. Both tests reported p-values < 0.001, indicating that the difference observed is very likely not due to chance. We also computed Cohen's d coefficient to estimate the effect size of playing on time perception error (using the formula provided in [20]). The value found indicates that the effect size is medium, suggesting that the difference found is of concrete significance. Table 6.2 reports the results of these tests in detail.

6.4.2 Correlations

To verify whether some of the variables we collected correlate with time perception error, we computed the Pearson's correlation coefficient r for some pairs of variables. This coefficient tests for linear correlation and can range between -1 and +1, it gives

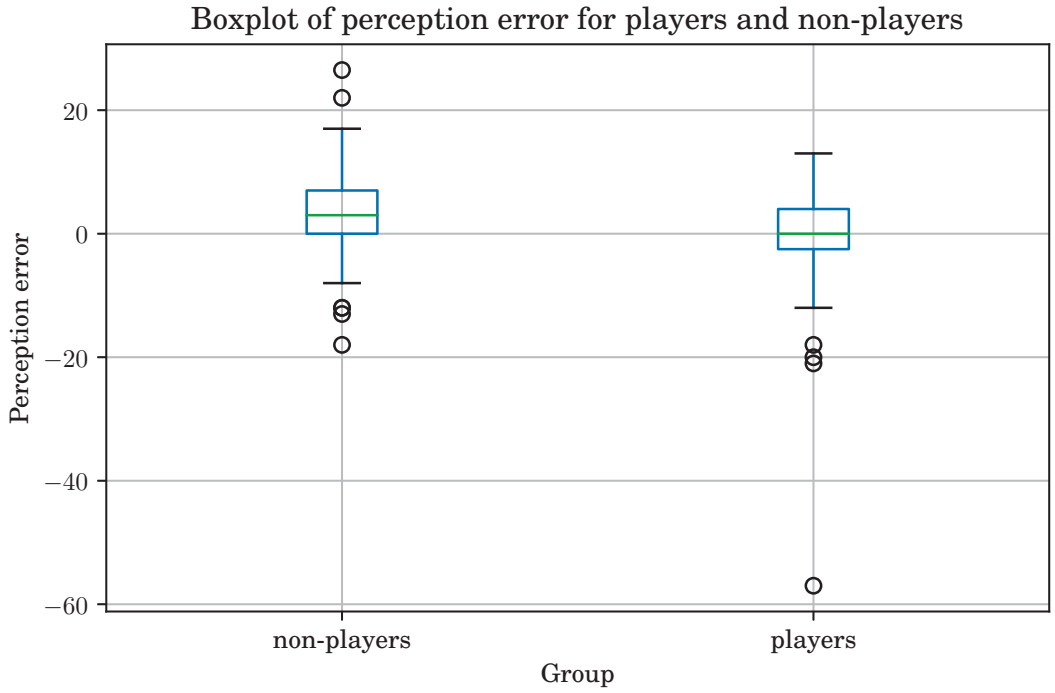


Figure 6.2 Box-plot of perception error for players and non-players

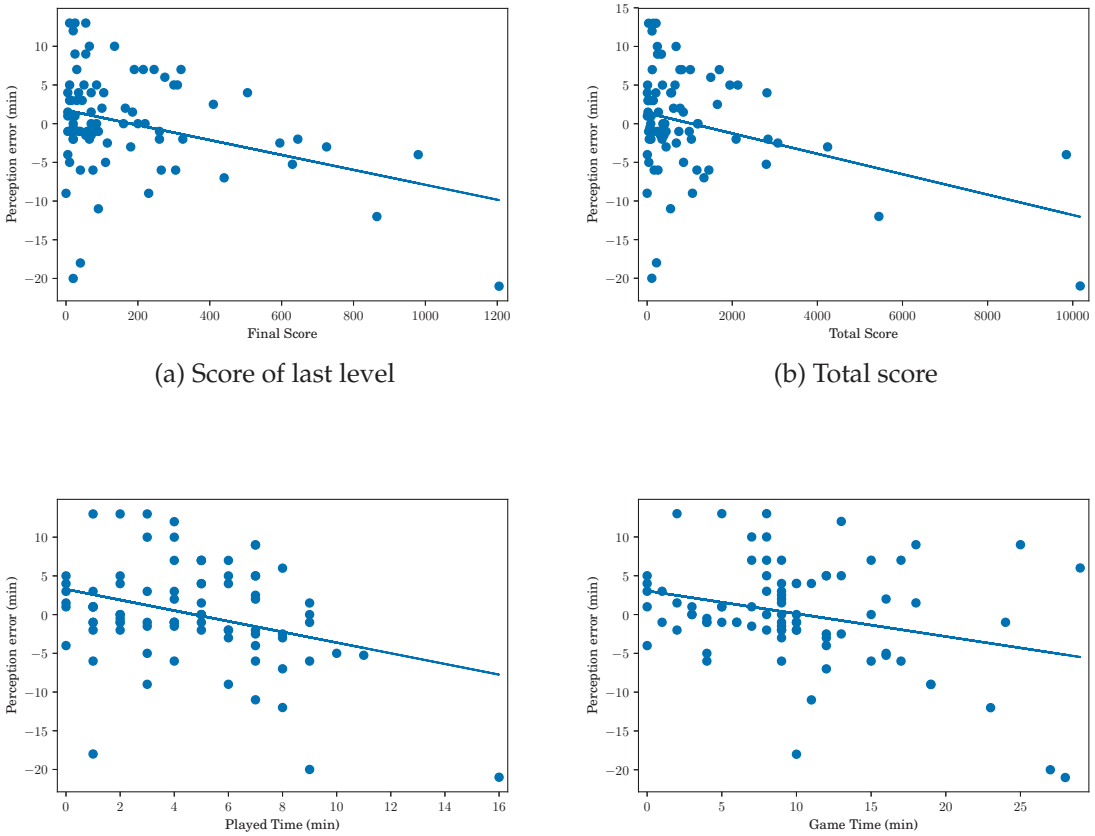
Group	Count	Mean (min)	Std. Dev. (min)	Median (min)
Non-players	100	3.74	6.74	3
Players	100	-0.41	8.53	0

Table 6.1 Statistical metrics of perception error for players and non-players. The two groups both count 100 participants, but these round numbers were not planned and are a coincidence.

Test	Result
Mann-Whitney rank test	$p < 0.001$ ($Ua = 6681.5$)
Cohen's d	$d = 0.54$

Table 6.2 Results of statistical significance tests and effect size test.

an indication of how well the data fits into a perfect line, where -1/+1 indicate a perfect negative/positive correlation and 0 indicates no correlation at all. We found some correlations to be statistically significant, which are shown in Table 6.3. All the variables we found to be significantly correlated to time perception error are proxies to how much attention a participant dedicated to the game over time. The fact that all these correlations are negative seems to indicate that an increase in attention to



(a) Score of last level

(b) Total score

(c) Played time

(d) Game time

Figure 6.3 Correlation between some variables and perception error.

Variable	**Result**
Time actually played vs Perception error	$r = -0.33$ ($p < 0.01$)
Time the game was running vs Perception error	$r = -0.30$ ($p < 0.01$)
Total Score vs Perception error	$r = -0.35$ ($p < 0.01$)
Score on last level) vs Perception error	$r = -0.34$ ($p < 0.01$)

Table 6.3 Correlations found significant, along with their r coefficient and the p-value.

the game correlates with a more negative perception error; that is, a lower estimate of elapsed time. Figures 6.3a through to 6.3d show scatter plots of the correlations. Although this correlation is not large, it is statistically significant; this is in agreement to the outcome of the statistical significance and effect size that were reported in the previous section.

6.4.3 Linear Regression

We fit a linear model to compute the magnitude of each contribution to the overall time perception under the assumption that the contributions are linear and independent. The results suggest that time spent playing and time spent not playing contribute in different magnitude to the overall perceived time. In mathematical terms, we computed A, B and C so that the following formula was as accurate as possible:

$$A \cdot \text{non-playing time} + B \cdot \text{playing time} + C = \text{perceived time} \tag{6.1}$$

To do this, we performed a basic linear regression. The coefficients can be interpreted as follows: every minute not playing (while in the queue) is perceived as A minutes; every minute playing is perceived as B minutes; people usually always perceive a wait of at least C minutes (that is, C is the intercept in the linear formula). We computed the three coefficients using three sets of data: players only, non-players only, and both players and non-players together. The three sets of coefficients we found are reported in Table 6.4, along with the R-squared statistic, which is a statistical measure between 0 and 1 indicating how closely the data matches the formula. Even though these findings are limited by the strong assumptions made and by the average R-squared statistic, the numbers suggest that, when estimating perceived time from playing/non-playing time, the contribution of the time spent playing to the overall the perceived time is many times lower than the contribution of the time spent not playing. Using the data from both groups, we found that the same amount of actual time was perceived roughly 5 times longer by non-players than by players.

6.4.4 Survey analysis

After playing during their wait in the queue, players were asked to complete a 10 question survey, each answer being given on a Likert scale; we used a 5 step Likert

Using	A	B	C	R^2	Ratio A/B
Players and non-players	0.87	0.17	5.49	0.6	5.24
Non-players only	0.94	0	4.73	0.64	N/A
Players only	0.75	0.24	6.27	0.5	3.15

Table 6.4 Results of the linear regressions conducted. A and B are, respectively, coefficients for non-playing time and playing time; C is the intercept. R^2 is a statistical measure $0 \leq R^2 \leq 1$ indicating how closely the data matches the formula A * non-playing time + B * playing time + C = perceived time.

Question	Mean	Std. Dev.
I played the game as much as possible in the queue.	3.92	1.17
Playing the game made time pass more quickly.	4.09	0.92
I enjoy waiting in long queues.	1.76	1.13
I often use my mobile phone while waiting in long queues.	3.72	1.28
I would appreciate a game like this being available in other long queues.	3.87	1.02
I would rather play my favourite mobile game in a queue than the game that was provided.	3.4	1.07
Playing this game with a group makes time pass more quickly.	3.98	0.99
I/we explored different strategies in the game.	3.48	1.06
Progressing in the game made me feel better about my progress in the queue.	3.72	0.97
I would play this game again if I re-entered this queue.	3.78	1.19

Table 6.5 Statistics of the survey responses that we collected. Responses, in a Likert scale, are assigned values 1 to 5 which represent, respectively, strong disagreement or strong agreement.

scale, including “Strongly disagree”, “Disagree”, “Neutral”, “Agree”, “Strongly agree”, along with an extra option “Not applicable”.

Table 6.5 shows mean scores for every question. Players seem to agree on the positive impact of the game, with high mean scores on questions such as “Playing the game made time pass more quickly” and “I would appreciate a game like this being available in other long queues”. However, some player stated that they would prefer playing some other game. The data seems to support that some of the strategies are indeed effective, such as playing in group and giving players a sense of progression. Figure 6.4 show the distribution of answers to selected items of the survey.

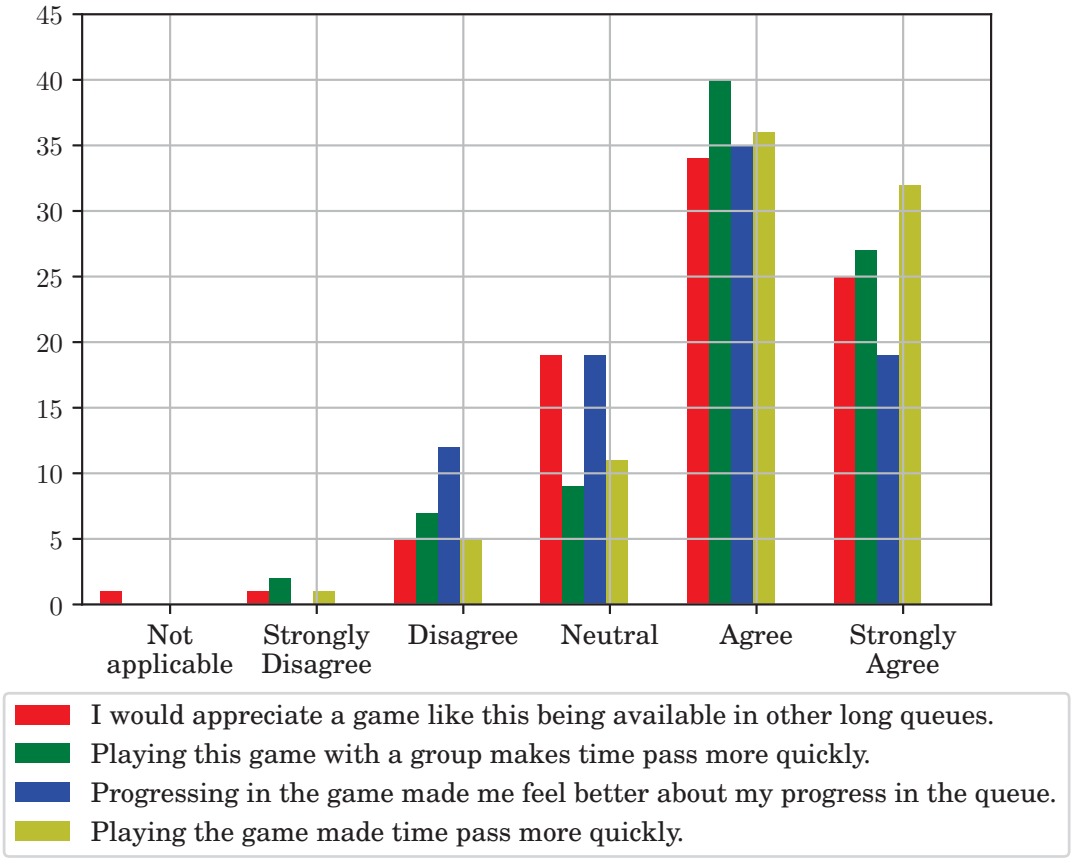


Figure 6.4 The figure shows the distribution of answers for selected questions of the survey administered to participants.

6.5 Limitations and Future Work

Studies in the wild dramatically differ from controlled in studies in that several parameters or aspects of a study cannot be fully controlled by the research team. This study was no exception and indeed the current limitation of our work primarily stem from this.

Due to operational constraints of the theme park (i.e., peak season) being operated by our industry partner, we were not able to carry out the entire study where we had originally planned (the "Justice League" area) and in fact we had to switch to a new area (the "Wild Wild West" ride area). This was responsible for some thematic disconnection in the game content, which could however be easily fixed as suggested in the previous sections on. Similar constraints also limited the amount of time that we devote to our experiments and ultimately impacted how many people ended in our experimental samples. In our future work, we hope to carry out additional

experiments in the wild with content better aligned to a predetermined theme park area. Preliminary discussions with our industry partner seem to indicate that a solution can be found to overcome our previous logistical issues.

Also, we did not deploy any of the content we had originally planned to include in-between levels e.g., a "shop" to purchase new weapons and power-ups, which could have had a positive impact on longevity as well as on perceived waiting times. We were hesitant adding this new feature could have impacted the stability of our game prototype: we leave this as future work, as well.

Finally, we did not compare time perception in participants playing their favourite mobile game in the queue as this would have meant a supplementary layer of complexity in tracking each individual user behaviour on the provided mobiles (which, in fact, were sand-boxed for security reasons). We feel this could be an interesting experiment to run, which, however, would not weaken the strength of our results but rather it could put them in a broader perspective. Moreover, it could be argued that while time perception was an essential element in our experience and that other games could fare well in this respect, it would not be easy to reproduce the kind of social experiences our AR game prototype delivered without extensive deliberate design, ruling out the vast majority of existing mobile games.

6.6 Summary

This chapter presented the work done with a theme park industry partner: an Augmented Reality (AR) video game prototype was developed and deployed it in a queue area. We presented our findings in terms of users time perception. The data collected suggests that video games in queues can bring great benefits to customers and therefore to theme parks. The results of the experiment indicate that time perception is significantly different (slower) for players compared to non-players, with a medium sized effect. In particular, there seems to be a correlation between variables related to the degree of attention and the time misperception. Fitting the data through a linear model shows that playing can give up to a 5x speedup to perceived time.

Despite the overall positive results, some participants reported that they would rather play their favourite game in the queue. This could be partially due to the unfamiliarity of people with the new game, which could make someone reluctant to put an effort in the new experience when they could instead play their favourite mobile game. Additionally, some participants played a shorter amount of time than

they waited, which seems to suggest that the game can get boring after a period of time. These are indications that the video game could benefit by the introduction of Dynamic Difficulty Adjustment (DDA). DDA would in fact lower the entry barrier to the game, allowing a more gentle introduction that would benefit less confident players. DDA would also scale the difficulty up for the more skilled players so that the game would keep them entertained for a longer period of time, possibly long enough for them to reach the head of the queue.

A final but important benefit, this can serve in extending the ride experience to the queue area, making the whole theme park experience more immersive for customers.

Chapter 7

Conclusions

The focus of this thesis was Artificial Intelligence (AI) applied to games, which is attracting a lot of research from very big players in the technology scene. This is because of two distinct reasons: first, games are an excellent test field for AI techniques, being non-trivial but at the same time not containing all the nuances of the real world; second, top video games, with budgets in the order of tens of millions US dollars, want to offer players an all-round realistic experience. The first part of the thesis focused in particular on learning systems, based on the Reinforcement Learning paradigm, while the second part focused on AI techniques aimed at enhancing player experience in video games.

The objectives of this thesis were to study novel approaches and methods to the problems of learning behaviours in large and complex environments and of Dynamic Difficulty Adjustment (DDA) in video games. Furthermore, the thesis offered an assessment of the potential for DDA in the context of theme parks digitally augmented areas.

The studies in this thesis answered four research questions:

Q_1 Can goal-oriented options be learned from expert demonstration by using the concept of surprise? Will such options accelerate the learning process?

A_1 *Yes, and they can both accelerate the learning process as well as produce better final performance in a reasonable learning time.* The key contribution in answering this question is an algorithmic solution to learn options from expert demonstrations based on the notion of "surprise". This algorithm detects unexpected expert decisions and infers the expert motivating goal. Extracted goals are used to create goal-oriented options that bias learning in a positive way, increasing performance during the learning process. The use of the concept of surprise had not been investigated before in the problem of learning options.

Q_2 Can an agent refine its abstraction on the knowledge of the environment over time?

A_2 *Yes, by progressively choosing a more refined abstraction when there is enough data to be confident in its reliability despite data sparsity.* The key contribution in answering this question is an algorithmic solution to perform online state abstraction in Markov Decision Processes using standard Student's t test. This was coupled with an ad-hoc heuristic exploration strategy to further improve agent performance. These techniques increased performance in the learning process by avoiding the problem of data sparsity. This approach of progressively using finer-grained abstractions by considering more features had not been investigated before in RL.

Q_3 Can an agent playing in a game target a 50% chance of victory while maintaining believability?

A_3 *Yes, in the context of a fighting game, an agent reducing the HP difference between players can achieve a victory rate around* 50%. The key contributions in answering this question are a set of algorithmic solutions to target a 50% chance of victory in games. These algorithms use Monte Carlo Tree Search to choose the action most likely to produce the desired outcome, while maintaining a believable behaviour. These approaches built on well-established ideas in the research communities of psychology and video games AI and improved on the state of the art.

Q_4 Is there evidence to suggest that adjusting the difficulty of a game played in a queuing environment can improve time perception?

A_4 *Yes, given the large effect on time perception of playing a game while queuing and the feedback from players themselves, there is evidence to make a case for difficulty adjustment.* The key contributions in answering this question are the analysis and discussion of real-world data that suggests that applying DDA to games played in a theme park queuing area could benefit time perception of players, making them feel like they had been waiting a shorter time span than they actually had. A quantitative analysis of data collected in a theme park queuing environment had not been performed before. This can inform theme parks managers on how to improve the experience of their customers.

This thesis explored techniques to improve the efficiency of Reinforcement Learning. In particular, tabular state-action representations were used; further research is desirable to apply the same approaches to the modern visual-based Deep Learning

architectures, which have the advantage of not requiring a special state representation, further generalising the approach. Moreover, the individual techniques can be studied further: the proposed state abstraction technique can be mixed with other orthogonal approaches, while options discovery can be improved by beans of automatic features selection.

The latter part of the thesis focused on DDA in games. In particular, one study proposed a technique based on MCTS to tackle the problem. While MCTS has the advantage of not requiring any learning time, it is limited in that it is computationally expensive at run time. Further research is required to apply the same idea to a RL-based approach. Furthermore, DDA can be used to tailor groups of players sharing some traits of personality, or even tailor individual players.

Finally, we reported the results of a study that measures the effects of time perception on theme park customers queuing for an attraction. We argued that DDA has a large potential in these venues; further studies are required to verify this with an actual DDA-powered game deployed in a theme park queue.

7.1 Future Works

In this section we touch on directions for future work that build upon the different contributions.

Learning Options from Demonstrations

Chapter 3 presented an algorithm to extract options, which are high-level behaviours, from expert demonstrations. The proposed algorithm detects a goal every time an expert makes an unexpected decision; a pruning step is used to reduce this large set. Unfortunately, this latter part is computationally expensive; this can be solved by providing the pruning method with a list features to be used for aggregation. Automatic approaches to completing this task are avenues for future work. Another important limitation of this work is that the approach was only applied to tabular Q-learning and not to approximated Q-learning. Further study is necessary to explore the suitability of the method with approximated Q-learning.

Dynamic Choice of State Abstraction

Chapter 4 introduced a novel approach to state abstraction, useful when knowledge of the state space is sparse. The method stores experience with as much detail

as possible, but abstracts away most details when taking decisions; over time, as knowledge become less sparse, the abstractions used for decision-making become less sever. The proposed algorithm takes advantage of similarity in Q-values of similar states. In particular, factored state spaces where states in (hyper-)rectangular regions share similar values are a good fit for the algorithm. However, this performs well only if the entirety of the state space follows this "hierarchy of importance" of the features; on the other hand, it performs poorly if there are different regions of the state space with different hierarchies of importance. For example, in the case of Pac-Man, in states where a threatening ghost is nearby, food direction is less important ghost direction, while in states where all threatening ghosts are distant, the opposite is true. To optimise the use of redundancy in these cases, an algorithm would need to consider multiple, separate sets of features. This means that the input should not be a sequence of abstractions but rather a lattice of abstraction. This is because a list allows for only one "line of specialization", whereas a lattice allows for more possibilities. In other words, the proposed algorithm does not have a choice in *what* information to add over time: it can solely choose *when* to add it. Using a lattice, different sets of features could be selected for different states and they would still all converge to the abstraction with all features in the end. This is currently the strongest limitation of the algorithm and is an important direction for future research.

Monte Carlo Tree Search for Dynamic Difficulty Adjustment

Chapter 5 presented a novel approach to achieve Dynamic Difficulty Adjustment (DDA) in video games, allowing for more balanced automatic opponents. The technique works on top of any mechanism that can compute the value of all actions in a given state in term of a given reward function; both Reinforcement Learning and Monte Carlo Tree Search are suitable techniques (in the study, the latter was used for convenience). The approach can be used in various flavours, but the essence is to always choose the action that leads to the smallest difference in terms of games score between the two players; this is a proxy for a 50% victory rate, which should indicate a balanced game, one where the outcome is uncertain. This is motivated by research on *flow* [25], a mental state of deep immersion: a necessary condition to allow someone to enter the state of flow is dealing with a task not too easy nor difficult. However, a 50% win rate may not be the best fit for that definition, and further research is required to reach a deeper understanding.

Opportunities for AI Techniques in Theme Parks

Chapter 6 reported the results of a study where an Augmented Reality video game was deployed in a theme park queuing area. In the context of the experiment, time perception of players and non-players was measured, showing an important difference between the two, with players feeling time passed more quickly. At the same time, some participants reported that they would have rather played their favourite mobile game. This suggests the potential for games in theme parks and shows current problems that can be addressed by DDA (to ease introduction) opponents and other more challenging AI opponents (to keep players challenged). Notice that, beside difficulty, other player-customised dynamic adjustments in queue video games could bring a positive effect. For example, players showing a preference for action could be given tougher enemies to defeat, while players who seem to like customising their weapons could be given a larger variety of items and power-ups. These dynamic customisation could mitigate the challenge of game designers of producing a game that is entertaining to a large and varied audience, spanning different age ranges, gaming experience, culture, and gender.

Bibliography

[1] Ahmadi, R. H. (1997). Managing capacity and flow at theme parks. *Operations research*, 45(1):1–13.

[2] Alexander, A. L., Brunyé, T., Sidman, J., and Weil, S. A. (2005). From gaming to training: A review of studies on fidelity, immersion, presence, and buy-in and their effects on transfer in pc-based simulations and games. *DARWARS Training Impact Group*, 5:1–14.

[3] Andrade, G., Ramalho, G., Gomes, A. S., and Corruble, V. (2006). Dynamic game balancing: An evaluation of user satisfaction. *AIIDE*, 6:3–8.

[4] Andrade, G., Ramalho, G., Santana, H., and Corruble, V. (2005). Challenge-sensitive action selection: an application to game balancing. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 194–200. IEEE.

[5] Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pages 119–125.

[6] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

[7] Avery, P. M. and Michalewicz, Z. (2010). Adapting to human gamers using coevolution. In *Advances in Machine Learning II*, pages 75–100. Springer.

[8] Baldwin, A., Johnson, D., and Wyeth, P. A. (2014). The effect of multiplayer dynamic difficulty adjustment on the player experience of video games. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 1489–1494. ACM.

[9] Bellman, R. (1957). A markovian decision process. Technical report, DTIC Document.

[10] Birenboim, A., Anton-Clavé, S., Russo, A. P., and Shoval, N. (2013). Temporal activity patterns of theme park visitors. *Tourism Geographies*, 15(4):601–619.

[11] Bonarini, A., Lazaric, A., and Restelli, M. (2006). Incremental skill acquisition for self-motivated learning animats. In Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J., Marocco, D., Meyer, J.-A., Miglino, O., and Parisi, D., editors, *From Animals to Animats 9*, volume 4095 of *Lecture Notes in Computer Science*, pages 357–368. Springer Berlin Heidelberg.

[12] Bradley, S. P., Hax, A. C., and Magnanti, T. L. (1977). *Applied mathematical programming*, chapter Dynamic Programming, pages 320–362. Addison-Wesley.

[13] Broadbent, D. E. (1977). The hidden preattentive processes. *American Psychologist*, 32(2):109.

[14] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

[15] Burton, L. J., Westen, D., and Kowalski, R. (2009). *Psychology: Australian and New Zealand*. John Wiley & Sons Australia, Ltd.

[16] Casini, L., Macar, F., and Grondin, S. (1992). *Time Estimation and Attentional Sharing (short communication)*, pages 177–180. Springer Netherlands, Dordrecht.

[17] Černỳ, M., Plch, T., Marko, M., Gemrot, J., Ondráček, P., and Brom, C. (2015). Using behavior objects to manage complexity in virtual worlds. *arXiv preprint arXiv:1508.00377*.

[18] Chen, J. (2007). Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34.

[19] Cobo, L. C., Zang, P., Isbell Jr, C. L., and Thomaz, A. L. (2011). Automatic state abstraction from demonstration. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1243. Citeseer.

[20] Coe, R. (2002). It's the effect size, stupid: What effect size is and why it is important. `https://web.archive.org/web/20170327212724/http://www.leeds.ac.uk/educol/documents/00002182.htm`. Accessed: 2017-03-27.

[21] Cope, R. F., Cope III, R. F., Bass, A. N., and Syrdal, H. A. (2011). Innovative knowledge management at disney: human capital and queuing solutions for services. *Journal of Service Science*, 4(1):13.

[22] Corder, G. W. and Foreman, D. I. (2014). *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons.

[23] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2 edition.

[24] Cowley, B., Charles, D., Black, M., and Hickey, R. (2008). Toward an understanding of flow in video games. *Computers in Entertainment (CIE)*, 6(2):20.

[25] Csikszentmihalyi, M. (1990). Flow: The psychology of optimal performance. *NY: Cambridge University Press*.

[26] Csikszentmihalyi, M. and Csikszentmihalyi, I. S. (1992). *Optimal experience: Psychological studies of flow in consciousness*. Cambridge university press.

[27] Şimşek, O., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd International Conference on Machine learning*, ICML '05, pages 816–823, New York, NY, USA. ACM.

[28] Davis, M. M. and Maggard, M. J. (1990). An analysis of customer satisfaction with waiting times in a two-stage service process. *Journal of Operations Management*, 9(3):324–334.

[29] Dayan, P. and Watkins, C. (1992). Q-learning. *Machine learning*, 8(3):279–292.

[30] Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., and Mueller, F. F. (2017a). Measuring player skill using dynamic difficulty adjustment. In *Proceedings of the Australasian Computer Science Week Multiconference*. ACM.

[31] Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., and Mueller, F. F. (2017b). Monte carlo tree search based algorithms for dynamic difficulty adjustment. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE.

[32] DeNero, J. and Klein, D. (2010). Teaching introductory artificial intelligence with pac-man. In *Proceedings of the Symposium on Educational Advances in Artificial Intelligence*.

[33] Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine learning*, 43(1-2):7–52.

[34] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In Simoudis, E., Fayyad, U., and Han, J., editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231. AAAI Press.

[35] Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345–349.

[36] Fraisse, P. (1963). *The psychology of time*. Harper & Row.

[37] Gallagher, M. and Ryan, A. (2003). Learning to play pac-man: an evolutionary, rule-based approach. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 4, pages 2462–2469 Vol.4.

[38] Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192.

[39] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182.

[40] Hallak, A., Di-Castro, D., and Mannor, S. (2013). Model selection in markovian processes. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–382. ACM.

[41] Hao, Y., He, S., Wang, J., Liu, X., Huang, W., et al. (2010). Dynamic difficulty adjustment of game ai by mcts for the game pac-man. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 8, pages 3918–3922. IEEE.

[42] Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

[43] Herbrich, R., Minka, T., and Graepel, T. (2006). Trueskill™: A bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576.

[44] Hernandez-Maskivker, G. and Ryan, G. (2014). Could managers consider waiting times as something positive. In *2014 International Conference on Global Economy, Commerce and Service Science (GECSS-14)*, pages 352–355, Amsterdam, Netherlands. Atlantis Press.

[45] Hsu, F.-h. (1999). IBM's deep blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81.

[46] Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM.

[47] Hunicke, R. and Chapman, V. (2004). Ai for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91–96.

[48] Jennett, C., Cox, A. L., Cairns, P., Dhoparee, S., Epps, A., Tijs, T., and Walton, A. (2008). Measuring and defining the experience of immersion in games. *International journal of human-computer studies*, 66(9):641–661.

[49] Jiang, N., Kulesza, A., and Singh, S. (2015). Abstraction selection in model-based reinforcement learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 179–188.

[50] Jong, N. K., Hester, T., and Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, pages 299–306, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[51] Jong, N. K. and Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *IJCAI*, pages 752–757. Citeseer.

[52] Jonsson, A. and Barto, A. (2006). Causal graph based decomposition of factored mdps. *J. Mach. Learn. Res.*, 7:2259–2301.

[53] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg.

[54] Konidaris, G., Kuindersma, S., Grupen, R., and Barreto, A. S. (2010). Constructing skill trees for reinforcement learning agents from demonstration trajectories. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1162–1170. Curran Associates, Inc.

[55] Ledbetter, J. L., Mohamed-Ameen, A., Oglesby, J. M., and Boyce, M. W. (2013). Your wait time from this point will be... practices for designing amusement park queues. *ergonomics in design*, 21(2):22–28.

[56] Leigh, R., Schonfeld, J., and Louis, S. J. (2008). Using coevolution to understand and validate game balance in continuous games. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1563–1570. ACM.

[57] Limentani, G. B., Ringo, M. C., Ye, F., Bergquist, M. L., and MCSorley, E. O. (2005). Beyond the t-test: Statistical equivalence testing. *Analytical Chemistry*, 77(11):221 A–226 A.

[58] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163.

[59] Maister, D. H. (1985). The psychology of waiting lines. In Czepiel, J. A. and Solomon, M. R., editors, *The Service Encounter: Managing Employee/Customer Interaction in Service Businesses*, chapter 8, pages 113–123. Lexington Books, Lanham, MD, USA.

[60] Malone, T. W. (1980). What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM.

[61] Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning*, ICML '04, pages 71–78, New York, NY, USA. ACM.

[62] McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196.

[63] McCallum, R. A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395. Citeseer.

[64] McCallum, R. A., Tesauro, G., Touretzky, D., and Leen, T. (1995). Instance-based state identification for reinforcement learning. *Advances in Neural Information Processing Systems*, pages 377–384.

[65] McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 361–368, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[66] Menache, I., Mannor, S., and Shimkin, N. (2002). Q-cut—dynamic discovery of sub-goals in reinforcement learning. In Elomaa, T., Mannila, H., and Toivonen, H., editors, *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages 295–306. Springer Berlin Heidelberg.

[67] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[68] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[69] Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. In Ang, Marcelo H., J. and Khatib, O., editors, *Experimental Robotics IX*, volume 21 of *Springer Tracts in Advanced Robotics*, pages 363–372. Springer Berlin Heidelberg.

[70] OpenAI (2017). Dota 2. `https://web.archive.org/web/20171123200000/https://blog.openai.com/dota-2`. Accessed: 2017-12-28.

[71] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830.

[72] Pickett, M. and Barto, A. G. (2002). Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 506–513. Morgan Kaufmann.

[73] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[74] Raffe, W. L., Tamassia, M., Zambetta, F., Li, X., and Mueller, F. F. (2015a). Enhancing theme park experiences through adaptive cyber-physical play. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 503–510. IEEE.

[75] Raffe, W. L., Tamassia, M., Zambetta, F., Li, X., Pell, S. J., and Mueller, F. F. (2015b). Player-computer interaction features for designing digital play experiences across six degrees of water contact. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pages 295–305. ACM.

[76] Risi, S., Vanderbleek, S. D., Hughes, C. E., and Stanley, K. O. (2009). How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 153–160, New York, NY, USA. ACM.

[77] Robles, D. and Lucas, S. M. (2009). A simple tree search method for playing ms. pac-man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 249–255. IEEE.

[78] Rohlfshagen, P. and Lucas, S. (2011). Ms pac-man versus ghost team cec 2011 competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77.

[79] Russell, S. J. and Norvig, P. (2009). *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall.

[80] Samothrakis, S., Robles, D., and Lucas, S. (2011). Fast approximate max-n monte carlo tree search for ms pac-man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):142–154.

[81] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.

[82] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[83] Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, ICML '04, pages 95–102, New York, NY, USA. ACM.

[84] Spronck, P., Sprinkhuizen-Kuyper, I., and Postma, E. (2004). Difficulty scaling of game ai. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37.

[85] Stahl, A. E. and Feigenson, L. (2015). Observing the unexpected enhances infants' learning and exploration. *Science*, 348(6230):91–94.

[86] Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In Koenig, S. and Holte, R., editors, *Abstraction, Reformulation, and Approximation*, volume 2371 of *Lecture Notes in Computer Science*, pages 212–223. Springer Berlin Heidelberg.

[87] Stone, P. and Sutton, R. S. (2001). Scaling reinforcement learning toward robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 537–544, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[88] Subramanian, K., Isbell, C. L., and Thomaz, A. L. (2011). Learning options through human interaction. In *In 2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT*. Citeseer.

[89] Suguru, I., Ishihara, M., Tamassia, M., Tomohiro, H., Thawonmas, R., and Zambetta, F. (2017). Procedural play generation according to play arcs using monte-carlo tree search. In *Game-ON'2017: 18th International Conference on Intelligent Games and Simulation*.

[90] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

[91] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044.

[92] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[93] Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181 – 211.

[94] Sweetser, P. and Wyeth, P. (2005). Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3.

[95] Tadić, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine learning*, 42(3):241–267.

[96] Tamassia, M., Zambetta, F., Raffe, W., and Li, X. (2015). Learning options for an MDP from demonstrations. In Chalup, S., Blair, A., and Randall, M., editors, *Artificial Life and Computational Intelligence*, volume 8955 of *Lecture Notes in Computer Science*, pages 226–242. Springer International Publishing.

[97] Tamassia, M., Zambetta, F., Raffe, W., Mueller, F. F., and Li, X. (2016a). Dynamic choice of state abstraction in q-learning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press.

[98] Tamassia, M., Zambetta, F., Raffe, W., Mueller, F. F., and Li, X. (2016b). Learning options from demonstrations: A pac-man case study. *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*.

[99] Tange, O. (2011). Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

[100] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685.

[101] Tesauro, G. (1995). Td-gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer.

[102] Togelius, J., De Nardi, R., and Lucas, S. M. (2006). Making racing fun through player modeling and track evolution. In *Proceedings of the SAB Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, pages 61–70.

[103] Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690.

[104] Vigorito, C. M. and Barto, A. G. (2010). Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development*, 2(2):132–143.

[105] Walsh, T. J., Li, L., and Littman, M. L. (2006). Transferring state abstractions between mdps. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

[106] Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

[107] Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge.

[108] Welford, B. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420.

[109] West, D. (1979). Updating mean and variance estimates: An improved method. *Communications of the ACM*, 22(9):532–535.

[110] Yannakakis, G. N. and Hallam, J. (2006). Towards capturing and enhancing entertainment in computer games. In *Hellenic Conference on Artificial Intelligence*, pages 432–442. Springer.

[111] Yannakakis, G. N. and Hallam, J. (2009). Real-time game adaptation for optimizing player satisfaction. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(2):121–133.

[112] Yannakakis, G. N. and Togelius, J. (2011). Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161.

[113] Zakay, D. (1998). Attention allocation policy influences prospective timing. *Psychonomic Bulletin & Review*, 5(1):114–118.

[114] Zakay, D. (2014). Psychological time as information: the case of boredom. *Frontiers in psychology*, 5.

[115] Zambetta, F., Raffe, W. L., Tamassia, M., Mueller, F. F., Li, X., Dang, D., Quinten, N., Patibanda, R., and Satterley, J. (2017). Reducing perceived waiting time in theme park queues via an augmented reality game. *ACM Transactions on Computer-Human Interaction (TOCHI)*.

[116] Zang, P., Zhou, P., Minnen, D., and Isbell, C. (2009). Discovering options from example trajectories. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1217–1224, New York, NY, USA. ACM.