“Design and Refinement of NPC Rules in Digital Board Games”

Diploma Thesis

by

Swen Gaudl

supervised by

Priv.-Doz. Dr.-Ing. habil. Rainer Knauf

Technical University Ilmenau
Faculty of Computer Science and Automation
Artificial Intelligence Group

Summer Semester 2009
May 11, 2009

Registration No.: 2009-05-04/023/IN00/2238

Contents

List of Figures

Examples

Abstract

What is the big difference between playing a game against a human being or against a computer generated player? Why were “LAN parties” such a big success in the mid-1990s? And why do many people believe that it is more challenging to play against human beings than to play with an artificial player? At the beginning of this new century “LAN parties” are in the middle of a slight regression, but only because there is a constantly growing number of people which celebrates playing massive multi-player games. Consequently, players have moved from staging games as a local event, where it is possible to play against other human beings to the World Wide Web, where it is possible to play with and against other people on a daily basis. All these developments appear to be based upon human behavior. Based upon this evidence the current state of game AI is unsatisfactory if compared to the performance of human players. The following work presents the reader with a tool that was developed for analyzing basic computer games with incorporated AI modules. These incorporated AI modules contain the strategies to perform the behavior of artificial players.This sets the stage for a systematic evaluation and refinement of rule based game AIs.

1. Introduction

Imagine a computer game with a programmed adversary NPC[1] that is represented as a rule based agent. The NPC behaves so well that it passes an adapted TURING Test successfully. Now imagine that you delete just one rule from the rule base and the resulting NPC fails the TURING Test.

As a consequence, before removing this last rule the NPC was represented as a minimal version of a rule base including the human factor.

By analyzing the differences between human behavior and intelligent rule-based behavior, I am searching for a way to create an artificial human agent. In his paper on machine learning [16] ALAN TURING describes an approach to validate a system that imitates a human being. Accordingly, there are different kinds of approaches. One of the most popular approaches was WEIZENBAUM's ELIZA program [17].

I believe that computer games offer the most efficient way to gather a suitable number of test persons, because they have the capability to reach a vast amount of volunteers. Consequently, they present an ideal stage for the analysis of machine game strategies performed by NPCs against human beings. However, commonly used game logics are mostly based on fixed programmed rules. Through the evaluation and refinement of adaptable rule-based NPCs in digital games it should be possible to see the differences between human and artificial behavior.

With this goal in mind, I created an environment to perform TURING Tests with NPCs and human beings. This environment serves as a corner stone for running experiments which aim at a systematic analysis and refinement of programmed game strategies performed by NPCs.

Human behavior shows a certain level of complexity. Therefore, straightforward strategies (encoded in very few rules) will most likely be identified as non-human. Huge amounts of com-

[1] NPC is a commonly used term in the game community and stands for "non-player character".

plicated rules on the other hand do not necessarily reflect the quintessence of human intelligence when playing games. Therefore, I try to find a model of human behavior, which enjoys minimality in terms of the rule base's size and complexity, but models human behavior reasonably well.

Since it is difficult to accomplish this goal for popular "state of the art" online computer games such as WORLD OF WARCRAFT or EVE ONLINE, we decided to start with a digitalized version of a board game called "JOSTLE 2007" [10], which I shall describe later.

One may ask: "Why is it difficult to create an NPC's knowledge base for these mentioned modern computer games?"

In my search for a suitable answer, I divided this question into several subtopics and found the following reasons:

- Since a TURING Test requires the separation and identification of test persons, most games preclude us to fully control the game environment and setting. Therefore, I am often unable to separate one proband from an other.

- Since communication is such an important factor in online games, it would be too much at this stage to combine both, communication and game strategies, into this first prototype. For an existing online game I would have to combine the two mentioned aspects into a first prototype. If not, the probands could easily identify the game strategy and this would ruin the test scenario.

- We cannot influence the development of further upgrades or features in external software. Therefore, I started with our own basic game which we can incrementally upgrade in well defined steps. This enables the rule base to be adapted more easily to the current degree of game complexity. It avoids anomalies, mostly redundancies, in the rule base and keeps it maintainable, while becoming more and more mature at the same time.

- In order to derive credible insights into human game behavior, many test persons are required. This means that many game copies must be used. Using a commercial product may exhaust the budget quickly in regards to the license fees or provoke illegal actions.

- By using a “state of the art game” my created game content would have to be on the same quality level in regards to graphical representation and maturity as a full grown commercial product. Otherwise, its content could easily be identified.

After weighing pros and cons, I am able to provide an answer to the previous question.

“ It is too difficult for me to create a good NPC knowledge base for a modern computer game, because it would require insights into the developers code base and change this code frequently to support our needs. It would also require a tremendous effort on the artistic and design level to support our created NPC.”

This is why I began with a new project, namely small self developed games and my own analyzing interface.

With my client-server architecture, which is capable to communicate over the INTERNET, and the first basic game integrated into it, it will be possible to collect all required data from a proband even if he or she is on the opposite side of the world.

This gives me the opportunity to validate my theory with a reasonably high proband base and it allows us to refine the NPC’s rule base according to the validation results.

The chapter on our GRINTU framework provides more information about the capabilities of this tool.

1.1. Artificial Intelligence

1.1.1. Definition

Artificial Intelligence, or short AI, can be understood as the attempt to create a machine or program which acts intelligently. Features like problem solving or logical closure fall all under the field of Artificial Intelligence.The science of Artificial Intelligence tries to transfer rational and human-like thinking and behavior onto machines. For further information on Artificial Intelligence, please refer to [14, p. 17f]. In everyday language Artificial Intelligence stands for both the Science of Artificial Intelligence and the attempt to create intelligent machines.

1.1.2. TURING Test

ALAN TURING drafted in his work on machine learning "'Computing Machinery and Intelligence"' [16] a test which can be described as follows: the test is based on an imitation game in which a person communicates with two other people through text messages. TURING's set-up was the following:

The imitation game was played by three people namely a male or female interrogator (A), a man (B), and a woman (C). The interrogator was placed in another room, separate from persons (B) and (C). The goal of the imitation game is that the interrogator has to determine which of the two persons is (B) and which is (C). (A) knows the two persons only by X and Y, and by the end of the game (A) has to state:

"X is (B) and Y is (C) or X is (C) and Y is (B)."

In the course of the game, the interrogator may ask questions like the following:

"A: X, can you please tell me the color of your hair?"

If (B) is X, the man has to answer the question. It is the goal of players (B) and (C) to mislead (A). (B)'s answer could be as follows:

"B: I have brown hair with bleached highlights and black bangs."

TURING's idea was now to replace (B) with a machine and watch how often (A) now misjudges (B), the machine for (C), and vice versa. What could this mean, if the interrogator (A) misjudges equally often?

Today, people break the TURING Test down into the following scenario:

If the interrogating person cannot decide upon the asked questions and answers wether the respondent is a human or a machine, would this not suggest that the machine must beconsidered intelligent in some way? In fact, this should be the case, because human beings judge other human beings by the same criteria.

For this work I chose an adapted version of the TURING Test which does not claim to be a full-fledged TURING Test. In this adapted version, I want to determine if the behavior of an artificial intelligence is judged as "human-intelligent" or not. Our working definition of "human-intelligent" is that by not detecting an artificial intelligence, which means that the human player cannot decide during a game if the artificial player is human or not, the artificial intelligence has the "human factor".

> *The human factor is a hypothetical construct which, if present, creates the impression of human behavior in a artificial intelligence.*

During my adapted TURING Test, two probands will play together with a artificial intelligence. After each game, the probands who do not know each other and who can not communicate with each other, have to judge the two other remaining players. Each proband has to decide between the human or the artificial intelligence.

Overview The thesis is organized as follows. The subsequent chapter describes the board game "JOSTLE2007" [10], which is used in the first setting, using the GRINTU Framework. It is followed by a chapter about Jostle play strategies, which are/will be employed by an NPC. It provides a closer look at the implementation of these rules using the PROLOG language. Then, I shall discuss the GRINTU framework and describe its value as a tool for game analysis. The prototype will be capable of analyzing any board game with defined steps. Finally, I will take a look at the evaluation and refinement possibilities of such a rule base including a short summary and outlook of what still needs to be done.

2. JOSTLE 2007

The first game which will be integrated into the GRINTU framework is"JOSTLE 2007". It was selected, because JOSTLE is easy to learn and understand in terms of game rules and its complexity to play. The board has a clear set-up, which allows even people without much experience with digital games or children to play the game.

2.1. Board Design

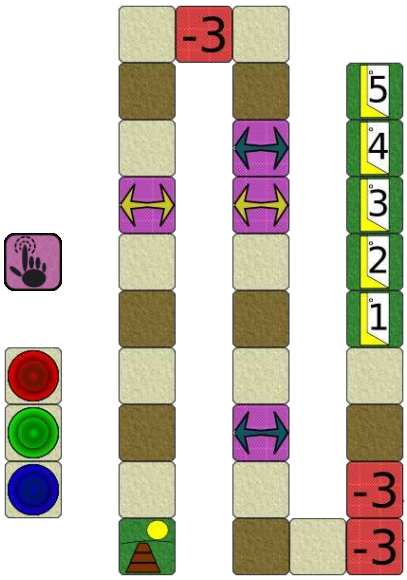


Figure 2.1.: Jostle Board

"JOSTLE 2007" is part of a board game family called JOSTLE, which focuses the aspect of jostling stones on the board. JOSTLE 2007 was developed by Dr. Klaus P. Jantke in report [10]. In this version, the game is played by three players. Each player owns a total of three gaming pieces. The board consists of 31 track fields, to which the gaming pieces can be moved, and a stack field for each player.

By looking at the board, one can recognize four different sorts of special fields, namely"START", "SWITCH", "MINUS THREE", and "GOAL".

Figure 2.2.: START tile

The "START" field is the first field any piece must visit when entering the game. On his or her first turn the player has to select a piece from the stack, which is situated to the left of the track, and move his or her piece onto the track. By doing so, the play piece's first field becomes the "START" field.

Figure 2.3.: SWITCH tile

The next special field is called "SWITCH", which is marked with a colored double arrow. There are always two switch fields bidirectionally connected. Once a gaming piece moves onto a switch field, the actual piece is instantly transported to the connected switch field. A special game rule takes place when jostling a piece onto a switch field. I shall describe this later in my the game rule section.

Figure 2.4.: MINUS THREE tile

If a piece stops on a “MINUS THREE” field, the piece is moved three fields backwards. If that particular field is occupied already, then the piece that occupied the field is jostled backwards.

Figure 2.5.: GOAL tile

The last special field type is the “GOAL” field. There are five goal fields, which are all marked with numbers, ranging from one to five.

2.2. Game Rules

The board is designed for three players with three play pieces each. The first player, either human or a NPC, begins by throwing a dice[1]. I modified JOSTLE2007 that whenever a six is rolled, the player has the option to roll the dice once more, but only once each turn.

After rolling the dice (once or twice), the player selects one of his pieces and moves it along the track towards the direction of the “GOAL” area in accordance with the number of points.

In the original version of JOSTLE2007*, the player has to select and move a piece after rolling the dice once. If he rolled a six, he can roll the dice once more, but he must move the already selected piece. This original part of the Game Rules allows the game to be a little more random, because not all moves can be completely planned. My modification allows the players to plan their actions better, which I preferred for this special setting.*

If a six was rolled and the dice was rolled once more, the player must move one piece for the complete amount of points rolled. That means the player is not allowed to partition the rolled points or to move two different pieces.

Is the destination field already occupied by another piece, then the piece that was situated on the destination field before the move is jostled back in row. The result of this jostling is that the

[1] In this version of the game we only use a six sided dice.

jostled piece is moved along the track towards the “START” field until it arrives at an unoccupied field.

A field is considered as occupied if a piece is standing on top of it. Special fields are treated a little differently. If one piece occupied a special field during its turn, then all special fields are treated as occupied until the end of said turn. This game rule is necessary to avoid infinite cycles.

An exception to this game rule are the “GOAL” fields and the “START” field. These fields are handled like normal fields regarding jostling. If there is no unoccupied field during jostling on the way towards the “START” field, the jostled piece is moved back onto its owner’s stack.

If a player was able to move a gaming piece and was successful, then his turn ends. Then it is the next player’s turn. If all players who were able to move a piece had their turn, a new round starts, beginning with the first player.

The players’ cycles end, when all goal fields are occupied, because then the last round begins. This means, that every capable player has one last turn. After this last round, each player counts the points of his occupied goal fields. The player with the highest score wins the game.

Further information on JOSTLE can be found in [10]. The following chapter describes various game strategies for JOSTLE and shows the derived PROLOG models. The tremendous differences between a formalized language and an informal description of a play strategy will also be highlighted, because it shows how difficult it is to write the correct artificial agent for a complex system such as a computer game.

3. Play Strategies

> *Strategies are what makes a person either win or loose a game, but the best strategy still requires a cunning mind to apply it.*

I shall start this chapter by introducing the basic terms and working definitions used from now on throughout this thesis to describe the AI, its implementation in PROLOG, and its behavior. With these it will be possible to present a closer and more refined view at strategies for the previously described game JOSTLE. These strategies are one focus of this paper.

All examples are tested and written using the ISOPROLOG implementation TUPROLOG [13]. The complete example code as well as the commonly shared rule base and a set of test data can be found in the appendix.

> "What can be said at all can be said clearly, and what we cannot talk about we must pass over in silence." *Wittgenstein* [18]

3.1. Game Play Notation

If somebody wants to discuss different play strategies, it is in most instances required to replay games or to talk about certain states of a game. This is not possible if there is no uniform notation of how a move is represented. For JOSTLE and most board games it is quite easy to create such a notation. I will now introduce my basic notation, which allows to reconstruct any JOSTLE game easily. The notation is similar to the notation of chess, but slightly easier.

Each move of a gaming piece is represented by an object consisting of three items. The first item is representing the moving piece. It is a number representing the internally given pieceID of the specific piece. For matters of convenience, all pieces in play have a unique number,

controlled by the game and not by the player. The next item is a small letter identifying the player. Since there are three players in JOSTLE who use red, green, and blue game pieces, the letters are r, g, and b. The last item is the field on which the player will move his gaming piece based on the outcome of the rolled dice. After each round, three dashes mark the end of a round which is useful to determine individual rounds, in case a player cannot make a move. As one might recognize, the pieceID is given at the start of each game and is depending on the color of the player. The redundancy to keep the player identification in the notation is due to the fact that it allows an easier understanding and reading of the notation for the case when human players wish to analyze a game by parsing through a game play book.

Example:

The setting is as follows: the JOSTLE *board is set-up as in figure 2.1 on page 7. There are three players, red, green, and blue. Each player has his or her turn in the given order. All players own three gaming pieces each. Since player red is the first player*[1]*, he or she controls the pieces with the pieceIDs 0, 1, 2. Player green controls the pieces with ids 3, 4, 5, and player blue, finally, commands the gaming pieces represented by the pieceIDs 6, 7, 8.*

Starting with player red, each player has his turn after the previous player has finished. Player red rolls the dice and moves piece zero to field number four. Then player green begins his turn by rolling the dice and by moving piece four to field six. Then, player blue moves gaming piece number six to field two. After all players have finished their turn, player red starts a new round by rolling the dice. Player red now moves piece zero to field eight. Player green moves his first piece, piece number three, to the "START" field after having rolled a one. The last player in the round is player blue, who moves piece number six to field four. . . .

The resulting representation of these previous moves is the following:

[1] If player blue had been the first player, he would control the pieces with the pieceIDs 0, 1, 2.

```
1   0r4
2   4g6
3   6b2
4   ——
5   0r8
6   4g0
7   6b4
8   ——
```

Example 3.1: Game Play Book

This so-called game play book represents a complete game and can be used to replay all moves. In this notation the time needed by each player to make a move is not recorded in the game play book.

3.2. Implementation Issues

Definitions:

- **Variable Vs. Atom**: PROLOG offers two ways to store information in so-called Terms. Information can be statically stored in atoms, which means that the atom has a certain value. Atoms can be numbers, letters, or strings.
  Examples: a, 123, 'Time_1212', "String"
  The last two examples are simple Strings, not to misunderstand with variables such as **Time is** 1212 and _12. For easier usage, atoms should always be marked by dashes and a variable should always start with an upper-case letter or an underscore. Here, I will adjust this convention a little. A variable will always start with an upper-case letter, or an underscore and a letter. We shall never use an arrangement such as the above _12 for a variable, because it is very hard to understand the meaning behind such a cryptic variable.

  Variables are "placeholders" for atoms which do not have a value yet.

- **Structured Terms** are not used in this thesis and therefore omitted here.

- **Interfaces** define the predefined call for certain predicates. Since PROLOG does not specify algorithms but just relations in-between data, every predicate can be called by using

every variable as in- or output. The usage of a predicate in regards to each parameter that is an input or an output is called flow pattern. Generally, we call certain predicates, such as the worth predicate in the subsequent example with a predefined flow pattern. In accordance with this convention, we are allowed to communicate slightly better between PROLOG and JAVA.

- **Clauses** consist of a conclusion part that heads a clause, for example haus(street ,number), followed by an "implied by" operator :– and end with a condition part, for example map(street), side (number)., which is the clause body.

  I define three different types of clauses:

  – **Rules** are constructed following a certain *"if-condition-then do-this"* logic. They are always arranged in a top-down order, which means that, by copying a rule and arranging it under the original rule, the original version is executed first and the copied version later on, if the original one failed. With this in mind, it is possible to give certain rules a higher preference regarding other rules by arranging them higher in the script.

  – **Facts** are clauses with an empty clause body, which is interpreted as true.

  – **Questions** are hypotheses and subject to check whether or not they are a logic conclusion from the rules and facts in a knowledge base.

- **Predicates** are defined by clauses with the same name attached to them and the number of arguments. The following two clauses, for example, are combined under the same predicate.

  student (A,B):– studies (A,Subject).

  student (**Name**,Age):–person(**Name**,Age),happy(**Name**).

  The clauses prof (N,A):–person(N,A),busy(N) and prof (N,U,A):–person(N,A),busy(N) are not combined under the same predicate.

- **Rule Sets** are combinations of different predicates. In our case, they define explicitly the strategy of one particular NPC. It is essential that a rule set is checked on correctness and completeness to guarantee failure security.

- **Matching** in PROLOG refers to the unification of predicates. Prolog solves all questions by unifying the clause's head of a question with an atom by replacing variables with other terms such as atoms, variables, or structured terms. This is done until all variables are replaced by atoms.

- **Strategies** in this paper will be used to describe formal or informal definitions for the complete behavior of a given rule set. They are more or less formal representation of a rule set.

Having established the terminology to describe certain aspects of coded strategies, I shall start discussing several of them. The terminology is necessary, because understanding AI rules that are given informally by natural language does not necessarily mean that an coded representation of these rules correctly represents them. It often is a challenge and bears the risk to lose or misinterpret information in the translation process.

The next example tries to highlight the above mentioned misinterpretation of a strategy. In the following examples the predicate worth defines a simple strategy of an NPC which looks for a piece that can be moved a maximum amount of fields.

For this and all of the following implementations of strategies, I used TUPROLOG [13]. This seemed to be the appropriate PROLOG engine, because of the following reasons:

- TUPROLOG is a feasible and stable PROLOG engine, which is still maintained.

- TUPROLOG is free to use even in commercial applications.

- TUPROLOG is a JAVA light-weight interference engine, which allows it to be used in Internet applications.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
8          goalTile(Xa),
9          worth1(R,[X,Y,Z],Best).
10 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
11         not(goalTile(X) ),X > Xa,  !,
12         worth1(R,[X,Y,Z],Best).
13 worth1([_|R],A,Best) :- worth1(R,A,Best).
```

Example 3.2: ADVANCER

The worth predicate receives a list of all gaming pieces [S|L] and as a result returns the one which was best according to the strategy by using an accumulator variable to hold the "best so far".

In 3.2, the core of the strategy can be seen starting from the eleventh line. There, the worth predicate compares the X value of two gaming pieces. After comparing all pieces in pairs, it selects the one most advanced that is not in the "GOAL" area (**not**(goalTile (X))) yet.

A change of X > Xa to X < Xa alters the complete behavior of the NPC. With X < Xa, the NPC always moves the most backward piece, which results in the fact that all pieces are moved relatively close to each other in so called clusters.

With this knowledge in mind, the implementation of the worth predicate reveals that the implemented predicate does not always move the correct piece. In the upper implementation (example 3.2) the "SWITCH" fields[2]were not comprised. With these fields it is possible to bypass long distances.

Conclusivly, a smarter implementation would look as follows:

[2]"SWITCH" fields transport a gaming piece onto the connected field of the same type. These fields allow movement in both directions towards the "START" field and towards the "GOAL" area.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1(L,Stone,Stone):-
8          die(D),
9          [A,_,_]=Stone,
10         B is A+D,
11         minTile(B,'switch@'),!.
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not( backTile(X)), not(goalTile(X) ),X > Xa,  !,
17         worth1(R,[X,Y,Z],Best).
18 worth1([_|R],A,Best) :- worth1(R,A,Best).
19
20 minTile(N,String):- board(N,B),
21     startsWith(String,B),
22     firstTileOf(N,B).
23
24 firstTileOf(N,String):-
25     board(N2,StringB),
26     startsWith(String,StringB),
27     N2 < N, !, fail.
28 firstTileOf(_,StringB).
29
30 backTile(X):-
31     die(D), B is X+D,
32     board(B,Na),
33     startsWith('back@',Na).
```

Example 3.3: Observing ADVANCER

By analyzing example 3.3 and its predicate worth, one can see that the minTile(B,'switch@') predicate checks if the next possible position (B) is a "SWITCH" field, whereas the backTile(X) predicate checks if the next position (X) is a "BACK-3" field[3]. Using this knowledge to avoid the "BACK-3" fields and preferring pieces which can move onto a "SWITCH" field allows to move a piece

more suited to the intended behavior of the strategy with its informal definition.

Obviously, coding rules that are given informally by natural language is a challenge and bears the risk to lose information in the translation process.

In previous examples 3.2 and 3.3, the first implementation (example 3.2) does not fulfill its informally given definition completely, because it does nothing more than moving the furthermost piece.

The second implementation, example 3.3, which comes closer to the natural intention, is a little more complex, because it is composed of three essential parts. The first part (lines four to six) is to avoid selecting a piece from the "GOAL" area, which can no longer move. The second part (lines seven to eleven) is to find a piece, owned by the NPC, which can move onto a "SWITCH" field by using the rolled number of points. If such a gaming piece is not available, the third part (lines twelve to eighteen) of the predicate worth1 will be used. This part searches the furthermost piece which will not move onto a back throwing field.

Part one and three of the second implementation (example 3.3) have been slightly adjusted in regards to example 3.2. The only new part is the second one. So the difference between both implementations could be understood as an enhancement of the strategy in example 3.2.

Knowingly combining and enhancing easy strategies leads to a new technique, which I will call AI STRUCTURE COMPOSITION. This technique will be described in the next section by giving different examples in code form to illustrate my theory.

3.3. AI STRUCTURE COMPOSITION

The concept behind the AI STRUCTURE COMPOSITION is, that all strategies can be grouped into two categories. The first category contains unstructured strategies. These Strategies are either poorly written, extremely complex, or evolved into an unstructured state.

It is my belief that most of the unstructured strategies can be converted into the second category, structured strategies. This restructuring of category one strategies can be extremely complex and will be part of some later work. For matters of convenience, I will from now on only take a look at category two strategies.

[3] A "BACK-3" field transports a piece three fields towards the "START" field.

AI Strategies:

- Category one: unstructured strategies
  - simple (*transformable*)
  - complex (*transformation could be non polynomial*)
  - evolved (*transformation could be non polynomial*)
- Category two: structured strategies
  - Type one:simple
    - basic
    - symbiont
  - Type two:complex

Category two contains strategies which are structured. By definition, there are again two types of category two strategies, simple and complex ones. The simple ones consist of two subtypes, basic strategies and symbiotic strategies.

Basic and symbiotic strategies follow a certain naming convention. The basic strategies are all labeled with nouns and written completely in upper-case letters. It serves the purpose to imply that basic strategies are actors and can behave without the support of other strategies. In contrast to them, symbiotic strategies still require a basic strategy to work properly.

The symbiotic strategies are labeled with adjectives to emphasize their dependence on other strategies. The first letter of symbiotic strategies is in the upper-case, the rest in lower-case letters to achieve a visible separation between basic and symbiotic strategies.

3.3.1. Basic Strategies

All later basic strategies are implemented for the game JOSTLE2007 using PROLOG, but it is easy to see that these basic strategies can be implemented for any other board game by using the informal definition of the strategy and implementing it for a specific game.

Basic Strategies:

- RANDOMIZER
- ADVANCER
- PALS
- EQUALIZER

RANDOMIZER is the first strategy I shall discuss here. It is an interesting basic strategy, because it acts unpredictably without being necessarily complex. The biggest advantage and, at the same time, its downside is its randomness. Here for our purpose, I shall assume that the given random function is adequate and generates usable results. In example 3.4, a PROLOG implementation of RANDOMIZER is shown. Whenever the NPC can make a move, the strategy picks a random gaming piece of the NPC and moves it. According to the so-called "Theory of Mind" [3], any random player assumes that the strategy must have an intention to react in this specific way. Since there is no such intention, it seems complex and unpredictable to the other players. By using this knowledge, the probands might suspect a hidden motive for these moves and assume that the artificial intelligence might be a human being. It is still not my goal to only use a random strategy, but using unpredictable behavior at some point might come in handy. Thus, combining RANDOMIZER with other strategies may produce valuable results.

```
1  worth([S|L],Stone):-
2          length([S|L],A),
3          rand_int(A,B),
4          worth1(L,S,B,Stone).
5
6  worth1(_,A,0,A):-
7          [X,_,_]=A,
8          not(goalTile(X) ).
9  worth1([[X,Y,Z]|R],S, Count, Best):-
10         not(goalTile(X) ),
11         C is Count -1, !,
12         worth1(R,[X,Y,Z],C,Best).
13 worth1([A|R],B,C,Best) :-
14         append(R,[B],L),
15         worth1(L,A,C,Best).
```

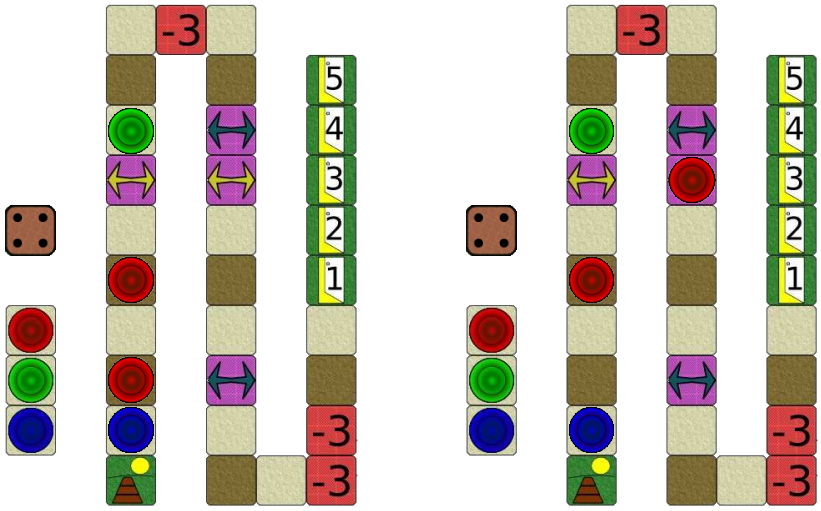
Example 3.4: RANDOMIZER

The Prolog implementation of RANDOMIZER (see example 3.4) starts with the call of the worth predicate in line 1. The first thing worth([S|L],Stone) calculates is the number of gaming pieces the player (in that case the rule based system) owns. This is done by calling the built in TUPROLOG predicate **length**([S|L],A) with the list [S|L] and the variable A. If A was already set to a specific value, then **length** returns **True** if the length of the list matches the value otherwise it returns False. If A was not set to a value, then **length** matches A with the length of the list and returns **True**.

rand_int(A,B) is the next built in TUPROLOG predicate which generates a random integer value B between zero and A, under the precondition that A represents an integer value. The generated random number is used in the second clause of worth1 to determine which gaming piece the NPC selects. The last thing worth does is to call worth1 with the tail of the list of gaming pieces, the first element of the list, the random number, and the return value Stone.

The worth1 predicate for RANDOMIZER consists of three clauses, whereas the first two clauses are straightforward. If the first element of the list is selected by the random number, then the first clause is called. If random has a greater value than zero, then the second clause is called. It selects the gaming piece from the provided list, using the random number as a counter. If the counter reaches zero, the first clause is called using the current list element. The third clause has the function to rotate the elements of the list if some elements are already in the “GOAL” area.

It becomes clear that RANDOMIZER selects a random gaming piece and returns it.



(a) RANDOMIZER before move *(b)* RANDOMIZER after move

Figure 3.1.: RANDOMIZER play strategy

Figure 3.1 illustrates the previously mentioned interesting behavior of a random strategy. With a setting of the board as in (a), RANDOMIZER chooses a random piece and moves it four fields. Since the chances for one piece to be selected amounts one-third, it is possible for RANDOMIZER to make the best move in this situation by selecting the second most advanced piece. By doing so, the gaming piece moves onto a “SWITCH” field that allows it to be transported to a connected field. After completing the move RANDOMIZER moved the second piece a total of twelve fields. Since a one-third chance is not impossible in this particular situation, RANDOMIZER creates the impression of observing the board and calculating the best possible move. I do not want to stress this more than necessary, because chances of selecting the last or the first piece were also each one-third each. The aspect which should be emphasized in this situation is that the randomness can create all kinds of different impressions and it could also be used to hide an intended strategy by making a random move at times.

ADVANCER our second strategy, was already introduced in example 3.2 on page 16 to illustrate the difficulty of translating an informal definition into a formal representation.

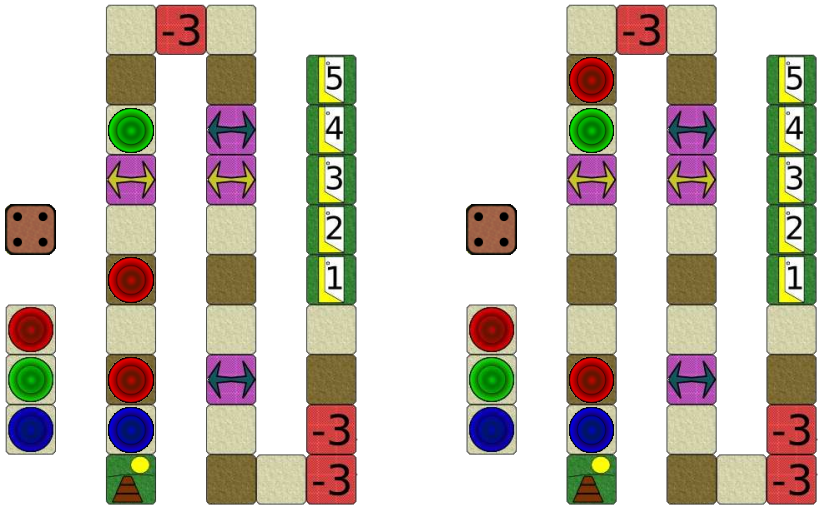
```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
8          goalTile(Xa),
9          worth1(R,[X,Y,Z],Best).
10 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
11         not(goalTile(X) ),X > Xa,  !,
12         worth1(R,[X,Y,Z],Best).
13 worth1([_|R],A,Best) :- worth1(R,A,Best).
```

Example 3.5: ADVANCER

The goal of ADVANCER is to advance as much as possible. Therefore, it tries to move the furthermost piece. In order to keep the strategy on a basic level and to allow easy modification, it does not analyze the board structure. Therefore, it does not act according to the layout of the board or the behavior of other players, except for the fact that it monitors the “GOAL” area for reasons of convenience.

The import part of the ADVANCER implementation is the third clause of worth1, where comparisons($x > x_a$) in twos between the advancement x and x_a of all gaming pieces are made and the furthermost piece is chosen.

In figure 3.2a, a match of the game JOSTLE is presented, where ADVANCER controls the red player. The player has currently two gaming pieces on the track and already rolled a dice, thus winning the opportunity to move a piece four fields. Player blue has one gaming piece on the track and player green none. As implemented, ADVANCER selects the most advanced piece and moves it four fields towards the “GOAL” area as shown in figure 3.2b. By doing so, ADVANCER disregards the possibility to move onto a “SWITCH” field, which would allow the second piece to advance even further.



(a) ADVANCER before move *(b)* ADVANCER after move

Figure 3.2.: ADVANCER play strategy

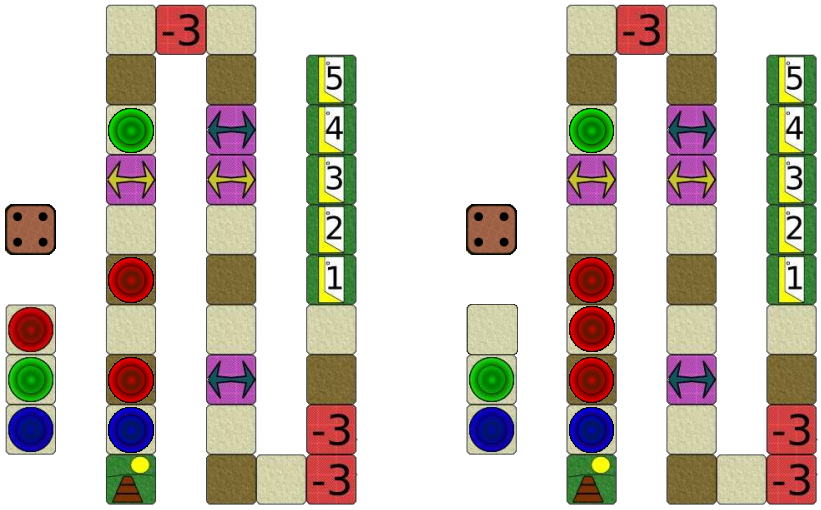
PALS emphasizes the aspect of fairness and closeness. In its basic form it selects the trailing piece. By doing so, all pieces are moved relatively close to each other in a so-called cluster and no piece is left behind. The name PALS originates from the visible behavior of the strategy. Due to the clustered movement, it seems as if all gaming piece are chummy and move like a flock. Since it is only a basic strategy, some aspects are not fully developed to allow a good compatibility between different strategies. In other environments an advanced strategy based on PALS might perform flock behavior of birds or other creatures.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
8          goalTile(Xa),
9          worth1(R,[X,Y,Z],Best).
10 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
11         not(goalTile(X) ),X < Xa,  !,
12         worth1(R,[X,Y,Z],Best).
13 worth1([_|R],A,Best) :- worth1(R,A,Best).
```

Example 3.6: PALS

The only difference between PALS and ADVANCER in regards to their code representation appears in line 11, namely `X < Xa`. This change of comparison operator results in PALS behavior to select the least advanced piece. This is why the gaming pieces move in a cluster for the most part. I believe that, clustered movement creates the impression that no piece is left behind. Therefore, the strategy was named PALS, because friends do not abandon each other.

After understanding the implementation of PALS, I would like to illustrate PALS's decision-making process by providing a simple example. PALS starts its turn in figure 3.3a and ADVANCER in figure 3.2a by having rolled a dice showing four points. This allows PALS to move one of its three gaming pieces four fields towards the "GOAL" area. Since PALS always moves the last piece, it selects and moves the piece from the stack as shown in figure 3.3b. This situation also allows me to show a cluster of four, namely the three red pieces and one blue piece without any space between them. Clusters appear more frequently in strategies like PALS, because this and other related strategies move all pieces relatively close to each other.



(a) PALS before move *(b)* PALS after move

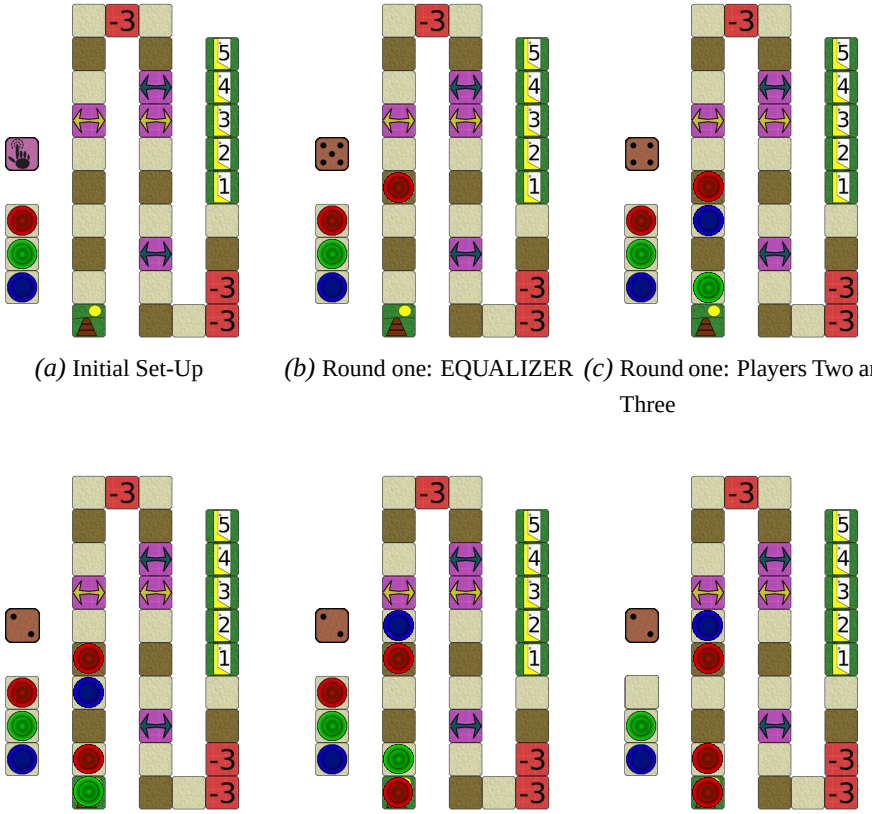
Figure 3.3.: PALS play strategy

EQUALIZER is the last basic strategy I would like to discuss here. As one can see in example 3.7, the implementation has grown significantly in comparison to previous strategies such as ADVANCER. This is due to the fact that EQUALIZER memorizes previous moves in history (X), where x is a sorted list of of gaming pieces that have yet to move. EQUALIZER is a strategy which basically moves all pieces equally, without any attention to the covered distance. Equally in this context means (1) equally often, i.e. the number of moves made by the NPC differ only minimally and (2) in a specific order. This does not necessarily create clusters. But by selecting all pieces equally, the distance normally between the first and the last piece is less than the distance of the pieces when ADVANCER is used. By moving all pieces in such a fashion the strategy overall is moving a little slower than ADVANCER, but still faster than PALS. The following example 3.7 shows the implementation of EQUALIZER, using a basic history that remembers the order in which the pieces are moved.

```
1  worth(L, Stone):-
2          worth1(L, Stone).
3
4
5  worth1(L, Best):-
6          history([H|T]),
7          length([H|T],A),
8          length(L,B),
9          0 < A,
10         A =< B,
11         worth11(L,H, Best),
12         setworthHistory(T), !.
13 worth1(L, Best):-
14         history(A),
15         length(A,0),
16         getworthHistory(L,B),
17         setworthHistory(B), !,
18         worth1(L, Best).
19 worth1(L, Best):-
20         not(history(A)),
21         getworthHistory(L,B),
22         setworthHistory(B), !,
23         worth1(L, Best).
24
25 worth11([[X,Y,Z]|L],Y,[X,Y,Z]):-
26         not(goalTile(X) ).
27 worth11([[X,Y,Z]|L],Y, Best):-
28         goalTile(X),
29         worth12(L, Best).
30 worth11([K|R],H, Best):-
31         append(R,[K],L),
32         worth11(L,H, Best),!.
33
34 worth12([[X,Y,Z]|T],[X,Y,Z]):-
35         not(goalTile(X) ).
36 worth12([_|L], Best):-
37         worth12(L, Best).
```

Example 3.7: EQUALIZER

The worth predicate calls the worth1 predicate. This predicate is composed out of three different clauses. The first clause is used if the history is working and in use. This means that the list has elements for a minimum of one gaming piece and a maximum of elements that equals the total of all gaming pieces of the NPC. If this is the case then the first element of the history(X) list is checked. This is done by comparing the piece's ID H with all elements of the given list of owned pieces of the NPC in worth11([[X,Y,Z]|L],H,[X,Y,Z]) and by checking that the gaming piece is not already in the goal area **not**(goalTile(X)). If these conditions are not fulfilled, the second clause is called for. This clause tests if the length of the element list is zero, which means that all gaming piece were equally moved in previous play rounds. If this is the case, a new history(X) list will be created by getworthHistory(L,B), setworthHistory(B) that includes all gaming pieces of the NPC that are utilizing this strategy. The last clause of worth1 is used for the initial set-up of the history.



(a) Initial Set-Up *(b)* Round one: EQUALIZER *(c)* Round one: Players Two and Three

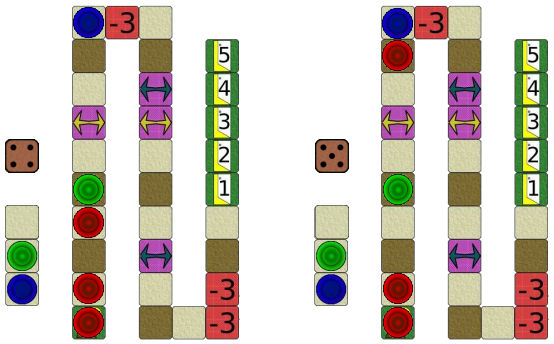
(d) Round two: EQUALIZER *(e)* Round two: Players Two and Three *(f)* Round three: EQUALIZER

Figure 3.4.: EQUALIZER play strategy part one

Through figure 3.4, I would like to illustrate the behavior of EQUALIZER by presenting a sample of a replayed game. Using EQUALIZER to control player red and ADVANCER to control players green and blue the gamer is initialized as shown in figure 3.4a. The game play book can be found in the appendix, example A.3 on page 70.

EQUALIZER's behavior in figure 3.4 illustrates very well how all pieces are moved in this strategy. The first move, figure 3.4b, is interesting, because all pieces are equally far away from the "GOAL". So EQUALIZER first defines an order of how all pieces will be moved during the

next rounds by creating a list that contains all gaming pieces owned by the player. The next figure, figure 3.4c, shows the board after player green and player blue have moved their gaming pieces. Both players are in this setting controlled by ADVANCER as a matter of convenience. Player green rolled two points and player blue four, and both moved a piece controlled by them according to their inherited strategy. In figure 3.4d, EQUALIZER moves its second piece from the stack instead of moving the piece already on field four. By doing so, EQUALIZER misses the opportunity to use a "SWITCH" field. This demonstrates that EQUALIZER chose the second element of the list of yet-to-move pieces. The following moves of green and blue as presented in figure 3.4e show that green has jostled EQUALIZER's second piece from field two to field one, and the blue player's piece advanced two fields towards the "GOAL" area. In the last figure, figure 3.4f, EQUALIZER draws its last piece from the stack onto the board. Up to this point, the behavior of EQUALIZER and PALS would be identical, because PALS would also try to move all pieces from the stack onto the board. The main difference here is that EQUALIZER will continue to keep the order of moving pieces in the same way as it is, demonstrated in figures 3.4g and 3.4h, whereas PALS tries to move only the least advanced piece, if it can.



(g) Round three: Players Two and Three *(h)* Round four: EQUALIZER

Figure 3.4.: EQUALIZER play strategy part two

EQUALIZER could thus be called a medium speed strategy, in comparison to ADVANCER and PALS. ADVANCER would then be a full speed strategy, and PALS moves with the lowest

advancement, by getting equally good dice rolls. RANDOMIZER, on the other hand, is not classifiable in terms of movement speed and advancement of its pieces, because due to its random behavior.

The next section is about the second subtype of Type One strategies, the symbiotic strategies.

3.3.2. Symbiotic Strategies

> *Over the long term, symbiosis is more useful than parasitism. More fun, too. Ask any mitochondria. [Larry Wall]*

Symbiotic strategies or short symbionts are strategies that are not able to exist, or, more accurately, to be used alone. Their name was chosen because of their special nature. Symbionts require another strategy as a host, just as a symbiont cannot live without an host animal or plant in nature. Symbionts can be understood as positive and negative enhancement strategies. I explicitly use the term negative enhancing symbiont instead of parasite, because the main goal remains still the same, namely that both sides are still benefiting from the effect of the enhancement. In fact, I belief that a parasitic strategy gains a positive effect without a positive effect for the host.

Positive and negative features are present in nature as well to allow for a greater variety of possible actions/ reactions. A penguin, for example, is quite clumsy and slow when it comes to walking on land, but it moves fast and elegantly when in its element, the water. By having such negative features in games, the recognized behavior varies enormously and this allows more possibilities for creating believable AI. In the virtual world, it would be possible to create an almost perfect strategy. Perfection, however, is not desired in this case on the games domain nor any other domain where believable AI agents are used to model human behavior because human beings strife for perfection, but are never perfect themselves. Strategies inheriting the human factor represent a certain kind of behavior. This behavior also shows faults and flaws.

Therefore, even negative symbionts are needed to create this particular impression.

Symbiotic Strategies:

- Observing
- Aggressive
- Lingering

I shall discuss different strategies wand their pros and cons by first giving their informal definition. Then I will explain more details by providing examples. In contrast to basic strategies, symbionts are conjoined with their hosts. In some later process it will be important to extract these essential representations of the symbionts and create clear interfaces between the different building blocks of a strategy. This allows us to use automatic tools to combine different strategies. But like in nature, host and symbiont are very closely connected, and it is hard to separate both without destroying some part. In my examples, I try to highlight the different parts and show the bridges between host and symbiont.

Observing is a the first symbiont which I want to portray here. It is a true symbiont, because it does not make any sense to use Observing during a game as stand-alone strategy. The main idea ob observing is, as its name suggests, to observe the game environment. In my setting, Observing is capable of finding a better solution for a move by reordering the preference list of possible moves. By doing so, Observing changes the order in that way that pieces which can move onto a special field will be treated special. So, when the base strategy aims at a certain goal like ADVANCER to advance the most, it will prefer a piece that can move onto a “SWITCH” field over pieces which could move onto a “BACK-3” field.

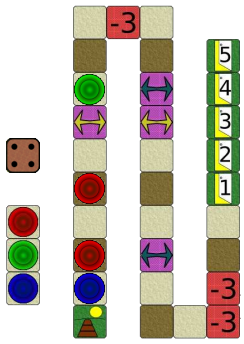
Observing ADVANCER

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1(L,Stone,Stone):-
8          die(D),
9          [A,_,_]=Stone,
10         B is A+D,
11         minTile(B,'switch@'),!.
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not( backTile(X)), not(goalTile(X) ),X > Xa,  !,
17         worth1(R,[X,Y,Z],Best).
18 worth1([_|R],A,Best) :- worth1(R,A,Best).
19
20 minTile(N,String):- board(N,B),
21     startsWith(String,B),
22     firstTileOf(N,B).
23
24 firstTileOf(N,String):-
25     board(N2,StringB),
26     startsWith(String,StringB),
27     N2 < N, !, fail.
28 firstTileOf(_,StringB).
29
30 backTile(X):-
31     die(D), B is X+D,
32     board(B,Na),
33     startsWith('back@',Na).
```

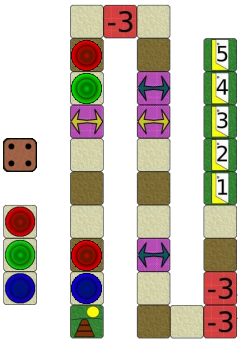
Example 3.8: Observing ADVANCER

An implementation of Observing is visible in Observing ADVANCER in 3.8. This strategy was already introduced at the beginning of this chapter as a more advanced version of ADVANCER. This is due to the fact that the difference between ADVANCER and Observing ADVANCER is the symbiotic strategy Observing. Observing is used to extract essential information

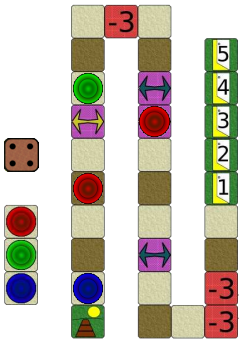
about the layout of the board and then use this knowledge to gain additional benefit for it. The current version of Observing used in this context does not feature an analysis of the movement of other players, because OBSERVING focuses on the environment and especially on the layout and advantages gained by using the terrain.



(a) Observing ADVANCER before move



(b) ADVANCER after move



(c) Observing ADVANCER after move

Figure 3.5.: Observing ADVANCER play strategy

By analyzing figure 3.5 one can see the different result between ADVANCER's and Observing ADVANCER's move decision. In the first figure (figure 3.5a) the situation before the move is shown. Both strategies use the same setting and dice value, but make a different decision due to the extra information Observing delivers. Since ADVANCER does not recognize any board

features, it chooses the most advanced gaming piece as shown in figure 3.5b by analyzing the given limited information. Observing allows now to gain additional knowledge and broaden the horizon of ADVANCER towards Observing ADVANCER. Thus, Observing ADVANCER can select another gaming piece knowing that this selected piece will advance the most in the current turn, see figure 3.5c.

So by knowingly enhancing ADVANCER using Observing as enhancement created a more efficient strategy. This new strategy Observing ADVANCER still allows us to understand the indented basic strategies behind it.

Observing PALS utilizes Observing in a different way than Observing ADVANCER. ADVANCER uses Observing to gain additional advancement of fields that would otherwise have been ignored. PALS, on the other hand, is not intended to advance at all cost. Instead of winning a match, the main goal of PALS is to keep all gaming pieces as close together as possible. Observing then is employed to minimize the risk of breaking a cluster of its own pieces. By doing so, it alters the selection of gaming pieces from always selecting the last piece to selecting the pseudo-optimal piece for minimizing the distance between the first and the last piece. Yet, the goal of Observing PALS is still to reach the “GOAL” area. Thus, the distance between all pieces of this strategy will sometimes not be minimal. Otherwise, the NPC’s, controlled by Observing PALS, would try not to move towards the “GOAL” area.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
8          die(D),
9          B is Xa+D,
10         minTile(B,'switch@'),
11         worth1(R,[X,Y,Z],Best).
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not( backTile(X)), not(goalTile(X) ),X < Xa,  !,
17         worth1(R,[X,Y,Z],Best).
18 worth1([_|R],A,Best) :- worth1(R,A,Best).
19
20 minTile(N,String):- board(N,B),
21     startsWith(String,B),
22     firstTileOf(N,B).
23
24 firstTileOf(N,String):-
25     board(N2,StringB),
26     startsWith(String,StringB),
27     N2 < N, !, fail.
28 firstTileOf(_,StringB).
29
30 backTile(X):-
31     die(D), B is X+D,
32     board(B,Na),
33     startsWith('back@',Na).
```

Example 3.9: Observing PALS

Comparing Oberserving ADVANCER (example 3.8) and Observing PALS, which is shown in example 3.9, allows me to demonstrate the similarities of both Observing implementations. As mentioned before, symbiotic strategies are really strong interweaved with basic strategies. This complicates the extraction of the symbionts and makes a comparison difficult. Therefore, I used ADVANCER and PALS, which are, judging from their code representation, quite similar

but from their intended behavior somewhat different.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
8          die(D),
9          B is Xa+D,
10         minTile(B,'switch@'),
11         worth1(R,[X,Y,Z],Best).
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not( backTile(X)),
17         not(goalTile(X) ),X < Xa, !,
18         worth1(R,[X,Y,Z],Best).
19 worth1([_|R],A,Best) :- worth1(R,A,Best).
```

a) Observing PALS

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1(L,Stone,Stone):-
8          die(D),
9          [A,_,_] =Stone, B is A+D,
10         minTile(B,'switch@'),!.
11
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not( backTile(X)),
17         not(goalTile(X) ),X > Xa, !,
18         worth1(R,[X,Y,Z],Best).
19 worth1([_|R],A,Best) :- worth1(R,A,Best).
```

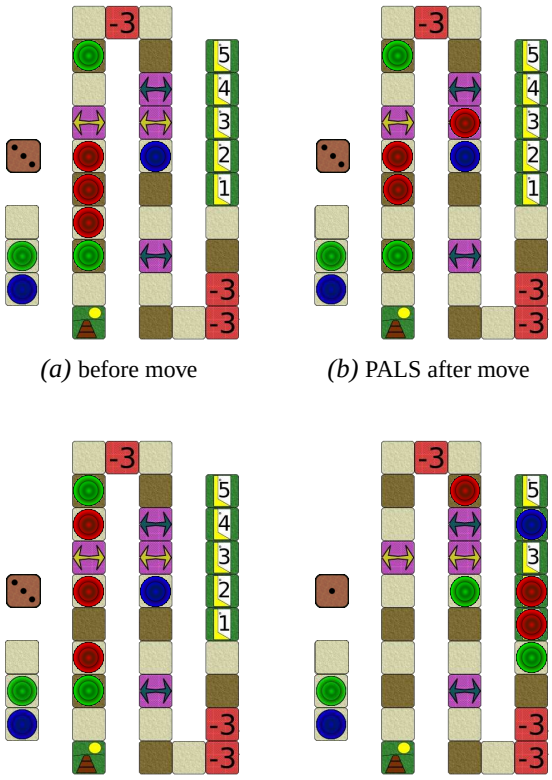
b) Observing ADVANCER

Example 3.10: Observing

In example 3.10, the most important parts of Observing ADVANCER and Observing PALS are contrasted. The common parts of both rule bases have been toned down, whereas the differences of both implementations are in bold. The difference between ADVANCER and PALS is visible in line seventeen. All other differences between both strategies are based upon Observing's different behavior. It is due to Observing's support of PALS that the solutions differ from ADVANCER's. To be more exact, Observing PALS does recognize "SWITCH" fields as does Observing ADVANCER, but treats them in a different way. In 3.10a, the third clause of the predicate worth1(L,S1,S2) keeps track of the "SWITCH" fields. This clause allows Observing PALS to choose a gaming piece which will not move onto such a field, because using "SWITCH" fields would destroy a cluster.

A very important difference noticable to an observer appears in lines ten and eleven. In line ten, the cut operator ! is omitted. This allows backtracking, in case line eleven does not

provide a sufficient solution. Line eleven is important, because it implements the previously mentioned intended behavior of Observing PALS, which is, not to use a “SWITCH” field if it can be avoided. If the cut operator in line ten was still be present in Observing PALS, situations could occur in which Observing PALS could not return a solution, because it would not select a piece which must go onto a “SWITCH” field.



(a) before move *(b)* PALS after move

(c) Observing PALS after move *(d)* Observing PALS dead end

Figure 3.6.: Observing PALS play strategy

Previously, a dead end situation was mentioned in which Observing PALS cannot return a result. This situation could occur if in example 3.10a in line ten the cut operator was still present. One of these situations is illustrated in figure 3.6d. Because two pieces are already in the “GOAL” area, they cannot move. The last available gaming piece would have to move

onto a "SWITCH" field which Observing PALS normally tries to avoid. If the cut operator was present, then the strategy using this implementation would be stuck in a dead end, without the cut operator alternative solutions are still possible. This example again reveals that translating the informally given definition of a strategy into a formal language can be tricky and bears the risk of including or omitting behavior.

Keeping in mind that even small changes to the implementation can have a tremendous impact on the resulting behavior, I want to show the result of enhancing PALS with Observing, creating Observing PALS. Figure 3.6a shows a situation before a move was made. The dice has already been rolled and the NPC can select which of its pieces will be moved three fields. The intended behavior of PALS is to move all pieces relatively close together and move all gaming pieces in a cluster, if possible. Having a board which does not allow pieces to stand longer on certain fields, such as the "BACK-3" fields breaks a cluster. But it is still possible to move all pieces close together.

Having altered two JOSTLE matches to fit the setting of figure 3.6a allows us to see the different behavior of PALS and Observing Pals. Because PALS is not capable of analyzing or understanding the layout of the board it will always move the last gaming piece to achieve its goal. By doing so, it is easy to see that some moves will lead to a separation of the gaming pieces, although it could have been avoided. In figure 3.6b the result is shown when using PALS with the setting from figure 3.6a. The presented move separated the gaming pieces,thus breaking the cluster, and created a maximum distance of nine fields between the first and the last gaming piece. This move would have made sense for ADVANCER, but not for PALS. Utilizing Observing in figure 3.6c, Observing PALS is in the situation of avoiding this long distance of its controlled pieces and creating a maximum distance of six fields between the first and the last gaming piece. This move still breaks the cluster, but keeps all owned pieces closer together as the move of PALS did.

Aggressive is an enhancement strategy focusing on jostling. Its main "GOAL" is to use given opportunities to jostle the gaming pieces of other players. Observing had focused on a more fitting solution for the basic strategies it enhances, but Aggressive alters the intended behavior of the basic strategy a lot. In contrast to a constructed strategy based on Observing the

resulting strategy using Aggressive does not directly benefit from using Aggressive.

One might now say that Aggressive is not a symbiotic, but a parasitic strategy. Yet, I would have to disagree with this assessment. The main goal of all strategies discussed here is to create the impression of a human player. This is the most important goal of all strategies. Aggressive does contradict its host strategy in a way, but still fulfills the intended "super" goal to create this impression. It then adapts the behavior of its host still in a beneficial way.

Aggressive ADVANCER is an ideal example for an adapted strategy that opposes the intended behavior of its host. ADVANCER is designed to select and move a piece which, by using the constrained view on the environment, can advance the most. Aggressive alters this behavior and would rather move a piece that can jostle a piece of an opponent player. The resulting Aggressive ADVANCER is implemented in example 3.11 and selects pieces which can jostle other players pieces. If this is not possible than it will select a piece which can advance the most.

```
1  worth([S|L],Stone):-
2          worth1(L,S,Stone).
3
4  worth1([],A,A):-
5          [X,_,_]=A,
6          not( goalTile(X) ).
7  worth1(L,Stone,Stone):-
8          die(D),
9          [X,_,Z]=Stone,
10         B is X+D,
11         findOpponentPiece(B,Z,Za),!.
12 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
13         goalTile(Xa),
14         worth1(R,[X,Y,Z],Best).
15 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
16         not(goalTile(X) ),X > Xa,  !,
17         worth1(R,[X,Y,Z],Best).
18 worth1([_|R],A,Best) :- worth1(R,A,Best).
19
20 findOpponentPiece(X,Z,Za):-
21         stateSearch(X,_,Za),
22         Z \= Za.
```

Example 3.11: Aggressive ADVANCER

The symbiont Aggressive is embedded in the second clause of the worth1 predicate. The intended behavior is created by comparing the possible next positions of all owned pieces and then checking if these positions are occupied by pieces of other players using the findOpponentPiece(B,Z,Za) predicate. If this is the case then findOpponentPiece(B,Z,Za) not only acknowledges this move but also identifies the player whose piece Aggressive ADVANCER is going to jostle.

In a second, enhanced version of Aggressive we shall call it **Revenging** it would be possible to differentiate between opponents. Therefore, only those pieces could be jostled which were hostile against the NPC, controlled by this enhanced version. From it, it is easy to see that by clearly defining strategies it is possible to enhance special parts, which can be extracted and evaluated separately.

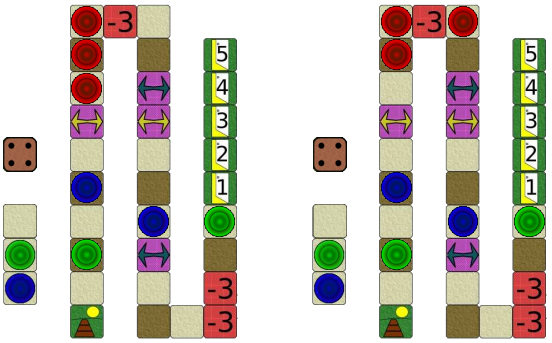
Lingering is the last symbiotic strategy I want to describe in this context, because the already presented selection of strategies should be sufficient to demonstrate their enormous diversity. Due to the fact that this will be the last strategy described here, I decided to choose Lingering. This is a more than interesting strategy, because it observes not the complete board, but only the "GOAL" area and depending on the situation switches between PALS and ADVANCER. This shift of behavior is instantly visible.

```
1  worth([[_,_,Z]|L],Stone):-
2          schizoid(Z,'goal@'), !,
3          worth1(L,S,Stone).
4  worth([[_,_,Z]|L],Stone):-
5          worth2(L,S,Stone).
6
7  worth1([],A,A):-
8          [X,_,_]=A,
9          not( goalTile(X) ).
10 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
11         goalTile(Xa),
12         worth1(R,[X,Y,Z],Best).
13 worth1([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
14         not(goalTile(X) ),X > Xa, !,
15         worth1(R,[X,Y,Z],Best).
16 worth1([_|R],A,Best) :- worth1(R,A,Best).
17
18 worth2([],A,A):-
19         [X,_,_]=A,
20         not( goalTile(X) ).
21 worth2([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
22         goalTile(Xa),
23         worth2(R,[X,Y,Z],Best).
24 worth2([[X,Y,Z]|R], [Xa,Ya,Za], Best):-
25         not(goalTile(X) ),X < Xa, !,
26         worth2(R,[X,Y,Z],Best).
27 worth2([_|R],A,Best) :- worth2(R,A,Best).
28
29 findOpponentPiece(X,Z,Za):-
30         stateSearch(X,_,Za),
31         Z \= Za.
32
33 schizoid(Z,Name):-
34         findOpponentPiece(X,Z,Za),
35         board(X,Name1),
36         startsWith(Name,Name1).
```

Example 3.12: Lingering

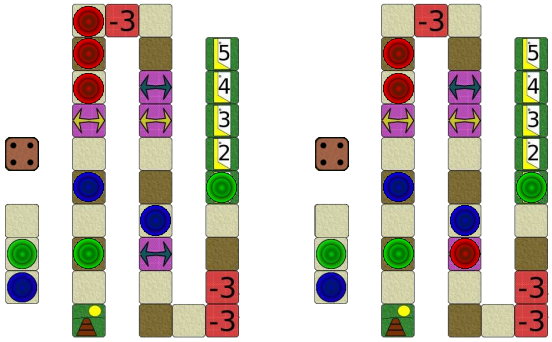
In example 3.12, a version of Lingering is shown in which Lingering switches between PALS and ADVANCER when a certain field or field type is in use by other players. This switch is visible in line two, where the `schizoid(Z,'goal@')` predicate is called. This call of `schizoid` can be interpreted as:

> *" When any other player has one of his/her gaming pieces in the "GOAL" area, schizoid switches from PALS to ADVANCER to speed up the advancement because the game will end soon."*



(a) Setting One:before move *(b)* Setting One: after move

Figure 3.7.: Lingering play strategy Part One



(c) Setting Two: before move *(d)* Setting Two: after move

Figure 3.7.: Lingering play strategy Part Two

In order to make this behavior visible, I chose two different settings of JOSTLEthat differ only in one piece that is situated in the "GOAL" area.

In this small example, the power of such enhanced strategies becomes clear. By writing strategies which react on a certain event or behavior of other players, I think that these players will recognize a behavioral reaction. Thus, they might create a "Theory of Mind", as this is called in Cognitive Science [3], for the NPC strategy.

For easy and consistent creation of rule-based NPC behavior, it is important to validate changes in existing systems or evaluate new rule sets, which should aim at the intent of the NPC's designer.

The term designer was used on purpose, because in modern computer games most of the people take an artistic perspective when they create the content of games. This approach only tries to utilize science, but without the correct tools there is no utilization.

The behavior of various combinations of a basic and a symbiotic strategy is summarized in figure 3.8. In this table, Lingering uses ADVANCER as a second strategy to which it switches after a certain event trigger.

	Observing	**Aggressive**	**Lingering feat. ADVANCER**
RANDOMIZER	not meaningful	random jostling	straightened behavior after switch
ADVANCER	better advancement	advances slower but hinders others players	not meaningful
PALS	higher cluster rate	advances faster, hinders other players but breaks clusters	breaks clusters while in eleventh hour panic
EQUALIZER	not meaningful	breaks order in which stones are moved but hinders other players	breaks order while in eleventh hour panic

Figure 3.8.: Combination Table of Simple Strategies

3.4. Summary

By summarizing this chapter, I want to emphasize certain points mentioned earlier. The chapter started with a notation that allows the system and even a human to analyze different matches. This introduction was necessary to allow the reader to understand the examples provided later on. Because I think that all of the strategies used in digital games can be structured and subdivided, I first provided examples of elemental strategies and, later on, presented a way to combine them.

By using basic elements, such as basic strategies, and enhancement strategies, such as the symbionts, I hope it is possible to create a better understanding of artificial intelligence in digital games and improve NPC AI in digital games. In order to achieve this, the AI STRUCTURE COMPOSITION might prove to be a useful theory for constructing NPC behavior. I also think that the AI STRUCTURE COMPOSITION is not suited for the creation of certain game AI, focusing on path-finding or language processing because for these categories of AI, efficiency is the main focus, not human behavior. An NPC created through AI STRUCTURE COMPOSITION is capable of selecting a path that a human being would choose, but not capable to create such a path.

The next chapter introduces and discusses the GRINTU framework.

4. GRINTU-Framework

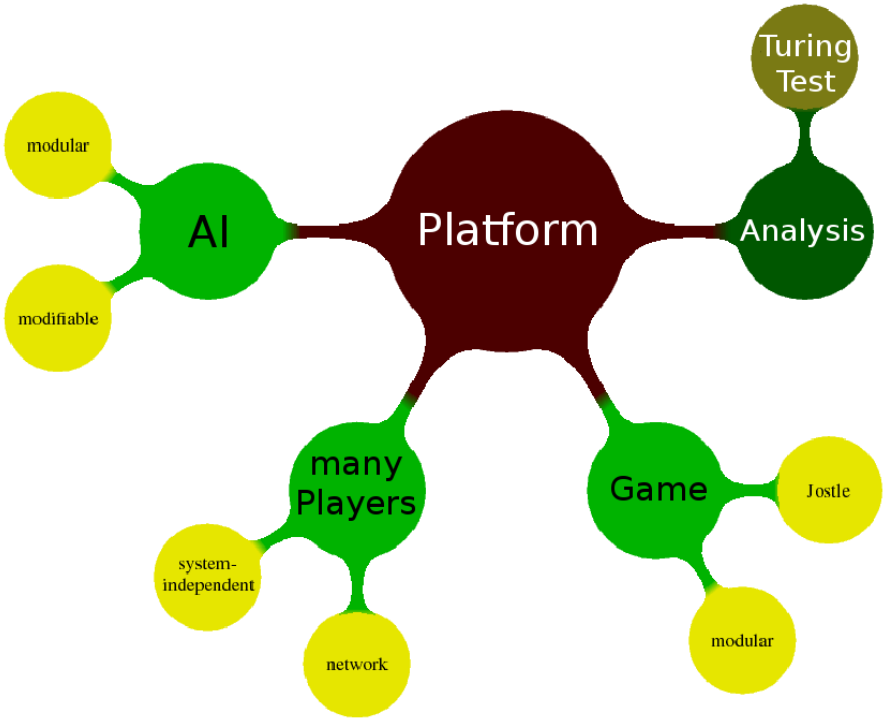


Figure 4.1.: Mindmap of GRINTU

The GRINTU framework was designed to meet various requirements. Due to its main requirement to emphasize the aspect of performing TURING Tests with different game strategies, the framework was given the name GRINTU, which is an anagram for TURING. The platform is strictly separated into different parts (Figure 4.1), namely the network, the game library, and the

analysis package.

In order to meet the requirement of handling many users, we choose a client server architecture, which gives us the possibility to store the data the user creates in our data base. A different option would have been to create different deliverables and distribute them, relying on the probands to send us the data that is generated while using the software. Since this option is more risky and the return rate would probably be low, we decided against it.

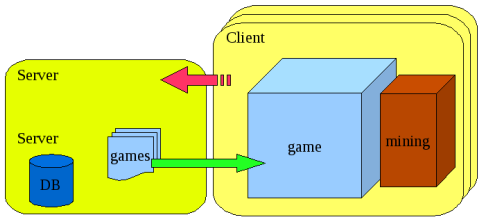


Figure 4.2.: GRINTU Overview

An object view of the resulting software appears in figure 4.2. The figure shows that there are different clients and only one server. All games and nearly all calculation takes place on the client side so that the server is able to manage a lot of connections and still can coordinate the communications between different clients. If one starts a game, the server receives a request from a specific user and sends all required data to start a session. The server will use the database to determine which NPC should play against which proband. After a successful game the server collects the results of those games.

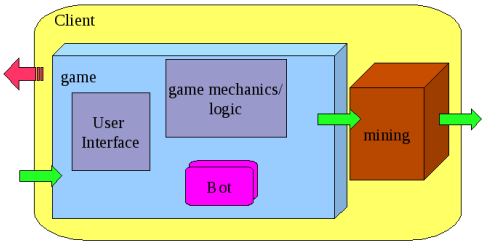


Figure 4.3.: GRINTU Client

In figure 4.3 on page 45, the constellation of a client is illustrated. A game object consists of the user interface, which isvquite obvious, the game mechanics, and a local version of the adver-

sary NPC. Running the NPCs by the clients reduces the server load exponentially, because in the current setting of our experiments, each client should play against two NPCs. This means that if 40 probands are playing at a time, the server has to calculate 40 to 80 NPCs simultaneously.

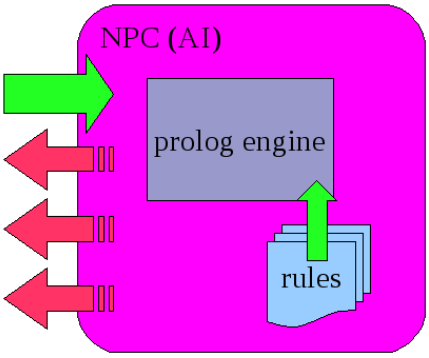


Figure 4.4.: GRINTU NPC

Figure 4.4 illustrates the implementation of the current PROLOG NPC module. With this module it is possible to run native ISOPROLOG scripts, which comes seems quite useful, because with PROLOG it is easier to analyze rule based NPC strategies than to write the strategies in JAVA or C. After getting feedback from the probands, we can easily modify the rule base and rerun the TURING Test for further refinement.

The data mining module (see 4.3) is still under development, but due to the construction of the client, it is possible to read nearly all users interactions with the game, such as mouse movements, pauses, and key inputs. We are also able to log the course of gaming for a later analysis. Another part of the data mining module is the questionnaire, which is shown after each game and is used for a survey regarding the behavior of the other players. The data collected by the data mining module is send to the server and stored in the database, using cross references for all participating players, both human players and NPCs.

In the following chapter, we shall discuss the advantages of this tool and show ways to use the GRINTU framework for the aforementioned purpose.

5. Evaluation & Refinement

In chapter 3 I introduced rule-based strategies which are used to control the behavior of NPC's. These first strategies are at an elemental level and do not carry the human factor. The human factor was defined at the beginning of this work as an hypothetical construct allowing me to determine the essence of human play behavior. By definition every strategy which successfully passes my adapted TURING Test (p. 5) contains the human factor. The creation of such a construct as the human factor was necessary because the true nature of human play behavior is unknown. The crucial question now is whether a strategy passes the TURING Test or not. To answer this question, an test is necessary which needs to be viewed as a process of evaluating if a strategy is human believable or not. These answers are in nature non deterministic or probabilistic. To validate the results of the TURING Test, I will set up an evaluation carried out by the GRINTU framework. The GRINTU framework represents the system I used for the creation of the games and of the match-making, as well as the data-mining utilized during the evaluation. This evaluation is essential, because, without it, the credibility of the TURING Test cannot be guaranteed. However, the results of the evaluation and the data is not only used to validate the TURING Test and certify the existence of the human factor in certain strategies, but it is also used later on to refine the rule bases and thus increase their quality in terms of human believability.

5.1. Evaluation

In contrast to verification, which is usually understood as thoroughly formal, evaluation is not possible without expert knowledge. Since our research field is in the games sector, each player is considered an expert for us.

There is no play without action; there is no game without interaction.

Interaction can take place between a player and some opponent, which can be either an NPC or another player. Therefore, most players have an intuitive understanding of what to expect from an opponent.

5.1.1. Preparatory Work

This understanding of games or, in other words, this specific domain knowledge is refined the more often a person plays games. For our purpose I am limiting this domain to board games, and since I want to research digital games for which the GRINTU framework is specialized, I want to use digital board games.

As mentioned above, the more often games are consumed in a healthy way[1], the better the domain knowledge. We use this special domain knowledge to evaluate our strategies and then use this gained expert base to refine the rule based system.

In chapter 3 different strategies were presented as well as a way to create completely new strategies. This new theory of constructing complex rule based strategies was called AI STRUCTURE COMPOSITION. By creating a versatile pool of basic and enhancement strategies it should be possible to create nearly all kinds of strategies. What is the big advantage if, after having accumulated these masses of strategies, they cannot be ranked by some useful metric. My ranking scale is the believability, or in other words, the quality of a strategy in comparison to a human player. By using a way of validating the construction process and the corresponding strategies, I hope it will be possible to:

[1] I would like to give a short definition of a healthy way by defining its counter part. Here an “unhealthy way”, is defined as the extreme consumption of games. Due to matters of time and space, I cannot take other aspects into account. It is my belief that extreme consume of media in our case digital games leads to discrepancies in social behavior, which represent error sources for this research. Also, the question of how the consumption of games changes the perception of behavioral values has to be included into some other research.

1. create a better understanding of AI
2. create more believable AI
3. extract the human factor mentioned in the introduction

"Who is an expert?", one might ask. Even our parents and grand parents have or had most likely expertise in the games domain, because it comes to us naturally in some way.

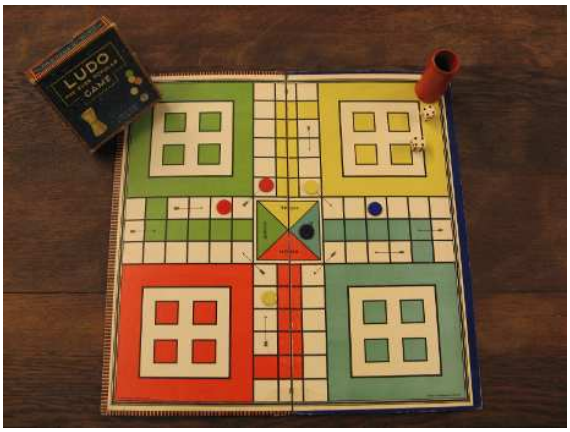


Figure 5.1.: Ludo board *photographed by M.L. Rieser, 2007*

Imagine following setting:

One hundred years ago a grandmother is sitting with her two grandchildren around a table and all three participate in a game of "Ludo" [12], see Figure 5.1 for the board layout[2]*. Both children, even when still young, expect their grandmother to behave in a certain way after some games, because they correlate the grandmothers real-life behavior with her "in-game" behavior. Like playing non aggressively and letting the children make mistakes and patiently play on. It would be obvious for both children, if their grandmother would start playing in a totally different way, that something is not how it used to be. But if the grandmother would only make little changes in her behavior over time, like adapting to the increasing skill of the children, the children would probably not recognize it.*

[2] A version of Ludo is mostly called "Mensch ärgere dich nicht" in Germany or "Trouble" [11] in the USA, after a little redesign during the 1960's.

Evaluation Approach

To achieve the previously mentioned goal of creating a believable rule-based game AI, the pool of strategies has to be continuously evaluated. To achieve this, human judgment on these pools is used as quality measurement for the evaluation. If one does not evaluate these strategies and their process, the system would become unable to judge whether a certain strategy is better or not, and create strategies which are unable to fulfill their intended purpose. To do this, I use a tool called "the Questionnaire", which is integrated into the GRINTU-framework to evaluate the quality and believability of all used rule based strategies.

To achieve this, the Questionnaire has to guarantee certain quality standards. Therefore I used [8] as my guideline. To collect the data properly, the evaluation process of all used rule based game AI has to offer some of the following features:

- **Utility** is meant to identify and represent the interests of the indented users. During the design of the evaluation, the intentions and wishes must be clarified and taken into consideration. The evaluation report should be comprehensible and clear.
- **Feasibility** is another important feature, because the evaluation should be diplomatic, thought through, and cost efficient. It is import to keep the intrusion and burden on the respondent to a minimum when gathering the information. Is is also important to minimize conflicts of interest under the different user groups. Thus, when designing a questionnaire for different scientific problems, the blending should be minimal.
- **Propriety** standards are required because every evaluation has to take the responsibility upon ethical or legal questions. Because of the data collected from the user groups, it is important to protect and respect the dignity and welfare of all involved persons. All participants have to be informed what kind of information about them is gathered and, if possible, should be allowed to view the findings.
- **Accuracy** is the fast standard that needs to be fulfilled. The evaluation purpose has to be declared as clear and exact as possible. All information sources have to be documented as well as error sources and evaluation procedures. In the end, a justified summary and all collected data has to be archived.

When an evaluation is performed, all previously mentioned feature standards cannot be applied. But according to "Forschungsmethoden und Evaluation" [4] it is advisable to fulfill as many standards as possible. By following these standards, the quality of an evaluation and the number of findings increase greatly. I also believe that an evaluation planned and carried out according to these principles attracts more volunteers, because of the professionalism and seriousness of such an evaluation.

To evaluate the recognition of the the behavior of an NPC controlled by a rule based strategy, it is necessary to set up a test that keeps the all controllable possible errors to a minimum. Naturally, this is an obvious statement, but it is quite difficult to succeed in the creation and execution of such a test. As mentioned before, I aim at an evaluation of rule based strategies and their behavior. This is required for a refinement of those strategies to create more believable NPCs. By doing so, our main goal is to extract the human factor[3].

Creating a test for this setting would be really easy if human interaction was not required. For a rating of efficient strategies or the comparison of path finding algorithms tests can be automated and at the end the result is returned by a clear result. The difficulty of comparing rule based systems with human behavior is that one has to involve actual people as judges.

As noted earlier, in the chosen domain of digital games, every human who plays games is an expert. So to get the required information, I have to design an error robust test evaluating system, utilizing the human player's judgment.

5.1.2. Test Construction

If one focuses on the domain of classic test theory than on repeating the test with the same individual, the error margin in the results should minimize gradually. If so, the reliability of the answers given is proven and the results can be expected to be true. Normally, when repeating a test in accordance with my setting, the answers will differ because of measurement errors. The respondent could be under stress, is distracted, or cannot concentrate. Perhaps the respondent is even providing wrong answers on purpose. By taking this into account, the domain of classic test theory might not be the right domain for this approach. To eliminate these measurement

[3] See page 6 for the definition of the human factor.

errors or to be more exact to reduce them, I am adopting techniques from the social sciences to create more trustworthy results.

Test Items

As for the test itself, it is a composition of questions the respondent can answer in different ways. From now on for matters of convinience, I will call the questions of the test “items”. It is useful to view a question as an “item”, because a question is an item of the set of questions which form the test. The goal of such a composition of items is:

a) to get answers to the implicitly formulated problem

b) to keep the respondents from guessing the answers and thus falsifying the answer to a minimum

To achieve this goal, the first thing one has to do is to look at the items themselves. To do so, I will introduce four kinds of items:

- **Open Answer**: Questions from this type allow the respondent to answer freely without any restrictions. Thus, the answer can be given in a playful or artistic way, meaning a picture or longer text passages. These items are very useful as material basis for later tests because they offer a lot of insights to the respondent and his/her opinion on the subject of the question. Open answers are not very good for automated analysis and require a lot of subsequent human input.

- **Half-Open Answer**: As it is the case with open items, half-open items allow free answers, but with a certain restriction. Open items do not necessarily have right or wrong answers, and allow thus more creativity. Half-open answers, on the other hand, can only be correct or not. The problem then is that the evaluator has to decide afterwards what answer is right. This sometimes depends on fine nuances in the evaluators interpretation. Another big disadvantage of half-open items is that they are not analyzable through automation. These items are very popular with respondents, because they offer much liberty. On the other hand, they are not very suited for large or automated evaluations.

- **Multiple Choice**: Multiple Choice questions are very good for a computer enhanced evaluation, because the respondent has to pick an answer out of a given pool of answers. The problem with multiple choice is that the questions and answers must be compiled in a way that the right answer is not obvious. Another disadvantage of multiple choice is that it only requires recognition of answers instead of reproduction when it comes to open and half-open items. The biggest disadvantage is the risk of guessing answers. The impact of guessing for the reliability denotes [4] three factors, namely duration of test, success probability, and number of persons which were instructed to guess.

- **Correlation:** This type of item is similar to Multiple Choice items, but yields additional information. There are mostly two sets of elements and the respondent has to correlate elements of both sets. The elements mentioned can be statements, pictures, or sounds. The result can be evaluated equally well as it is the case with multiple choice items.

As it is possible to see, the two items best suited for an questionnaire offer the possibility to easily guess the correct answer. Another aspect that has not been addessed, yet, is if somebody intentionally provides a false answer or is randomly picking answers to speed up the answering process. In [4], they propose a “guessing adjustment” by giving negative points for each wrong item and positive points for each right item. However, in my case, this would not be feasible, because the respondent does not receive a score or a grade after passing my questionnaire. An interesting statement concerning multiple choice items is given in [4]. Items allowing the respondent to have a neutral answer are very hard to analyze, because of their meaning. If such elements are present, it should be made clear to the respondent if this neutral element stands for:

a) The respondent does not know the answer.

b) The respondent is not sure what to select.

c) The respondent does not want to answer this question.

d) The respondent cannot decide between two answers.

This demonstrates the difficulty of having to deal with neutral answers. Their use should thus be carefully evaluated.

Above I stated that every gamer is an expert. This was not entirely correct. There are some people who are able to expose some strategies as artificial player and some who cannot expose said same strategies. These so-called super experts who are able to identify strategies reliably, are not there to expose all strategies, but are needed as upper limit and fix points during the refinement. To extract these super experts, I want to use a theory called Item-Response-Theory, abbreviated as IRT [9], which allows me to group people according to their abilities. A model for this theory is the Rasch Model [7], which allows easy extraction of abilities. After obtaining these, it is possible to group them by abilities, extract the super experts, and analyze the newly constructed user space.

After gathering all of this information about how an evaluation test should be done, I will present the first evaluation test setting.

5.1.3. Alpha Test

The first setting will involve all previously mentioned strategies starting from RANDOMIZER (p. 21) to Lingering (p. 41). They will be used as a basis for further refinement and are used for comparing the advancement of future strategies. The user base for this evaluation should be around 100 persons. I hope that at least 25 percent of all participants will be available and willing to participate in more than one game, because every absolved game minimizes the error value of wrong answers and allows me to extract the requirements for super experts.

After every finished game, the questionnaire will pop up and ask questions regarding the following topics:

- human or artificial behavior of a player
- enjoyment of the game
- visibility of other players strategy or goal
- level of challenge regarding certain players

The items regarding the enjoyment of the game will be Multiple Choice, including a neutral position. The items regarding the decision between human or artificial player will also be multiple choice, including an option that the respondent can pick if he cannot differentiate between them. The items regarding the visibility of the strategy will be Open Answers. In this case I decided against Multiple Choice, because I want to gather the insights of the players' decision. The items for the challenge will be a Multiple Choice with four options:

a) not challenging

b) somewhat challenging

c) overstraining

d) unsure

whereas there is no real neutral position. Some challenge items will also be Half-Open answers allowing the user to state his decision.

The test will be repeated after every game, and it will offer the player to skip the questionnaire after the second time. Although this might reduce the respondents filling out the questionnaire, this approach will minimize the error margin of the respondents who are intentionally providing wrong answers. I hope to gather a small group of people who will play the games frequently in order to enhance the results of the evaluation.

I believe that I can accomplish this by communicating the goal of the evaluation and by informing the users regularly on the progress of this project on its website. The users will be given an account with a password when they join the website. The only information I require is a valid email address. Then, the users can answer a small questionnaire about themselves. This is done to create the dimensions for the IRT.

In contrast to other approaches, I will not force them to enter this information and allow them to fill out what they are comfortable with reminding them that wrong information might jeopardize the outcome of the evaluation. Changes regarding their personal settings will be monitored to guarantee error safety. For example, if a person changes his/ her gender information more than twice, the person will be marked as possible falsifier. Removing attributes will not set

such a mark, but the exact moment will be recorded because it might be a point where people start to loose trust or interest in the project.

In addition to the data from the questionnaire, the progress of each game will also be logged in the game notation which was introduced earlier in this thesis. This notation allows to replay the game, but it will also allow me to validate some information from the questionnaire regarding the challenge level. In some future setting I hope to use more of that additional data to gain more insights into the play behavior of the users as well as validation information regarding the enjoyment of the game by analyzing, for example, the reaction time and mouse movements.

After obtaining all information from the questionnaire and the additional data from the game, I will be able to group the users according to their answers and their behavior during the matches. These groups will allow me to select the super experts and groups which are receptive for certain strategies. Super experts may be good at identifying artificial players, but may not be the most interesting group when it comes to recognizing and reacting on different generated behavior.

All users will also be given a weight in correspondence with their accuracy, when it comes to detecting strategies. This weight will be provided after it is clear which person reacts to which strategies. The super experts are an exception. They have an equal weight regarding all strategies. This weight is analyzed over time to view the progress of detecting the strategies as well as detecting falsifiers. All users start with a zero weight at the beginning, and over time the weight is adjusted according to their detection rate.

The validity of the collected data depends more or less on the expertise of the respondents. By increasing the positive weight, the super experts can be validated or normal users can be added into this group because of their high identification rate.

An approach to adjust weights may be Exponential Smoothing [5]. I think that this approach is quite natural, because it ranks more recent data higher than old data in the evaluation process and therefore, it models the humans' ability to learn by gaining experience, too.

So basically, after every match, the questionnaire collects data which is analyzed by the IRT and weighted by an approach utilizing Exponential Smoothing.

Selection Criteria I have already mentioned that some respondents are better at detecting certain strategies and good at detecting all strategies. The only issue I have not addressed, yet, is how the matches are organized. Before the refinement takes place for the first time all matches are set up in a certain way so that every user will play at least once against every strategy. After a user has finished the game against the last strategy, a new list is created including those strategies he/she misjudged and a selection of strategies other people in his/her group misjudged as well. This satisfies two functions (1) validating right decisions of the user by re-judging the strategies and (2) validating the group affiliation. When the refinement takes place, this list of strategies which all users have is adjusted according to the refinement process defined below.

Strategy Weighting The strategies themselves are also weighed according to the detection rate of their identity and the fun and challenge they generate for all players. This generates a three dimensional space in which all strategies can be placed.

After placing the strategies into the three dimensional space and validating their position over several iterations of the evaluation, it is possible to minimize nearly all error values created through the questionnaire or the personal disposition of the respondents. On the other hand, the test users are learning step-by-step to identify the artificial players. If I stopped at this point, the gathered information would be useless.

The next step is then to use this knowledge to refine the rule based system which controls the NPC to make it harder to detect them.

5.2. Refinement

Having the results of the evaluation and the different weighted strategies, it is possible to start the refinement. In this part of the process, the respondents do not need to be involved, because all work is either done by human analysts or is automated.

Refinement has two principle directions. The first direction is to create strategies which pass the TURING Test and are therefore credible human in nature. This means increasing the level of complexity of strategies. The second direction is to take these complex strategies which pass the TURING Test successfully and minimize them to a level where they still pass the test. This creates minimal rule bases which carry the human factor.

5.2.1. Phase One

The first phase of the refinement is done after a certain amount of iterations of the evaluation to minimize errors such as answers that have been guessed in the questionnaire or misjudged strategies. Most importantly the amount of iterations needed depends on the fact that all strategies should have been ranked by at least one super expert. To achieve this the basic settings in the evaluation include that every user receives a set of all strategies. After all strategies have been rated, the first refinement will be triggered.

Now there are two scenarios that can take place:

Scenario a) All strategies have been rated nearly equally often in the questionnaire. If a strategy is at that point in time not rated by at least one super expert scenario b) will be used. Otherwise the second phase of the refinement can begin.

Scenario b) If not all strategies have been rated due to the exclusion of users[4]or other factors, the lists of all users will be adjusted. This also involves new users or users joining later. The remaining unrated strategies will be rearranged in the set of strategies as the next strategies the users will be confronted with. This method guarantees that all strategies will be rated in the

evaluation, thereby achieving an equal use of strategies during the first phase.

Using the rating produced by the Exponential Smoothing allows now to select a group of strategies which were least likely estimated as artificial. These strategies have also the highest potential to evolve into a state containing the human factor. This highly rated group of strategies will most likely still contain strategies which were detected as non-human due to the early phase of refinement.

5.2.2. Phase Two

After obtaining enough information about the strategies for their rating and establishing that rating, it is possible to adjust the newly found group of more believable strategies. As mentioned in chapter 3, on implementing the informal definition of strategies, the combination of different strategies has to be done by hand. I still hope to develop an automated combination, but this has to be done in some future setting because of its complexity.

Initialization The now presented concept is based on the Darwinian approach on evolution [6], adapted to rule based strategies and in particular on game strategies. I will follow the style of evolutionary strategies (see [2, pp 48]) for a while until a set of strategies is fully undetectable as artificial. By taking each presented strategy from chapter 3.3 into the evaluation, I create an initial population $P(t = 0)$, with a size of μ elements. As proposed in [1, p.83], the default diversions σ_k for all strategies k are initialized with an integer value of three. The meaning of this diversion σ_k will be described later on.

The result ρ_k which a fitness-function would return is directly given through the implicit target function as the rating done by the respondents of the evaluation. This indicates that my evolutionary strategy does not have a directly implemented fitness-function due to the nature of human behavior which is not computable. If I knew in advance, how a human being behaved and reacted, then it would be obvious that I already have extracted the human factor.

[4] If the users do not stop evaluating the strategies there would be a point in time at which the first super user finishes his/her set of strategies. Thus, all strategies have been rated at least once. But taking into account that even super users will drop out due to various reasons is important. Otherwise it could occur that entering later phases of the refinement process would become impossible or to erroneous.

Creation of Offspring The next step creates a set of offspring (the size of the set will be λ) by randomly selecting partners, whereas every strategy has an equal chance of $\frac{1}{\mu}$ to be selected as a partner.

In this step it is possible to mate a strategy with itself, allowing it to stay unchanged, if it is fit enough, in the process.

Traditionally it was advised ([15]) to have a ratio for $\frac{\mu}{\lambda}$ of $\frac{1}{7}$ at the end of the selection. After selecting all partners in pairs (partners can be selected multiple times), the recombination and replication of new strategies takes place.

Normally, this process recombines numerical values. But for my purpose, it will select the basic and symbiotic strategies present in the parents which were selected. This process is normally called discrete recombination, because it takes either an elemental strategy from the first or from the second parent. The section of parts of the parent is possible, because every parent is constructed of at least one elemental (basic or symbiont) strategy. In the current setting, the recombination will be done manually. Thus, during the replication, a set of strings is returned specifying the basic or enhancement strategies and their order in which they should be implemented in the child strategy.

Mutation Next, the mutation of the newly created strategies takes place. For my scenario, I decided to choose a mutation which takes the midpoint of both parent diversions. To this value a random number between zero and one is added to symbolize light changes over time between parents and children. The new diversion for the child k' of both parents $k1$ and $k2$ is:

$\sigma_{k'} = \frac{1}{2} * (\sigma_{k1} + \sigma_{k2}) * N(0,1)$

This slight change, represented as the random value generated by $N(0,1)$, is added to the diversion of k', symbolizing a drift of the mutation over time. The function $N(0,1)$ returns an normally distributed random value between zero and one including both border values.

As mentioned before, each strategy k consists of a diversion σ_k, a fitness value ρ_k, and an ordered set of l elemental strategies α_k. Each $a_r, a_r \epsilon \alpha_k$ is either represented as a basic strategy b or symbiotic strategy s.

$\alpha_k = [a_1, \ldots, a_l]; a_r \epsilon b, s, 1 \leq r \leq l$

The elemental strategies $a_x, x \epsilon \mathbb{N}$ ($\mathbb{N}$ represents the set of all natural numbers) are true subsets

of Ω where Ω is the pool of all elemental strategies.

$\alpha_k \subsetneq \Omega$

The new $\alpha_{k'}$ is calculated by using $\sigma_{k'}$ to add elemental strategies to this set. This is done by the mutation $\pi_{k'}$ where $\pi_{k'}$ is represented as an integer which determines how many elemental strategies will be added.

$\pi_{k'} = \lceil N(0, \sigma_{k'}) \rceil$

When $\pi_{k'} > 0$ then $\pi_{k'}$ strategies will be added. If $\pi_{k'} = 0$ then no strategies will be added to $\alpha_{k'}$, meaning no mutation takes place. The elemental strategies are randomly picked from the pool of available elemental strategies Ω for adding them to $\alpha_{k'}$ at an random position inside α_k.

Selection After having an intermediate pool of strategies $P'(t = 0)$, the selection process starts. The most successful strategies from $P(t)$ will be directly transfered to $P(t + 1)$. This selection of successful strategies from the previous time step t is done by selecting the top rated $\frac{1}{7}$ percent from $P(t)$. As of the initial phase the population size of $P(t = 0)$ was μ_0 all following populations will have an fixed size of $\lambda = 2 * \mu_0$ this is required to guarantee the evaluation of all strategies. In some future setting it might be a good idea to evaluate the upper limit of λ' and allow the populations to grow bigger.

The rest of the remaining space in $P(t + 1)$ is filled with strategies from $P'(t)$ according to their fitness value ρ'_k. This is done by calculating an preliminary ρ'_k for all strategies $k \epsilon P'(t)$.

$\rho'_k = \frac{1}{2} * (\rho_{k1} + \rho_{k2})$ where $k1$ and $k2$ are the parent strategies of k.

Now that $P(t + 1)$ is completely filled with strategies, a new evaluation cycle can begin until the point is reached when a group of strategies is recognized as human players by all super experts reaching a minimum percentage of all strategies. In our setting, twenty percent seems to be a reasonable percentage but it needs to be evaluated empirically. If this happens, phase three of the refinement can begin.

5.2.3. Phase Three

At this particular point, a group of strategies in $P(t)$ has gained enough complexity and behavioral substance[5]. This implies that these strategies have reached a point in evolution where they have developed the human factor sufficiently. The next step is to stop the evolution of the strategies and adjust the complete process into a nearly reverse setting.

As mentioned in the introduction, my goal is to extract the human factor. To achieve this, I first started to create a pool of strategies $P(0)$. Then I evolve this pool of strategies over iterations into a pool of strategies which were able to mimic human behavior quite well. This pool of new strategies is the selected subset $P_{opt}(t) \subseteq P(t)$, where t is the time of the iteration which is accomplished, when phase three is initialized. $P_{opt}(t)$ includes all strategies which were able to mimic a human behavior on a level where no super expert could find a difference between a human player and these strategies. For later experiments it could be useful to accept a detection rate of strategies higher than zero percent to eliminate the noise which might still be present due to user created errors.

Decomposition Using this subset $P_{opt}(t)$, a new refinement cycle starts by recursively removing elemental strategies in each iteration.

By applying this new function to the refinement process, the complexity of the strategies will slowly decrease over the iterations. This process seems at first a bit senseless, because it seems to destroy the over iterations of evolution created believability. But this is not the case. Only by doing this, the goal to find a minimal rule set which yet carries the human factor is reachable.

In each iteration t, the fitness which is representing the believability of each strategy k is checked. If the believability drops, which means that the fitness decreases rapidly, the strategy k is removed from $P_{opt}(t)$ and thus from each future generation. The predecessor k_{pre} of this specific strategy fulfilled the believability in $P_{opt}(t-1)$, otherwise it would have been removed earlier. This means that the essence of the human factor must have been destroyed in k. By knowing that k_{pre} still had the human factor, we add k_{pre} to P_{final} which represents a set of most minimalistic strategies including the human factor.

[5] With behavioral substance I mean behavioral patterns which create a trustworthy impression of human behavior.

Because now there is a new free slot in $P_{opt}(t+1)$, a random strategy of $P_{opt}(t)$ is picked and gains the possibility to create two clones each with another removed elemental strategy. This is done for every free slot in $P_{opt}(t+1)$. This means that in $P_{opt}(t+1)$ there are strategies $k1$ and $k2$ which share the same predecessor. Thus, when $k1$ and $k2$ do not fulfill the fitness criteria anymore, the predecessor will only be added once for both strategies.

The refinement stops when $P_{opt}(t+1)$ is empty which means that in $P_{opt}(t)$ there must have been at least one strategy k, which was believable. Thus, k is added to P_{final} and P_{final} is ordered by the degree of complexity of the strategies k, where $k \epsilon P_{final}$. Complexity in this context may be estimated by the number of elemental strategies.

Now the refinement process can be stopped and the strategy with the lowest complexity is returned as optimal solution.

5.3. Summary

In this chapter, I have illustrated two parts of a circular process. The first part was the evaluation of strategies and TURING-testing them. By doing so, we created a ranking of these strategies, which is used in the second part, the Refinement. During the refinement, the strategies were modified and even new strategies were created by using the old strategies as a basis. These newly created strategies were then used in a new evaluation, creating a cycle of evaluation and refinement of strategies to enhance the resulting behavior of these used strategies. The process is stopped at a point when a reasonable number of strategies inherits the human factor, defined in the introduction (p. 6).

The complete process of Evaluation and Refinement is also illustrated in figure 5.2 to allow for a better understanding.

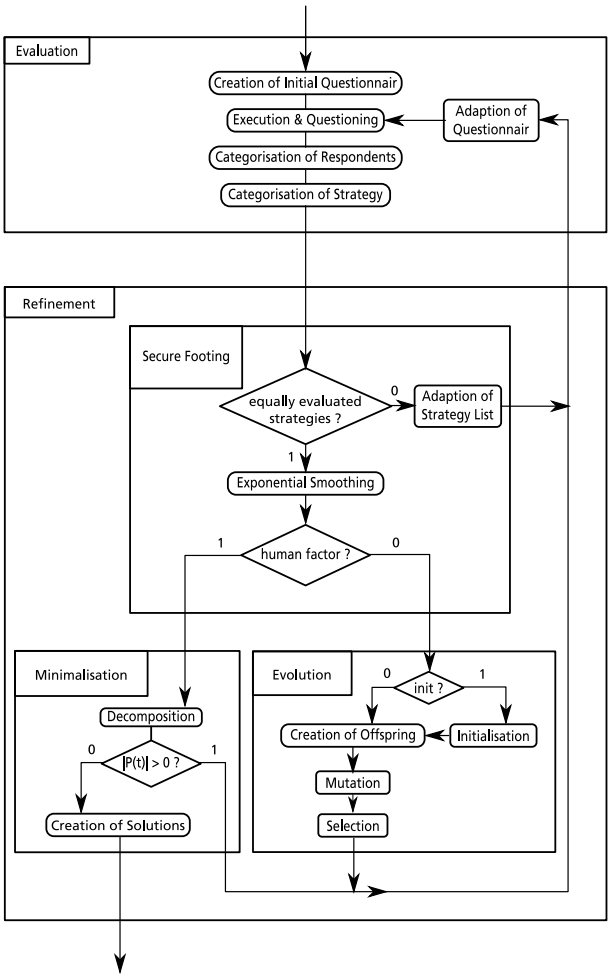


Figure 5.2.: Evaluation & Refinement process

6. Perspective

At the beginning of this thesis, I presented the idea of the human factor, a construct which allows us to determine human behavior in digital environments. This so-called human factor represents the technical essence of human behavior, which means that a person cannot differentiate between a human being or an NPC, controlled by a strategy which has adopted the human factor. The main idea behind the creation of a construct such as the human factors was that for a game in this example the board game JOSTLE a minimal rule set can be found, which inherits the ability to mimic human behavior reasonably well. In chapter 3 a basic pool of strategies was presented, which was created as a base system for developing more complex strategies based on the AI STRUCTURE COMPOSITION theory presented in the same chapter. In the last chapter, I constructed an iterative process of evaluation and refinement, utilizing techniques from the social or human sciences and from evolutionary research to create a system able to find such a minimal believable strategy. This is done by evolving the given strategies into a state where they inherit the human factor.

Throughout the presented thesis, new interesting research fields opened up allowing (a) a deepening of the presented research or (b) a solidification and validation of presented processes and theories.

The next possible step derives from the field of social and human sciences and is an executing of a first evaluation and refinement process. An inclusion of an complete evaluation in this work was impossible because a sophisticated and implemented evaluation is equal in space and time to the present work.

Another important step is the enhancement of computational techniques by creating modules for automated combination of and decomposition towards basic strategies, which were presented in chapter 5.

A future project will also be the inclusion of different games into the GRINTU framework to check and validate the transfer of knowledge from game AI to different environments. If this transfer of knowledge is detectable, a step-by-step program is required to increase the complexity of the used games to achieve the creation of modern game AI which inherits the human factor.

A. Additional Information & Source Code Repository

A.1. Additional Information

Handling Inactive Users If a user does not log in after a period of time, the account is set inactive and some time later removed. Copying time guidelines from actual online games such as WORLD OF WARCRAFT or PUZZLE PIRATES allows me to specify these time restrictions. A user will be marked inactive after not having logged in for about four weeks. After six month the user data will be marked to be deleted. In this purging process only user related information will be deleted, not his/her ratings and answers as long as they do not concern private information.

A.2. Prolog Sources

```
1  %X = field number
2  %Y= Piece ID
3  %Z = Owner ID
4  stateSearch(X,Y,Z) :-
5          state(X, List),
6          member([Z,Y], List).
7
8  newState(f(X,Y),Z):-stateSearch(X,Y,Z).
9
10 allStates(L,Z):-
11         findall(S,newState(S,Z),L1),
12         reformat(L1,Z,L).
13
14 reformat([],_,[]).
15 reformat([f(X,Y)|R],Z,[[X,Y,Z]|R1]):-
16         reformat(R,Z,R1).
17
18 boardsize(N) :- board(N,_),
19                          largest(N).
20
21 largest(N1) :- board(N2,_),
22         N2 > N1, !, fail.
23 largest(_).
24
25 backTile(X):-
26         die(D),
27         B is X+D,
28         board(B,Name),
29         startsWith('back@',Name).
30
31 goalTile(X):-
32         board(X,Name),
33         startsWith('goal@',Name).
34
35
36 goalsize(S) :- boardsize(Max),
37         board(N, 'goal@1'),
38         S is Max - N.
39
40 minTile(N, String):- board(N,B),
41         startsWith(String,B),
```

```
42          firstTileOf(N,B).
43
44 firstTileOf(N, String):-
45          board(N2, StringB),
46          startsWith(String, StringB),
47          N2 < N, !, fail.
48 firstTileOf(_, StringB).
49
50 startsWith(A,B):-
51          atom_chars(A,X), atom_chars(B,Y),
52          startsWith1(X,Y).
53
54 startsWith1([],L).
55 startsWith1([A|Tail1],[A|Tail2]):-
56          startsWith1(Tail1, Tail2).
57
58
59
60
61 % the external call functions which requires the users ID
62 % and returns the piece which should be moved
63 setDie(X) :-
64          retractall(die(_)),
65          assert(die(X)).
66
67 addDie(X) :-
68          die(Y),
69          Z is Y + X,
70          setDie(Z).
71
72 move(ID, Stone):- allStates(L,ID),
73          worth(L, Stone).
74
75 rollDieAgain(ID):- allStates(L,ID),
76          worth(L,[X,Y,ID]),
77          boardsize(Max),
78          X < Max - 12.
```

Example A.1: Shared AI Rule Base

EQUALIZER used this following additional rule base on page 27.

```
setworthHistory(X) :-
        retractall(history(_)),
        assert(history(X)).

getworthHistory(L,A):-
        getworthHistory1(L,[],A).

getworthHistory1([],A,A).
getworthHistory1([[_,Y,_]|L],A,B):-
        getworthHistory1(L,[Y|A],B).
```

Example A.2: Additional Rule Base EQUALIZER

The following gameplay book was used in figure 3.4 on page 28.

```
0r4
3g1
6b3
——
1r1
3g1
6b5
——
2r1
3g4
6b9
——
0r8
...
```

Example A.3: EQUALIZER gameplay book

```
% die rolled once with an total of five points
die(5).

% Layout of the Jostle board
board(-1,n).
board(0,start).
board(1,n).
board(2,n).
board(3,n).
board(4,n).

```

```
12 board(5,n).
13 board(6,'switch@1').
14 board(7,n).
15 board(8,n).
16 board(9,n).
17 board(10,'back@3').
18
19 board(11,n).
20 board(12,n).
21 board(13,'switch@2').
22 board(14,'switch@1').
23 board(15,n).
24 board(16,n).
25
26 board(17,n).
27 board(18,'switch@2').
28 board(19,n).
29 board(20,n).
30 board(21,n).
31 board(22,'back@3').
32
33 board(23,'back@3').
34 board(24,n).
35 board(25,n).
36 board(26,'goal@1').
37 board(27,'goal@2').
38 board(28,'goal@3').
39
40 board(29,'goal@4').
41 board(30,'goal@5').
42 board(31,'goal@6').
43
44 % actual state of all gaming pieces for the three players
45 state(-1,[[2,7],[1,4],[1,6],[2,8],[2,9]]).
46 state(0,[[1,5]]).
47 state(25,[[0,1]]).
48 state(26,[[0,2],[0,3]]).
```

Example A.4: Test Data

Bibliography

[1] BÄCK, Thomas: *Evolutionary algorithms in theory and practice*. Oxford University Press US, 1996

[2] BÄCK, Thomas ; FOGEL, David B. ; MICHALEWICZ, Zbigniew: *Basic Algorithms and Operators*. Bristol UK: IOP Publishing Ltd., 1999 (Evolutionary Computation 1)

[3] BARON-COHEN, Simon: Precursors to a theory of mind: Understanding attention in others. In: *In A. Whiten (ed) Natural theories of mind*, 1991, S. 233–251

[4] BORTZ, Jürgen ; DÖRING, Nicola: *Forschungsmethoden und Evaluation : für HumanundSozialwissenschaftler*. 3., überarb. Aufl., Nachdr. Heidelberg: Springer, 2005. – ISBN 3–540–41940–3

[5] BROWN, Robert G. ; MEYER, Richard F.: The fundamental theorem of exponential smoothing. In: *Operations Research* 9 (1961), Nr. 5, S. 673–687

[6] DARWIN, Charles: *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life [microform] / by Charles Darwin*. 5th thousand. J. Murray, London :, 1860. – folded leaf of plates :ix, 502 p. S

[7] DIMITZ, Erich ; FORMANN, Anton K.: *Programm zur Schätzung der Item- und Personenparameter im dichotomen logistischen Modell von Rasch sowie Modellkontrollen und ein interaktives System zur Eingabe und Änderung der Programmparameter, Steuerbefehle und Daten.*, Wien: Institut für Höhere Studien., Diss., 1979. – 115 S

[8] GESELLSCHAFT FÜR EVALUATION E.V., DeGEval: *Standards für Evaluation (DeGEval-Standards)*. Website, 2001. – Available online at `http://www.degeval.de/`

index.php?class=Calimero_Webpage&id=9023; last visited on March 25th 2009.

[9] HAMBLETON, Ronald K. ; SWAMINATHAN, Hariharan. ; ROGERS, H. J.: *Fundamentals of item response theory / Ronald K. Hambleton, H. Swaminathan, H. Jane Rogers*. Sage Publications, Newbury Park, Calif. :, 1991. – x, 174 p. : S. – ISBN 080393646 0803936478

[10] JANKTE, Klaus P.: Jostle 2007. / TU Ilmenau, IfMK. 2007 (29). – Forschungsbericht

[11] KOHNER BROTHERS: *Trouble.* 1965. – initially manufactured by Irwin Toy Ltd., later by Milton Bradley (now part of Hasbro).

[12] MASTERS, James: *Pachisi (Ludo etc.).* Website. http://www.tradgames.org.uk/games/Pachisi.htm. Version: 1997. – last visited on November 20th 2008.

[13] RESEARCH GROUP aliCE: *tuProlog*. Website. http://www.alice.unibo.it/xwiki/bin/view/Tuprolog. Version: 2008. – last visited on March 2th 2009.

[14] RUSSELL, Stuart J. ; NORVIG, Peter: *Artificial Intelligence: a modern approach*. 2nd international edition. Prentice Hall http://aima.cs.berkeley.edu

[15] SCHWEFEL, Hans-Paul: Collective phenomena in evolutionary systems. In: *Problems of constancy and Change – The Complementarity of Systems Approaches to Complexity* Bd. 2, International Society for General System Research, 1987, S. 1025–1033

[16] TURING, Alan M.: Computing machinery and intelligence. In: *Mind* 59 (1950), S. 433–460. – Reprint:Feigenbaum & Feldman, Computers and Thought, 1963, luger

[17] WEIZENBAUM, Joseph: ELIZA—a computer program for the study of natural language communication between man and machine. In: *Commun. ACM* 9 (1966), Nr. 1, S. 36–45. http://dx.doi.org/http://doi.acm.org/10.1145/365153.365168. – DOI http://doi.acm.org/10.1145/365153.365168. – ISSN 0001–0782

[18] WITTGENSTEIN, Ludwig ; 1922 ROUTLEDGE AND KEAN PAUL LTD., London (Hrsg.): *Tractatus logico-philosophicus*. Ungekuerzte Buchgemeinschafts-Lizensausgabe. Suhrkamp Verlag, 1974 (Klassiker des Modernen Denkens)