



NTNU – Trondheim
Norwegian University of
Science and Technology

Case-Based Reasoning for Adaptive Strategies in Texas Hold'em Poker

Eivind R Solbakken
Jan Berge Ommedal

Master of Science in Computer Science

Submission date: June 2012

Supervisor: Agnar Aamodt, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Most of the existing poker agents using case-based reasoning (CBR) are based on imitation of other poker agents and have weak capabilities of adapting their own strategies to different opponents or playing styles. We address these concerns in the development of UpperCase, a heads up no-limit Texas Hold'em poker agent representing a new approach to the application of CBR in poker. Using methods of perfect information hindsight analysis, the poker agent attempts to more accurately determine the quality of poker decisions. Through extensive exploration of the quality of different decisions, UpperCase is able to invent new poker strategies. The agent also tries to recognize different opponents by observing their actions and perform adaptation accordingly. Experimental results suggest that the agent is able to successfully create new profitable strategies, as well as achieve increased performance by dynamically changing its strategy during play.

Sammendrag

De fleste eksisterende poker-agenter som benytter seg av case-basert resonnering (CBR) er basert på imitasjon av andre poker-agenter og har svake evner til å kunne tilpasse sine strategier til forskjellige motstandere og spillertyper. Dette problemet har vi adressert i utviklingen av UpperCase, en heads up no-limit Texas Hold'em poker-agent som representerer en ny tilnærming til anvendelsen av CBR i poker. Ved å benytte *perfect information hindsight analysis* forsøker poker-agenten å produsere mer nøyaktige målinger av kvaliteten på avgjørelser i poker. Gjennom omfattende utforskning av kvaliteten på ulike avgjørelser er UpperCase i stand til å utvikle nye pokerstrategier. Agenten forsøker også å gjenkjenne forskjellige motstandere ved å observere deres handlinger og tilpasse seg i henhold til dette. Våre forsøk antyder at agenten er i stand til å utvikle nye lønnsomme strategier, samt oppnå økt ytelse ved å dynamisk endre sin strategi under spill.

Acknowledgments

We would like to thank our supervisor, Prof. Agnar Aamodt, for his helpfulness, guidance and good advice. His knowledge and contacts in the field, his constructive feedback and his inspirational, friendly and positive attitude have been greatly appreciated. We would also like to thank our friends and family for help and support during the process of writing this thesis.

Contents

List of Tables	xi
List of Figures	xii
I Background and Related Research	1
1 Introduction	3
1.1 Artificial Intelligence in Games	3
1.2 Background and Motivation	4
1.3 Research Approach and Goal	5
1.4 Report Structure	6
2 Texas Hold'em Poker	7
2.1 Basic Terminology	7
2.2 Possible Actions	8
2.3 Game Structure	9
2.4 Betting Rules	11
2.5 Additional Terminology	12
3 Artificial Intelligence in Poker	13
3.1 Strategies	13
3.2 Different Approaches	15
3.2.1 Knowledge-Based Approach	15
3.2.2 Game Tree Approach	16
3.2.3 Simulation-Based Approach	16
3.2.4 Other Approaches	17
3.3 Measuring Performance	17
3.3.1 Luck	18
3.3.1.1 Duplicate Poker	19
3.3.2 Evaluating Quality of Decisions	20
4 Case-Based Reasoning	21
4.1 The CBR-Cycle	21
4.2 Case-Based Reasoning in Poker	23
4.2.1 Casey	23
4.2.2 CASPER	25
4.2.3 SARTRE	26
4.2.4 BayCaRP	28
4.2.5 Discussion	30

II	Tools and Frameworks	33
5	DIVAT	35
5.1	Motivational Example	36
5.2	Definitions and Metrics	37
5.3	The DIVAT Policies	39
5.4	Evaluating Quality of Decisions Using DIVAT	40
6	UniPoker	43
6.1	Motivation	43
6.2	High-Level Structure	44
6.3	Framework-Module	44
6.3.1	Common Data Classes	44
6.3.2	Common Utility Classes	45
6.4	Agents-Module	46
6.4.1	UniPoker Agent Implementations	47
6.5	Simulation-Module	47
6.5.1	Simulation Example	48
7	UpperCase - Initial Design	51
7.1	High-Level Design	51
7.2	EQ	53
7.3	EX	53
III	Results	55
8	Improvements to UniPoker	57
8.1	Improved hand evaluation	57
8.2	Integration with the ACPC protocol	58
8.3	Improvements of the Poker Simulator	58
8.4	Web-Interface	58
9	UpperCase	61
9.1	High-Level System Architecture	61
9.2	Differences from Previous Design	62
9.3	Quality-Based Reasoning	63
9.3.1	Quality of Decisions	64
9.3.2	QBR Case Structure	64
9.3.2.1	UpperCase Action-Types	66
9.3.2.2	Situation-Key Format	67
9.3.3	Similarity	70
9.3.4	QBR Retrieval	70
9.3.5	QBR Reuse	71
9.3.6	QBR Training - Revise & Retain	72
9.3.6.1	Hand Simulation	73
9.3.6.2	Hand Evaluation	74

9.4	Decision Quality Evaluation	74
9.4.1	Profit Evaluator	75
9.4.2	AIVAT Evaluator	75
9.4.3	Equity Evaluator	78
9.4.4	Comparison By Example	79
9.5	EQ	81
9.6	EX	82
9.6.1	Known Opponent-Types	83
9.6.2	Characterization of Poker Strategies	84
9.6.3	EX-CBR	86
9.6.3.1	Retrieve - Opponent Classification	86
9.6.3.2	Reuse - Opponent Adaptation	87
9.6.4	Adaptation Rationale	89
9.7	System Summary	91
10	UpperCase Web-Interface	95
10.1	Case-Base List	95
10.2	Case-Base View	96
10.3	Player List	97
10.4	Player View	98
10.5	Hand List	99
10.6	Hand View	100
11	Experimental Results	101
11.1	EQ - Baseline Strategy	101
11.1.1	Test Structure EQ	102
11.1.2	Test Results EQ	103
11.1.2.1	Profit Evaluation Results	103
11.1.2.2	AIVAT Evaluation Results	103
11.1.2.3	Equity Evaluation Results	104
11.2	EX - Adaptive Strategy	104
11.2.1	Test Structure EX	104
11.2.2	Test Results EX	105
11.2.2.1	Test 1: Unknown opponent	106
11.2.2.2	Test 2: Known opponent	108
11.2.2.3	Test 3: Known easily exploitable opponent	109
12	Discussion	111
12.1	Discussion of EQ Test Results	111
12.2	Discussion of EX Test Results	113
12.3	Summary	114
13	Conclusions and Future Work	115
13.1	Future Work	116
13.1.1	Further Testing and Development of UpperCase	116
13.1.2	Further Development of the UniPoker Framework	117

Bibliography	119
Appendices	125
A Setting up UniPoker	125
A.1 Setup 1: Externalized development	125
A.2 Setup 2: Integrated development	126
B Setting up UpperCase	128
C Simulation Results EQ	130
C.1 Profit Evaluation Results	130
C.2 AIVAT Evaluation Results	132
C.3 Equity Evaluation Results	134
D Poker Hand Rankings	137

List of Tables

3.1	Results after simulating 1,000 hands.	18
3.2	Results after simulating 1,000,000 hands.	18
3.3	Duplicate Poker Example	19
4.1	The Casey poker agent	24
4.2	The CASPER poker agent	26
4.3	The SARTRE poker agents	28
4.4	The BayCaRP poker agent	29
5.1	Explanation of betting sequence.	37
5.2	Standard DIVAT settings from Billings and Kan [12].	39
6.1	Simulation example	48
9.1	Explanation of betting pattern.	67
9.2	Values for different 5-card hands.	68
9.3	Most significant tie-breaking value for the different 5-card hand classes.	68
9.4	Conversion of face-values.	69
9.5	Pre-flop analysis	76
9.6	Flop analysis	76
9.7	Turn analysis	77
9.8	River analysis	77
9.9	Output from PE, AIVAT and EE in the motivational example.	80
9.10	Weights for the different metrics of SVs in different stages.	87
9.11	The UpperCase poker agent.	92
11.1	Results of the EQ-tests using profit evaluation.	103
11.2	Results of the EQ-tests using AIVAT evaluation.	103
11.3	Results of the EQ-tests using equity evaluation.	104
11.4	Results from EX-test 1. Against an unknown opponent.	107
11.5	EX strategy adaptation against an unknown opponent.	107
11.6	Results from EX-test 2.	108
11.7	EX strategy adaptation against a known opponent.	109
11.8	Results from EX-test 3.	110
11.9	EX strategy adaptation against a known easily exploitable opponent.	110
12.1	Summary of EQ test results.	111
12.2	Summary of EX test results.	113

List of Figures

2.1	Preflop poker illustration	9
2.2	Flop poker illustration	10
2.3	Turn poker illustration	10
2.4	River poker illustration	11
3.1	Simulation example of 100,000 hands	19
4.1	The CBR cycle	22
5.1	DIVAT high-level	40
6.1	UniPoker high-level structure	44
6.2	UniPoker class diagram	45
6.3	Simple poker agent	46
6.4	Simulation example	48
6.5	Simulation example in UniPoker	49
7.1	UpperCase initial design	52
8.1	Screenshot of a poker-game using the UniPoker web-interface.	59
9.1	High-level view of UpperCase.	61
9.2	System Architecture	62
9.3	Relationship between similarity, utility and quality.	64
9.4	The relationship between situations and experiences.	65
9.5	A QBR-module case.	65
9.6	QBR retrieve and reuse process.	71
9.7	Example of an aggregated QBR solution.	72
9.8	Training UpperCase.	73
9.9	EQ	81
9.10	EX	82
9.11	A Strategy Vector (SV).	84
9.12	An EX case.	86
9.13	EX solution	88
9.14	Lesser Adaptation	90
9.15	Greater Adaptation	90
9.16	System overview of the UpperCase poker agent.	91
10.1	The case-base list.	95
10.2	The case-base view.	96
10.3	The player list.	97
10.4	The player view.	98

10.5	The hand list.	99
10.6	The hand view.	100
11.1	Simulation graph of EX-test 1. Against an unknown opponent.	107
11.2	Simulation graph of EX-test 2. Against a known opponent.	108
11.3	Simulation graph of EX-test 3. Against a known easily exploitable opponent.	109
A.1	A simple passive poker agent	126
A.2	A simple aggressive poker agent	127
C.1	EQ simulation using profit evaluation. Test 1.	130
C.2	EQ simulation using profit evaluation. Test 2.	131
C.3	EQ simulation using profit evaluation. Test 3.	131
C.4	EQ simulation using profit evaluation. Test 4.	132
C.5	EQ simulation using AIVAT evaluation. Test 5.	132
C.6	EQ simulation using AIVAT evaluation. Test 6.	133
C.7	EQ simulation using AIVAT evaluation. Test 7.	133
C.8	EQ simulation using AIVAT evaluation. Test 8.	134
C.9	EQ simulation using equity evaluation. Test 9.	134
C.10	EQ simulation using equity evaluation. Test 10.	135
C.11	EQ simulation using equity evaluation. Test 11.	135
C.12	EQ simulation using equity evaluation. Test 12.	136
D.1	Poker hand rankings	137

Part I

Background and Related Research

Introduction

1.1 Artificial Intelligence in Games

Artificial intelligence (AI) in games is an interesting topic for research and has been so for quite some time. Ever since the birth of computer games there has been a need for intelligent opponents that can match human players. Everything from first-person shooters, role-playing games, strategy games, sport games and more, all need some type of AI. We have now grown accustomed to intelligent behavior in computer games that can closely imitate human behavior. Research on AI applied to computer games is a great way to study the field of intelligent systems because we have extensive domain knowledge on this subject and computer games can model many types of real-world challenges or problems that we are trying to find solutions to.

There has been substantial research done on AI applied to various games. In 1997, IBM's Deep Blue beat the world's chess champion, Garry Kasparov, in a six-game match [33]. This proved that AI can be successfully applied to the game of chess and outsmart the best human players. Chess is not a simple game, but it is a game of what we call *perfect* information. This means that the entire state of the game is known to all players. In chess, both players can see the whole state of the board and the different chess pieces. No information is hidden. This makes it different from *imperfect* information games.

In games with imperfect information the players involved do not have access to the entire state of the game. Applying AI to problems with imperfect information is different and can quickly become more complex. Poker is a game of imperfect information as the opponents' cards are hidden from the player. It is also a stochastic game, meaning that there is an element of chance or non-determinism involved. This comes from the dealing of cards which is impossible to predict. Imperfect information and the stochastic nature of the game make poker a challenging and interesting topic for AI research. Poker is also a well known game and a growing business, so domain knowledge is easily accessible.

Poker has grown from a regular card game into a whole industry with huge televised tournaments from casinos all over the world. One of the important contributions towards this growth is online poker. The online poker industry has helped the game become available for everyone with a computer and there are online tournaments

with large money prizes. The interest in this game has also reached AI researchers worldwide. Many computerized poker players, also called *poker agents*, have been developed. These poker agents make decisions by applying different types of AI techniques and statistics. There are annual tournaments where poker agents created by researchers and hobbyists can compete against each other. One of the most popular is called the Annual Computer Poker Competition (ACPC)¹ with contributions from all over the world.

Poker can have different applications within AI research. It can be used as a model for other problems with imperfect information and stochastic behavior in which domain knowledge is weak or difficult to obtain. With this approach, poker is used as an understudy to explore the capabilities of AI applied to a certain field (e.g. medical research). However, in this thesis we research the possibility of creating a strong poker agent that can match human players and other existing poker agents by applying a method called case-based reasoning (CBR). In CBR, previous experiences (called cases) are stored in a case-base and reused in future problem solving. We focus on the most popular variation of poker today called Texas Hold'em [27].

1.2 Background and Motivation

We have worked with and researched the field of artificial intelligence applied to poker in previous projects. In 2010 we created a case-based poker agent in a pattern recognition class at the University of California, Santa Barbara. This agent used hand history from an experienced human player as its case-base. The agent attempted to imitate the human player by reusing solutions from the matching cases it found. It was able read the game state of an online poker client using optical character recognition (OCR) and perform actions directly to the client. The case-based system did not perform as well as we hoped, but the pattern recognition part worked well.

In a research project prior to this thesis [26] we further studied the use of artificial intelligence in poker. That project resulted in a software framework for development of poker agents in addition to a high-level architecture of a CBR poker agent with adaptive capabilities. At the beginning of that project we were recommended a commercially available poker program called Poker Academy Pro (PAP). This program can be used as a learning platform for poker players and has been widely used in research on AI in poker. The interesting part about PAP is that it has a number of already implemented poker agents that users can play against. PAP uses a java-based API called Meerkat that lets users plug in their own poker agents and compete against others.

Unfortunately, we found that PAP is no longer available. This led us to develop our own open-source poker software framework. The framework, which we named UniPoker, lets developers implement their own Texas Hold'em poker agents in a simple way. The idea behind this framework was to make it easier for developers to

¹See <http://www.computerpokercompetition.org/>

try out an idea for a poker agent. UniPoker lets developers focus on their strategy while the logic behind the game is included and fully implemented in the framework (dealing of cards, betting structure, game structure, and so on). UniPoker contains a poker simulator for testing purposes that can be used to simulate poker games between different poker agents. The UniPoker framework is presented in chapter 6.

After finishing the UniPoker framework, and as part of the same pre-thesis project [26], we started working on the design of a CBR poker agent. CBR is a growing field of AI research and there are many different useful applications for it [22]. However, what we discovered after studying existing CBR poker agents is that most of these agents are not adapting well to dynamic opponents. Many agents use case-bases that contain hand history from another poker player or are trained up by playing against a certain player. The resulting strategy of these CBR agents is therefore the same strategy that the agent they were trained on applies. With an approach like this the agent does not adapt its play if the opponent changes playing style (unless new cases are stored during play).

In this thesis we wanted to focus on integrating adaptive strategies with CBR. If an agent is able to analyze its opponent it can exploit known weaknesses with the observed playing style. This adaptability can be very important when playing against different types of players in which static strategies can become ineffective. An adaptive strategy can also be the key to success when competing against other adaptive opponents.

1.3 Research Approach and Goal

The topic of this thesis is how CBR can be utilized in order to achieve adaptive capabilities in the game of poker. Our primary goal is to acquire knowledge and get practical experience with this topic. We have formulated the following three specific objectives:

- **Study related work and existing solutions**

We will research the topic and present relevant related work and existing solutions.

- **Create a new CRB-based poker agent**

We will develop and present a new solution to CBR applied to poker. The poker agent should have stronger adaptive capabilities than what we have seen in existing CBR solutions so far.

- **Reuse and improve/extend UniPoker**

In a project prior to this thesis we created UniPoker, an open poker software framework. We will utilize this framework and extend it as necessary.

1.4 Report Structure

This thesis is divided into three parts. Part one concerns background and related research. In chapter 2, an introduction to Texas Hold'em poker is given and chapter 3 includes a presentation of artificial intelligence in poker. Chapter 4 gives a short introduction to CBR and presents existing solutions to CBR in poker.

Part two presents the tools and frameworks used in this thesis. This includes a decision evaluation tool used for poker, called DIVAT, which is described in chapter 5. The UniPoker software framework for development of poker agents is presented in chapter 6, and the initial design of our CBR-based poker agent, named UpperCase, is described in chapter 7.

Part three includes the final results of our implementation. An overview of the improvements to the UniPoker framework is given in chapter 8. Then the UpperCase poker agent is described in detail in chapter 9. In section 10 we present a web-interface that can be used to observe the system state of UpperCase during runtime. In chapter 11, the experimental results of the UpperCase poker agent is presented, followed by a discussion of these results in chapter 12. Conclusions and ideas for future work are given in chapter 13.

Texas Hold'em Poker

Texas Hold'em poker is the most popular variation of poker today [27]. It is used worldwide in casinos, professional tournaments, online and between friends. The interest in poker has grown significantly after the introduction of online poker and televised poker tournaments. Now everyone can log on to their favorite poker client and play poker online with virtual money (play-money) or real money. In this chapter, the game of Texas Hold'em is introduced, which is the variation of poker used in this research. A presentation of the basic rules of the game is given, in addition to different definitions and terms that are helpful to understand.

2.1 Basic Terminology

First it is important to explain some basic poker terminology that is used later.

- **Hole cards**
Before every round of a game the dealer deals two *hole cards* face-down to each player around the table. These two cards are each player's personal cards and are not visible to opponents.
- **Hole cards classes**
There are 169 different hole cards classes ranging from pair of aces (AA) to seven and two (72), including suited and offsuited combinations (meaning cards of the same suit or different suits respectively) [32].
- **Community cards**
The community cards refers to the cards that are placed face-up on the table. Each player can use these cards in addition to their own hole cards.
- **Stage**
Texas Hold'em consists of four different *betting rounds* called *stages*. The four stages are named pre-flop, flop, turn and river. See section 2.3 for a detailed description.
- **Hand**
The term *hand* can have two different meanings in poker. First of all, it is often used to refer to the combination of cards that a player has (hole cards plus community cards). In Texas Hold'em a hand consists of at most five

cards. A player is not required to use any of his or her hole cards (can use all five community cards to create a five-card hand).

The second meaning of the term is the reference to a single round of game play, starting from the dealing of cards until a winner of the pot is declared (this can also be called a *deal*). A game of poker consists of playing a number of hands.

- **Hand strength**

There is a list of hand rankings in poker that ranks the hands from worst to best (see appendix D). The hand strength refers to how strong a given hand is in regards to all other possible hands. There are different ways to calculate hand strength depending on which factors that are taken into account (number of opponents, table position, etc.).

- **Blinds**

Before each hand of Texas Hold'em the player sitting on the left side of the dealer will place the small blind and the next player to the left will place the big blind. The big blind is twice the size of the small blind. The blinds can be considered regular bets. The meaning of blinds is to make sure that there is money in the pot from the beginning of a hand, and it also gives a cost to play as the dealer and blinds rotate around the table between hands. The blinds are only placed at the very beginning of a hand and new blinds will not be placed until a new hand has started and the dealer has rotated to the left.

2.2 Possible Actions

Players will be required to act and their different actions can be described as either *no play*, *passive play* or *aggressive play*:

- **No play (fold)**

If a player chooses to *fold* his cards, the player can no longer take part in the current hand and the player's hole cards are not used.

- **Passive play**

A passive play refers to either a *call* or a *check*. A call means that a player will match the current bet that has been placed by an opponent to continue to play the hand.

If no bets have been placed in the current stage (betting round), a player may check. A check is essentially the same as calling a zero-value bet. It can be used to continue to play the hand without placing any additional money in the pot.

- **Aggressive play**

Aggressive play refers to placing a *bet* or a *raise*. Placing a bet means to put an additional amount of money in the pot that all opponents must at least call in order to continue to play the hand. If player A places a 5 dollar bet and

player B answers by placing a 10 dollar bet, then player B's action is referred to as a raise. A raise exceeds the current value of the highest bet.

If a player bets all his or her money, this is called going *all-in*. If there are more than two players still active in a hand and one of the players (player A) goes all-in, the other players can continue to place bets exceeding player A's all-in bet creating a side-pot. Player A will not compete for the side-pot.

2.3 Game Structure

This section presents how a game of Texas Hold'em poker is played. Texas Hold'em has four different stages for each hand. The stages are called *preflop*, *flop*, *turn* and *river*. Screenshots taken from the online poker client PokerStars¹ are presented to illustrate the different stages of a game.

Pre-flop

In the pre-flop stage the blinds are placed and each player around the table is dealt two hole cards. See figure 2.1. The player to the left of the big blind will begin betting and it continues left around the whole table. The betting does not stop until all players have either called or folded. If all players fold but one, this player wins the hand (common for all stages of a game). Otherwise the hand continues to the next stage of the game with the remaining players still in the hand.



Figure 2.1: Screenshot from PokerStars showing the **preflop** stage.

Flop

Now the dealer will place three community cards face-up on the table. See figure 2.2. These cards can be used by all players in order to make a five-card hand. As on the pre-flop, the player to the left of the big blind will begin betting. Now the players have more information regarding possible hands, both for themselves and

¹<http://www.pokerstars.com/>

the opponents. If there are more than one player remaining after the betting round, the game will continue to the next stage.



Figure 2.2: Screenshot from PokerStars showing the **flop** stage.

Turn

The dealer places a fourth community card face-up on the table. See figure 2.3. Like always, the betting begins with the player to the left of the big blind and continues until each player has either called or folded. The players can use all community cards combined with their hole cards to create the best possible five-card hand. If there are more than one player remaining after betting, the game proceeds to the final stage.



Figure 2.3: Screenshot from PokerStars showing the **turn** stage.

River

Here the dealer places the fifth and last community card face-up on the table. See figure 2.4. If there is more than one player that has not folded after the betting round, then a winner must be declared. This is called the *showdown*. If player A

placed the bet that was called by the remaining players, player A must show his/her cards. Any player that can beat player A's hand must show their cards to prove it. Players with a losing hand can choose to *muck* their hand, which means that they simply fold it without showing the cards. Mucking a hand can be done in order to prevent revealing information about a losing hand. However, showing a losing (but strong) hand can be used as a strategy for demonstrating strength.



Figure 2.4: Screenshot from PokerStars showing the **river** stage.

The player with the best hand wins the entire pot. If there is a tie-situation, the players involved will share the pot evenly between them. After the showdown is over, the dealer and blinds rotate one step to the left. The game will then start over again from the pre-flop stage where new blinds are placed and new hole cards are dealt.

2.4 Betting Rules

There are different betting rules in Texas Hold'em and the three most common ones are presented below.

- **No-limit**

In no-limit poker a player's bet can be any size between the big blind value and the size of the player's stack. There are no set limits to the size of a bet. If a player bets his or her whole stack then this is referred to as going *all-in*.

- **Fixed-limit**

In fixed-limit poker the size of all bets are predetermined. The fixed size can vary between different stages of a hand. In this type of poker, players only have to decide whether or not to bet. This type of poker is often just referred to as *limit poker*.

- **Pot-limit**

In pot-limit poker no bets can be larger than the size of the current pot on

the table.

2.5 Additional Terminology

The following list defines some additional terms used throughout this thesis.

- **Heads up**
This refers to a game of poker between two players only.
- **Pot odds**
This is the ratio between the cost of a call and the size of the pot that can be won. Pot odds is used to decide if *a call is worth the cost* by comparing it to the player's probability of winning. The calculation is: $callsize / (potsize + callsize)$.
- **Pot equity**
Pot equity is used to decide *when to bet*. The equity of your hand is basically your "ownership" of the pot; what you are entitled to in regards to your win chance [44]. A 65% win chance means a 65% equity. The calculation of pot equity is given by $(equity * potsize)$. You should bet when your pot equity is larger than the cost of betting.
- **Bluff**
A bluff is a bet or raise that is placed in an attempt to intimidate opponents. A bluffing strategy is to pretend that your hand is stronger than it actually is by playing aggressively. Large bets can push your opponents into folding and let you get away with the pot.
- **Small blinds per hand (sb/h)**
This is a way to measure the performance of a player. Small blinds per hand is the amount of money, represented by the number of small blinds, that a player has won in each hand on average. This means that if a player has won \$1000 in total after playing 100 hands with small blinds of \$1, the sb/h equals 10. If instead the small blinds were \$10, the sb/h would equal 1. This way of measuring performance scales according to the small blind value.

Artificial Intelligence in Poker

Games is an interesting and good topic for AI research. Games have well defined rules and goals and can model real-life problems concerning stochastic behavior and hidden information. In this thesis we focus on the game of Texas Hold'em poker. As stated previously, the interest in poker has grown very fast in the last years. There has been extensive research done on the field of AI applied to poker, and Rubin and Watson present a comprehensive review of computer poker in [38]. This chapter presents some of the different approaches towards poker AI identified in their review.

3.1 Strategies

Computer poker strategies can be divided into two categories; *Nash equilibrium* strategies and *exploitive* strategies [38]. The term *Nash equilibrium* is named after its inventor John Forbes Nash, an American mathematician. A Nash equilibrium strategy means that no player has an incentive to deviate from his or her chosen strategy because any slight changes will theoretically lead to a more negative outcome [18]. This implies that all players involved in the game are following a Nash equilibrium strategy and no players diverge from it. If all players follow their strategies, the set of strategies will be in a Nash equilibrium which is collectively the optimal solution. This is essentially a strategy where the goal is to minimize loss rather than maximize profit.

An exact Nash equilibrium strategy is currently considered impossible to apply to poker because of the high complexity of the game [30, 38]. Instead, an approximation called ϵ -Nash equilibrium (or near-equilibrium) can be used [19, 38, 41]. The ϵ refers to the maximum possible gain that a player can achieve by changing his or her strategy. By following an equilibrium strategy, a player will not focus on exploiting potential weaknesses in opponents. This means that the player will keep losses down, but also miss opportunities for higher profits using exploitation.

Algorithms exist that are used for creating near-equilibrium based strategies in poker. Counterfactual Regret Minimization (CFRM) [46, 19] is an iterative algorithm created by the University of Alberta Computer Poker Research Group (CPRG)¹. *Regret* is considered the loss due to not choosing the optimal solution

¹<http://poker.cs.ualberta.ca/>

at all times. The CFRM algorithm attempts to minimize this regret by separating the overall regret into different information sets which are then minimized independently [46]. Gilpin, Hoda, Peña and Sandholm [15] developed an iterative algorithm based on Nesterov’s *excessive gap technique* [23]. The algorithm requires $O(1/\epsilon)$ iterations in order to calculate an ϵ -Nash equilibrium.

An exploitive strategy takes advantage of weaknesses found in opponents. This means that it requires a method for observing and modeling the opponent, often referred to as *opponent modeling*. In poker there are certain traits that can suggest what type of player an opponent is. An example can be a playing style called *loose aggressive* which is a player that plays many hands (large number of bets and calls). This leads to frequently playing weak hands, so a good strategy against this type of player is to bet large when you have good hands. An exploitive strategy deviates from an equilibrium strategy which makes itself more vulnerable towards being exploited. However, exploitive strategies can potentially be more profitable than equilibrium strategies.

The key to creating a good exploitive strategy lies in the analysis of opponents. A player must be able to correctly identify an opponent’s playing style and take advantage of the opponent’s weaknesses. It is also important for the player to adjust the strategy in regards to his or her own weaknesses. If a player has correctly identified the opponent as *loose passive*, it means that the opponent calls frequently. Bluffing is therefore not an option against this type of opponent. An opponent model can be created by training the agent against the opponent or by analyzing the opponent online (during live play). By training against a certain opponent, the agent should be able to create a good and comprehensive model if a sufficient number of hands are played between them during training. This leads to a strategy that is very efficient against this particular opponent’s playing style. However, the strategy might not perform well against other types of players.

An agent that analyzes its opponent online will most likely not achieve the same degree of exploitation as an agent that is trained against this certain opponent, but it will be better prepared to handle different opponents and playing styles. With online opponent analysis there is a time span between the first encounter with a new opponent and the time the agents actually starts adapting its playing style. This means that it is important to have a good baseline strategy that is used until the exploitive strategy takes over.

There are different types of exploitive strategies, some are *static* which means they are pre-programmed and will not change during a game. Others are *adaptive* and have the ability to dynamically change during a game. Adaptive strategies are more robust, as they can better handle opponents that change their playing style over the course of a game (adaptive opponents). Adaptive strategies must be able to apply more weight to the most recent actions taken by the opponent in order to better adapt to the opponent’s changing playing style.

3.2 Different Approaches

The field of AI is large and there are many different approaches and techniques that can be used to solve a problem. The study of poker agents show a large variety of AI techniques that have been applied. Some poker agents depend on expert domain knowledge and are called knowledge-based systems [38]. There are approaches that employ simulation and game tree search, and others that apply techniques like neural networks, Bayesian networks and case-based reasoning. Combinations of different techniques have also been researched. This section presents some different approaches and examples of existing poker agents.

3.2.1 Knowledge-Based Approach

Knowledge-based agents require good domain knowledge in order to work properly. In a rule-based system the poker agent will make decisions by following certain rules that apply to the different game situations. The rules will most likely be static rules that have been developed using domain knowledge. A certain rule can be to fold your hand pre-flop if your hole cards do not make a pair and no card is higher than an eight. There is an extremely large number of different situations that can arise in poker, so it is not possible to create specific rules for each separate situation. Instead, a group of situations must be handled by the same rule. Because rule-based agents play according to a static set of rules they are highly vulnerable to being exploited.

A formula-based agent is also a knowledge-based system, but instead of playing according to strict rules it collects a number of inputs that provide better understanding of the game state and then uses this information for decision making. Useful inputs can be hand strength, pot-odds, pot equity, etc. The system can output a probability triple which consists of the probability for choosing each decision (fold, check/call, bet/raise) [38].

CBR is also considered a knowledge-based method in general. The different processes of CBR (see section 4.1) can be guided and supported by models of domain knowledge [1]. However, there are different types of CBR methods and some types do not rely on domain knowledge and are therefore not knowledge-based (e.g. instance-based methods). Section 4.2 presents some of the approaches to CBR in poker.

Some of the most well known poker agents, Loki [28, 10] and Poki [13, 9], are both knowledge-based agents. They were developed by the University of Alberta CPRG. Loki uses expert domain knowledge to make rules to guide its decision making. It also successfully applies opponent modeling to increase its performance. Poki is a rewritten version of Loki that uses a formula-based approach. Poki performs well against other poker agents as well as human players. A version of Poki won the Annual Computer Poker Competition (ACPC) in the 6-player Texas Hold'em limit tournament in 2008 [2].

3.2.2 Game Tree Approach

Game trees can be used to represent the state of a game in a tree structure. In a game tree a node corresponds to a certain game state and a branch between two nodes represents the action which leads to the new state. The leaf nodes in the tree correspond to the final outcomes. By carrying out a game tree search, an agent can foresee the outcome of each possible action. The problems with using a game tree to represent poker are the hidden information and the complexity of the game.

We cannot represent all possible states in poker using a game tree as the size of the tree would be too large. Simplifications or abstractions are needed in order to apply game theoretic principles to the game [38]. Elimination of betting rounds is one way to reduce the number of game states. Another abstraction is to group different hands together using their hand strength. Many different hands can have approximately the same winning chances and can be treated the same way. It is also possible to handle the different stages of Texas Hold'em poker (pre-flop, flop, turn and river) in separate models to reduce the size of the game tree.

A number of poker agents created by the University of Alberta CPRG apply near-equilibrium strategies that have been developed using game abstractions [8]. PsOpti is the name of a collection of poker agents applying this strategy and it has proven to be very successful. The Hyperborean poker agent [46], which is a combination of PsOpti agents, uses the CFRM algorithm mentioned above and performs very well against strong opponents. Hyperborean has won many of the ACPC tournaments since the beginning in 2006.

3.2.3 Simulation-Based Approach

Considering that poker is a game of imperfect information, it is impossible to know the exact game state. Some type of simulation must occur that randomly chooses or predicts the hidden information.

Simulation-based agents simulate the outcome of certain decisions in order to predict the most profitable actions. Before a decision is made, an agent can simulate what the outcome of the decision will be by predicting the opponent's hole cards and adding random future community cards. When the simulation reaches the end of the hand, the outcome is known. In order for such a simulation-based approach to work properly there must be a large number of simulations done for each situation. As poker is a complex game with a huge number of possible game states, simulating every possible situation and outcome is infeasible. Hence, a simplification is required here as well.

Monte-Carlo simulation is a method used in a wide variety of fields to approximate the outcome of a decision by using random variables in multiple simulations [17]. This method can also be applied to poker. If an action is to be made in poker, the Monte-Carlo simulation will make a decision and simulate the rest of the hand by adding random community cards and opponent cards. A sufficiently large number

of simulations will be executed with different community cards and opponent cards for each decision. The average outcome from the simulations represent the expected value for the given decision.

An improved version of the poker agent Loki, called Loki-2 [11], uses a modified approach to the Monte-Carlo simulation. The opponent’s most probable hole cards are calculated based on the opponent’s actions. The combinations of hole cards that have higher probabilities are given more weight in the sampling used in the simulation. This approach is called *selective sampling*. The results from using this approach were promising and the creators proposed this technique to become standard for games with imperfect information and nondeterminism. However, after comparing the performance between a formula-based Poki agent and a simulation-based Poki agent, the results were not as convincing [9]. The simulation-based strategy worked well against weak opponents, but performed worse than a formula-based strategy against stronger opponents. The simulation-based agent lost to the formula-based agent in self-play experiments and also performed worse in heads up play against human players.

A poker agent called AKI-RealBot [43] uses the Monte-Carlo simulation approach combined with an exploitive strategy. The Monte-Carlo simulation provides the expected values of different decisions and this information is post-processed and adapted towards exploiting weak opponents. The AKI-RealBot placed second, behind Poki, in the ACPC 6-player Texas Hold’em limit tournament in 2008 [2]. One of the main reasons for its success in this tournament was the ability to exploit weak opponents, in particular the weakest one.

3.2.4 Other Approaches

A poker agent called MANZANA, created by Marv Andersen, uses artificial neural networks (ANNs) to play poker. MANZANA was trained on the hand history of a strong poker agent and it placed first in the 2009 ACPC Texas Hold’em limit bankroll tournament [3]. BPP is a poker agent based on Bayesian decision networks [24]. It uses decision networks to model its own hand, the opponent’s hand and the opponent’s playing style.

3.3 Measuring Performance

The stochastic nature of poker makes it difficult to accurately measure the performance of poker agents. There could be situations where an agent makes good decisions, but still loses. This means that it is not always sufficient to look at an agent’s total profit to analyze its performance. In this section we present some of the challenges that arise when measuring a poker agent’s performance.

3.3.1 Luck

The element of chance coming from the shuffling and dealing of cards makes luck an important factor in poker. Like mentioned above, a player can make good decisions and be unlucky and lose. A lucky card can often be the difference between a huge loss and a huge profit. This is why there are discussions about whether poker is a game of skill or luck. Levitt and Miles [21] suggest that poker indeed is a game of skill, examining results from the 2010 World Series of Poker. They compared the total return on investment (ROI) between players who are identified *a priori* as high-skilled and the other players. The results showed that the high-skilled players had an average ROI of over 30 percent while the others averaged -15 percent.

However, the stochastic nature of poker will influence the measurement of performance. What we need to do in order to more accurately measure the performance is to play a large number of hands to reduce the variance. Fortunately, this is significantly easier when it comes to computerized poker agents compared to human players. Most poker agents have the ability to play millions of hands in a short amount of time. With a large sample size we can reach a satisfactory level of statistical confidence.

To test this assumption we did an experiment using our poker simulation tool integrated in the UniPoker framework (presented in chapter 6). The two tables below show results from poker game simulations between two simple poker agents with identical strategies. Since the agents are identical they should play evenly against each other. This means that the total profit should be close to zero for both agents. Table 3.1 shows the results after 1,000 simulated hands and table 3.2 shows the results after 1,000,000 hands. The bankroll represents how much a player has won or lost in total and the sb/h is the win-rate represented by the number of small blinds won per hand on average (see section 2.5).

Agent	Bankroll	Win-Rate (sb/h)
SimpleRuleAgent	3 268.0	3.26800
SimpleRuleAgent	-3 268.0	-3.26800

Table 3.1: Results after simulating 1,000 hands.

After simulating 1,000 hands the win-rate of one agent is more than 3 small blinds per hand on average. This is a much larger number than expected.

Agent	Bankroll	Win-Rate (sb/h)
SimpleRuleAgent	-13 840.0	-0.01384
SimpleRuleAgent	13 840.0	0.01384

Table 3.2: Results after simulating 1,000,000 hands.

After simulating 1,000,000 hands the win-rate has been reduced to a number close to zero. This illustrates that a larger sample size will lead to a more accurate measurement of performance. This experiment suggests that luck plays an important role

in a short-term perspective, but evens out between players in a long-term perspective. The role of skill is therefore especially important over a large number of hands. Hence, it is essential to simulate a considerable amount of hands when measuring the performance of poker agents.

Figure 3.1 shows the resulting simulation graph of the experiment with 1,000,000 played hands. This graph illustrates the total bankroll (the amount of money won or lost) of the two agents on the y-axis and the number of played hands on the x-axis. We can see that neither of the agents seem to play with a consistent profit. Instead the graphs are constantly fluctuating, which means that the winner of this game is fairly random. This is a result of both agents applying identical strategies.

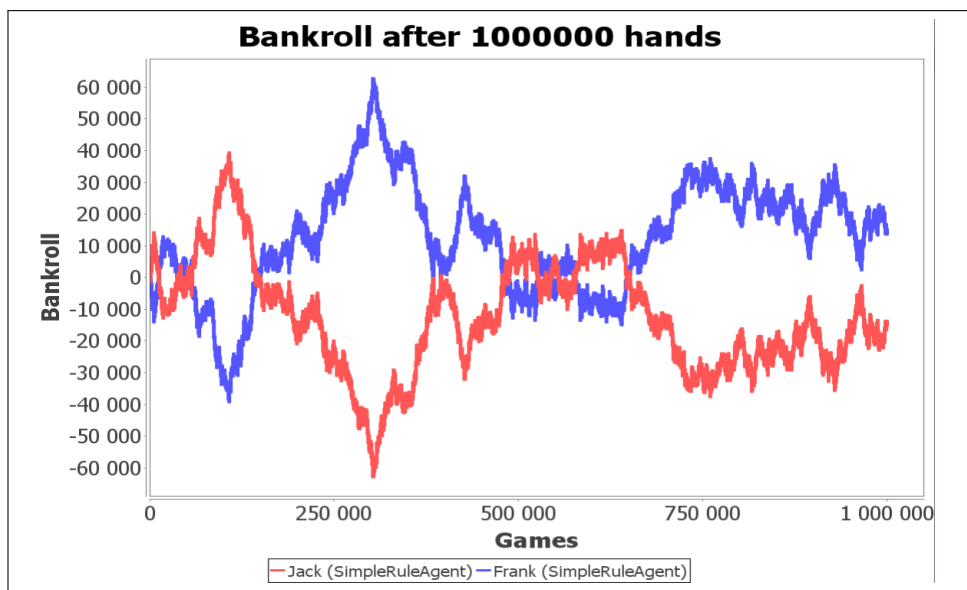


Figure 3.1: Simulation graph after 100,000 played hands.

3.3.1.1 Duplicate Poker

Another effort used to reduce the variance in poker is called *Duplicate Poker* [25]. The idea behind this technique is to let different players play the same hand and then compare their performance. We present a simple example in table 3.3 of two agents (A and B) and two different hands (X and Y). This example illustrates how duplicate poker can be used in heads up poker.

	Agent A	Agent B
Run 1	X	Y
Run 2	Y	X

Table 3.3: Duplicate Poker Example

In run number one, agent A is dealt hand X and agent B is dealt hand Y. After the first run is finished, any learning performed during this run is removed from the agents' memories. Then in round number two, agent A is dealt hand Y and agent B

is dealt hand X. The outcome of the two runs can then be compared and the agent with the highest total profit is the winner.

Duplicate Poker reduces variance by letting the poker agents play the same exact hands against each other. This means that if agent A is lucky and is dealt good cards throughout a game, this will not necessarily affect the final outcome because agent B will be dealt these same cards in the next run. The final outcome reflects the agents' total performance after playing both hands.

Duplicate Poker is used in the ACPC tournaments to provide more accurate results [4]. A series of Duplicate Poker matches are played between poker agents in order to declare the winning agent. All memory is removed from the agents between each match. In the 2010 ACPC heads up no-limit tournament, each agent played 200 matches of Duplicate Poker against every other opponent in which each match consisted of 3000 hands [40]. This brings the total number of played hands between two agents to: $200 \cdot (3000 \cdot 2) = 1,200,000$ (with Duplicate Poker each hand is played twice). This corresponds to the concept of playing a large number of hands to achieve a more accurate measurement of performance, as mentioned above.

3.3.2 Evaluating Quality of Decisions

The quality of decisions in poker are difficult to evaluate because of stochastic behavior and hidden information. Few hands are actually played until a showdown, so most of the decisions we evaluate do not include information about the opponent's cards. A common way to evaluate a played hand is to use the final outcome. If a player won, the outcome is good. In a case-based approach the evaluation of decisions becomes very important because we only want to reuse the best decisions. The idea of reusing the decisions that have led to the best outcomes (highest profits) is called a *best-outcome reuse policy*. This policy sounds reasonable, but the variance in poker makes this policy noisy. Results from [37, 41, 45] show that the performance of a best-outcome reuse policy is inferior to other policies (further explained in section 4.2).

When evaluating the quality of decisions made by a poker agent we must take luck into account. Some decisions are good, but still lead to a negative outcome. If we could reduce the variance in poker we would be able to give a more accurate evaluation. In chapter 5 we present DIVAT [12], an evaluation tool that focuses on this issue, and in section 9.4.3 we present another approach to decision evaluation.

Case-Based Reasoning

Case-based reasoning (CBR) is a growing research field in AI that originated in the US. It separates itself from many other major AI approaches by utilizing specific knowledge of previously experienced situations, rather than solely relying on general knowledge of a problem domain or making associations between problem descriptors and its conclusions [1]. CBR can still work well without extensive domain knowledge and can therefore be very useful in problems with hidden information or weak domain knowledge.

In this chapter, a brief introduction to CBR is given in addition to a presentation of some existing approaches to CBR in poker. Section 4.1 presents the CBR-cycle and section 4.2 gives an overview of a few different CBR poker agents.

4.1 The CBR-Cycle

CBR stores experienced situations, called cases, in a case-base for later use. This can resemble how humans do problem solving. When humans encounter a situation that requires a decision or action to be made, we often look back at previous similar situations to help the decision making. An example of this can be a physician's treatment of a patient, or a court's decision in a law suit. CBR takes advantage of this idea of looking back at previous experiences to make informative decisions. The CBR approach can be presented as a cycle consisting of four separate processes as identified by Aamodt and Plaza in [1] and illustrated in figure 4.1. The four processes are *retrieve*, *reuse*, *revise* and *retain*.

Retrieve

In the first step of the cycle we have the retrieve process. This process involves recognizing the current situation and retrieving the closest matching cases from the case base. The input to a CBR system is a new situation (new case) described by different *features*. These features help the system match the new case with already stored cases by using some similarity function. The type of features used and the matching between them is very important for the performance of the CBR system. Different weights can be given to each feature based on importance in order to better calculate the similarity. The one closest matching case can then be retrieved or a group of the k closest cases.

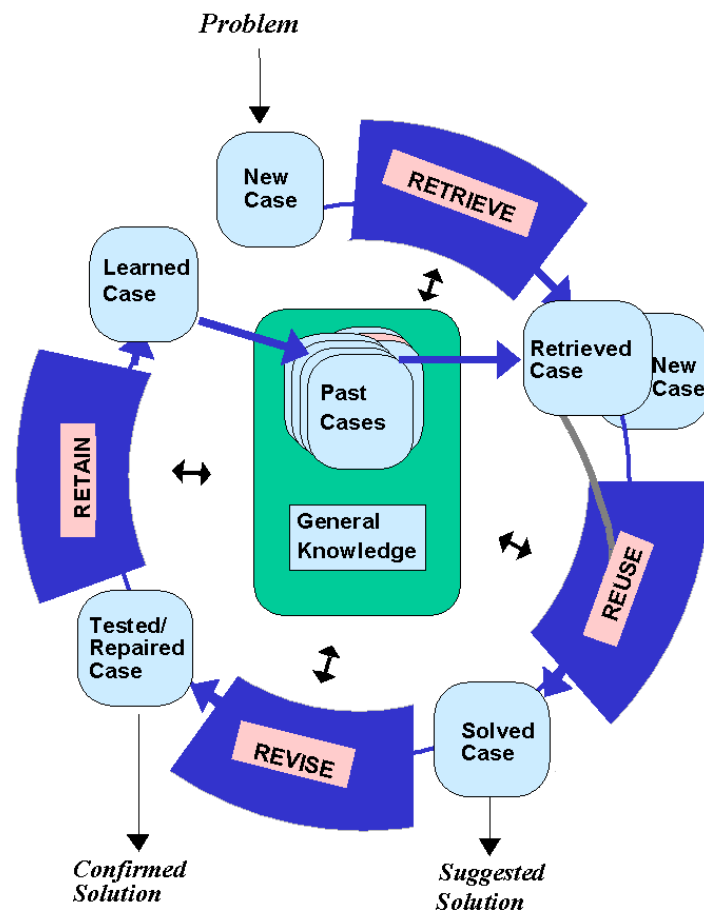


Figure 4.1: The CBR cycle illustrating the four processes; retrieve, reuse, revise and retain from [1].

Reuse

In this process the closest matching case has been retrieved and is combined with the new case to create a suggested solution to the problem. The system must consider the differences among the retrieved case and the new case and choose whether to copy the retrieved case completely or adapt it. An adaption should be made if the retrieved case is not considered a good solution to the new problem.

Revise

After receiving a suggested solution from the reuse process this solution should be tested. Some type of evaluation must occur. The suggested solution can be tested by applying it to the real environment. If the evaluation shows that the suggested solution failed, the solution must be repaired to better solve the problem at hand. To repair a solution the system must be able to explain the errors that are found.

Retain

In the retain process the final tested and repaired solution is stored in the case base. This solution is now part of the case base and used in future problem solving. A challenge in CBR is to decide how large a case base should be in order to achieve desired performance. With very specific cases one will most likely be able to achieve good performance, but a large case base is required to find matches in all types of situations. A larger case base results in a much more time consuming training phase and retrieval step.

4.2 Case-Based Reasoning in Poker

Experienced poker players are able to recognize situations and use their experience to guide them in their decision making. This experience-based approach is what CBR focuses on. As we play poker we discover new situations and learn the outcome of our decisions. This can be applied to poker agents as well, and this section presents some of the work that has been done on CBR in poker.

In order to compare and analyze the different poker agents, we investigate a set of important factors and how these are implemented and used in each poker agent. The performance of the agents has not been emphasized, but rather the possibility of adapting strategies and not relying completely on imitation of other expert agents. The factors are presented in the following list:

- **Case-base structure**
How are cases described? Is there more than one case-base? How is the solution presented?
- **Training**
How is the agent trained?
- **Reuse policy**
How does the agent choose which solution to reuse?
- **Adaptive capabilities**
Is the agent able to adapt its strategy during live play to exploit its opponents?
Does the agent store new cases during play?

There are other factors that are important as well, like an agent's run-time, memory footprint and obviously its performance, but considering that this thesis deals with adaptive strategies in CBR we have not prioritized these factors.

4.2.1 Casey

Sandven and Tessem [42] researched the potential of CBR in poker with their poker agent Casey that plays fixed-limit Texas Hold'em. Casey uses separate case-bases for pre-flop and post-flop play. There are two types of features used to describe a

Casey	
Case-base structure	Separate case-bases for pre-flop and post-flop play. Matching cases constitute a generalized case. Solution is represented by a strategy.
Training	Starts with empty case-base. Makes random decisions to grow case-base.
Reuse policy	Best-outcome
Adaptive capabilities	Cases are stored during play which makes its playing style adaptive. However, it does not separate between different types of opponents.

Table 4.1: The Casey poker agent

case; indexed features and un-indexed features. The indexed features are used for retrieving cases from the case-base and contain hand strength, position, number of opponents and bets to call. The un-indexed features are contextual information that further help describe a case, but are not used for retrieval.

Casey uses different simple strategies in order to change its play according to the game context. A strategy consists of an initial action and a follow-up response if possible. This could for example be a bet/raise followed by a re-raise. Deceptive strategies are also available. A bluff consists of a bet/raise followed by a fold if re-raised. Casey makes decisions by consulting a generalized case which consists of information from all matching cases. If the number of matching cases that make up the generalized case is sufficiently large, the most profitable strategy is used (best-outcome). If not enough matches are found, a random play is made to build up the case-base.

Casey starts off with an empty case-base and builds it up during play. The results from a learning experiment consisting of 50,000 played hands showed that after about 20,000 hands Casey was able to play consistently against its opponents. The experiment was done against 3, 5 and 7 simple rule-based agents. Casey was able to play profitably on the 4-player table, but was losing on the 6- and 8-player tables. However, the results from all tables showed that Casey was able to learn over time and improve its play, suggesting that CBR can successfully be applied to poker.

Table 4.1 shows the description of Casey according to the factors presented earlier. Casey has no cases stored in the case-base before playing against a new opponent (in the experiment presented in [42]). This means that Casey goes through a learning phase in the beginning of a game. Until the case-base has grown sufficiently large, Casey will mostly make random decisions resulting in bad performance initially. As the case-base grows, Casey begins to learn what decisions to make. This makes Casey able to adapt its play towards different opponents. However, this requires a way of overwriting old cases with new ones or resetting the case-base between games if the case-base has a given maximum size, if not the strategy becomes static. Sandven and Tessem used a maximum of 100 cases to create a generalized case [42].

If Casey plays against a set of different opponents, it will not change its playing style according to each opponent. Instead the decisions that have yielded the best results during play (independent of opponent) will be reused. This might lead to a playing style that is effective against one type of opponent, but ineffective against another. Also, the best-outcome reuse policy is influenced by luck and is not considered a very good policy.

4.2.2 CASPER

CASPER is a CBR poker agent that plays fixed-limit Texas Hold'em and was created by Rubin and Watson [34]. CASPER's training data is constructed by observing 7,000 played hands between the poker agents Simbot and Pokibot, both created by the University of Alberta CPRG. Each decision made in the training data was stored as one single case. CASPER uses separate case-bases for each stage of the game (pre-flop, flop, turn and river). When it is time to act, a target case is constructed by looking at different features of the game state. Then the appropriate case-base is searched for cases that match this target case. All cases that exceed a 97% similarity threshold are retrieved. If no such matches are found, the 20 most matching cases are retrieved instead.

The decisions found in the retrieved cases are all summarized and used to create a probability triple. This triple represents the share of decisions that are fold, check/call, or bet/raise. CASPER's decision is made probabilistically using the probability triple. This means that if the probability triple is (0.1, 0.2, 0.7) CASPER will fold 10% percent of the time, check/call 20% of the time, and bet/raise 70% of the time.

CASPER was tested against strong adaptive poker agents, non-adaptive poker agents and human players. Two versions of CASPER was tested against other agents where CASPER02 is the same agent as CASPER01, but with a larger case-base constructed by observing 13,000 played hands. Both agents were able to play evenly against strong adaptive opponents. CASPER02 played profitably against non-adaptive opponents, while CASPER01 did not. Rubin and Watson suggest the reason behind this could be that the training data consists of play between adaptive agents, so a larger case-base is needed in order to show good results against non-adaptive opponents.

Against human players, CASPER was able to play profitably on play-money tables with an average profit of \$2.90 for every hand. However, on real-money tables CASPER was not able to play profitably resulting in an average loss of \$0.02 per hand. This suggests that human players play differently with real money. There were fewer matching cases found during real-money play compared to play-money play which supports this claim.

As we can see in table 4.2, CASPER does not support adaptability. It is trained by observing two adaptive poker agents and simply reuses their decisions probabilistically. This means that the strategy is not adapting to different opponents. However,

CASPER	
Case-base structure	Separate case-bases for each stage. Target case is created to find matching cases. Solution is a probability triple.
Training	Uses hand history from games between Pokibot and Simbot.
Reuse policy	Probabilistic
Adaptive capabilities	Trained on adaptive agents, but has no adaptive capabilities.

Table 4.2: The CASPER poker agent

the decision made in two identical situations will not always be the same considering the probability triple used as a solution. This can help decrease the chance of being exploited by an adaptive opponent.

4.2.3 SARTRE

SARTRE is another CBR poker agent created by Rubin and Watson that specializes in fixed-limit heads up Texas Hold'em [35, 36, 37]. Its case-base is constructed by observing the strongest opponents of the Annual Computer Poker Competition (ACPC). A case consists of three features; hand type, betting sequence and board texture. The hand type feature describes the strength of the hand and its ability to improve. Betting sequence describes the betting that has occurred up to this point in the hand. The board texture summarizes the state of the community cards, including possibilities for flush and straight. The solution for each case is represented by a probability triple for the chosen decisions and an outcome triple that shows the average profit for each decision.

To retrieve matching cases a k -nearest neighbor algorithm (k-NN) was used. Rubin and Watson created two types of SARTRE agents, called SARTRE-1 and SARTRE-2. SARTRE-1 looks for exact matches using the k -NN algorithm where the k is only bounded by the number of matching cases. If no exact match is found a default policy of always calling is used. In SARTRE-2 the k is set to 1 and an exact match is not required. This means that the one best match is always used as the solution. The similarity metric used for the betting sequence feature also varies between the two SARTRE agents. SARTRE-2 includes four levels of similarity between betting sequences whereas SARTRE-1 only accepts exact matches.

Rubin and Watson experimented with three different policies for reusing cases from the case-base [37]. The *probabilistic policy* selects the decision by using the probability triple in the case-solution. The *majority-rules policy* selects the decision that has been made the majority of the time in the given situation. The *best-outcome policy* selects the decision that has given the largest average profit.

SARTRE-1 was trained on hand history from the poker agent Hyperborean-Eqm that won one of the 2008 ACPC heads up limit tournaments [2]. SARTRE-2 was

trained on hand history from MANZANA that won the 2009 ACPC limit bankroll tournament [3]. As SARTRE-1 requires exact matches between the target case and the solution, its case-base of approximately 1,000,000 cases is more than ten times the size of SARTRE-2's case-base. The results from self-play experiments between the two agents show that SARTRE-2 (with a majority-rules policy) was able to win 6 out of 10 games, where one game resulted in a draw. However, Rubin and Watson report that the evaluation did not achieve statistical significance [37].

In a self-play experiment between the three different reuse policies, the most profitable policy proved to be majority-rules. Rubin and Watson believe that this could be because the type of opponent being challenged is Nash-equilibrium based [37]. They explain this by pointing out that a Nash-equilibrium strategy only profits when opponents make mistakes. This means that a majority-rules policy applied to a Nash-equilibrium case-base is a safe strategy against another Nash-equilibrium opponent. The worst performance was given when using the best-outcome policy. This suggests that a good outcome does not necessarily mean a good betting sequence, as pointed out earlier. SARTRE-1 participated in the 2009 ACPC limit heads up tournament with a majority-rules policy and placed 6th out of 12 competitors in the bankroll event [3].

SartreNL is an extended version of the SARTRE agent that specializes in no-limit heads up Texas Hold'em [40]. There is an important difference between creating limit agents and no-limit agents as the size of bets must be taken into account in no-limit poker. Playing limit poker can therefore be considered a less complex task than playing no-limit poker. A good no-limit poker agent must be able to decide on a reasonable bet size before making a bet. It must also consider the size of an opponent bet before calling.

SartreNL is able to convert hand history into generalized cases that can be used when playing no-limit poker. A quantitative bet is translated into a certain bet category like quarter pot, half pot, three quarter pot and so on. This abstraction is done to reduce the number of available actions that the agent can perform. Hand strength buckets are also used in order to reduce the state space. An additional case feature was also introduced called stack commitment. Stack commitment represents how much of a player's total stack size that has been invested in the current pot. This is important to consider when making decisions later in a hand.

SartreNL was trained on hand history from the best no-limit poker agent from the 2009 ACPC (Hyperborean). SartreNL performed well at the 2010 ACPC and achieved a 2nd place in the bankroll instant run-off tournament [5].

As we can see from table 4.3 the SARTRE agents have no adaptive capabilities. SARTRE reuses cases from other expert poker agents and do not adapt its strategy towards different opponents. By using the majority-rules policy the agent keeps its losses down against equilibrium opponents, but it will also make it easily exploitable against adaptive opponents. This is because the majority-rules policy makes the strategy completely static so an adaptive opponent can learn how to play profitably against it.

	SARTRE-1	SARTRE-2	SartreNL
Case-base structure	Separate case-bases for pre-flop and post-flop play. Solution consists of outcome triple and action triple.		
Training	Hyberborean	MANZANA	Hyperborean
Reuse policy	Majority-rules	Majority-rules	Probabilistic
Adaptive capabilities	No adaptive capabilities.		

Table 4.3: The SARTRE poker agents

The probabilistic policy used in SartreNL is not static in the same sense as a majority-rules policy. This is because of the element of chance involved in choosing a decision. However, a strong adaptive opponent should still be able to exploit SartreNL’s strategy as this strategy will not adapt to a changing opponent and will remain fairly consistent.

4.2.4 BayCaRP

BayCaRP is a poker agent created by Sebastian Helstad Unger that combines a Bayesian network (BN) with CBR to play fixed-limit Texas Hold’em [45]. The BN is used to predict the opponent’s hand types, which could be low pair, two pair, three of a kind and so on. There is one BN for the pre-flop stage and one for the post-flop stages. The output of the BN is the opponent’s three most probable hand types. This prediction is then used as case-features in the CBR system.

The CBR system consists of separate case-bases for each stage of a hand. The pre-flop case-base is the only one that is not influenced by the BN prediction. The initial case-base was constructed by observing two poker agents, Pokibot and Simbot, playing approximately 10,000 hands of poker against each other. Each case included the solution (the decision made) and the outcome (the final win or loss) in addition to common case features. The total size of the initial case-base(s) was around 100,000 cases.

When BayCaRP is requested to act it creates a new target case using features describing the current situation (including predicted opponent hand types on post-flop stages). This target case is then matched with cases in the case-base. Matching cases are found by calculating the local similarities using the k-NN algorithm and thereby finding the global similarity over this set of weighted features. All cases with a similarity above a given threshold are retrieved from the case-base. The decisions made (solutions) in these retrieved cases are then used to create the solution for the target case. A best-outcome reuse policy was initially chosen for BayCaRP1. The target case is retained in the case-base as a new case after the outcome of the decision is known.

In an attempt to reduce the impact of luck in decision evaluation, BayCaRP planned to use a concept called G-bucks [29]. G-bucks calculates the expected value of a decision by multiplying the equity of a hand with the total pot size. Equity is found by simulating the outcome of playing a certain hand a number of times against other

BayCaRP	
Case-base structure	Separate case-bases for each stage. Target case is created to find matching cases. Decision made and final outcome are stored.
Training	Uses hand history from games between Pokibot and Simbot.
Reuse policy	Majority-rules (best-outcome initially)
Adaptive capabilities	Use of hand prediction influences strategy. Stores new cases during play.

Table 4.4: The BayCaRP poker agent

hands and record the results from each simulation. In BayCaRP, the most likely range of opponent hands is given by the BN. This was then to be used as input to the G-bucks calculation to assess the hand equity. Multiplying the equity with the total pot size gives the expected income from the hand and this can be compared to the cost of making a certain decision. A positive G-bucks score represents a decision that gives a positive income.

As the G-bucks approach was not implemented due to time constraints, Unger instead experimented with a different reuse policy in an attempt to reduce the impact of luck. BayCaRP2 uses a majority-rules policy instead of best-outcome. With this approach the agent will not base its decision making on the outcome of previous decisions, but will instead more closely imitate the Pokibot and Simbot agents by using their most common decisions.

BayCaRP was tested by playing 4,500 hands of poker on an 8-player table against five instances of Pokibot and two instances of Simbot. Unger reports that BayCaRP1 with a best-outcome reuse policy lost an average of 0.026 bb/h (big blinds per hand), placing 5th overall. BayCaRP2, using a majority-rules reuse policy, performed better than BayCaRP1 with an average profit of 0.056 bb/h, placing 3rd overall.

According to Unger, BayCaRP2 is able to outperform the CBR agent CASPER which is also trained on hand history from Pokibot and Simbot. This suggests that BN combined with CBR can be a good approach for developing strong poker agents. Unger reports that the three most likely hand types predicted by the BN includes the opponent's actual hand type 77% of the time. Knowing the opponent's hand type is very useful and this information can greatly increase the performance of the CBR system.

Table 4.4 shows a description of BayCaRP according to the factors presented earlier. The BN provides BayCaRP with the ability of modeling the opponent to a certain degree. It collects a number of inputs describing the opponent's playing style in order to predict the opponent's cards. As this hand type prediction is used as a case feature in the target case, this prediction will have an impact on which cases that are retrieved and reused. BayCaRP's strategy is therefore adaptive in the sense that it observes its opponents and plays according to their predicted hands.

BayCaRP also stores new cases in the case-bases during play. This means that

its strategy can potentially change over time as more and more cases are stored. However, this requires a different reuse policy than majority-rules as this policy will always reuse the most common decision and therefore not change. By using the G-bucks evaluation as planned, BayCaRP could instead use the best G-bucks score as a reuse policy. This could result in a strategy that can adapt over time and also be less influenced by luck (in comparison to the best-outcome policy).

4.2.5 Discussion

An important quality of a poker player is the ability to analyze the opponent and dynamically change playing style to exploit potential weaknesses. The CBR agents presented above do not implement strongly adaptive strategies. CASPER and SARTRE focus on imitating expert players. This approach works well against equilibrium-based opponents, but is not effective against strong exploitive opponents. CASPER and SARTRE prove that CBR can successfully be applied to a problem without requiring extensive domain knowledge. However, incorporating adaptive strategies with CBR through domain knowledge could potentially lead to increased overall performance.

Casey has the ability to adapt its playing style by continually storing new cases (starting from an empty case-base) and using a best-outcome reuse policy. BayCaRP also stores new cases during play as well as incorporating a type of opponent modeling. These qualities can result in more robust strategies that can handle different kinds of opponents.

In [39], Rubin and Watson experiment with combining decisions from different expert poker agents in order to maximize performance. In this system a decision is found by using either *ensemble voting* or *dynamic selection*. With ensemble voting, each expert votes on what decision to make and the decision with the majority of votes is chosen. With dynamic selection the optimal decision, found by using an algorithm called UCB1 [6], is chosen during play. The UCB1 algorithm calculates *regret* as the loss of profit due to not choosing the optimal solution at all times. Regret minimization is then used as a basis for decision making.

Considering that profit is not a good way to measure performance in poker, a different variable should be used in the UCB1 algorithm in poker applications. The decision evaluation tool DIVAT [12] (see chapter 5) was used in Rubin and Watson's experiment to reduce variance affecting the evaluations. The evaluation score given from DIVAT was used instead of profit as a variable in the UCB1 algorithm.

In [39], experiments were carried out in both fixed-limit and no-limit Texas Hold'em poker against two different opponents. Six expert poker agents were used as the basis for the system's decision making. The results show that use of either ensemble voting or dynamic selection appear to improve overall performance. In fixed-limit poker the dynamic selection technique seems to perform slightly better than ensemble voting. In no-limit poker the performance of dynamic selection and ensemble voting did not improve upon the Hyberborean agent with a majority-rules reuse policy.

Rubin and Watson suggest that additional opponents could be needed in order to see benefits.

In [20], Johanson, Zinkevich and Bowling research different counter-strategies in poker. Results show that agents that are trained to exploit certain opponents perform very well against these opponents, but against other opponents the performance decreases significantly. They introduced a new strategy technique called *restricted Nash response* that attempts to handle a variety of opponents. By computing an approximated Nash equilibrium for a modified game of poker using abstractions, they are able to apply regret minimization to create robust counter-strategies. This approach resulted in good overall performance, beating various opponents without being prone to exploitation. It was far superior to an agent that is trained only against a certain opponent.

What we aimed for in our own CBR poker agent (presented in chapter 9) is an agent that can play poker well with a baseline strategy in addition to being able to identify an opponents playing style and then adapt its strategy to take advantage of the opponent. With knowledge about different poker strategies and good counter-measures we can implement different types of strategies that can be applied when recognizing an opponent's playing style. This means that the seemingly best counter-strategy can be applied, and with this approach the agent does not rely on a strong static strategy that can easily be exploited.

An important part of the poker strategy in a CBR agent is the reuse policy. Making good evaluations of the quality of decisions can be critical and this is a challenge in poker (see section 3.3.2). Using a decision evaluation tool that reduces variance and provides a rational evaluation of quality can increase performance compared to a best-outcome policy using profit as the basis for evaluation. We present two such evaluation techniques in this thesis; DIVAT in chapter 5 and a different approach in section 9.4.3. We have experimented with applying both techniques in our CBR poker agent.

Part II

Tools and Frameworks

DIVAT

As discussed in section 3.3.2, the variance due to the stochastic nature of poker plays an important role when evaluating the quality of decisions. One can make bad decisions in poker and still come out as the winner of a hand. Consequently, one can also make good decisions, but still lose. This leads to difficulties when applying a CBR approach to make decisions in poker. If a bad play is stored in the case base with a positive outcome, this will be a misrepresentation of good decision making and it will most likely have a negative impact on performance. When constructing a case-base it is important to be able to represent as many situations as possible and also store cases that are based on good decisions. In this chapter we investigate a tool called the Ignorant Value Assessment Tool (DIVAT)¹ that attempts to reduce variance in the evaluation of poker decisions.

DIVAT is a tool developed by Billings and Kan from the University of Alberta and is presented in "A Tool for the Direct Assessment of Poker Decisions" (2006)[12]. It uses *perfect hindsight knowledge* to evaluate the quality of decisions made in heads up Texas Hold'em poker. Perfect knowledge (or perfect information) means knowing the complete state of the game, including the opponent's hole cards. By looking both at your own hole cards, the opponent's hole cards and the community cards, this tool is able to assess the quality of the decisions made in every stage within a played hand (pre-flop, flop, turn and river).

The way it works is by following a certain policy as to whether a bet or fold should be made in any given situation, and this can be represented by a betting sequence. Through hindsight analysis, DIVAT will calculate the difference between the actual value of a betting sequence and the betting sequence assessed by the DIVAT policy. If the difference is positive this means that the decision making was good and a negative difference means the opposite. The DIVAT difference can be summed up within a hand (the difference score from each stage) and create a total DIVAT score for a hand.

In order to use the idea behind DIVAT in our poker agent, we decided to implement our own version of DIVAT based on the description in [12]. An open source version of the DIVAT tool, called PVAT (Poker Variance Analysis Tool²), already existed, but when we discovered this we had already implemented our own version. Our version of the tool, which we named AIVAT (Another Ignorant Value Assessment

¹The first letter represents one of the author's first name.

²http://poker.cs.ualberta.ca/open_pvata.html

Tool), is strictly an imitation of the original DIVAT tool and is presented in section 9.4.2.

In this chapter we present the DIVAT tool and explain how this tool can evaluate the quality of poker decisions. A motivational example is given in section 5.1 followed by explanations of the different parts of the tool in section 5.2 and 5.3. Our own version of this tool applies the same techniques and rules as DIVAT, so this chapter can be used as an explanation of the AIVAT tool as well.

5.1 Motivational Example

We now look at an example showing the necessity of a poker evaluation tool that can disregard luck. Let us consider two players, Tom and Lisa, in a regular game of no-limit Texas Hold'em. Lisa is the dealer, the small blind is 1 dollar and the big blind 2 dollars. Bets must be at least the size of the big blind. Tom is given the hole cards $A\heartsuit, 10\heartsuit$ (ace of diamonds and 10 of diamonds), while Lisa gets $8\clubsuit, 5\heartsuit$ (8 of clubs and five of hearts).

Tom: $A\heartsuit, 10\heartsuit$ **Lisa:** $8\clubsuit, 5\heartsuit$

Pre-flop:

Lisa has a weak hand (low cards), but she has already placed the small blind and calling the big blind is not a big investment in order to see the flop. She decides to call. Tom has a strong hand and he chooses only to check so not to give away any information about his hand.

Flop: $A\heartsuit, 4\heartsuit, 7\heartsuit$

With a pair of aces, Tom has the strongest possible pair on the board. The other two cards on the flop are fairly low and Tom decides to bet his strong hand. Lisa has only an ace-high hand and she knows that her hand is weak. She goes for a bluff and raises Tom's bet. Tom decides to re-raise and Lisa's bluff fails. Lisa decides to call.

Turn: $A\heartsuit, 4\heartsuit, 7\heartsuit, 10\clubsuit$

The turn card is a very good card for Tom, and he now has two pairs; aces and tens. Again, Lisa has no more than an ace-high hand. Tom knows that Lisa often bluffs her weak hands, so he decides to go for a check to see how Lisa responds. Lisa has already invested a good amount of money in this pot and she knows that if the river card is a 6, she will get an 8-high straight. Lisa decides to place a bet after Tom's check in an attempt to scare Tom into folding.

Tom reads this bet as a bluff and again he decides to re-raise. Lisa now has the choice between folding a hand she has invested a lot of money in, or call the re-raise and hope for a lucky river card. She finally decides to call.

River: $A\heartsuit, 4\heartsuit, 7\heartsuit, 10\clubsuit, 6\heartsuit$

Character	Description
S	small blind
L	large blind (big blind)
C	call
K	check
B	bet
R	raise
F	fold
/	new stage of the hand

Table 5.1: Explanation of betting sequence.

The river card is a very lucky card for Lisa and consequently a horrible card for Tom. Now Lisa has the strongest hand with an 8-high straight. Tom still believes that he has the strongest hand with his two pairs. He decides to bet, and Lisa raises Tom’s bet knowing that she has a very strong hand. Tom becomes uncertain of whether or not he has the strongest hand. He recalls Lisa’s previous raises and wonders if she is just bluffing. He decides to call Lisa’s raise instead of making a re-raise. Lisa ends up winning the hand with an 8-high straight.

The betting sequence describing this example hand is presented below and an explanation of the format is given in table 5.1. The lowercase characters represent Lisa’s actions, while the uppercase characters represent Tom’s actions.

Betting sequence: sLcK/BrRc/KbRc/BrC

In this example, Tom had the winning hand all the way up until the river card, which turned out to be a lucky draw for Lisa. Tom had a fairly strong hand the whole game and he made the decisions of betting his strong hand. By looking back at this game, one cannot give too much blame on Tom for losing, considering the luck that Lisa had on the last card. If the decision evaluation would only consider the final outcome of a played hand, then this evaluation would give Lisa a good score (since she won) and Tom would be given a bad score. This would not be very helpful for future decision making, seeing as Tom actually played well considering the cards that were dealt.

In section 9.4.2 we show our AIVAT evaluation of this example hand, suggesting that this evaluation tool can provide a reasonable decision evaluation that is less influenced by luck compared to a regular outcome-based evaluation.

5.2 Definitions and Metrics

In this section we present a list of definitions and metrics that are used in DIVAT and are implemented in our AIVAT tool. The definitions and metrics are the same as in [12].

- **Immediate Hand Rank**

Immediate Hand Rank (IHR) is a way of assessing the strength of a hand without regarding any future community cards. It is simply a comparison between a given hand and all possible opponent hands at the current point in a game. The assessment is done by counting the number of times a given hand will beat all possible hands of the opponent. At the pre-flop stage there are 1,225 possible opponent hands (every two-card hand possible from 50 remaining cards). On the flop there are 1,081 possibilities, 1,035 on turn and 990 on river. The actual formula used to calculate the IHR is: $(wins + ties/2)/(wins + ties + losses)$. IHR is represented by a number between 0 and 1 where a higher number means a stronger hand.

- **7-card Hand Rank**

The 7-card Hand Rank (7cHR) is another way of assessing hand strength, but this time all future community cards are enumerated. This means that every hand that is ranked consists of seven cards (two hole cards + five community cards) in which the best possible 5-card hand is used. 7cHR is represented by a number between 0 and 1 where a higher number means a stronger hand. 7cHR contains a mixture of positive potential and negative potential of a hand. Positive potential considers the chance of increasing in strength as additional community cards are dealt, while negative potential means the opposite.

- **Effective Hand Rank**

The Effective Hand Rank (EHR) is used to even out the mixture of positive potential and negative potential of a hand. The 7cHR measurement implies that every hand will go to a showdown (five community cards are dealt) which is not the case. This will often overestimate weak hands because they have potential to increase in strength as additional cards are dealt. The DIVAT folding policy (see section 5.3) on the flop uses $EHR = (IHR + 7cHR)/2$, while the betting policy (see section 5.3) uses $EHR = \max(IHR, 7cHR)$.

- **All-in equity**

All-in equity (AIE) measures how many times a given hand will win, lose or tie against an opponent's hand. This measurement requires perfect information about the opponent's cards. All possible future community cards are enumerated and every showdown result is registered. There are 44 different possibilities on the turn (52 cards in total minus your hole cards, the opponent's hole cards and four community cards), 990 on the flop and 1,712,304 possibilities pre-flop. AIE is represented by a number between 0 and 1 where a higher number means a stronger hand.

- **Net gain/loss**

In DIVAT, AIE is used to calculate the *net gain or loss*. Net gain or loss is given by: $net = (AIE \cdot potsize) - invested$, where *invested* represents the amount of money invested into the current pot by the player in question. This calculation shows how much money a player will win or lose on a given hand on average.

	Fold Offset	Make1	Make2	Make3	Make4
Pre-flop	0.000	0.580	0.825	0.930	0.965
Flop	0.075	0.580	0.825	0.930	0.965
Turn	0.100	0.580	0.825	0.930	0.965
River	0.000	0.640	0.850	0.940	0.970

Table 5.2: Standard DIVAT settings from Billings and Kan [12].

5.3 The DIVAT Policies

The DIVAT policies are used to make betting sequences which again are used to calculate the DIVAT score. The folding and betting policies are rules that decide how many bets should be invested in a certain hand. The folding policy says that every hand that has a hand strength less than a certain threshold should be folded. More accurately, the folding policy used is given by:

$$\text{threshold} = \text{bet size} / (\text{total pot size} + \text{bet size} + \text{offset})$$

This means that if the EHR is less than the given threshold, the player should fold the hand. The *offset* is used to slightly increase the fold-threshold across different stages of a hand. As the 7cHR (used when calculating EHR) also considers future community cards in the calculation, this value is influenced by the draw potential (chance of getting the cards needed to make a hand). This draw potential has the largest impact on the 7cHR calculation in the flop and turn stages. In the pre-flop stage, all hands have sufficient draw potential and no offset is needed. In the river stage, no community cards remain to be dealt so there is no draw potential. This is why the offset is only introduced in the flop and turn stages. The different offsets are shown in the left column of table 5.2 and are the same as in [12]. The offsets are chosen based on the outcome of millions of simulations [12].

The betting policy works in somewhat the same way as the folding policy. If the hand strength is larger than a calculated threshold the player should bet/raise. The thresholds vary between the river stage and the other three stages. The threshold is highest on the river. The threshold for making a first bet (make1) is lower than the threshold for a raise (make2), which again is lower than the threshold for a re-raise (make3). The highest threshold is given for a re-re-raise (make4) (which is usually considered the last possible raise). The different thresholds are given in table 5.2 and are the same as in [12].

The gap between the fold threshold and the bet threshold can be considered the calling-gap. If a hand is not strong enough to raise, but not weak enough to fold, the player should instead check or call.

By following the DIVAT folding and betting policies, the betting sequence between two players can be created. This betting sequence represents the line of play between two honest players, without any deceptive plays (e.g. bluffing) or any consideration of betting history. This is called the *baseline sequence*. This baseline sequence can be assessed in all stages of a hand (pre-flop, flop, turn and river). The DIVAT policies

are based on a bet-for-value tactic meaning that a bet or raise always represents a positive expected value. The policies represent a realistic measurement of how much a player should invest in his hand at any given time.

5.4 Evaluating Quality of Decisions Using DIVAT

Evaluation of decisions using DIVAT is based on a comparison between the betting sequence of a non-DIVAT player (let us call this player *agent A*) and a DIVAT player. At the beginning of each stage of a hand, a betting sequence that represents the betting decisions made during this stage between two competing DIVAT players can be found. The DIVAT players have the same cards and stack sizes as the actual players in the game (agent A and its opponent). The two DIVAT players both follow the DIVAT policies explained in section 5.3 and represent two honest players following a bet-for-value strategy. The two DIVAT players play the hand from the beginning of the current stage until the end of this stage according to the policies. The resulting betting sequence from the DIVAT players makes up the *baseline sequence*.

After agent A has played the current stage of the hand against its opponent, the *actual betting sequence* is known. Now that we have the actual betting sequence and the DIVAT baseline sequence we can calculate the two net gains resulting from these two betting sequences (from the perspective of agent A and its corresponding DIVAT player). Since we have perfect information we are able to perform this calculation (calculation of net gain requires AIE). The actual net gain represents agent A and the baseline net gain represents DIVAT. These two values can now be compared in order to evaluate agent A's decision making. If the actual net gain is higher than the baseline net gain, agent A is given a positive DIVAT evaluation score. If the actual net gain is lower, the DIVAT evaluation score will be negative.

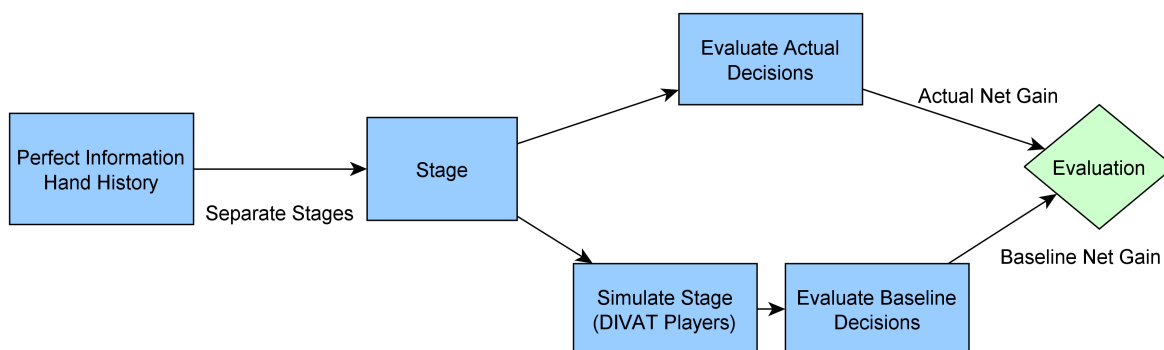


Figure 5.1: Illustration of DIVAT.

Figure 5.1 illustrates how the DIVAT tool works. The input is perfect information hand history in which the separate stages are treated one at a time. The actual net gain is calculated using the betting sequence found in the hand history. The DIVAT baseline sequence is found by simulating the stage using two DIVAT players.

The net gains from each betting sequence are then compared resulting in the final evaluation.

By evaluating agent A's decisions against the baseline provided by DIVAT, we are able to ignore luck. If agent A receives a lucky card on the river, then the corresponding DIVAT player will also receive this lucky card. This means that agent A must always perform better than the DIVAT player, given the same circumstances, in order to receive a positive evaluation score.

In section 9.4.2 we present an AIVAT analysis of the motivational example introduced in section 5.1. This will shed more light on the idea behind DIVAT and how variance in decision evaluation is reduced using this technique.

UniPoker

UniPoker is the result of one of our previous projects on AI in poker [26]. UniPoker is an open-source Java software framework which provides users with the ability to efficiently implement their own poker agents. We have used this framework in the development of our own CBR poker agent presented in chapter 9. In this chapter we present an overview and short description of the UniPoker framework based on [26]. In chapter 8, we give an overview of the extensions and improvements that have been done to UniPoker during the work on this thesis.

6.1 Motivation

Developing poker agents requires more than just an idea for a poker strategy. There are many components that are needed in order to have an agent playing poker. Game logic like dealing of cards, bets, stacks, different stages and so on, need to be provided. Also, a testing environment is required to be able to compete against other agents to measure the performance of a new strategy.

Many researchers have been using the Meerkat API for development and the Poker Academy Pro (PAP) software for testing their poker agents. The Meerkat API provides classes and methods used for developing poker agents. The PAP software is basically a platform for poker where both humans and poker agents can compete. PAP includes many different poker agents with varying performance and lets developers plug in their own agents using the Meerkat API. After discovering that that the PAP software was unavailable we felt the need for a new test-bed for poker agents. We were also lacking documentation of the Meerkat API. We therefore decided to implement our own poker software framework.

UniPoker includes an adapter for an open source poker project called *opentestbed*¹. Opentestbed implements the Meerkat API which means that poker agents that have been originally created for PAP can still be used with opentestbed. The integration with opentestbed in UniPoker allows us to use these poker agents in our framework as well.

¹See <http://code.google.com/p/opentestbed/> for more information.

6.2 High-Level Structure

The three most essential modules of the UniPoker framework are the *framework-module*, the *agents-module* and the *simulation-module* (see figure 6.1). The framework module includes common functionality that is used by the other modules and is described in section 6.3. The agents-module includes different types of poker agents. Some agents are developed by us while others are open-source agents we have included for testing purposes. The agents-module is presented in section 6.4. The simulation-module includes a simulator that is used for testing the different poker agents. The simulator implements the rules of Texas Hold'em poker and can simulate millions of hands in a short amount of time. The simulation-module is described in section 6.5.

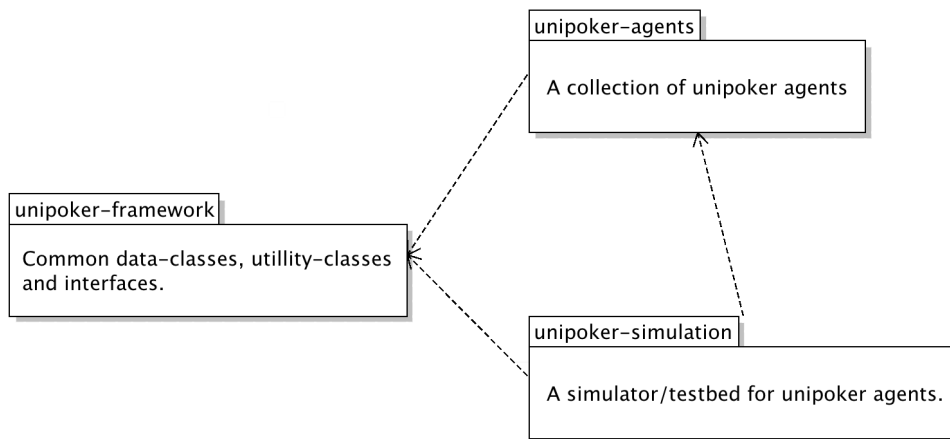


Figure 6.1: High-level structure of UniPoker from [26].

6.3 Framework-Module

The framework-module in UniPoker consists of common data classes and common utility classes. The common data classes implement important elements of poker and are presented in section 6.3.1. The common utility classes provide functionality that is needed for calculating hand strength, pot-odds, etc. The utility classes are described in section 6.3.2.

6.3.1 Common Data Classes

Figure 6.2 shows the most important common data classes and the relationship between them. These classes provide important elements of poker.

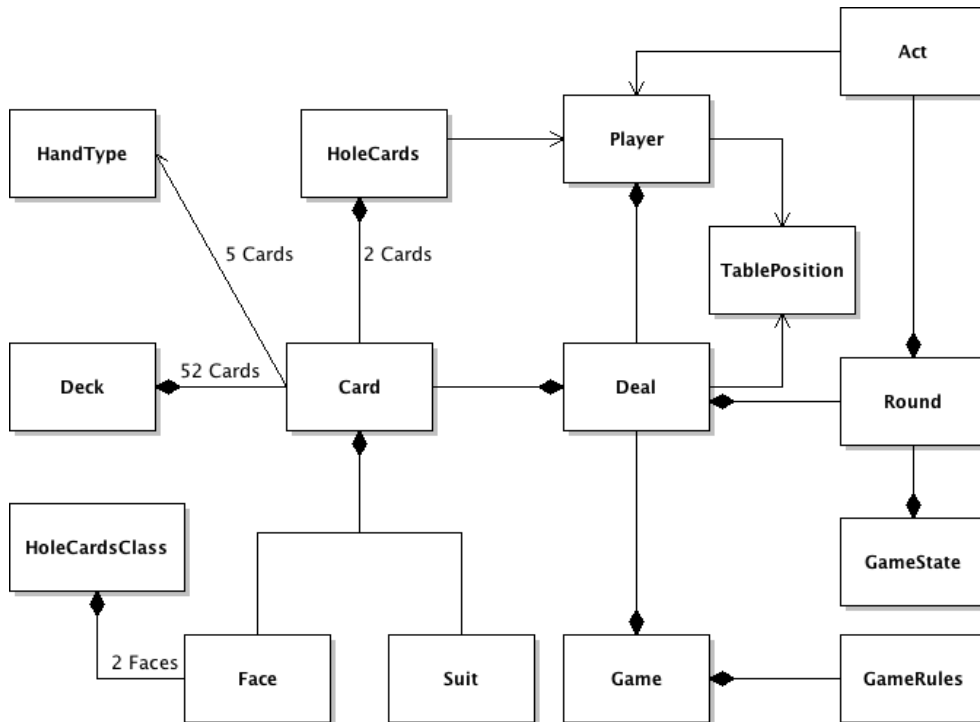


Figure 6.2: Class diagram from [26] showing the common data classes.

6.3.2 Common Utility Classes

The following list includes the different common utility classes in the framework-module.

- **HandEvaluator**
This class computes the best possible hand from the available hole cards and community cards. The result of the computation is an HandEvaluation describing the type of hand, the strength and possible flush and straight draws.
- **PotOdds**
This class computes pot odds represented by a value between zero and one (see section 2.5 for an explanation of pot odds).
- **HandStrength**
Calculates hand strength represented by a value between zero and one (see section 2.1 for an explanation of hand strength).
- **PreflopHandRollout**
This class computes the pre-flop winning probability by using hand rollout. Rollout is a technique used when calculating hand strength. Because poker is a game of imperfect information one does not know the complete state of a game (opponent cards and future community cards). Rollouts are simulations of future community cards and current opponent cards that can provide a more accurate calculation of hand strength.

6.4 Agents-Module

An agent in UniPoker is required to implement an interface including one single method that controls the agent's decision making. This method receives arguments describing the current game state and returns a single number representing the agent's action. A negative number results in a fold while a positive number indicates the agent's bet size. Returning a number that is as large, or larger, than the agent's stack size will result in going all-in. Returning zero represents a call independent of the amount needed to call. This means that if the opponent's current bet is larger than the agent's stack size, returning zero will also result in going all-in.

Figure 6.3 shows an example from [26] of the implementation of an agent applying a simple poker strategy. The SimpleAgent implements the PokerAgent interface which requires the implementation of the act-method. The strategy used by the SimpleAgent is as follows:

- Raise if the pot is less than 10 small blinds
- Else, always check/call

```
1) public class SimpleAgent implements PokerAgent{
2)
3)     @Override
4)     public double act(Game game,PokerGameRunner gameRunner,
5)         Player player, Act theAct){
6)         Deal currentDeal = game.getCurrentDeal();
7)         double pot = currentDeal.getPot();
8)         double targetPot = currentDeal.getSmallBlind() * 10;
9)         if(pot<targetPot){
10)             //Raise the difference between 10sb and the current pot
11)             return targetPot-pot;
12)         }else{
13)             //Check/Call
14)             return 0;
15)         }
16)     }
17)
18) }
```

Figure 6.3: Example of a simple poker agent.

The example in figure 6.3 shows the implementation of the SimpleAgent. The total pot size is found in lines 6-7 and the target pot size (10 small blinds) is calculated in line 8. If the pot size is less than 10 small blinds (line 9) the agent will raise the difference between the target pot size and the actual pot size (line 11). Else, the agent will check or call (line 14) depending on whether the opponent has already placed a bet or not.

6.4.1 UniPoker Agent Implementations

There are a few different poker agents implemented in the UniPoker framework that can be used when testing new strategies. Some agents are very simple while others are stronger and more complex. The different agents are presented in the list below. Some of the agents are taken from the opentestbed project.

- **Random Agent**
This agent plays with a random strategy. It folds 25% of the time. It will otherwise raise with a value between one and fifteen small blinds. If the raise is smaller than the current opponent bet it will call.
- **Simple Rule Agent**
This agent follows a simple rule-based strategy.
- **Advanced Rule Agent**
A rule-based agent that is more advanced than the simple rule agent.
- **Simple Statistics Agent**
This agent follows a strategy based on simple statistics. It collects information using rollouts, pot-odds and hand strength and uses this to calculate if it has a statistical advantage.
- **Demo Agent (opentestbed)**
The Demo Agent uses a rule-based strategy and can be found in opentestbed. It has a hard-coded pre-flop strategy and makes decisions based on pot odds and winning chance post-flop.
- **Monte Carlo Tree Search Agent (opentestbed)**
This agent is an implementation of a Monte Carlo Tree Search (MCTS) poker agent [14].
- **Chump Bot² (opentestbed)**
This agent models a loose-aggressive playing style. It plays many hands and will often raise. It can be found in opentestbed.
- **Flock Bot² (opentestbed)**
This agent models a very loose playing style and can be found in opentestbed. It applies a strategy of always calling except on the river.

6.5 Simulation-Module

The simulation-module in UniPoker is used for testing the performance of different poker agents. It includes the rules of Texas Hold'em poker and can simulate games between two or more players. It supports Duplicate Poker (see section 3.3.1) which reduces variance when testing agents. The user can choose which agents to include in the simulation and how many hands to simulate. During simulation, a graph is

²<http://oursland.net/projects/pabots/>

continuously updated that shows the bankroll (total money won or lost) of all players on the y-axis and the number of hands simulated on the x-axis. The simulator has the ability of simulating millions of hands in a short amount of time, but the actual simulation time depends on the agents' run time.

6.5.1 Simulation Example

Here we present an example from [26] illustrating typical use of the simulation-module. We want to simulate 100,000 hands between three poker agents. Figure 6.4 shows the code needed in order to begin the simulation. The number of hands to simulate is given as an argument to the Simulation constructor in line 5 and the three agents are added in line 7-9.

```

1) public class SimulatorDemo {
2)
3)     public static void main(String[] args) {
4)         //Number of hands to simulate as argument
5)         Simulation sim = new Simulation(100000);
6)
7)         sim.addPlayer(new RandomAgent(),"Jack");
8)         sim.addPlayer(new SimpleRuleAgent(),"Frank");
9)         sim.addPlayer(new AdvancedRuleAgent(),"Luke");
10)
11)        sim.openGui();
12)        sim.start();
13)    }
14) }

```

Figure 6.4: Example of testing three different agents in the simulator.

The progress of the simulation can be observed through a graphical user interface (GUI). This includes information about the estimated time remaining and number of hands simulated per second, in addition to the bankroll of each agent and number of hands simulated. When the simulation is over the results are printed out. Table 6.1 shows the results from the example simulation. This includes the total bankroll of each agent and the average win-rate per hand. Figure 6.5 shows a simplification of the GUI after the completed simulation.

Player	Bankroll	Win-Rate
RandomAgent	-13,263.0	-0.1326
SimpleRuleAgent	-3,957.5	-0.0396
AdvancedRuleAgent	17,220.5	0.1722

Table 6.1: Results after simulating 100,000 hands.

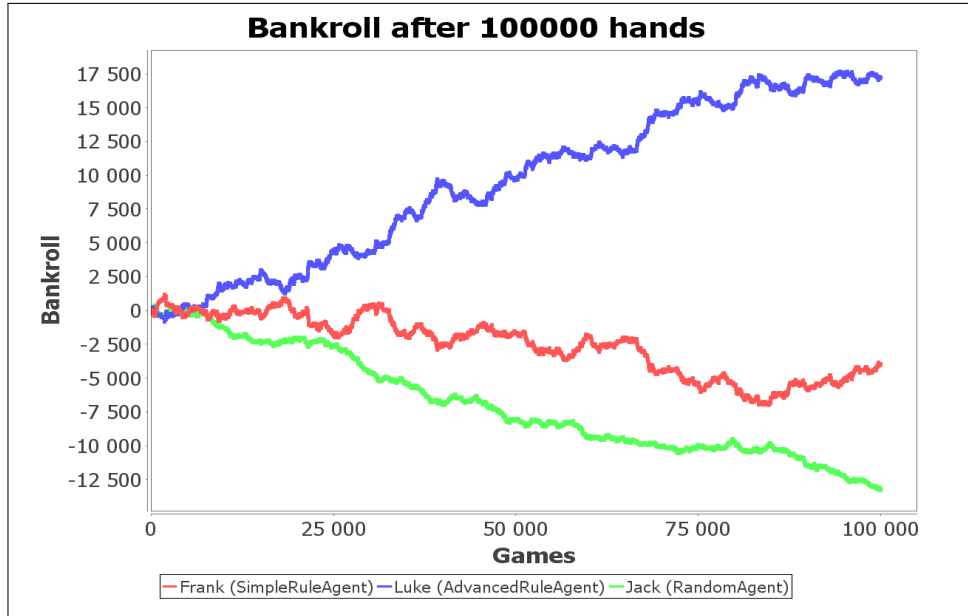


Figure 6.5: The graphical user interface after completed simulation.

We can see from table 6.1 that the advanced rule agent won the simulation with an average win-rate of \$0.1722 per hand. The random agent lost, while the simple rule agent ended up second. This example illustrates the uncomplicated use of the simulation-module in UniPoker. A guide for setting up and running UniPoker is given in appendix B.

UpperCase - Initial Design

In addition to the UniPoker framework described in chapter 6, a proposed design for a CBR poker agent was also created in [26]. CBR has proven to be successful in imitating strong poker agents, however, not many CBR-based agents apply adaptive strategies. Our agent, which we named UpperCase, was designed to use two collaborating CBR systems. One system would produce a static equilibrium strategy and the other an adaptive strategy.

UpperCase was designed to play heads up no-limit Texas Hold'em poker. The idea was to have a baseline strategy that performs well in general, as well as an adaptive strategy that is applied to increase performance after recognizing the opponent's playing style. In this chapter we give a short description of the proposed design. Some parts of the design are not relevant for this thesis and are therefore not fully explained. A more detailed explanation of the proposed design can be found in [26].

The baseline strategy uses the CBR system called EQ (equilibrium) and is presented in section 7.2. The CBR system used for creating an adaptive strategy was named EX (exploitive) and is presented in section 7.3. The final design and implementation of UpperCase separates itself from the initial design in some ways and is described in detail in chapter 9.

7.1 High-Level Design

Figure 7.1 illustrates the initial high-level design of UpperCase. UpperCase was designed to have two different training phases. One is based on the EQ-system and is unsupervised, while the other training phase, based on the EX-system, uses a human expert to guide its training. The unsupervised training is performed in the UniPoker simulator with perfect information. In the supervised training, the human expert can view information and statistics about the system through a web-interface and direct training towards situations where UpperCase has weak knowledge or experience.

The goal of the supervised training is to improve the domain knowledge of UpperCase, including the different concepts (tactics, metrics and profiles), that it can use to better adapt its strategy towards exploiting its opponent. The human expert can

analyze the performance of UpperCase by looking at which situations that demand more training and then force the agent to make decisions that lead to these.

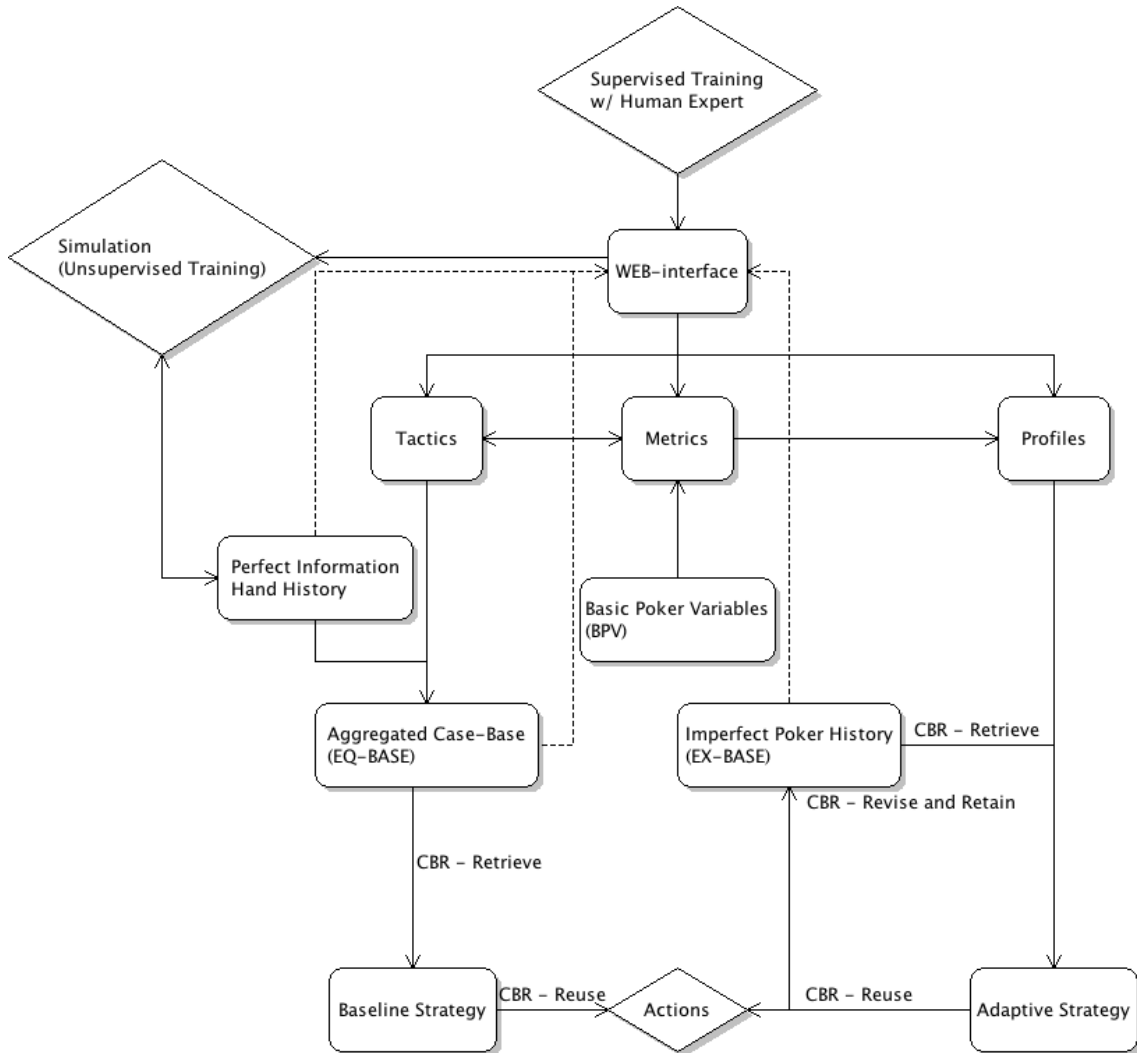


Figure 7.1: High-level view of the initial design of UpperCase from [26].

The dotted arrows in figure 7.1 represent data that is observable through the web-interface. The solid arrows illustrate the information flow between the different parts of the system. Tactics, metrics and profiles are defined by a human expert using the web-interface. These elements constitute the domain knowledge of UpperCase. The basic poker variables are attributes that describe the current game-state, like pot-size, hole cards, and stack sizes. These attributes are observable from a player's perspective. The main components of UpperCase, EQ and EX, are described below.

7.2 EQ

The EQ-system creates a baseline strategy based on DIVAT evaluation (see chapter 5). UpperCase is trained against a set of different opponents using perfect information hand history which makes the DIVAT evaluation possible. Different tactics are tested in simulations and the results are used to create the EQ case-base. New situations are explored using a random strategy during training which results in a large variety of situations. The objective of the training is to experience as many different situations as possible such that the case-base provides good coverage. If the agent encounters a large number of unknown situations during play the performance will degrade significantly. Statistics are kept over which actions and situations UpperCase has experienced.

Every decision is evaluated using the DIVAT tool and each case in the case-base includes a DIVAT evaluation score that can be used with a best-score reuse policy during play. This case-base created during training makes up the basis for the baseline strategy. The training of UpperCase is performed against a variety of opponents that have different playing styles in order to develop a robust strategy. The DIVAT tool provides reasonable evaluations of the decisions made during training.

7.3 EX

The initial design of UpperCase required strong domain knowledge taught by a human expert. Knowledge about different concepts in poker were important for the agent's ability to analyze its opponent. After recognizing the opponent's playing style, the adaptive strategy is applied.

The EX-system provides UpperCase with this adaptive strategy. When UpperCase meets a new and unknown opponent the EX case-base is empty. UpperCase will then rely on the EQ strategy for decision making. During play, new cases are stored in the EX case-base. As the case-base grows, the EX-system analyzes the opponent's playing style. The EX-system will then eventually classify the opponent's specific player profile based on the playing style. After this classification has been performed, the EX strategy takes precedence over the EQ strategy. This new strategy is tailored to exploit known weaknesses with the observed playing style and should therefore perform better than the baseline strategy. When UpperCase meets the same opponent in a new game, the old EX case-base designed for this specific opponent is reused.

Part III

Results

Improvements to UniPoker

During the work on this thesis we have added new functionality to the UniPoker framework when this was necessary for the implementation of UpperCase. We have also improved the original functionality and fixed some unresolved issues. These changes are presented in this chapter.

8.1 Improved hand evaluation

Comparing and classifying different poker hands is referred to as *hand evaluation*. This can be a very important feature for some poker agents. UpperCase uses hand evaluation extensively during the training phase, and by improving the efficiency of hand evaluation, we were able to reduce the time spent training the system. Hand evaluation is also required in order to declare the winner of a hand during simulation. Not only is the correctness of the evaluation paramount in this situation, but the efficiency greatly affects how fast poker hands can be simulated. Since we perform a large number of simulations in our testing of UpperCase, the simulation-speed was important to us. We have improved the performance of the original UniPoker hand evaluator significantly.

Sometimes, e.g. when determining the winner of a hand, it is sufficient to only produce a ranking of the different hands. A ranking does not have to include the type of hand or tie-breaking values, only a score that can be compared to the score of other hands. This can be done efficiently using a large collection of pre-computed hands. The *Steve Brecher's HandEval*¹ is a software-library able to do efficient hand rankings using this approach. This library has now been integrated with UniPoker.

¹Description and source code can be found on http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup#steve_brecher

8.2 Integration with the ACPC protocol

As mentioned previously, ACPC is the Annual Computer Poker Competition. In this competition, poker agents communicate with an ACPC server² using the ACPC protocol. We have now implemented support for the ACPC protocol in UniPoker so that the framework can be used both as a ACPC server. Additionally we have also implemented support for UniPoker poker agents to connect as clients to the original ACPC server. Now, poker agents communicating through the ACPC protocol can be used with the UniPoker framework. Poker agents, implemented using the UniPoker framework, would now be able to participate in future ACPCs.

8.3 Improvements of the Poker Simulator

A central element of UniPoker is the poker simulator, implementing the rules of Texas Hold'em poker and used to simulate poker hands. While implementing our poker agent we have improved this significant part of the framework in different ways.

Direct Control

While testing poker agents we discovered the value of being able to directly control the simulation. The simulator has been redesigned so that developers are able to run poker simulation while controlling aspects of the game, such as which cards to be dealt, which stages to be executed and which actions are performed. The possibility of creating test-cases, using controlled, partial hand-simulations, proved to be helpful during development.

Side-pots

Side-pots are now supported in the simulator. This is not needed for testing heads up poker agents (like UpperCase), but is important when simulating games between more than two agents.

8.4 Web-Interface

We have developed a web-interface where human players can challenge the poker agents available in the UniPoker framework. A screenshot, displaying poker being played using this interface, is shown in figure 8.1. This web-interface presents the player's hole cards and the community cards, and it lets users fold, check/call, bet/raise. This tool makes it possible to investigate how poker agents perform against real human competition.³

²Source code can be found on http://www.computerpokercompetition.org/index.php?option=com_rokdownloads&view=file&Itemid=59&id=137:acpc-2011-server-code

³This web-interface is currently hosted at <https://unipoker.agorait.no/> and will be maintained for a period of time after the submission of this thesis.

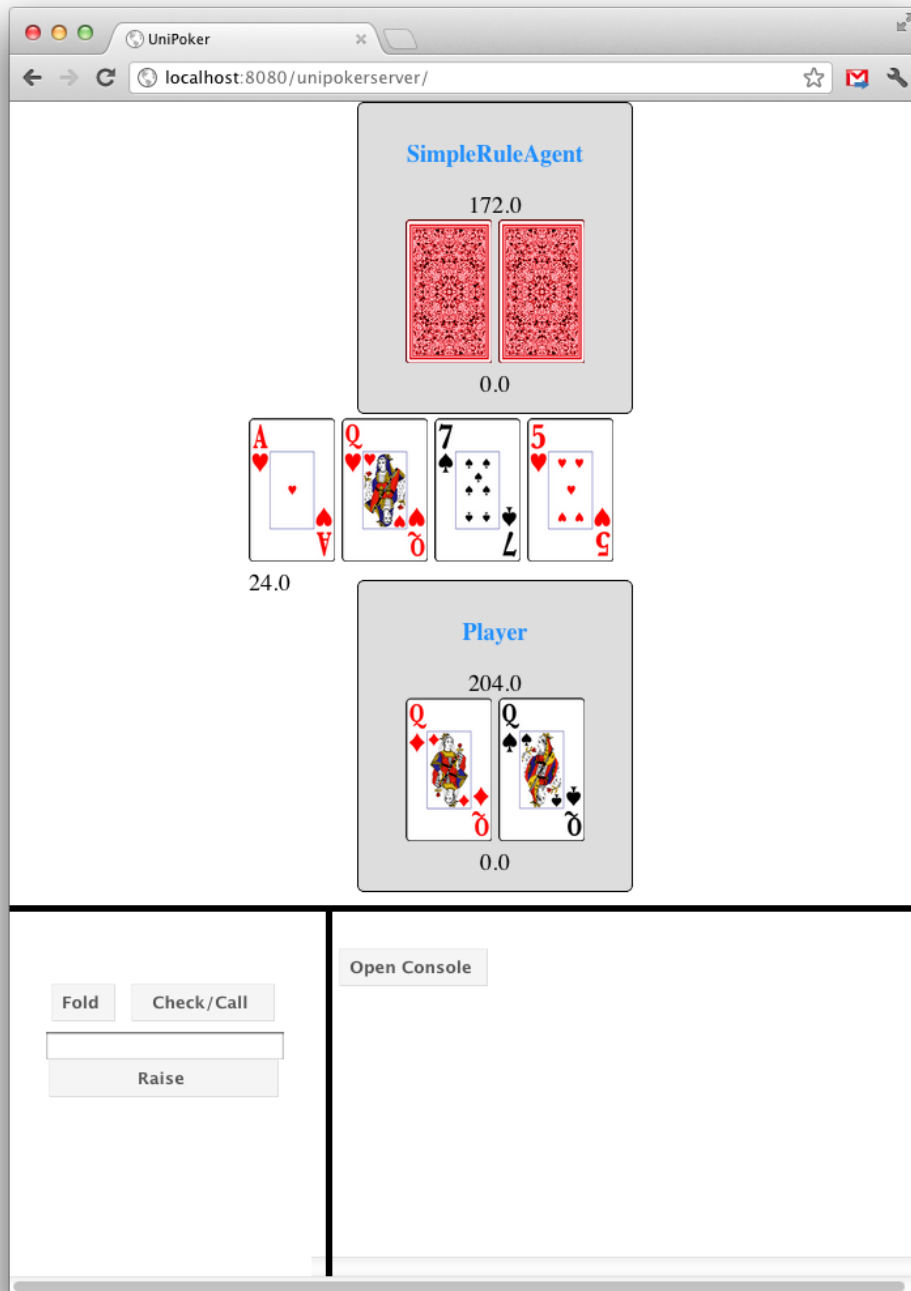


Figure 8.1: Screenshot of a poker-game using the UniPoker web-interface.

UpperCase

The design, implementation and testing of the UpperCase poker agent has been our main objective in this thesis. We have developed a system with a new approach to how CBR can be applied to the game of poker. Our approach focuses on developing adaptive capabilities, which is something that we have not observed to a great extent during our study of existing CBR solutions.

This chapter presents the design of UpperCase. First, we provide a high-level view of the system’s architecture in section 9.1. In section 9.2, we explain how the design has evolved since the system first was proposed in [26]. Section 9.3 presents QBR, a CBR-module within UpperCase. An explanation of *Decision Quality Evaluation* is given in section 9.4. Sections 9.5 and 9.6 explain EQ and EX, which are the two major components of the system. Finally, a summary of the system is given in section 9.7.

9.1 High-Level System Architecture

The task of a poker agent is to produce actions as output when different game-states are given as input. In order to do so, UpperCase employs two CBR-systems, named EQ (equilibrium) and EX (exploitive), that are able to make such decisions in a collaborative manner. When the agent is requested to make a decision, EQ will first suggest an action that is coherent with some static equilibrium-strategy. We call this the *baseline strategy* of UpperCase. With sufficient knowledge about the current opponent, EX will then adjust the baseline strategy in order to exploit the opponent. The final decision will be a combination of the output from the two individual systems. This process can be viewed in figure 9.1 below.



Figure 9.1: High-level view of UpperCase.

Figure 9.2 provides some more detail. This figure shows the architecture of the

system, with the major components, EQ and EX, in separate boxes. The EQ-system uses one separate case-base, while the EX-system employs multiple case-bases. Throughout this chapter we explain how the different parts of the system shown in figure 9.2 work and interact. We revisit this figure in section 9.7, which summarizes the chapter, to give a more detailed explanation of the figure.

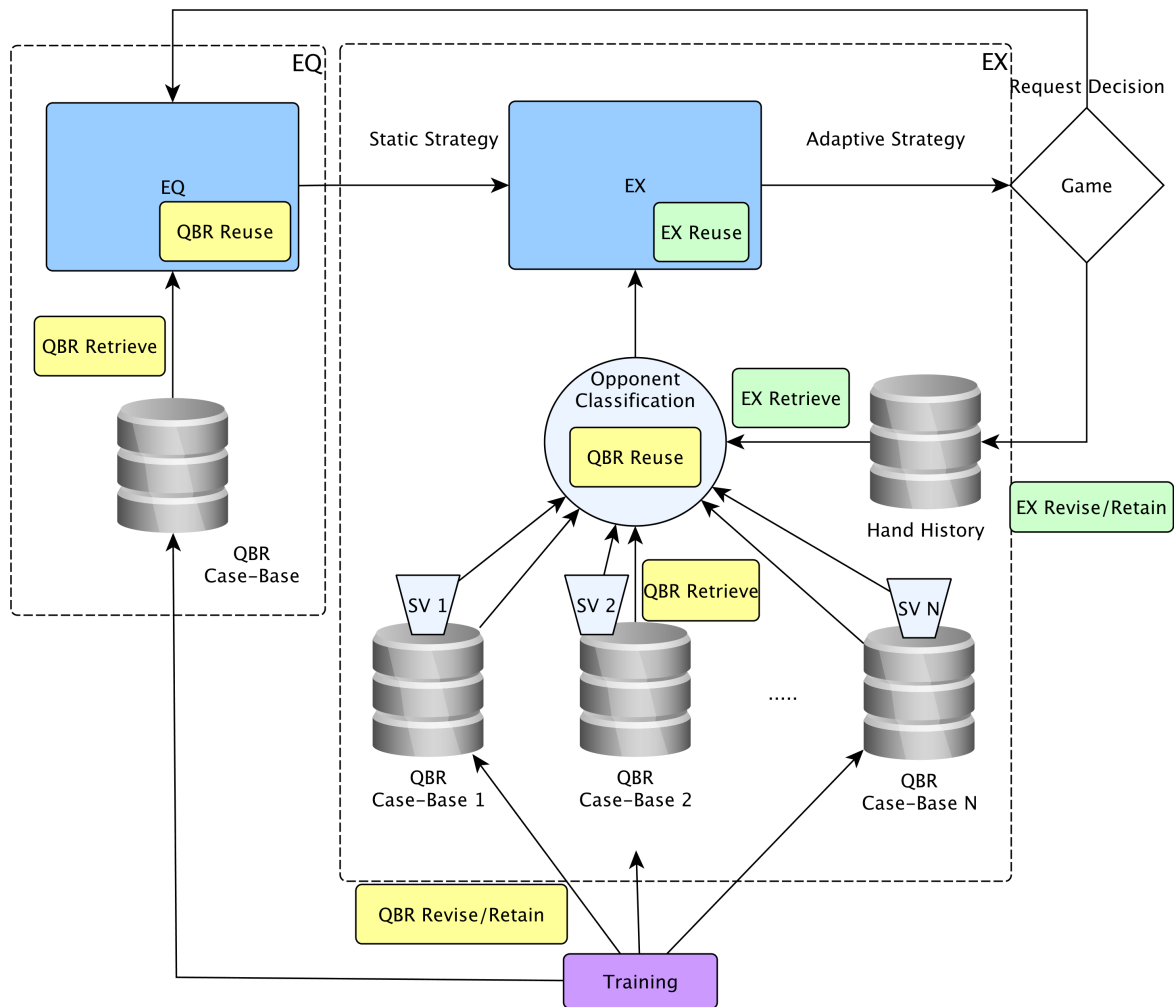


Figure 9.2: System Architecture

9.2 Differences from Previous Design

As explained in chapter 7, a proposed design for UpperCase was presented in [26]. Now, we have realized the idea behind UpperCase and implemented the system. During implementation and testing we have done some modifications to the initial design, which are presented in this section. We refer to the previous design as *UpperCase-1* and the current design as *UpperCase-2*.

The basic idea of two collaborating CBR-systems and their role in making a decision

remains unchanged from UpperCase-1 to UpperCase-2. If we compare figure 7.1 and 9.2 we see that the EQ and EX subsystems are present in both designs. EQ produces a *baseline strategy* or *static strategy* and EX produces an *adaptive strategy*. The important difference between the designs is that UpperCase-1 was envisioned to be a knowledge-intensive system, whereas UpperCase-2 is a data-intensive system using an instance-based approach to CBR.

Domain knowledge in UpperCase-1 is represented through *tactics*, *metrics*, *variables* and *profiles*, but these are no longer present in UpperCase-2. The idea of teaching UpperCase a small number of poker concepts using a human expert has been replaced by an approach that classifies the opponent based on a set of statistical poker metrics. This means that the system is no longer dependent on a human expert for guidance. Overall, this change mostly affects how EX functions. EQ remains more or less unchanged from UpperCase-1 to UpperCase-2.

The reasons for changing the initial design is based on the complexity of the knowledge-intensive approach of teaching UpperCase poker concepts. During implementation we quickly discovered great difficulties in representing such concepts as well as how they can be used to make decisions. Also, it would require much time and effort for a human expert to create enough cases to give good coverage of situations. These problems are rooted in the fact that poker is a very complex game with a large number of possible game-states, where it is difficult to create a small number of abstract concepts that handle this challenge well.

9.3 Quality-Based Reasoning

A building block in UpperCase is a module that is able to make poker decisions based on what we call *Quality Based Reasoning* or QBR. UpperCase uses several QBR-modules in different parts of the system.

Every QBR-module is an independent CBR-system using an instance-based approach to CBR. Each QBR-module also includes its own individual case-base, meaning there is multiple case-bases within the system as a whole. Our instance-based approach leads to a large number of cases and by separating the total knowledge of the system into different case-bases we are able to retrieve cases more efficiently from the appropriate case-base. The motivation behind this is increased performance during the retrieve process. This design also allows us to scale the system horizontally while adding more QBR-modules. This is important in a full-scale deployment of the system, where we want a large number of highly trained QBR-modules to achieve maximum performance.

If we look at the system from a different angle, we can still perceive the system as retrieving cases from one single case-base. The format of cases is identical across the different case-bases and all cases are stored in the same database. The separation into different logical case-bases happens at run-time. This is why it is possible to interpret the system as using either one single case-base or multiple case-bases. In our presentation of the system, we consider it to use multiple case-bases.

This section focuses on explaining how one single QBR-module works and in sections 9.5 and 9.6 we present how these modules are used in UpperCase.

9.3.1 Quality of Decisions

Previously executed poker decisions are retained in a case-base and reused based on their *quality*, which is a measurement of how good or bad the decision was in the previous situation. How the reuse process is performed is explained in section 9.3.5. Our concept of quality is related to the concept of *utility*, which in CBR-systems can be defined as the usefulness of a solution for solving a given problem [7]. The goal of traditional systems is to maximize utility of retrieved cases in order to be in the best possible condition to solve the current problem.

Our concept of quality is a discrete measurement describing the usefulness of the solution applied to the previous problem. This is not the exact same concept as utility, because utility is specifically related to the current problem. Figure 9.3 illustrates this relationship.

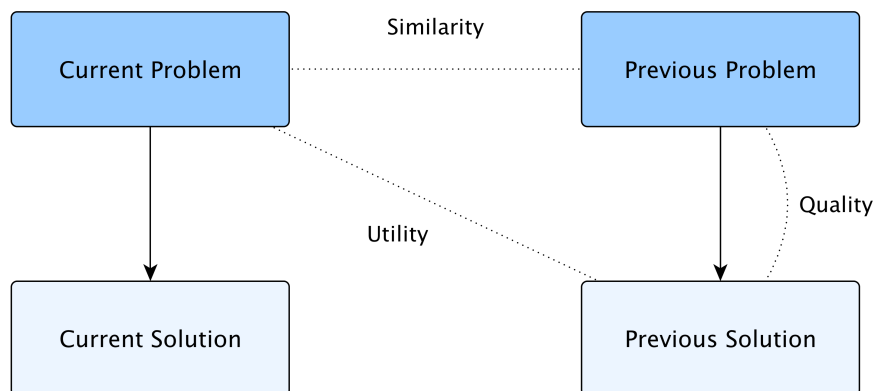


Figure 9.3: Relationship between similarity, utility and quality.

However, we are reusing cases based on quality while assuming a correlation between this and utility. In this sense, utility and quality are related.

9.3.2 QBR Case Structure

The QBR-module uses a two-level hierarchical case structure. We call the upper level a *situation* and the lower level a *solution*. A situation is an abstraction of a set of game-states that share some similarities. A solution is the result of an execution of a decision. There can be multiple solutions for each situation, but a solution can only be related to one situation. Figure 9.4 illustrates this relationship.

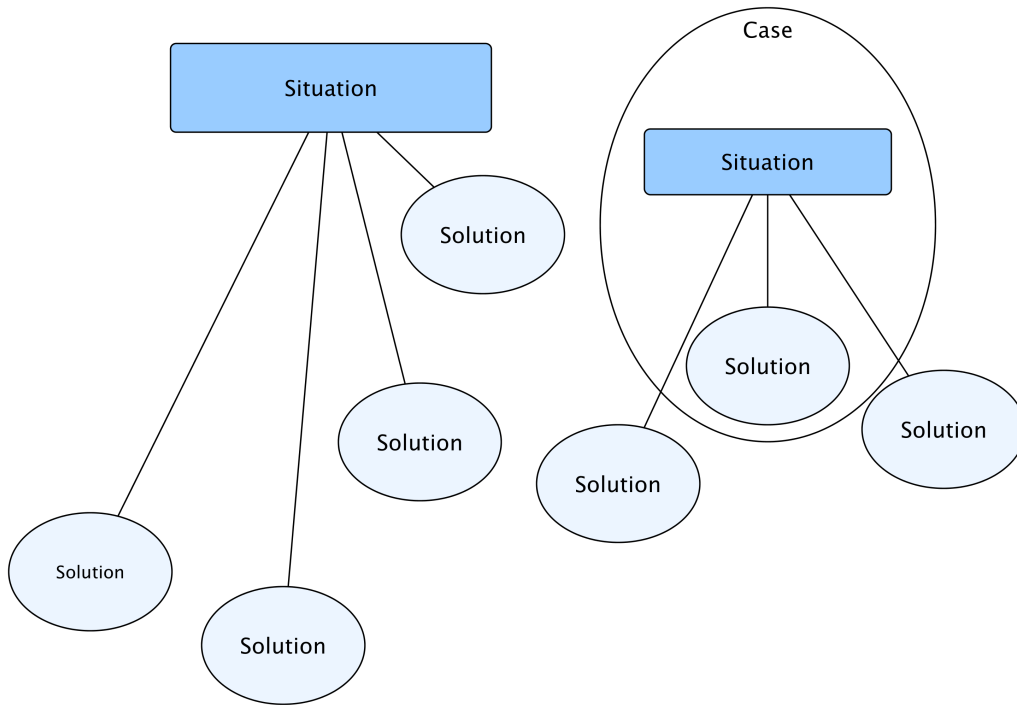


Figure 9.4: The relationship between situations and experiences.

A QBR-module is a CBR-system. Each case in the QBR-module is the combination of a situation and a solution. The features of a case are the combination of features from both the situation and the solution. Figure 9.5 illustrates the case-format in a QBR-module.

Case	
Index Features (Situation)	Situation-key Case-base-ID
Solution Features	Action-type Quality

Figure 9.5: A QBR-module case.

All index-features of the case, which are used in the retrieve-process, are contained in the situation. The index-features are:

- **Situation-key**

This index-feature is a string describing the situation. The string captures important aspects of the game-state such as the player's hand and previous actions in the round. Situation-keys are explained in section 9.3.2.2.

- **Case-base-ID**

UpperCase uses multiple QBR-modules and each module includes one individual case-base. This index-feature defines which case-base this situation belongs to.

The solution-features used in the reuse-process are contained in the solution. A solution contains the following features:

- **Action-type**

Action-type is the type of action that was executed, meaning the decision made in the previous problem. Possible values are defined in section 9.3.2.1.

- **Quality**

The quality of the action, describing if the action executed was good or bad. This value is determined using *Decision Quality Evaluation*, which is explained in section 9.4.

9.3.2.1 UpperCase Action-Types

Action-types are the possible decisions UpperCase understands. In a game of no-limit poker there is a large number of possible bets that can be made. We have reduced the possible decisions into four different types of actions:

- **No Play (Fold)**

A *No Play*-action ends a player's participation in the current hand.

- **Passive Play (Check/Call)**

A *Passive Play*-action is the action allowing continued participation in the hand with minimal investment.

- **Aggressive Play (Bet/Raise)**

Aggressive Play is the investment of more than the minimal amount required for continued participation in the hand. The upper bound for this action-type is three times the current pot size. When UpperCase uses this action-type to act, it will bet the same amount as the current size of the pot.

- **Very Aggressive Play (Large Bet/Raise)**

Very Aggressive Play is comparable to *Aggressive Play*, but the invested amount is equal or larger than three times the current pot size. When UpperCase uses this action-type to act, it will bet three times the amount of the current size of the pot.

9.3.2.2 Situation-Key Format

A QBR situation-key describes an abstraction of different game-states in a poker game. For example, the state *"On river with medium-high three-of-a-kind. Both me and my opponent have been mostly passive this hand"* could be expressed as: "AaP|pP|pP|p/33". The format of the key is:

$$\langle \textit{Betting Pattern} \rangle / \langle \textit{Hand Type} \rangle$$

The $\langle \textit{Betting Pattern} \rangle$ -section represents previous actions in the current hand. It contains a set of characters that defines which actions that occurred in which stages. Possible values for actions are the different action-types presented above (see 9.3.2.1) which are: *n* (no play), *p* (passive play), *a* (aggressive play) or *v* (very aggressive play). '|' represents the advancement to a new stage of the hand. This way it is possible to know which actions that belong to the different stages. The format of the $\langle \textit{Betting Pattern} \rangle$ -section is summarized in table 9.1 below.

Character	Description
N	no play
P	passive play
A	aggressive play
V	very aggressive play
	new stage of the hand

Table 9.1: Explanation of betting pattern.

Actions performed by UpperCase are capitalized to improve human readability of the format. As an example, consider the string "AaP|". This string would mean that UpperCase performed an aggressive play (raise), the opponent followed up with another aggressive play (re-raise) and UpperCase ends the current stage with a passive play (call).

By looking at betting patterns it is possible to give an approximation of the pot size of the hand. Obviously, the size of the pot of a hand with mostly checks and calls, which are passive plays, will be smaller than the size of the pot in a hand with mostly aggressive plays like bets and raises. Two hands with similar betting patterns can therefore be assumed to have approximately the same pot size.

The $\langle \textit{Hand Type} \rangle$ -section represents the type of hand the player holds in the current game-state. In the pre-flop stage, the value of $\langle \textit{Hand Type} \rangle$ is one of the 169 possible hole card hand-classes (see 2.1), for example 'AA', which means two aces.

In post-flop stages, the value of this section is two digits describing the best 5-card hand the player can make in the current situation:

- The value of the first digit represents one of nine possible classes of 5-card hands (see table 9.2).
- The second digit is the most significant tie-breaking value of the hand class (see table 9.3). This is a face with thirteen different possible values converted to an integer.

We do integer division by three on the tie-breaking value, reducing the number of possible values for this digit to five (see table 9.4). This reduces the number of possible values of the *<Hand Type>*-section from $9 \cdot 13 = 117$ to $9 \cdot 5 = 45$ for post-flop stages. This is motivated by the need to counteract a natural bias in training data. Since several poker hands never reach the final showdown stage, there will be less cases in the case-base for the later stages. Decreasing the number of possible values of the *<Hand Type>*-section for post-flop stages increases average number of retrieved cases in these stages.

Value	Hand Class
0	High-Card
1	Pair
2	Two Pairs
3	Three of a kind
4	Straight
5	Flush
6	Full House
7	Four of a kind
8	Straight Flush

Table 9.2: Values for different 5-card hands.

Hand Class	Most significant tie-breaking value
High-Card	Face of the highest card.
Pair	Face of the pair.
Two Pairs	Face of the highest pair.
Three of a kind	Face of the three cards of same kind.
Straight	Face of the highest card
Flush	Face of the highest card.
Full House	Face of the three cards of the same kind.
Four of a kind	Face of the four cards of the same kind.
Straight Flush	Face of the highest card.

Table 9.3: Most significant tie-breaking value for the different 5-card hand classes.

Face	Face value	Value after division
Two	0	0
Three	1	
Four	2	
Five	3	1
Six	4	
Seven	5	
Eight	6	2
Nine	7	
Ten	8	
Jack	9	3
Queen	10	
King	11	
Ace	12	4

Table 9.4: Conversion of face-values.

Some examples of situation-keys:

- **/76s**
The hand is in the pre-flop stage. The player's holecards are a seven and a six of the same suit ("s" represents suited). No actions have been performed yet.
- **a/AK**
The hand is in the pre-flop stage and the player is holding an ace and a king of different suits. He is responding to an aggressive play by the opponent.
- **pAp|/03**
The hand is in the flop stage and the best hand the player can make is a high-card hand with a jack, queen or king as the high-card. The "pAp|" tells us that the actions in the pre-flop stage were "opponent check", "player raise" and "opponent call".
- **aAaP|Av/34**
The hand is in the flop stage and the player has a hand of three aces. The player is responding to a very aggressive re-raise by the opponent. The actions in the pre-flop-stage were "opponent raise", "player raise", "opponent raise" and "player call".
- **aP|PaP|/10**
The hand is in the turn stage. The player has a pair of either twos, threes or fours. The actions in the pre-flop stage were "opponent raise", "player call" and the actions in the the flop stage were "player check", "opponent raise", "player call"

9.3.3 Similarity

A QBR-module performs direct matching of both index-features of the case, which means that the *similarity-function* of a QBR-module is a binary function. Either the case is a complete match or not. However, similarity between cases is captured by the format of the situation-key. We have designed the situation-key format so that different game states will be described by the same situation key when they share some important properties, as shown below:

- **Betting Pattern**

The *<Betting Pattern>*-section ensures that the following common properties are matched:

- Stage
- Table Position
- Approximate Pot Size
- Previous Action-Types

- **Hand Type**

The *<Hand Type>*-section ensures that the following common properties are matched:

- Hole Card Class (Pre-flop)
- Approximate 5-Card Hand (Post-flop)

9.3.4 QBR Retrieval

Like a regular CBR-system, retrieving cases from the case-base is the first step in the process of making a decision. At runtime, all situations are loaded into memory. Situations are divided into different collections according to their case-base-id index-feature, which tells which case-base it belongs to. Each QBR-module is associated with a collection of situations. A QBR case-base is basically a key-value store with situation-keys mapping situations. This design allows efficient retrieval of situations.

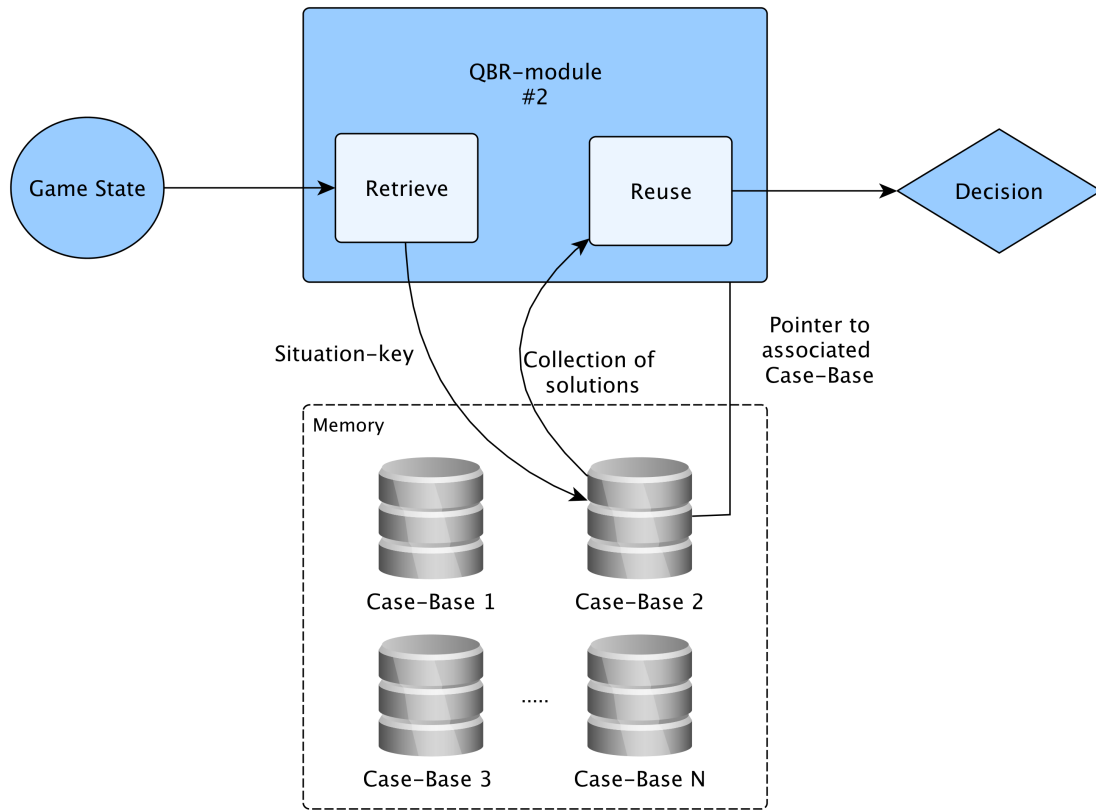


Figure 9.6: QBR retrieve and reuse process.

During retrieval, UpperCase converts the current game-state into a *situation-key* (explained in section 9.3.2.2) that is used to retrieve solutions from the associated collection. The retrieved solutions are viewed as a selection of cases, all sharing the same index-features which describes the current situation. This is illustrated in figure 9.6. Here we see that the situation-key is used to look up a collection of solutions in the associated case-base of the QBR-module (QBR-module #2 is associated with case-base #2). The retrieved cases (or solutions) are then passed on as input to the QBR reuse process.

9.3.5 QBR Reuse

As explained in section 9.3.4 above, the retrieve process extracts all cases sharing a *situation*. In the reuse process of QBR, the cases are grouped based on their *action-type* feature (see 9.3.2). This results in an aggregated QBR solution where the average quality of each action-type is calculated. Then the action-type of the group with the highest average quality is selected and becomes the chosen decision.

Aggregated QBR Solution (solutions grouped by Action-type)	
Action-Type	Average Quality
No Play	-4.0
Passive	-0.3
Aggressive	+1.5
Very Aggressive	+5.0

Figure 9.7: Example of an aggregated QBR solution.

Figure 9.7 shows an example of an aggregated QBR solution. Here we can see that the action-type *very aggressive* has the highest average quality and therefore will be reused in this example.

We call this a *best average quality* reuse policy. The quality of each solution is determined in the revise process by using *decision quality evaluation*. This is described in detail in section 9.4. Since quality determines which solutions that are reused, the decision quality evaluation has a large impact on the performance of the QBR-module.

9.3.6 QBR Training - Revise & Retain

The training of a QBR-module is performed in a two-phased process. In the first phase, a hand is simulated in the UniPoker simulator against a designated opponent. In the second phase, the results of the simulated hand is used to produce a collection of cases, that possibly will be revised and retained in a case-base. This process is repeated for a desired amount of time or until the case-base contains a desired amount of cases. The phases are explained in the sections below and can be viewed graphically in figure 9.8.

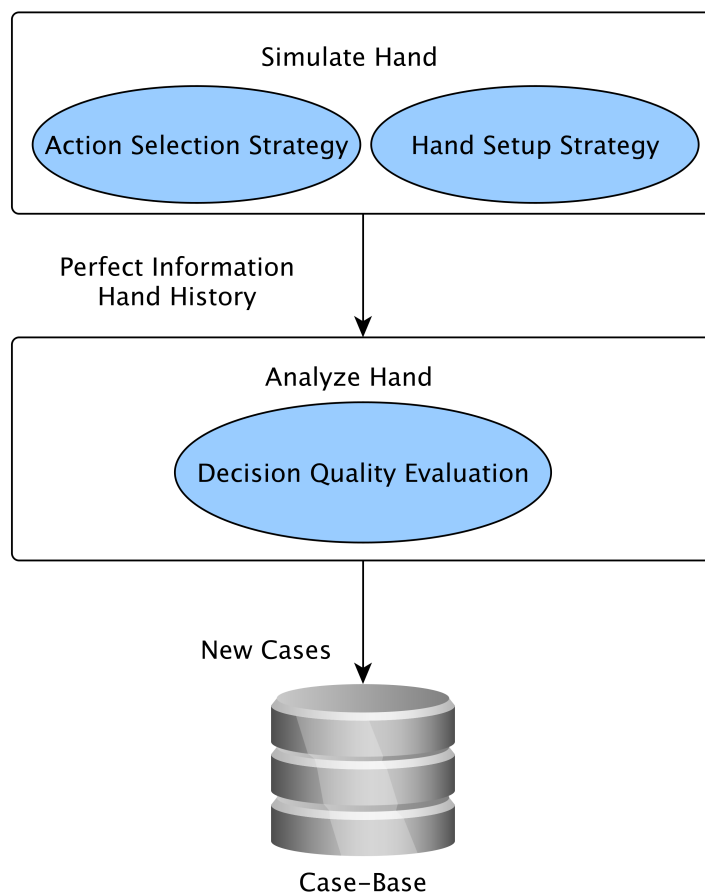


Figure 9.8: Training UpperCase.

9.3.6.1 Hand Simulation

How the hand plays out is controlled by an *Hand Setup Strategy*. This strategy can decide which cards should be dealt to the players in the different stages. This is important because it affects which cases that are created to later be revised and retained. Currently we have the following two *Hand Setup Strategies*:

- **Random**

Hands are executed in a random manner, like a regular poker game.

- **Hand-Class Permutation**

This strategy iterates through the 2-permutations of the 169 different possible hand-classes (see section 2.1) and deals an example hand to the players. In other words, the players receive samples of every possible hand-class and plays this hand-class against every other possible hand-class. After hole-cards have been dealt, the rest of the hand is executed in a normal, random manner.

Actions are selected according to an *Action Selection Strategy*. In the same way as the *Hand Setup Strategy*, this strategy is important because it affects which cases are produced by the simulation. For example, an *Action Selection Strategy* that

never raises can be a bad idea, since no situations with raises will then be evaluated and the system will have no information on the effect of such actions. Currently we have two *Action Selection Strategies*:

- **Random**
Actions are chosen in a random manner.
- **Poker Agent**
A poker agent chooses the action.

Both the *Hand Setup Strategy* and the *Action Selection Strategy* should try to fulfill the goal of producing a good distribution of cases that maximizes the coverage of possible situations. We are currently using the random strategy for both hand-setup and action-selection since this naturally gives a good distribution.

9.3.6.2 Hand Evaluation

In the Hand Evaluation-phase, the hand is converted to a set of cases for every action the player performed. Each case is revised by performing a *Decision Quality Evaluation* on the action. This results in a value describing the quality of the action in the current situation. How the evaluation is performed is explained in detail in section 9.4. The value is assigned to the case and then the case is retained in the case-base.

When a situation has occurred more than a given limit number of times, it is not evaluated or stored. Currently, we have a limit of 200 cases for each situation. This is done for training-efficiency reasons since revise and retain are time-consuming processes. This also counteracts a natural bias in poker hand history. There are both more different late-stage situations than different early-stage situations and also a lower frequency of late-stage situations due to that some hands are folded before late stages. After a satisfying number of cases in the given situation have been produced, additional cases are discarded. Early in the training phase, every case will be revised and retained, but in the end only infrequently experienced cases are revised and retained.

9.4 Decision Quality Evaluation

The variance introduced by the stochastic behavior of poker affects the outcome of decisions. Like discussed in section 3.3.1, this results in problems when evaluating the quality of a given decision in poker. Bad decisions can lead to a good final outcome if the player is lucky, and vice versa. Luck affects decision evaluation in traditional evaluation methods like using the profit as the basis for evaluation. Our decision quality evaluation techniques attempt to determine the quality of decisions independent of any luck. This can potentially lead to a stronger case-base in which only good decisions are actually reused by UpperCase.

In [37], Rubin and Watson investigate how different reuse policies affect performance. We have conducted a similar investigation of reuse policies using methods of *perfect information hindsight analysis* to evaluate decisions. These methods require hand history with perfect information of the game-states, which essentially means that we know our opponent's cards. As a consequence, these methods cannot be used during regular play or to analyze regular hand-history. The design and implementation of the *Profit Evaluator*, *AIVAT Evaluator* and *Equity Evaluator* are presented in the following sections. The results of testing the different evaluation techniques can be found in section 11.1.

9.4.1 Profit Evaluator

The Profit Evaluator (PE) is a simple reference-implementation of a *Quality Decision Evaluator*, similar to the best outcome reuse policy in Rubin and Watson's study. Every action in the same hand receives the same quality-score, which is the player's total profit of this hand (money won or lost). This evaluation technique can be used as a comparison to the other two approaches to see if they are able to improve the performance or not.

9.4.2 AIVAT Evaluator

We have created *Another Ignorant Value Assessment Tool* (AIVAT), which is our implementation of the DIVAT specification described in chapter 5.

We now use the motivational example introduced in section 5.1 to show how the analysis of Tom's performance is evaluated. We have looked at Billings and Kan's paper [12] and used this as a reference for the format and presentation of the analysis. As we recall, two players, Tom and Lisa, are playing no-limit Texas Hold'em, and Lisa is the dealer of the given hand. The equity calculations shown in the following tables are based on Tom's cards and the AIVAT difference represents Tom's performance compared to how two AIVAT policy players would play the same exact hand. The AIVAT difference is based on the difference between the *actual net gain* and the AIVAT *baseline net gain* (see section 5.2). The actual net gain represents Tom's net gain in each play, while the baseline net gain represents the net gain that a player following the DIVAT policies would get in the same play.

Tables 9.5, 9.6, 9.7 and 9.8 show the calculations done for each stage of the example hand.

Tom: $A\heartsuit, 10\heartsuit$ **Lisa:** $8\clubsuit, 5\heartsuit$

Pre-flop:

On the pre-flop, Tom decided to check after Lisa called the big blind. This check led to less money being invested in the pot by both players compared to the AIVAT baseline. In this stage, Tom had the best hand with an AIE of 0.6710, meaning that

Board:			
Pot size before betting: 0			
	IHR	7cHR	EHR
Tom: $A\heartsuit, 10\heartsuit$	0.9453	0.6709	0.9453
Lisa: $8\clubsuit, 5\spadesuit$	0.1592	0.4139	0.4139
AIE = 0.6710			
AIVAT baseline = sLcRc			
Actual sequence = sLcK			
Actual net gain = 0.6840			
Baseline net gain = 1.3680			
AIVAT difference = -0.6840			

Table 9.5: Pre-flop analysis

Board: $A\spadesuit, 4\heartsuit, 7\spadesuit$			
Pot size before betting: 4			
	IHR	7cHR	EHR
Tom: $A\heartsuit, 10\heartsuit$	0.9491	0.8215	0.9491
Lisa: $8\clubsuit, 5\spadesuit$	0.1036	0.3432	0.2234
AIE = 0.7970			
AIVAT baseline = Bf			
Actual sequence = BrRc			
Actual net gain = 4.7520			
Baseline net gain = 2.0000			
AIVAT difference = +2.7520			

Table 9.6: Flop analysis

he was entitled to a larger share of the pot. Tom's decision to check resulted in a negative AIVAT difference.

Flop: $A\spadesuit, 4\heartsuit, 7\spadesuit$

On the flop, Lisa tried to bluff Tom into folding, but Tom decided to re-raise and Lisa's plan backfired. The actual betting sequence resulted in a large pot investment. The AIVAT baseline shows that Lisa should have folded her hand after Tom's initial bet. The baseline net gain represents the amount that Tom would have won if Lisa folded (pot size - invested). The actual net gain is higher than the baseline net gain which led to a AIVAT difference of +2.7520.

Turn: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit$

On the turn, Tom decided to check and Lisa answered with a bet. Tom knew that his hand was strong so he decided to re-raise the bet. Lisa finally decided to call this re-raise in hope of a lucky draw on the river. The actual betting sequence resulted in a larger pot investment than the AIVAT baseline, and again the AIVAT difference is positive.

River: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit, 6\heartsuit$

Board: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit$			
Pot size before betting: 16			
	IHR	7cHR	EHR
Tom: $A\diamondsuit, 10\diamondsuit$	0.9903	0.9410	0.9903
Lisa: $8\clubsuit, 5\heartsuit$	0.1082	0.2303	0.2303
AIE = 0.9091			
AIVAT baseline = Bc			
Actual sequence = KbRc			
Actual net gain = 9.8184			
Baseline net gain = 8.1820			
AIVAT difference = +1.6364			

Table 9.7: Turn analysis

Board: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit, 6\diamondsuit$			
Pot size before betting: 24			
	IHR	7cHR	EHR
Tom: $A\diamondsuit, 10\diamondsuit$	0.9383	0.9383	0.9383
Lisa: $8\clubsuit, 5\heartsuit$	0.9955	0.9955	0.9955
AIE = 0.0000			
AIVAT baseline = BrC			
Actual sequence = BrC			
Actual net gain = -16.0000			
Baseline net gain = -16.0000			
AIVAT difference = 0.0000			

Table 9.8: River analysis

The river card was a lucky card for Lisa and she knew that her hand was strong and correctly raised Tom's initial bet. As Tom became unsure of whether Lisa actually had a strong hand or was just bluffing, he decided to call. This betting sequence matched the AIVAT baseline and therefore resulted in an AIVAT difference of 0.

By looking back at this played hand with the AIVAT scores for each stage, we can see that Tom actually played reasonably well considering the cards that were dealt. The pre-flop AIVAT difference was the only negative score that was given, and this was the result of Tom's decision to check instead of raising after Lisa's big blind call. It is difficult to say what the outcome would be if Tom instead decided to raise. Perhaps Lisa would then choose to fold her weak hand and the round would be over. The choice of checking instead of placing a bet can often be justified by a strategy of not giving away information about a hand, pretending that your hand is weaker than it is, or not wanting to scare the opponent into folding when there is a potential for bigger winnings. However, the fact that the AIVAT baseline resulted in a larger pot investment than the actual betting sequence led to a negative score for this play.

The flop and turn plays both resulted in positive scores. Tom had the strongest hand and he was able to increase pot investment compared to the AIVAT baseline. For the player with the strongest hand it is best to attempt to increase the pot size, as a larger pot means larger winnings.

The final and most interesting part of this analysis is the river stage. Up until the river, Tom had the strongest hand and both players had invested heavily in the pot. When Lisa gets the lucky straight on the last card, Tom's strong hand is no longer the strongest. Tom's decision to bet-call is the same as the one given by the bet-for-value tactic shown in the AIVAT baseline. Even though Tom loses the hand, the AIVAT score given on the river is still not negative. This reflects the disregard for luck, and Tom is therefore not punished for being unlucky.

If we look at the whole hand in total, Tom would get a total score of:

$$-0.6840 + 2.7520 + 1.6364 + 0 = +3.7044$$

This means that he played the hand well. The fact that he finally lost is irrelevant. If the score from each stage of the hand is to be stored in a case base, then the decisions made on the flop, turn and river would be considered good decisions.

This analysis tool has the potential of providing a more rational evaluation of decisions that are not as influenced by luck. If we were to use the final outcome of a played hand as the basis for our decision making, the example hand would give a score of -16, which is how much Tom lost. Instead we now have a score of +3.7044 in total (or 0 if we look at the river stage alone). This score is more reasonable considering how Tom played. A better decision evaluation score can assist in providing a stronger case-base for our CBR approach, where evaluations are not influenced by luck and good decisions are reused.

9.4.3 Equity Evaluator

The Equity Evaluator (EE) is our own approach to *perfect information hindsight analysis*. The idea is to investigate how one decision changes the player's equity of the pot related to how big investment the action represents. In other words, it determines if the result of performing the action is more valuable than the price of the action. The quality evaluation of the Equity Evaluator can be defined as:

$$DecisionQuality = PEAA - PEBA - Investment$$

Investment is the amount of money committed to the pot by the player's action. *PEBA* (Pot Equity Before Action) is computed by multiplying the current all-in equity, or AIE (see section 5.2), with the current pot size. Note that the current pot size excludes the most recent raise in the same hand, if any. We assume no ownership of the additional money contributed by this raise. Computing *Pot Equity*

After Action (PEAA) is performed in a similar fashion. However, additional care is taken if the opponent is responding to the player's chosen action. If the response is a fold, the player assumes complete ownership of the pot. Therefore, PEAA will be the value of the pot (which includes the player's investment). If the response is a call, the player is assumed to have partial ownership of the money contributed by this call. In this case, the amount required for the call is added to the pot (which the player has its percentage-based ownership of). In the case of the opponent responding with a raise, we still add the amount required for a call to the pot as before, but the evaluation score is then adjusted with a special offset to represent how the opponent's raise affects the player's ownership of the pot. This factor is defined as:

$$\text{Score Offset} = 2 \cdot (\text{AIE} - 0.5) \cdot \text{Raise}$$

Raise is the opponent's raise, excluding the amount required to make a call. Note that the value of *score offset* ranges from a positive value to a negative value depending on the *all-in equity*. The reasoning behind this offset is to award actions that make the opponent put more money in the pot when the player is winning and penalize actions where a winning opponent is undermining the player's ownership of the pot by raising.

If the decision made by UpperCase is to fold, the equity evaluation score will be set to zero for this decision. This means that the fold decision will often end up with the highest evaluation score in situations where folding is reasonable. This is because the other decisions (call, bet/raise) lead to investing more money in a pot with low equity, which results in a negative equity evaluation score.

9.4.4 Comparison By Example

We now present an example of how the three different decision quality evaluators work. We use the same motivating example as the one first introduced in section 5.1 and used again in the presentation of AIVAT.

One thing to observe is the difference in evaluation granularity between the different evaluators. The Profit Evaluator does not evaluate the different actions within the same hand individually. Since Tom ends up losing 16\$, all actions receive a -16.0 evaluation. AIVAT does not evaluate different actions within the same stage differently. This is because the comparison of the player's performance and the AIVAT baseline has to include the whole stage. We now look at Tom's actions to see how AIVAT and EE works differently.

Again, consider the two players, Tom and Lisa, in a regular game of no-limit Texas Hold'em, where we are evaluating Tom's actions. Lisa is the dealer, the small blind is 1\$ and the big blind 2\$. Tom is given the hole cards $A\heartsuit, 10\heartsuit$ (ace of diamonds and 10 of diamonds), while Lisa is given $8\clubsuit, 5\heartsuit$ (8 of clubs and five of hearts).

Pre-flop:

Tom’s first action is checking after Lisa called the big blind. EE gives a neutral evaluation because Tom’s actions make no change to his equity of the pot. AIVAT, on the other hand, gives a negative evaluation because the baseline received better results with calling instead of checking.

Flop: $A\spadesuit, 4\heartsuit, 7\spadesuit$

On the flop, Tom bets his strong hand, gets re-raised by a bluffing Lisa, re-raises her again and is answered by a call. Since he has a stronger hand, the increase of pot size is beneficial. Both AIVAT and EE assign high scores to the actions of this stage. EE does so because the actions increased Tom’s pot equity more than the cost of performing them. AIVAT gives a high score because Tom did better against Lisa in this stage than the DIVAT-player was able to against the DIVAT-opponent during simulation.

Turn: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit$

On the turn, Tom checks first, then re-raises Lisa’s bet, which she then responds to with another call. Again, the increase of pot size is desirable, and therefore leads to good scores. However, as explained above, AIVAT is not able to separate the two actions. This results in both actions receiving equal scores. EE recognizes that the main reason for the increase of Tom’s ownership of the pot is caused by his re-raise. This action therefore receives a significantly higher score.

River: $A\spadesuit, 4\heartsuit, 7\spadesuit, 10\clubsuit, 6\diamondsuit$

In the river stage, Tom calls Lisa’s bet. At this moment, Lisa has the best hand and will win a showdown. Therefore the call is a direct loss of money. This is reflected in the negative score given by EE. AIVAT does not punish this action with a negative score because it is identical to the baseline.

Stage	Action	Profit Evaluator	AIVAT Evaluator	Equity Evaluator
Preflop	Check	-16.0	-0.69	0.0
Flop	Raise (2)	-16.0	2.38	2.13
	Raise (4)	-16.0	2.38	2.38
Turn	Check	-16.0	1.64	1.49
	Raise (4)	-16.0	1.64	3.27
River	Raise (2)	-16.0	0.0	-2.0
	Call (2)	-16.0	0.0	-2.0

Table 9.9: Output from PE, AIVAT and EE in the motivational example.

Table 9.9 shows the results after evaluating the motivational example for all three evaluators. As we can see, the difference in evaluation granularity results in the profit evaluator giving the same score to all decisions within the same hand. AIVAT separates between each stage of the hand, while the equity evaluator gives a score to each separate decision. Overall, the AIVAT and equity evaluators are able to capture more information about the played hand and provide a reasonable evaluation

of the decisions. Seeing as Tom actually played fairly well with his strong hand, this not reflected in the profit evaluation score of -16.0. This example suggests that the AIVAT and equity evaluation techniques can be helpful when evaluating poker decisions for reuse in a CBR system. We have tested the performance of the difference decision evaluators and the results are presented in section 11.1 and discussed in section 12.1.

9.5 EQ

EQ is the module of UpperCase responsible for producing a static equilibrium strategy. The objective of EQ is to provide a strategy that performs relatively well against most opponents. This will be the baseline strategy of the system, which is dominating decision-making when there is none or little knowledge about the current opponent. EQ could be employed as an independent poker agent.

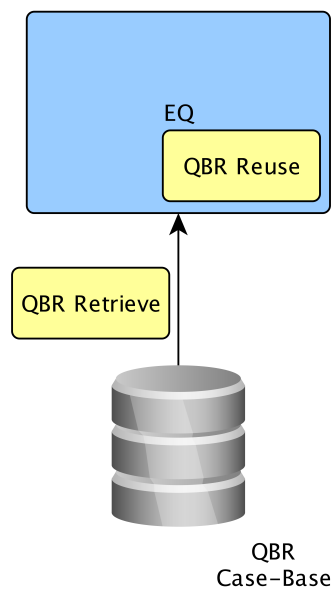


Figure 9.9: EQ

EQ is simply one single QBR-module (see figure 9.9). As explained in section 9.3.5, the result of the reuse process is an aggregation of solutions where the solutions are grouped after their action-type. This is the output of EQ, which will be processed by EX.

Training is also performed in the same manner as explained in section 9.3.6. However, since we want to produce a solid strategy it is important that we select training-opponents with care. A set of well-performing poker agents could be selected to produce a general baseline strategy, or we could train specifically against a single

opponent to create a more specialized baseline strategy. In most cases we want a general baseline strategy, and let EX adapt this to the specific opponent.

9.6 EX

EX is the module of UpperCase responsible for adjusting the equilibrium strategy produced by EQ in order to exploit the current opponent. The motivation behind doing so is that increased performance can be achieved by exploiting weaknesses in the opponent's strategy and eliminating weaknesses in UpperCase's baseline strategy.

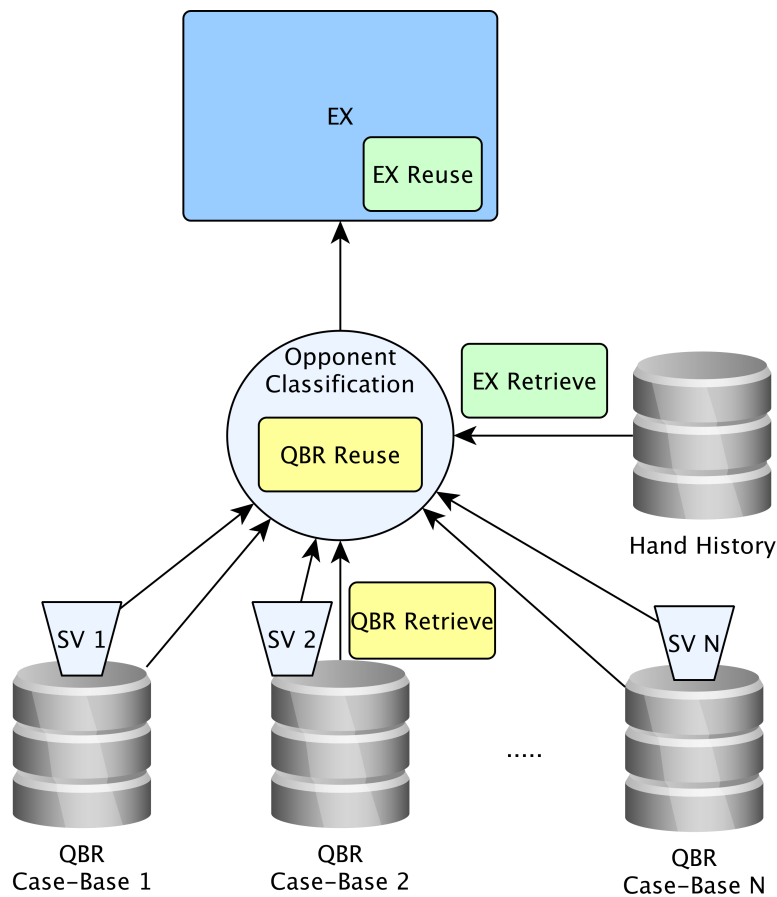


Figure 9.10: EX

Adaptation is performed by classifying the opponent among several pre-defined *opponent-types* (explained in section 9.6.1) and adjusting the baseline strategy accordingly. The classification is based on available knowledge about the opponent, which is extended during play (explained in section 9.6.3.1). EX employs several

QBR-modules, as illustrated in figure 9.10, each representing a specific opponent-type. When EX is requested to make a decision, one of these QBR-modules will be activated. The input from EQ and the decision produced by this QBR-module are both used to make the final decision.

9.6.1 Known Opponent-Types

These modules have been trained to perform well against such opponents. When the current opponent has been classified as one of the opponent-types, the associated QBR-module is activated and used to make a decision in the current game-state. EX is trained against a set of poker agents. For each agent provided, a QBR-module is created and trained against this agent as described in section 9.3.6. When training is completed, all the poker agents will represent *known opponent-types* of EX. Each QBR-module has developed strategies for playing well against strategies similar to the strategy of poker agent it was trained against. The strategies of the provided set of opponents determine which strategies EX is able to recognize and adapt to. These will be the known opponent-types of EX. We have experimented with different sets of opponents, but by default we use the following four poker agents, inspired by different playing-styles described by the professional poker player Phil Hellmuth in [16]:

- **Loose-Aggressive**
Players with a loose-aggressive strategy play a large number of hands, but in an aggressive way. The high number of hands played means they will be playing with weaker holdings on average. They may also often bluff. A player with this strategy can be referred to as a *jackal*.
- **Tight-Aggressive**
Players with a tight-aggressive strategy play a small number of hands, but in an aggressive way. Because these players play fewer hands, they will often have stronger holdings on average when making plays. A player with this strategy can be referred to as a *lion*.
- **Loose-Passive**
Players with a loose-passive strategy calls frequently. Consequently, bluffing this type of player is rarely a good idea. A player with this strategy can be referred to as an *elephant* or a *calling station*.
- **Tight-Passive**
Players with a tight-passive strategy will play a small number of hands, but in a passive way. A player with this strategy can be referred to as a *mouse*.

We have implemented these playing-styles as poker agents using UniPoker. These agents use the concept of hand strength (see 2.1) together with different fixed thresholds to decide which actions to perform. By adjusting the threshold-values we can adjust how aggressively and frequently these agents play.

9.6.2 Characterization of Poker Strategies

EX characterizes different poker strategies using a vector of poker metrics called a SV (Strategy Vector). The metrics are evaluated using regular hand history or *imperfect information hand history* (IIHH). It is important to notice that this is different from the *perfect information hand history* (PIHH) generated in the training phase. With IIHH, we do not know the opponent's cards unless they were displayed in a showdown. Consequently, the decision quality evaluation tools described in section 9.4 are not applicable for evaluating IIHH. However, the poker metrics contained in a SV can be computed from IIHH. The format of a SV is presented in figure 9.11.

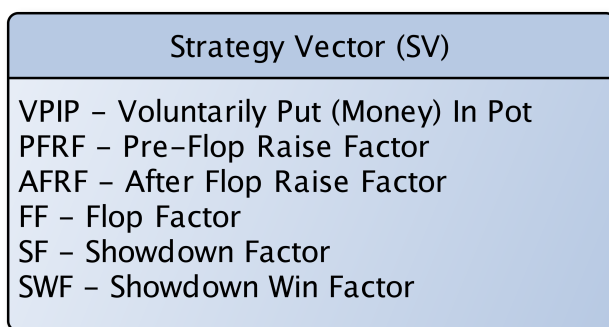


Figure 9.11: A Strategy Vector (SV).

Choosing the right metrics to put in the strategy-vector is an interesting task. SVs are used to recognize and differentiate poker strategies, so the set of metrics should capture important aspects of different poker strategies as well as being usable even after relatively few hands have been played. Also, the set of metrics should cover as many situations as possible. E.g., if the set of metrics only describe pre-flop situations, the classification of post-flop strategy will probably be weak.

We have chosen the following metrics in our SV:

- **Voluntarily Put Money In Pot (VPIP)**

VPIP characterizes how often a player performs actions that increases the size of the pot, e.g. calls or bets/raises. It is defined as:

$$VPIP = \frac{\# \text{ Actions with Cost} > 0}{\# \text{ Actions}}$$

- **Pre-Flop Aggression Factor (PFAF)**

PFAF characterizes how often a player plays aggressive in the pre-flop stage. It is defined as:

$$PFRF = \frac{\# \text{ Aggressive Actions on Preflop}}{\# \text{ Actions on Preflop}}$$

- **After Flop Aggression Factor (AFAF)**

AFAF characterizes how often a player plays aggressive on the flop and later stages. It is defined as:

$$AFRF = \frac{\# \text{ Aggressive Actions Post-Flop}}{\# \text{ Actions Post-Flop}}$$

- **Flop Factor (FF)**

FF characterizes how often a player participates in the flop stage. It is defined as:

$$FF = \frac{\# \text{ Hands reaching Flop}}{\# \text{ Hands}}$$

- **Showdown Factor (SF)**

SF characterizes how often a player participates in the showdown, given that he participated in the flop. It is defined as:

$$SF = \frac{\# \text{ Hands reaching Showdown}}{\# \text{ Hands reaching Flop}}$$

- **Showdown Win Factor (SWF)**

SWF characterizes how often a player wins the showdown. It is defined as:

$$SWF = \frac{\# \text{ Showdowns}}{\# \text{ Won Showdowns}}$$

This set of metrics constitutes in our opinion a balanced SV. VPIP, PFAF and FF cover pre-flop strategy. They give a good impression of the distribution of hands that the opponent takes into the flop stage as well as how aggressively he plays those hands. VPIP, AFAF, FF, SF are covering the later stages of the game. On the flop, turn and river stages it is important to describe how probable it is for the opponent to stay in the hand. This is described by VPIP and the difference between FF and SF. In addition, we want to express how aggressively the opponent plays in these stages using AFAF. In the last stage of the game we are interested in knowing if the opponent can be expected to have above or below average hand-strength at showdown. SF and SWF cover these concerns.

As mentioned earlier, one QBR case-base is created for each opponent-type in the

training phase. In addition, several test-games are executed against each opponent to produce hand-history from which a SV can be computed. These SVs are then assigned to the appropriate QBR case-base. One important aspect of the generation of these SVs is that some metrics may be affected by the player's actions. E.g., if the player would always fold, then FF would be close to zero. On the other hand, if the player would always call, then the FF would probably be close to 100% even if the opponent is identical in both cases. Therefore, this training is performed using the baseline strategy produced by EQ in order to achieve the best approximation of real play.

9.6.3 EX-CBR

As explained in the sections above, EX has a set of strategies to counter known opponent-types and a way of characterizing different poker strategies. In this section we present EX-CBR, a CBR-system that reuses the set of strategies in order to perform adaptation so that the current opponent can be exploited. A case in this CBR-system is the combination of a known opponent-type and the SV (see figure 9.12) describing this opponent-type. The opponent-type is associated with a QBR-module able to produce robust strategies against opponents similar to the opponent-type. The index-features of the case is the metrics of the SV and the solution-feature is the opponent-type.

EX Case
VIP - Voluntarily Put (Money) In Pot PFRF - Pre-Flop Raise Factor AFRF - After Flop Raise Factor FF - Flop Factor SF - Showdown Factor SWF - Showdown Win Factor
Opponent-Type

Figure 9.12: An EX case.

9.6.3.1 Retrieve - Opponent Classification

During play, EX will compute a SV that characterizes the current opponent's strategy based on available knowledge about him. The source of knowledge is hand history from this opponent. This hand history is continuously appended to when UpperCase plays against this specific opponent. It can also be populated from other sources, e.g. hand history generated by online poker clients.

The retrieve process proceeds with calculating the similarity between the current opponent and known opponent-types by comparing the computed SV against the SVs of all available cases. We use individual weights for the metrics of the SV and a weighted Euclidean distance as our similarity-function. The similarity-function is defined as:

$$s = \frac{\sqrt{\sum_{i=1}^N w_i (\vec{SV}_{(o,i)} - \vec{SV}_{(qbr,i)})^2}}{\sum_{i=1}^N w_i}$$

A different set of weights are used for each stage of a hand according to how important the given metric is in that stage. Greater weight causes the differences of a metric to have a greater impact on the resulting similarity, which means that this metric becomes more important. Our weights are shown in table 9.10.

Metric	Pre-flop	Flop	Turn	River
VPIP	1.0	0.75	0.75	0.75
PFAF	1.0	0.5	0.5	0.5
AFAF	0.5	1.0	1.0	1.0
FF	1.0	1.0	1.0	0.5
SF	0.5	1.0	1.0	1.0
SWF	0.5	0.75	0.75	1.0

Table 9.10: Weights for the different metrics of SVs in different stages.

The result of the retrieve process is the complete collection of cases with computed similarities to the current case.

9.6.3.2 Reuse - Opponent Adaptation

The reuse process receives the collection of cases and order them after their similarity. The process then identifies if at least one of the cases is reusable. The QBR-module associated with the opponent-type of the most similar case is activated and requested to produce an aggregated QBR-solution to the current game-state. If no matching QBR-case can be found in this QBR-module, the request is repeated for the next most similar case until one is found or all opponent-types been checked. If no QBR-module is able to produce a (QBR) solution, the EX reuse process ends and the output is the baseline strategy produced by EQ.

If a QBR-solution can be provided, EX calculates the utility of this solution based on how much knowledge is available about the current opponent. There will be a varying amount of knowledge about different opponents. When playing against an unknown opponent, UpperCase will have observed no actions, but after hundred thousands of played hands there will be many actions to consider. This determines the utility of the case. More knowledge about the opponent correlates with higher utility. We use the following formula to calculate utility:

$$u = \max\left\{0, \frac{a - T_a}{a}\right\}$$

a is the number of observed actions. T_a (Action Threshold) is a constant which determines the minimum required amount of observed actions before the case becomes usable. In the current implementation we use $T_a = 500$.

The similarity of the selected case is used later in the reuse process, but normalized after the following function:

$$\|s\| = 1 - \frac{\min\{T_s, s\}}{T_s}$$

At the moment adaptation occurs, we have two aggregated QBR solutions. One from EQ (EQ QBR) and one from the QBR module associated with the selected case (EX QBR). We calculate a weighted average for each action-type to produce a weighted EX solution. This process is illustrated in figure 9.13



Figure 9.13: EX solution

As we can see in the figure, the action-type with the highest average quality in the

EQ QBR solution is *very aggressive*. However, the selected action in the solution provided by EX QBR is *no play* (fold). With a weight of 0.3 applied to the EQ QBR solution and a weight of 0.7 applied to the EX QBR solution, the resulting weighted EX solution selects *passive* (check/call). This example illustrates how a baseline strategy can be adapted towards a specific opponent-type.

The weight (w) used in the adaptation is computed using the following formula:

$$w = \|s\| \cdot u$$

Since both $\|s\|$ and u ranges from 0 to 1, the resulting w is also a number between 0 and 1.

9.6.4 Adaptation Rationale

In this section we demonstrate the effects of the adaptation performed by EX, using an example illustrated in figures 9.14 and 9.15. In these figures, different strategies are shown as circles in a two-dimensional vector space. The actual strategy of the current opponent is represented by the red circle. Strategies of EX's known opponents is represented by green circles, while the baseline strategy is represented by a blue circle. The adapted strategy is represented with a black circle.

In both illustrations, EX has been able to classify the opponent as one of its known opponent-types and this match is shown with a line between the actual opponent strategy and the matched strategy. Utility is represented by the thickness of this line and similarity by the length of this line. A thicker line means higher utility and a shorter line means higher similarity. The resulting strategy will be somewhere along the line between the baseline strategy, and the matched strategy, shown with a dotted line in the illustrations.

In this example, the strategy is adapted towards *Strategy A*. This strategy is selected because it is the most similar strategy to the opponent's strategy. In figure 9.14 the utility is low due to weak knowledge about the opponent. Even though a relatively similar strategy has been matched, less adaptation is performed due to low utility.

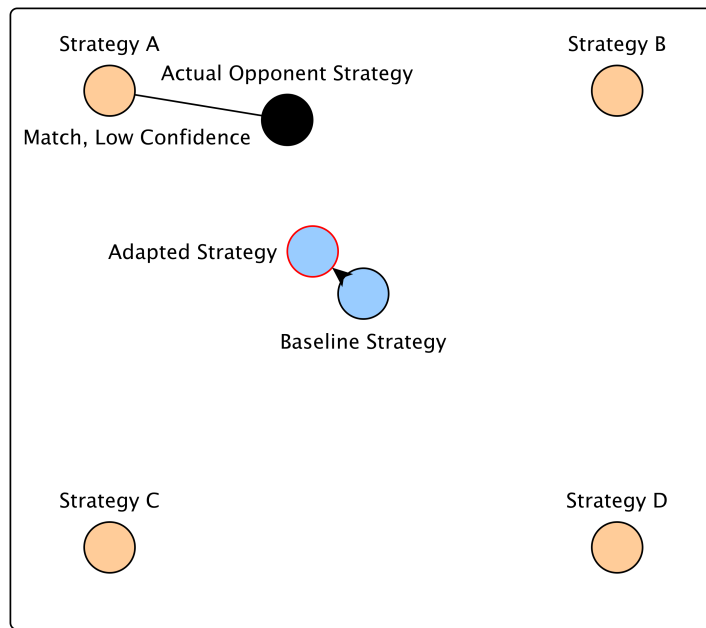


Figure 9.14: Lesser Adaptation

In figure 9.15, the strategy is again adapted towards *Strategy A*. However, this time the utility is greater since more knowledge about the opponent has been acquired. As a result, the adapted strategy is now closer to *Strategy A*, which also makes it closer to the actual opponent strategy. The result is a strategy that is better in counteracting the opponent.

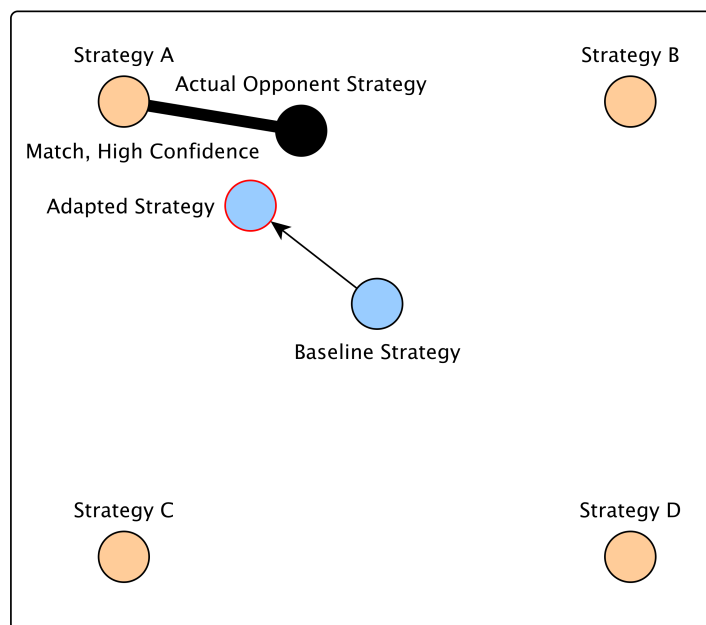


Figure 9.15: Greater Adaptation

In figure 9.15, the strategy is adapted towards the same type, but this time with higher confidence. This can be the same game as in figure 9.14, but at a later stage where more knowledge about the opponent increases CC. As a result, the adapted strategy is now moved further against 'Strategy A', making it closer to the actual strategy. The resulting strategy is now a more equal mix of 'Strategy A' and the baseline strategy.

9.7 System Summary

The UpperCase poker agent is a fairly complex agent that applies different CBR-systems to make decisions in poker. The two main components; EQ and EX, both use the underlying QBR-module for decision making. The concept of having a baseline strategy in addition to an adaptive strategy is illustrated in figure 9.16 which shows a detailed view of the system (same figure as in section 9.1).

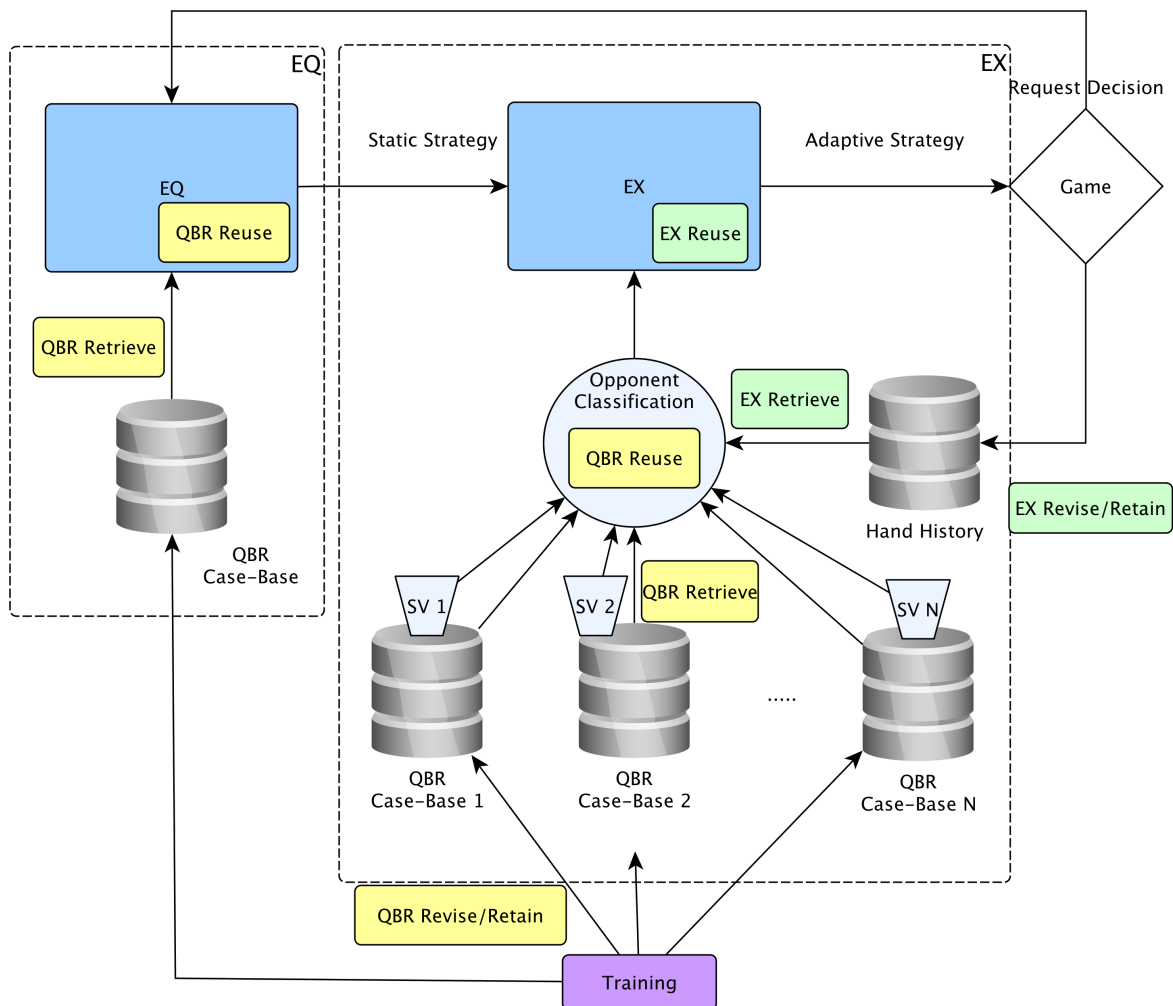


Figure 9.16: System overview of the UpperCase poker agent.

As we can see from this illustration, the EQ-system uses one single QBR case-base to create a static baseline strategy. UpperCase is not an expert-imitation agent that reuses the decisions made by other poker agents to imitate their behavior. Instead it creates its own case-base by training against appropriate opponents. Perfect information hand history is used during this training phase. This provides the possibility of applying *perfect information hindsight analysis* to assess reasonable evaluations of the decisions made. A decision quality evaluation is applied to every decision made during training. Each case is given a quality score before the case is retained in the case-base. In the reuse process, the decision with the best average quality is reused. This approach provides UpperCase with a sufficiently good baseline strategy that is used until the opponent's playing style is recognized.

The decision made by the EQ-system is forwarded to the EX-system. This system uses multiple QBR-modules in its adaptation process. Each QBR-module represents one opponent-type and includes a strategy for exploiting this specific opponent. During play, imperfect information hand history is stored and analyzed by the EX-system. The goal of the EX-system is to correctly classify the opponent's playing style and shift the strategy provided by EQ over to a more exploitive strategy tailored to the opponent. This provides UpperCase with an adaptive capability.

The output of the UpperCase system is a weighted average of the EQ-solution and the EX-solution. If the EX-system is sufficiently confident in its opponent classification, the EX-solution will be given a higher weight. This results in a strategy that dynamically adapts during the course of a game. This can increase the performance of the UpperCase agent by exploiting its opponents, and also prevent UpperCase from being easily exploited by others considering that it does not completely rely on a static strategy.

UpperCase	
Case-base structure	Multiple case-bases and case-base structures.
Training	Uses perfect information hindsight analysis for decision quality evaluation during training. Also stores cases during play to improve opponent classification.
Reuse policy	Best average quality.
Adaptive capabilities	UpperCase classifies opponents among known opponent-types in order to perform adaptation. Different opponents are recognized individually.

Table 9.11: The UpperCase poker agent.

The combination of a static equilibrium strategy provided by the EQ-system and an exploitive strategy provided by the EX-system, separates UpperCase from other existing CBR-based poker agents. Table 9.11 presents an overview of the UpperCase poker agent using the framework introduced in section 4.2. We can see that solutions in the case-base are represented by the expected quality of the actions. Most other CBR poker agents apply a reuse policy based on outcome or on the decisions made by other agents that they attempt to imitate. The goal of using

decision quality evaluation is to decrease the effect that luck has on the evaluation of decisions in poker. A more reasonable evaluation of decisions could result in a stronger poker agent. The use of perfect information hindsight analysis is an interesting approach. However, perfect information requires a poker simulator that can provide this information which is not common in many poker clients.

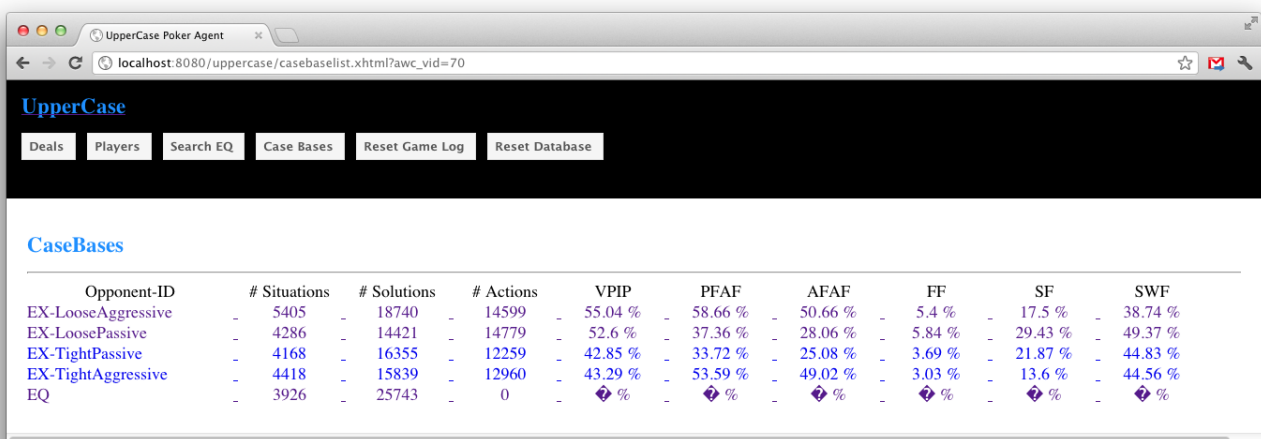
Chapter 10

UpperCase Web-Interface

We have developed a web-interface that allows us to monitor UpperCase. By looking into the different case-bases of the system, we can achieve an understanding of the different decisions made by the system. In this section, the most important parts of this tool are presented.

10.1 Case-Base List

Figure 10.1 displays an example of the *case-base list*, which shows a list of the QBR case-bases in UpperCase. For each case-base in the list, the metrics of the EX strategy vector (SV) characterizing the case-base is shown. The number of actions used to compute these metrics are listed as well. We can also see the number of situations and solutions contained in each case-base. The figure shows the case-bases after a small amount of training of has been performed. We can see that the SV metrics describing the EQ case-base are undefined. This is because EX does not compute a SV for the EQ case-base.



The screenshot shows a web browser window titled "UpperCase Poker Agent" with the URL "localhost:8080/uppercase/casebaselist.xhtml?awc_vid=70". The page has a navigation bar with buttons for "Deals", "Players", "Search EQ", "Case Bases", "Reset Game Log", and "Reset Database". Below the navigation bar, the "CaseBases" section contains a table with the following data:

Opponent-ID	# Situations	# Solutions	# Actions	VPIP	PFAF	AFAF	FF	SF	SWF
EX-LooseAggressive	5405	18740	14599	55.04 %	58.66 %	50.66 %	5.4 %	17.5 %	38.74 %
EX-LoosePassive	4286	14421	14779	52.6 %	37.36 %	28.06 %	5.84 %	29.43 %	49.37 %
EX-TightPassive	4168	16355	12259	42.85 %	33.72 %	25.08 %	3.69 %	21.87 %	44.83 %
EX-TightAggressive	4418	15839	12960	43.29 %	53.59 %	49.02 %	3.03 %	13.6 %	44.56 %
EQ	3926	25743	0	◆ %	◆ %	◆ %	◆ %	◆ %	◆ %

Figure 10.1: The case-base list.

10.2 Case-Base View

Figure 10.2 shows the *case-base view*, which displays relevant information about one single QBR case-base. In this view it is possible to search after situations. A QBR situation-key is submitted in the search-form, a QBR retrieve process is executed and the QBR aggregated solution is displayed. By using this function it is possible to investigate the decisions this QBR case-base will produce in different situations. In the figure we searched for "/AA" which is the best possible starting hand a player can have. As we can see from the results, a *very aggressive* action-type has been selected in this situation. All the retrieved cases are shown at the bottom of the page.

UpperCase

Deals | Players | Search EQ | Case Bases | Reset Game Log | Reset Database

QBR Case-Base: EQ

Situation-Count 3926
Solution-Count 25743

/AA

Solution-Count 26
Selected Action VERY_AGGRESSIVE
Expected Quality 8.514999999999999

Aggregated Solution

Action-type	Average Quality	Case-Count
VERY_AGGRESSIVE	8.514999999999999	4
AGGRESSIVE	4.645714285714285	7
PASSIVE	2.8440000000000003	10
FOLD	0.0	5

Retrieved Cases

Action-Type	Quality
PASSIVE	4.0
AGGRESSIVE	4.38
PASSIVE	2.7800000000000002
PASSIVE	2.7800000000000002
PASSIVE	4.0
AGGRESSIVE	2.92
PASSIVE	0.8199999999999998
PASSIVE	2.7800000000000002
FOLD	0.0
VERY_AGGRESSIVE	9.18

Figure 10.2: The case-base view.

10.3 Player List

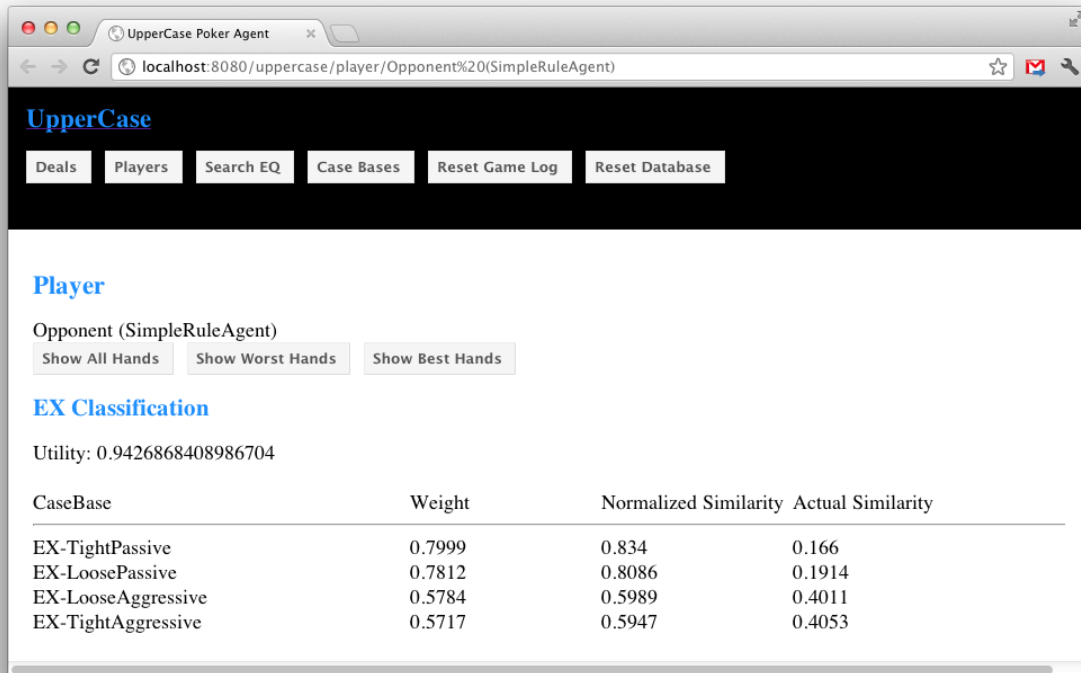
Figure 10.3 shows the *player list*, which displays a list of players that UpperCase has collected hand history from during play or training. For each player, the SV metrics, number of hands, number of actions and the total winnings are displayed. The list shown in the figure is relatively sparse, containing only the opponents UpperCase faced in the training phase (IDs prefixed with "UpperCase-Training-Opponent"), UpperCase itself (the remaining entries prefixed with "UpperCase") as well as one single opponent faced in real play ("Opponent (SimpleRuleAgent)").

Player ID	# Hands	# Actions	BB/100	VPIP	PFRF	AFRF	FF	SF	SWF	Total Winnings
Opponent (SimpleRuleAgent)	4999	8724	455.98	46.95 %	23.87 %	33.39 %	19.66 %	16.08 %	62.44 %	2279450.0
UpperCase (UpperCasePokerPlayer)	4999	8396	-455.98	50.79 %	59 %	49.45 %	19.66 %	16.08 %	40.55 %	-2279450.0
UpperCase-Training (UpperCasePokerPlayer)	3611	7364	-554.5	56.8 %	49.22 %	48.23 %	7.34 %	30.6 %	41.36 %	-40046.0
UpperCase-Training-Logging (UpperCasePokerPlayer)	30972	47638	-145.06	44.98 %	51.44 %	36.39 %	4.11 %	18.54 %	63.15 %	-89856.0
UpperCase-Training-Opponent (LooseAggressive)	10000	14599	155.86	55.04 %	58.66 %	50.66 %	5.4 %	17.5 %	38.74 %	31173.0
UpperCase-Training-Opponent (LoosePassive)	7241	14779	332.52	52.6 %	37.36 %	28.06 %	5.84 %	29.43 %	49.37 %	48155.0
UpperCase-Training-Opponent (RaiseAgent)	0	0	◆	◆ %	◆ %	◆ %	◆ %	◆ %	◆ %	0.0
UpperCase-Training-Opponent (TightAggressive)	10000	12960	148.26	43.29 %	53.59 %	49.02 %	3.03 %	13.6 %	44.56 %	29652.0
UpperCase-Training-Opponent (TightPassive)	7342	12259	142.48	42.85 %	33.72 %	25.08 %	3.69 %	21.87 %	44.83 %	20922.0

Figure 10.3: The player list.

10.4 Player View

Figure 10.4 shows the *player view*, which displays the EX opponent classification ordered by similarity. In the figure we see that the opponent has been classified as most similar to the "TightPassive" EX opponent type. We can proceed to see all hands, the most profitable hands and the least profitable hands from UpperCase's point of view against this player.



UpperCase

Deals Players Search EQ Case Bases Reset Game Log Reset Database

Player

Opponent (SimpleRuleAgent)

Show All Hands Show Worst Hands Show Best Hands

EX Classification

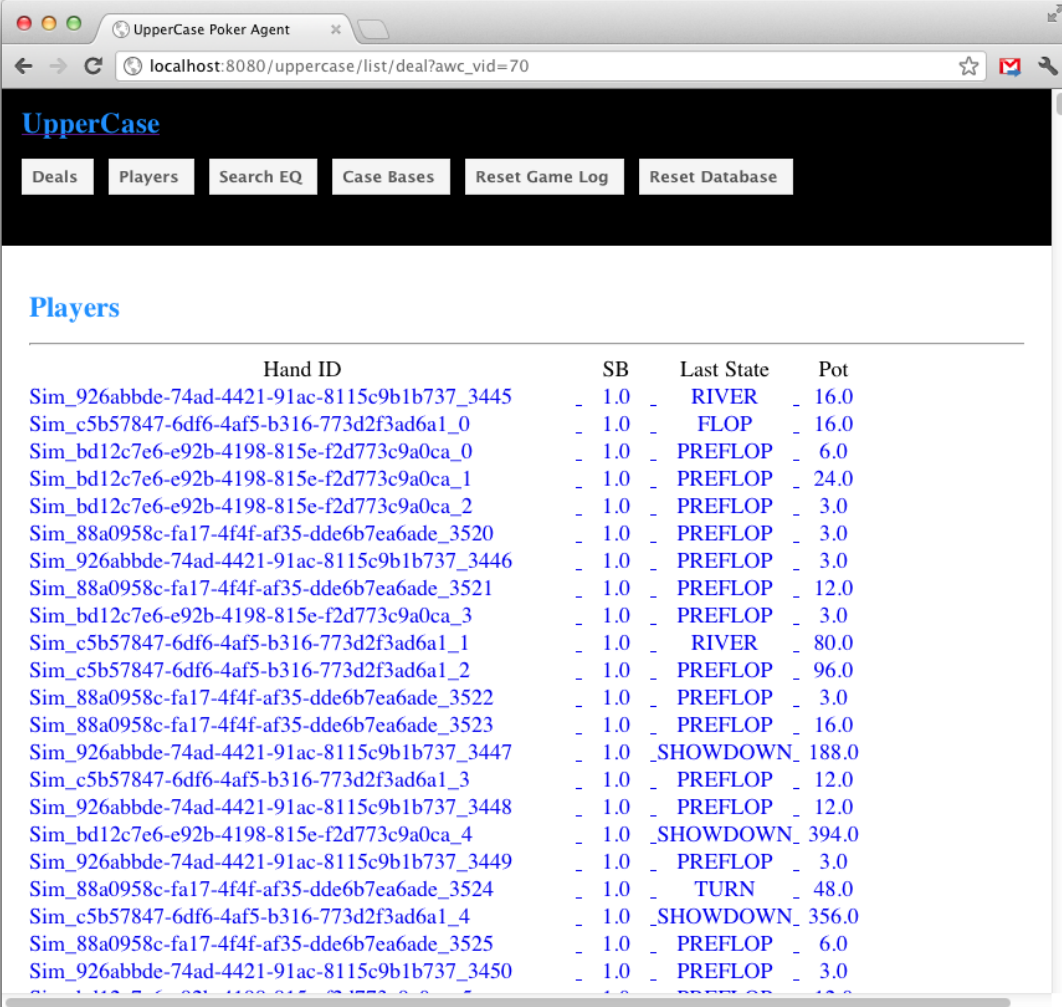
Utility: 0.9426868408986704

CaseBase	Weight	Normalized Similarity	Actual Similarity
EX-TightPassive	0.7999	0.834	0.166
EX-LoosePassive	0.7812	0.8086	0.1914
EX-LooseAggressive	0.5784	0.5989	0.4011
EX-TightAggressive	0.5717	0.5947	0.4053

Figure 10.4: The player view.

10.5 Hand List

Figure 10.5 shows the *hand list*, which displays as list of hands in which a chosen player has participated. The list shows the hand-id, size of small-blind, the last stage, the final pot size as well as the player's profit in the hand.



The screenshot shows a web browser window titled "UpperCase Poker Agent" with the URL "localhost:8080/uppercase/list/deal?awc_vid=70". The page features a navigation menu with buttons for "Deals", "Players", "Search EQ", "Case Bases", "Reset Game Log", and "Reset Database". The "Players" button is selected, and the page displays a table titled "Players" with the following columns: "Hand ID", "SB", "Last State", and "Pot". The table contains 20 rows of data, each representing a hand with its ID, small blind size, final state, and pot size.

Hand ID	SB	Last State	Pot
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3445	1.0	RIVER	16.0
Sim_c5b57847-6df6-4af5-b316-773d2f3ad6a1_0	1.0	FLOP	16.0
Sim_bd12c7e6-e92b-4198-815e-f2d773c9a0ca_0	1.0	PREFLOP	6.0
Sim_bd12c7e6-e92b-4198-815e-f2d773c9a0ca_1	1.0	PREFLOP	24.0
Sim_bd12c7e6-e92b-4198-815e-f2d773c9a0ca_2	1.0	PREFLOP	3.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3520	1.0	PREFLOP	3.0
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3446	1.0	PREFLOP	3.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3521	1.0	PREFLOP	12.0
Sim_bd12c7e6-e92b-4198-815e-f2d773c9a0ca_3	1.0	PREFLOP	3.0
Sim_c5b57847-6df6-4af5-b316-773d2f3ad6a1_1	1.0	RIVER	80.0
Sim_c5b57847-6df6-4af5-b316-773d2f3ad6a1_2	1.0	PREFLOP	96.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3522	1.0	PREFLOP	3.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3523	1.0	PREFLOP	16.0
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3447	1.0	SHOWDOWN	188.0
Sim_c5b57847-6df6-4af5-b316-773d2f3ad6a1_3	1.0	PREFLOP	12.0
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3448	1.0	PREFLOP	12.0
Sim_bd12c7e6-e92b-4198-815e-f2d773c9a0ca_4	1.0	SHOWDOWN	394.0
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3449	1.0	PREFLOP	3.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3524	1.0	TURN	48.0
Sim_c5b57847-6df6-4af5-b316-773d2f3ad6a1_4	1.0	SHOWDOWN	356.0
Sim_88a0958c-fa17-4f4f-af35-dde6b7ea6ade_3525	1.0	PREFLOP	6.0
Sim_926abbde-74ad-4421-91ac-8115c9b1b737_3450	1.0	PREFLOP	3.0

Figure 10.5: The hand list.

10.6 Hand View

Figure 10.6 shows the *hand view*, which displays information about a hand in which a chosen player has participated. Key information such as hand-id, final pot size, small-blind, last stage reached and final community cards are displayed. The view also contains a list of actions in the hand, showing the QBR situation-key, action-type, the pot size of the before the action, the amount required for a check/call and the investment caused by the action. In the bottom of the page, the actual output of the UniPoker simulation is displayed. By reading the output, we can identify the hole cards of each player as well as the winner of the hand.

UpperCase

Deals | Players | Search EQ | Case Bases | Reset Game Log | Reset Database

Hand

ID: Sim_d8a9013d-6916-4baf-bd5d-433f17709a40_9999
 Pot: 16250.0
 LastStage: SHOWDOWN
 TableCards: 7h,Ad,Qd,4d,9s

Actions

PlayerId	Pot	To Call	Action	ActionType	Situation-key
Opponent (SimpleRuleAgent)	75.0	50.0	50.0	PASSIVE	/Q7
UpperCase (UpperCasePokerPlayer)	125.0	0.0	200.0	AGGRESSIVE	P/Q7
Opponent (SimpleRuleAgent)	325.0	200.0	200.0	PASSIVE	Pa/Q7
UpperCase (UpperCasePokerPlayer)	525.0	0.0	600.0	AGGRESSIVE	PaP/23
Opponent (SimpleRuleAgent)	1125.0	600.0	2400.0	AGGRESSIVE	PaPla/23
UpperCase (UpperCasePokerPlayer)	3525.0	1800.0	3600.0	AGGRESSIVE	PaPlaA/23
Opponent (SimpleRuleAgent)	7125.0	1800.0	7150.0	AGGRESSIVE	PaPlaAa/23
UpperCase (UpperCasePokerPlayer)	14275.0	5350.0	1900.0	PASSIVE	PaPlaAaA/23

Output

```
Deal: Sim_d8a9013d-6916-4baf-bd5d-433f17709a40_9999 - sb=50.000000
Opponent (SimpleRuleAgent) pays the small blind (50.000000)
UpperCase (UpperCasePokerPlayer) pays the big blind (100.000000)
PREFLOP - 600.000000
Opponent (SimpleRuleAgent) calls. State=PREFLOP
UpperCase (UpperCasePokerPlayer) raises 200.000000. State=PREFLOP
Opponent (SimpleRuleAgent) calls. State=PREFLOP
flop: 7h Ad Qd
FLOP - 15650.000000
UpperCase (UpperCasePokerPlayer) raises 600.000000. State=FLOP
Opponent (SimpleRuleAgent) raises 2400.000000. State=FLOP
```

Figure 10.6: The hand view.

Experimental Results

In this chapter we present the results from testing UpperCase. Considering that UpperCase includes both a baseline strategy and an adaptive strategy we have performed different types of tests to measure the performance of the agent. There are two main test categories; *EQ-tests* and *EX-tests*. The EQ-tests measure the performance of the baseline EQ-strategy in UpperCase, while the EX-tests measure the adaptive part of UpperCase in which the agent attempts to classify the opponent's playing style and exploit it.

Each separate test consists of playing poker games of 1,000,000 hands using the technique of Duplicate Poker [25]. As discussed in section 3.3.1, the large number of played hands helps reduce the variance in order to more accurately measure the performance. The blinds are set to \$50 small blind and \$100 big blind.

The default-strategy used in all test is *always fold* (no case-match leads to a fold). This can be considered a fairly weak default-strategy compared to another simple strategy like always call. The reason for choosing the always fold default-strategy during testing is that we want to minimize the positive effect that the default-strategy has on the results. If the default-strategy is strong then this might influence the performance of the agent in a positive way and interfere with our investigation of the different evaluation techniques and adaptive strategies. By folding in every situation where a matching case cannot be found, an equal basis for testing the different parts of the system can be achieved.

The results of the EQ-tests and EX-tests are presented in sections 11.1 and 11.2 respectively. A discussion of the results can be found in chapter 12.

11.1 EQ - Baseline Strategy

In this section we present the tests measuring EQ's ability to develop solid baseline strategies. As explained in section 9.4, EQ evaluates every action performed by the agent during the training phase. The overall reuse-policy is highly affected by the evaluations given in the different decision quality evaluation techniques. We have therefore tested the performance of each of the three techniques described in section 9.4. The three evaluators are summarized in the following list:

- **Profit**

This approach uses a best-outcome reuse policy. This means that UpperCase will reuse the decision that has given the highest average profit from all matching cases. Outcome has been used as the basis for decision evaluation in other existing CBR poker agents and is included as a benchmark for measuring the performance of the other two reuse policies.

- **AIVAT**

This is an imitation of the decision evaluation tool described in chapter 5. Every decision is given an AIVAT score, where a higher score means better quality. A best-quality reuse policy is applied. This means that when UpperCase is requested to act, all matching cases from the EQ case-base are retrieved and the decision with the highest average AIVAT score is reused.

- **Equity**

This decision evaluation tool is described in section 9.4.3. Each decision made during training has been given an equity score based on the quality of the decision. A best-quality reuse policy is applied here as well. The decision that has given the highest average score from all matching cases will be reused.

11.1.1 Test Structure EQ

To test all three decision quality evaluators we have performed 12 tests in total. There are two different agents, *RaiseAgent* and *AdvancedRuleAgent* both included in the UniPoker framework, that have been used in the training phase and during regular play. *RaiseAgent* is an agent made for testing purposes that always raises the current size of the pot, while *AdvancedRuleAgent* is the strongest agent currently included in the UniPoker framework. Each decision quality evaluator (profit, AIVAT, equity) has been tested in the following manner:

1. Train against R, play against R
2. Train against R, play against A
3. Train against A, play against R
4. Train against A, play against A

R refers to **RaiseAgent**.

A refers to **AdvancedRuleAgent**.

In these tests we investigate how performance is affected if EQ has been trained against its opponent or not. In test-group 1 and 4, the agent used in training is also the opponent during actual play. The opposite is found in test-group 2 and 3.

The training phase consists of training 1,000,000 hands, which should be sufficient to measure the decision evaluators' abilities to develop decent poker strategies. The case-base is reset between each test.

11.1.2 Test Results EQ

This section presents the test results from all EQ-tests. The results are represented by the UpperCase agent’s win-rate in small blinds per hand (see section 2.5), in addition to the case-match percentage in parenthesis. For completeness, the simulation graphs from each individual test can be found in appendix C. This section only presents the results in a brief manner, while a discussion of the results is given in section 12.1.

11.1.2.1 Profit Evaluation Results

The results from testing EQ using the profit evaluation technique is presented in table 11.1.

	Play against R	Play against A
Train against R	9.2021 (99.85%)	0.2290 (58.83%)
Train against A	-4.0935 (34.33%)	1.0065 (99.56%)

Table 11.1: Results of the EQ-tests using profit evaluation.

In test 1 we can see that UpperCase is able to play very profitably, winning an average of 9.2021 sb/h. It is also able to create a winning strategy against AdvancedRuleAgent, even with a low case-match percentage, as seen in test 2. Test 3 shows that UpperCase does not play well against RaiseAgent after training against AdvancedRuleAgent. However, when playing against AdvancedRuleAgent, UpperCase plays well with an average win-rate of 1.0065 sb/h.

11.1.2.2 AIVAT Evaluation Results

The results from testing EQ using the AIVAT evaluation technique is presented in table 11.2.

	Play against R	Play against A
Train against R	4.6387 (99.67%)	0.3496 (47.94%)
Train against A	-4.1685 (33.90%)	1.1487 (99.67%)

Table 11.2: Results of the EQ-tests using AIVAT evaluation.

Test 1 and 2 show that UpperCase is able to create a winning strategy against RaiseAgent and AdvancedRuleAgent after training against RaiseAgent. The win-rate against RaiseAgent is lower than that of the profit evaluator. However, against AdvancedRuleAgent, the win-rate is higher than with profit evaluation, even though the case-match percentage is almost 11% lower. Again, in test 3 the performance is much worse. UpperCase plays very well against AdvancedRuleAgent resulting in a average win-rate of 1.1487 sb/h.

11.1.2.3 Equity Evaluation Results

The results from testing EQ using the equity evaluation technique is presented in table 11.3.

	Play against R	Play against A
Train against R	5.3951 (<i>99.36%</i>)	0.3051 (<i>59.51%</i>)
Train against A	-4.3023 (<i>33.92%</i>)	1.2167 (<i>99.67%</i>)

Table 11.3: Results of the EQ-tests using equity evaluation.

The results of the equity evaluation test resemble that of the other two tests. UpperCase is able to create a good strategy against RaiseAgent after training against it. It does also beat AdvancedRuleAgent after the same training phase with an average win-rate of 0.3051 sb/h. This is slightly below the AIVAT evaluator, but higher than with profit evaluation. A high profit can be seen when playing against AdvancedRuleAgent. The average win-rate of 1.2167 sb/h is the highest of all evaluation techniques against AdvancedRuleAgent.

11.2 EX - Adaptive Strategy

The goal of the EX-system is to classify the opponent's playing style and adapt UpperCase's strategy towards exploiting the opponent. During play, hand history is stored in a case-base and analyzed by the EX-system. Different poker metrics are observed and calculated in order to classify the current opponent-type (explained in section 9.6.1). As the classification becomes sufficiently reliable, the adapted strategy should begin to take precedence over the baseline strategy provided by the EQ-system. In this section we present the results from testing the performance of the EX-system. What we attempt to identify is the effect that the adaptation has on the overall performance.

11.2.1 Test Structure EX

In all EX-tests, the EX-system is completely disabled during the first half of the test. This makes it easier to compare the performance of UpperCase, with or without EX enabled. As soon as we are halfway through the test, the EX-system is enabled and UpperCase begins to store new cases in its EX case-base. The EX-system will then attempt to classify the opponent-type and begin adapting its baseline strategy towards exploiting the opponent. We have included the simulation graphs to visually display this difference. The simulation graphs show the total bankroll (amount of money won or lost in total) on the y-axis and the number of hands played on the x-axis. If the graph shows an improved win-rate (more positive graph) halfway through the test, we can assume that the adaptation has worked well and a better strategy has been applied.

We have used a weak EQ baseline strategy in order to more easily present the performance of EX. This means that in the first half of each test, the performance of UpperCase is expected to be poor. We have trained the EQ-system by playing 150,000 hands of poker, using equity evaluation, against RaiseAgent (we could also have reused any of the case-bases constructed in the EQ-tests).

We have trained EX by playing 150,000 hands of poker, using equity decision evaluation, against each of the four opponent-types described in section 9.6.1. This provides EX with knowledge on how to exploit these four opponent-types. In test number 2, UpperCase is specifically trained against the SimpleRuleAgent by playing 150,000 hands. The reason for lowering the number of hands played during training from 1,000,000 in the EQ-tests to 150,000 in the EX-tests, is that the EX-system trains against multiple opponent-types and this is more time-consuming. Ideally, the EX-system should also be trained with 1,000,000 hands (or more) against each opponent-type, but in this test we shortened the training phase for time-constraint reasons. However, 150,000 hands is a fairly large number, so the effect of the EX-system is still evident in all tests.

The different EX-tests are described in the following list:

1. **Unknown opponent**

In this test, UpperCase plays against an unknown opponent which means that there is no existing EX case-base for this specific opponent. This means that EX must classify the opponent's playing style to one of the existing four opponent-types in the EX-system and figure out how to best adapt its baseline strategy towards exploiting this unknown opponent. The unknown opponent used in this test is the SimpleRuleAgent included in the UniPoker framework. This agent applies a simple rule-based strategy.

2. **Known opponent**

In this test, UpperCase has already trained against the opponent during the EX training phase. This means that an EX case-base already exists for this specific opponent before the test begins. The test tries to identify if the EX-system is able to recognize the opponent and adapt its baseline strategy towards the counter-strategy developed during the EX training phase. We have used SimpleRuleAgent as the opponent in this test as well.

3. **Known easily exploitable opponent**

In this test, UpperCase plays against an easily exploitable opponent that it has trained against in the EX training phase. We use a *loose-passive* opponent that applies a simple rule-based strategy. This means that UpperCase should be able to quickly classify the opponent's playing style and adapt its strategy towards the counter-strategy developed during the EX training phase.

11.2.2 Test Results EX

The results of the three tests described in section 11.2.1 are given below. All tests consists of playing 1,000,000 hands of Duplicate Poker. The simulation graph show-

ing the total bankroll of each player is presented for each test. UpperCase is represented by a red line in the simulation graphs, while the opponent is represented by a blue line.

In addition to the simulation graph, a table showing the average win-rate in small blinds per hand (sb/h) is also presented for each test. This table includes the sb/h for the first 500,000 played hands (when EX is disabled), and the sb/h for all hands played from hand number 500,001 until 1,000,000 (when EX is enabled). This illustrates the change in average win-rate for the two strategies (with and without EX). The total win-rate for all 1,000,000 played hands is also given in the table.

The number of times a given EX case-base has been used to select an action is referred to as the *activation count*. During play, EX attempts to classify the opponent using a strategy vector (see section 9.6.3.1) and the case-base associated with the opponent-type with the highest similarity is activated. The activation count and percentage of all total activations are shown in a table in each of the three EX-tests.

The following sections present the results from each test while a discussion of the results is presented in section 12.2.

11.2.2.1 Test 1: Unknown opponent

Figure 11.1 shows the simulation graph resulting from playing against an unknown opponent. There are two consistent win-rates (both negative) that we can identify from the figure. The first win-rate shows the performance of UpperCase before EX is enabled. After 500,000 played, the graph begins to flatten slightly which illustrates increased performance. This is when EX is enabled and strategy adaption can begin. This second win-rate is a result of the adjustments made by EX.

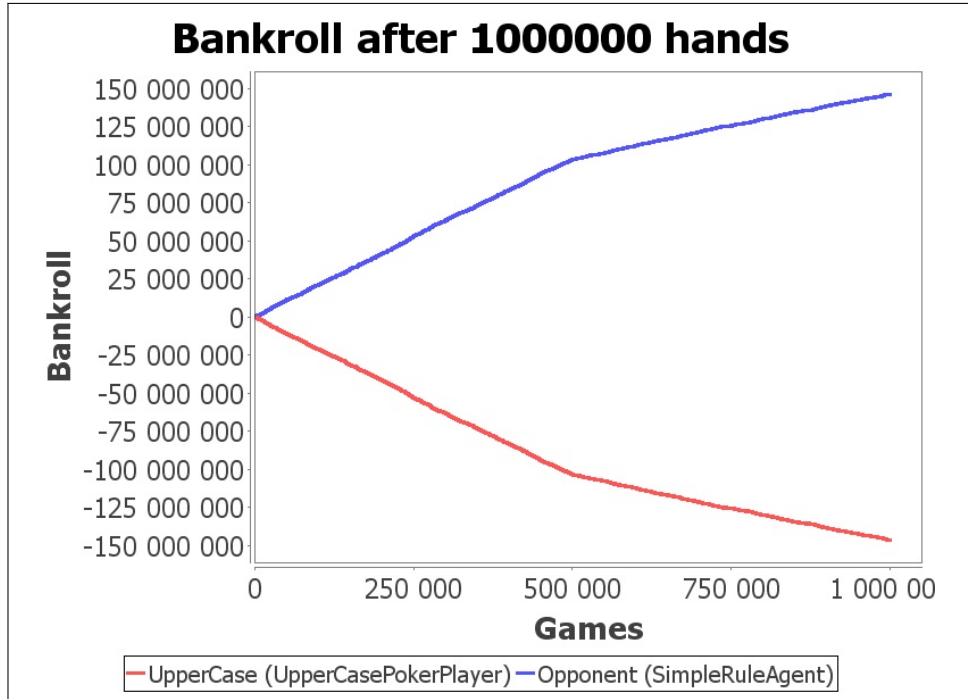


Figure 11.1: Simulation graph of EX-test 1. Against an unknown opponent.

	sb/h
EX disabled	-4.1333
EX enabled	-1.7143
Total	-2.9238

Table 11.4: Results from EX-test 1. Against an unknown opponent.

Table 11.4 shows that with EX disabled, UpperCase was losing 4.1333 sb/h on average. After EX is enabled, UpperCase improves its performance. However, the sb/h of -1.7143 shows that UpperCase is not playing well enough to beat this unknown opponent.

Opponent-type	Activation count	Activation %
Loose-Passive	681,417	99.71%
Tight-Passive	1,083	0.16%
Loose-Aggressive	899	0.13%
Tight-Aggressive	31	≈ 0%

Table 11.5: EX strategy adaptation against an unknown opponent.

Table 11.5 shows that EX classified SimpleRuleAgent as a loose-passive opponent-type in 99.71% of all activations.

11.2.2.2 Test 2: Known opponent

In this test, UpperCase is playing against SimpleRuleAgent which it has already played against during the training-phase of EX. We can see from the simulation graph in figure 11.2 that UpperCase is increasing its performance significantly after EX is enabled. The first win-rate shows a consistent loss, but after EX is enabled a profitable strategy is applied.

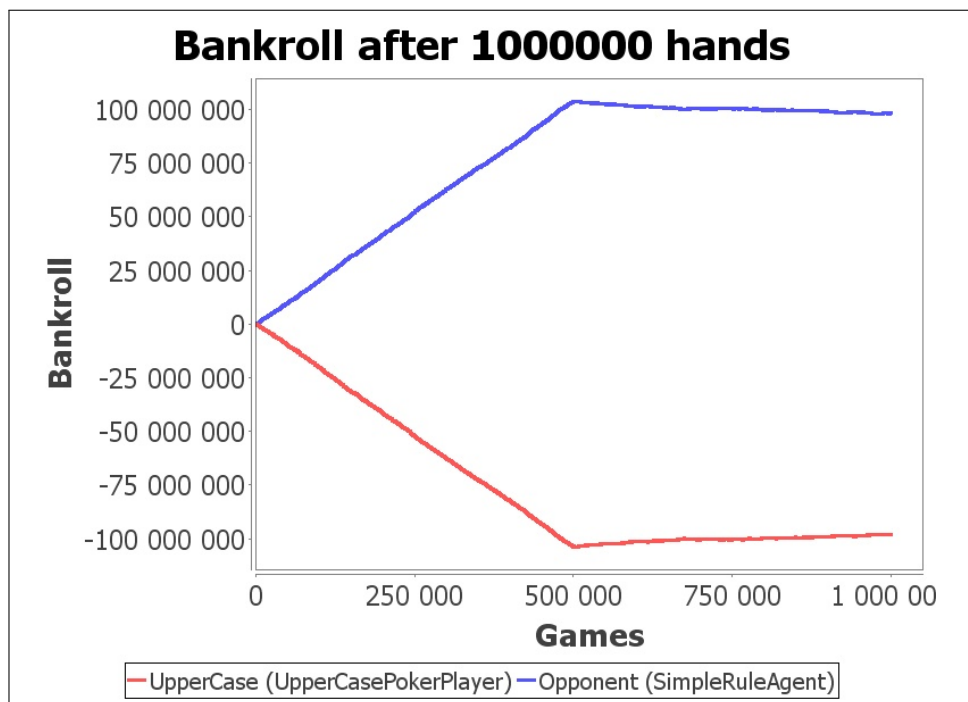


Figure 11.2: Simulation graph of EX-test 2. Against a known opponent.

	sb/h
EX disabled	-4.1439
EX enabled	0.2150
Total	-1.9645

Table 11.6: Results from EX-test 2.

Table 11.4 shows that the win-rate increases to a positive rate after EX is enabled. This is when EX attempts to recognize the opponent and apply a counter-strategy provided during training.

Opponent-type	Activations	Activation %
SimpleRuleAgent	675,038	99.86%
Loose-Passive	702	0.10%
Tight-Passive	172	0.03%
Loose-Aggressive	54	$\approx 0\%$
Tight-Aggressive	14	$\approx 0\%$

Table 11.7: EX strategy adaptation against a known opponent.

Table 11.7 shows that EX correctly classified SimpleRuleAgent in 99.86% of all activations.

11.2.2.3 Test 3: Known easily exploitable opponent

From figure 11.3 we see that UpperCase is able to play very profitably against its opponent after EX is enabled. The resulting counter-strategy from the training phase proved to be very successful in exploiting the weaknesses of this opponent. We can see from the simulation graph that UpperCase is able to turn the consistent loss into a good profit after EX is enabled.

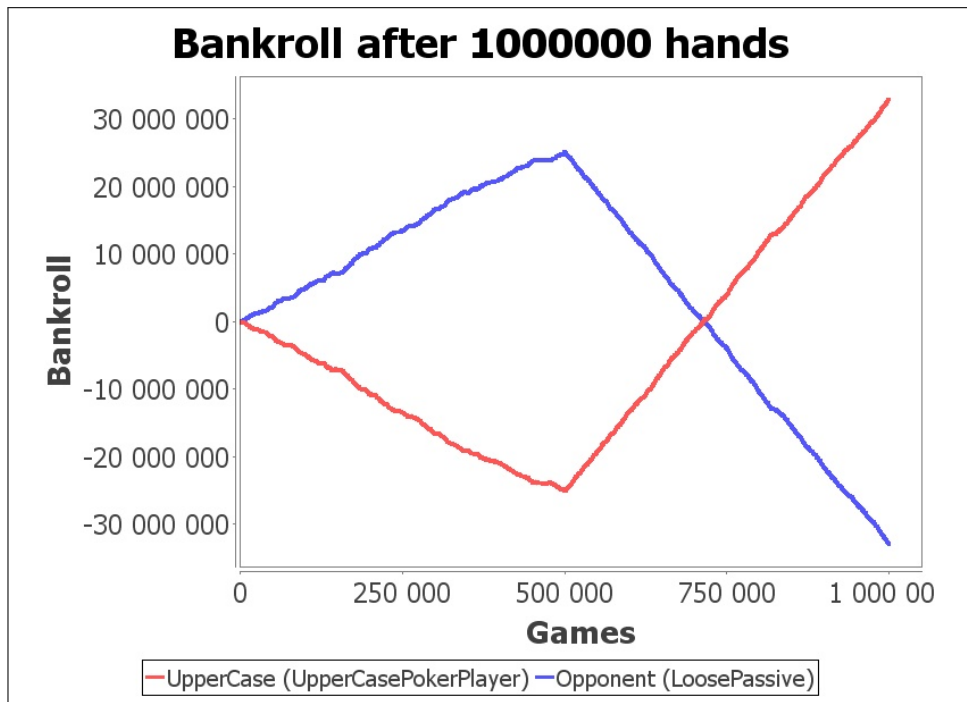


Figure 11.3: Simulation graph of EX-test 3. Against a known easily exploitable opponent.

Table 11.8 shows that after EX is enabled the average win-rate increases significantly, going from -1.0046 sb/h to 2.3233 sb/h. After losing consistently the first 500,000 hands, UpperCase is able to play very well the last part of the test, which leads a total win-rate of 0.6594 over the 1,000,000 hands played.

	sb/h
EX disabled	-1.0046
EX enabled	2.3233
Total	0.6594

Table 11.8: Results from EX-test 3.

Opponent-type	Activations	Activation %
Loose-Passive	720,307	99.71%
Tight-Passive	1,368	0.19%
Loose-Aggressive	648	0.09%
Tight-Aggressive	76	$\approx 0\%$

Table 11.9: EX strategy adaptation against a known easily exploitable opponent.

Table 11.9 shows that EX correctly classified the opponent as loose-passive in 99.71% of all activations.

Discussion

In this chapter we discuss the test results and performance of the UpperCase poker agent. Section 12.1 covers the EQ test results and section 12.2 covers the EX test results.

12.1 Discussion of EQ Test Results

The EQ-system is very important to the overall performance of UpperCase. EQ provides the strategy used by UpperCase until the opponent's playing style has been classified by EX and the adaptive strategy takes over. This means that UpperCase depends on a good EQ strategy when playing against different and unknown opponents. The goal of EQ is to produce an overall acceptable strategy.

One unexpected discovery in the test results displayed in table 12.1 is the results of test-group 1 (train against R, play against R). In this situation, the use of profit evaluation resulted in a much higher average win-rate compared to the other two evaluators. The profit evaluator may be considered a weak decision evaluation technique, as mentioned in section 3.3.2 and discussed in [37, 41, 45], as its evaluations are greatly affected by luck. However, this test suggests that when competing against a weak and static opponent, the profit evaluator can produce good results.

Evaluator		Play against R	Play against A
Profit	Train against R	9.2021 (<i>99.85%</i>)	0.2290 (<i>58.83%</i>)
	Train against A	-4.0935 (<i>34.33%</i>)	1.0065 (<i>99.56%</i>)
AIVAT	Train against R	4.6387 (<i>99.67%</i>)	0.3496 (<i>47.94%</i>)
	Train against A	-4.1685 (<i>33.90%</i>)	1.1487 (<i>99.67%</i>)
Equity	Train against R	5.3951 (<i>99.36%</i>)	0.3051 (<i>59.51%</i>)
	Train against A	-4.3023 (<i>33.92%</i>)	1.2167 (<i>99.67%</i>)

Table 12.1: Summary of EQ test results.

When we look at the results of test-group 2 and 3, we see that the different evaluators yield approximately the same results. One important observation is the low case-match percentage in these tests. When trained against RaiseAgent, the case-base will contain no cases of the opponent checking or calling. As a result, when playing against AdvancedRuleAgent the number of matching cases will be very low. A

similar tendency can be seen in test-group 3, where the case-match percentage is only approximately 34%. The case-base after training against `AdvancedRuleAgent` does not sufficiently cover the extremely aggressive playing style of the `RaiseAgent`. In both tests, EQ therefore applies the default-strategy when no match is found, resulting in the hand being folded. By applying a stronger default-strategy, the system would most likely be able to achieve better performance in these situations. This illustrates the importance of using balanced opponents during the training of EQ to produce an equilibrium baseline strategy that works sufficiently well in a broad spectrum of situations.

We consider test-group 4 to be most important in the comparison of the different decision quality evaluators. Our justification of this is based on the assumption of the `AdvancedRuleAgent` representing a more normal poker player as opposed to `RaiseAgent`. `AdvancedRuleAgent` is also currently the strongest agent in the UniPoker framework, besides `UpperCase`. In this test-group, the results suggest that the equity evaluator is able to provide the most profitable strategy. However, the difference between the two methods using perfect information hindsight analysis (equity and AIVAT) and the profit evaluator is still not as large as expected.

If we compare the results of this test to the results of test-group 1, we see an improvement to the equity and AIVAT evaluators compared to the profit evaluator. Test-group 1 and 4 are similar in the sense that `UpperCase` is trained against the same opponent as the one used during play, but there is a significant difference in the quality of the opponents. As suggested above, profit evaluation may work well against weak opponents, but when facing stronger opposition, the method of perfect information hindsight analysis becomes more efficient. It would be interesting to see the difference between the evaluators when tested against even stronger opponents than the `AdvancedRuleAgent`.

An important property to observe when comparing different decision evaluators is their ability to consider both a short-term perspective and a long-term perspective of the evaluation. The short-term perspective relates to the immediate results of an action, while the long-term perspective relates to how an action can affect a series of later situations. A reason for the different performances achieved by the three evaluators in test-group 4 may be explained in regards to this property. As discussed in section 9.4.4, the profit evaluator considers the complete hand, AIVAT considers each stage and the equity evaluator considers each separate decision. In the results, we see a correlation between shorter perspective and higher performance.

However, in poker sub-optimal decisions can be made in some situations with the intention of achieving larger gains later in the game. An example of this could be the tactic of *slow-play*. Instead of aggressively raising with a strong hand and risking that the opponent folds, passive play can lead to more money being invested in the pot by the opponent. In this example, the whole context of the situation must be taken into account when deciding which decision to make. A decision evaluation technique that only looks at separate decisions will not be able to address this.

All three evaluation techniques can prove to be useful in creating a good poker strategy, but the performance of the resulting strategy depends on the opponent used

during training. Training against a specific opponent can lead to a good strategy against this opponent, but it might not work as well when competing against other stronger opponents. It is therefore important to choose the right poker agents to train against in order to create a case-base with good coverage which can provide a robust and strong poker strategy.

12.2 Discussion of EX Test Results

EX is responsible for adapting the baseline strategy towards a new strategy that exploits the recognized opponent-type. EX should be trained against various opponents in order to achieve good coverage of possible opponent-types. The results from the EX-tests presented in section 11.2 show that the EX-system performed well overall. Table 12.2 presents a summary of the results from the these tests.

sb/h			
	Unknown	Known	Known exploitive
EX disabled	-4.1333	-4.1439	-1.0046
EX enabled	-1.7143	0.2150	2.3233
Total	-2.9238	-1.9645	0.6594

Table 12.2: Summary of EX test results.

All tests show an increased win-rate when EX is enabled compared to when it is disabled. This suggests that EX is able to successfully adjust the strategy provided by EQ to increase performance against the current opponent. The EX-system provides UpperCase with adaptive capabilities that can be very beneficial when facing new and unknown opponents.

In all tests, the activation of the different EX case-bases seems to be consistent and accurate. In test 1 (unknown opponent), EX classifies the unknown opponent as the loose-passive opponent-type in 99.71% of the situations. After a new QBR-module is added to the EX-system as a result of training against this opponent (test 2), the classification is correctly switched into selecting this QBR-module in 99.86% of the situations. This means that EX is able to successfully differentiate the loose-passive opponent-type from the SimpleRuleAgent opponent-type. An explanation of the increased win-rate from test 1 to test 2, may be that the selected QBR-module in test 1 was not trained against a sufficiently similar opponent-type as the one used in test 2.

However, test 2 and 3 show that by training against an opponent that closely matches the current opponent, higher performance can be achieved. This suggests that in order to adapt well to a broad range of opponents, EX must be trained against strong set of opponents that provide good coverage of different strategies in an accurate manner. This will also make UpperCase better equipped for competing against unknown opponents. Additionally, the EX-tests performed in section 11.2 include 150,000 played hands during the training phase. With more training, the EX-system might be able to further increase the performance.

12.3 Summary

From what we have seen in the tests performed and presented in chapter 11, UpperCase is able to play well against the opponents provided in the UniPoker framework. The baseline strategy works well and the results from using *perfect information hindsight analysis* in decision quality evaluation suggest that evaluations less influenced by variance in poker can be beneficial. The EX-system is able to classify opponent-types and adjust the baseline strategy towards given counter-strategies developed for exploiting these specific opponents.

We have seen that having one static strategy might work well against one type of opponent, but it can be a weak strategy against other opponents. This is evident in the EQ-tests where training against RaiseAgent resulted in a great strategy against this opponent, but it was not as strong when playing against AdvancedRuleAgent. However, this was influenced by the low case-match percentage.

Perfect information hindsight analysis is used in when creating the EQ strategy. This means that it requires a simulation- or training tool that can provide this perfect information. This can be a drawback considering that not all poker software tools provide this functionality. This means that the EQ-system cannot be trained against other strong poker agents unless perfect information hand history is available.

To fully measure and understand the performance of the UpperCase agent we need to compete against stronger opponents. Close to the submission of this thesis we did an attempt to test UpperCase against the SartreNL poker agent created by Rubin and Watson [37]. We were given an executable version of SartreNL and proceeded with implementing support for the ACPC-protocol in UniPoker. However, with the limited time available we were unfortunately not able to make the SartreNL agent work properly in our testing environment. Successfully testing UpperCase against other stronger agents remains the natural next step in this project.

Conclusions and Future Work

In this thesis we have investigated how to create poker agents with adaptive capabilities using case-based reasoning (CBR). Adaptive capabilities allow a poker agent to dynamically adjust its strategy in order to exploit different opponents. This property is considered necessary to achieve optimal performance against various types of opponents.

In our study of existing CBR-based poker agents, we discovered that a common approach among such agents is imitation and reuse of decisions made by other poker agents. This approach has been successfully applied to create strong poker agents. We also identified a sensitive point in reuse-policies relying on the evaluation of poker decisions. The properties of poker makes it difficult to accurately evaluate decisions and this can have a negative effect on such reuse-policies. Investigation of methods using *perfect information hindsight analysis* suggests that such methods can achieve improved accuracy of decision evaluations.

Our main goal in this thesis was to create a new approach to the application of CBR in poker. This has resulted in the implementation and testing of UpperCase. This poker agent is built on a principal idea of two CBR-systems, collaborating in the task of making decisions in the game of poker. One system, named EQ, provides a static equilibrium strategy while the second system, named EX, adapts to different opponents in order to exploit their strategies. These two system both employ an instance-based approach to CBR.

Experimental results have shown that both systems are able to invent strategies able to beat other poker agents. Our results also show that EX is able to classify different opponents based on their previous actions and achieve increased performance by adjusting the strategy accordingly. This means that the principal idea has been realized successfully.

In order to reach final conclusion about how well the system is able to perform, more experiments need to be conducted. There are still questions to be answered about optimal configurations and training of the system. We have demonstrated that the system is able to win consistently against the strongest agents currently available in the UniPoker framework. However, it is still unclear how well the system will work against stronger opposition.

We have also shown how the UniPoker framework can be utilized in the process of developing and testing an advanced poker agent. During this process, the function-

ality of the framework has been improved and extended. Overall, the use of the framework has been beneficial for our work with this thesis.

13.1 Future Work

In this section we present ideas for future work regarding UpperCase and the UniPoker framework.

13.1.1 Further Testing and Development of UpperCase

UpperCase was implemented and tested in this thesis. However, there are still some issues left to be researched and experiments to be performed. The following list presents some of our ideas for future work:

- **Additional Testing**

Although we have tested UpperCase against the strongest poker agents in the UniPoker framework, additional testing against even stronger opponents would be beneficial. Well-known poker agents from the Annual Computer Poker Competition (ACPC) would be ideal opponents, as this would help us understand how UpperCase performs in a comparison to the agents we have studied during this research.

- **Improving Situation-keys**

The situation-key format of the QBR-module (see 9.3.2.2) captures important similarities between game-states. However, the chance of improving the best 5-card hand in post-flop stages is not covered by the format. The current format recognizes the similarity between situations that have approximately equally strong hands, but fails to differentiate between situations where the player has different chances of improving it. E.g., a player in the flop stage with a weak 5-card hand is in a different position depending on if there is a chance for improving his hand into a flush or a straight. Since the current format does not cover this property, the system will not treat these situations differently. Changes made to the situation-key format, addressing this issue, can improve the strategies produced by the QBR-module.

- **Dynamic Adaptation**

EX currently considers all available knowledge about an opponent when characterizing his strategy. This means that the characterization is averaged across all the opponent's actions experienced by the system. Poker players may however dynamically change their strategy during play. E.g., a tight player may suddenly switch and become loose. The average characterization produced by EX will, in this case, not accurately describe the changes in the opponent's strategy. One solution to this problem could be the introduction of *dynamic adaptation*. With this approach, EX can treat the most recent actions as more

important when characterizing the opponent's strategy, e.g. only consider the N last actions.

13.1.2 Further Development of the UniPoker Framework

UniPoker can be a useful tool when developing poker agents. However, there is still a lot that can be done to improve the framework. Here are some of the improvements we consider being most beneficial to the framework in its current state:

- **Review the Meerkat-adapter**

UniPoker contains an adapter allowing poker agents implementing the Meerkat-API to be used with the framework. At the time the adapter was implemented, we were unable to obtain any documentation of the API. As a consequence, we have low confidence in the correctness of the implementation. However, while working on this thesis we received the documentation of the API and this is now included as part of UniPoker.

- **Additional Poker Agents**

A motivation for using the UniPoker framework is the ability to test poker agents against different opponents. There are currently few strong agents available in the framework and by developing or integrating additional agents with the framework, it can become more useful to its users.

- **Documentation and examples**

The API of UniPoker is, in our opinion, intuitive but the framework would still benefit from more documentation and examples on how it can be used.

Bibliography

- [1] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, March 1994.
- [2] Annual Computer Poker Competition. 2008 Poker Bot Competition Summary. <http://webdocs.cs.ualberta.ca/~pokert/2008/results/index.html>.
- [3] Annual Computer Poker Competition. 2009 Poker Bot Competition Summary. <http://webdocs.cs.ualberta.ca/~pokert/2009/index.php>.
- [4] Annual Computer Poker Competition. Duplicate Poker. http://www.computerpokercompetition.org/index.php?option=com_content&view=article&id=55:duplicate-poker&catid=40&Itemid=64.
- [5] Annual Computer Poker Competition. Results 2010. http://www.computerpokercompetition.org/index.php?option=com_content&view=article&id=79:results-2010&catid=36:results&Itemid=61.
- [6] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3):235–256, May-June 2002.
- [7] Ralph Bergmann, Michael M. Richter, Sascha Schmitt, Armin Stahl, and Ivo Vollrath. Utility-Oriented Matching: A New Research Direction for Case-Based Reasoning. *Proceedings of the 9th German Workshop on Case-Based Reasoning, GWCBR'01*, 2001.
- [8] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.
- [9] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The Challenge of Poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.
- [10] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent Modeling in Poker. *AAAI/IAAI*, pages 493–499, 1998.
- [11] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Using Probabilistic Knowledge and Simulation to Play Poker. *AAAI/IAAI*, pages 697–703, 1999.
- [12] Darse Billings and Morgan Kan. A Tool for the Direct Assessment of Poker Decisions. 2006.
- [13] Aaron Davidson. Opponent Modeling in Poker: Learning and Acting in a Hostile and Uncertain Environment, 2002. Master thesis, University of Alberta.

- [14] Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo Tree Search in Poker using Expected Reward Distributions. *ACML*, pages 1–15, 2011.
- [15] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding Nash equilibria in extensive form games. *Proceedings of the Eighteenth International Conference on Game Theory*, 2007.
- [16] Phil Hellmuth. *Play Poker Like the Pros*. HarperResource, 2003.
- [17] Investopedia. Monte Carlo Simulations. <http://www.investopedia.com/terms/m/montecarlosimulation.asp>.
- [18] Investopedia. Nash Equilibrium. <http://www.investopedia.com/terms/n/nash-equilibrium.asp>.
- [19] Michael Johanson. Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player, 2007. Master thesis, University of Alberta.
- [20] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing Robust Counter-Strategies. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.
- [21] Steven D. Levitt and Thomas J. Miles. The Role of Skill versus Luck in Poker: Evidence from the World Series of Poker. *NBER Working Paper No. 17023*, 2011.
- [22] Stefania Montani and Lakhmi C. Jain. *Successful Case-based Reasoning Applications*, volume 305 of *Studies in Computational Intelligence*. Springer, 1st edition, 2010.
- [23] Yu Nesterov. Excessive Gap Technique in Non-smooth Convex Minimization. *SIAM Journal on Optimization*, 16:235–249, 2007.
- [24] Ann E. Nicholson, Kevin B. Korb, and Darren Boulton. Using Bayesian Networks to Play Texas Hold'em Poker. 2006.
- [25] International Federation of Poker. Duplicate poker. http://www.imsaworld.com/uploads/pdf/Duplicate_Poker_Guide.pdf.
- [26] Jan Berge Ommedal and Eivind Røttereng Solbakken. Developing Autonomous Poker Agents, 2011. Intelligent Systems Depth Study, Norwegian University of Science and Technology.
- [27] Online Casino Advice. Poker Types. <http://www.onlinecasinoadvice.com/poker/types/>.
- [28] Denis Richard Papp. Dealing with Imperfect Information in Poker, 1998. Master thesis, University of Alberta.
- [29] Phil Galfond. 'G Bucks' Conceptualizing Money Matters. <http://www.bluffmagazine.com/magazine/%27G-Bucks%27-Conceptualizing-Money-Matters.-Phil-Galfond-985.htm>.

- [30] Poker AI Wiki. Nash Equilibrium. http://pokerai.org/wiki/index.php/Nash_Equilibrium.
- [31] Poker Times. Poker Rankings. http://www.pokertimes.com.au/poker_rankings.htm.
- [32] Preflophands.com. Preflophands.com. <http://www.preflophands.com/>.
- [33] IBM Research. Deep Blue. <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>.
- [34] Jonathan Rubin and Ian Watson. CASPER: A Case-Based Poker-Bot. *In AI 2008: Advances in Artificial Intelligence, 21st Australasian Joint Conference on Artificial Intelligence*, pages 594–600, 2008. Springer.
- [35] Jonathan Rubin and Ian Watson. A Memory-Based Approach to Two-Player Texas Hold'em. *In Proceedings of AI 2009: Advances in Artificial Intelligence, 22nd Australasian Joint Conference*, pages 465–474, 2009.
- [36] Jonathan Rubin and Ian Watson. SARTRE: System Overview, A Case-Based Agent for Two-Player Texas Hold'em. *ICCBR-09 Workshop*, 2009.
- [37] Jonathan Rubin and Ian Watson. Similarity-Based Retrieval and Solution Re-use Policies in the Game of Texas Hold'em. *18th International Conference on Case-Based Reasoning*, pages 465–479, 2010. Springer-Verlag.
- [38] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 145(5-6):958–987, April 2011.
- [39] Jonathan Rubin and Ian Watson. On Combining Decisions from Multiple Expert Imitators for Performance. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain*, July 2011.
- [40] Jonathan Rubin and Ian Watson. Successful Performance via Decision Generalisation in No Limit Texas Hold'em. *19th International Conference on Case-Based Reasoning*, 2011.
- [41] Jonathan Rubin and Ian Watson. Case-Based Strategies in Computer Poker. *AI Communications*, 25(1):19–48, March 2012.
- [42] Arild Sandven and Bjørnar Tessem. A Case-Based Learner for Poker. *The Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006), Helsinki, Finland*, 2006.
- [43] I. Schweizer, K. Panitzek, S.H. Park, and J. Fürnkranz. An Exploitative Monte-Carlo Poker Agent. *Technical Report, Technische Universität Darmstadt*, 2009.
- [44] The Poker Bank. Poker Equity (Pot Equity). <http://www.thepokerbank.com/strategy/mathematics/equity/>.
- [45] Sebastian Helstad Unger. Integrating CBR and BN for Decision Making with Imperfect Information, 2011. Master thesis, Norwegian University of Science and Technology.

- [46] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.

Appendices

Setting up UniPoker

The following sections present two different ways of using UniPoker. Knowledge about and access to the following freely available tools is required:

- Mercurial¹
- Maven²
- Eclipse³
- Eclipse Maven Integration⁴

A.1 Setup 1: Externalized development

This setup is recommended if you will be using the features of the framework without making any changes to it.

1. Setup a new Maven-project in Eclipse.
2. Add the Agora-repository as a repository in your pom:

```
<repository>
  <id>agora</id>
  <url>http://repo.agorait.no/content/groups/agoracontext</url>
</repository>
```

3. Add the framework-module as a dependency in your pom:

¹Mercurial is a distributed source-control management tool. It can be downloaded from <http://mercurial.selenic.com/>.

²Maven is a project management and comprehension tool. It can be downloaded from <http://maven.apache.org/>.

³Eclipse is an IDE (Integrated Development Environment). It can be downloaded from <http://www.eclipse.org/>.

⁴Eclipse Maven Integration is a plugin that adds support for Maven within Eclipse. It can be installed from the "Eclipse Marketplace" within Eclipse or downloaded from <http://www.eclipse.org/m2e/>.

```
<dependency>
  <groupId>edu.ntnu.unipoker</groupId>
  <artifactId>unipoker-framework</artifactId>
  <version>1.0</version>
  <scope>compile</scope>
</dependency>
```

4. Add the simulator-module as a dependency in your pom:

```
<dependency>
  <groupId>edu.ntnu.unipoker</groupId>
  <artifactId>unipoker-simulation</artifactId>
  <version>1.0</version>
  <scope>compile</scope>
</dependency>
```

5. Start implementing your poker agent by subclassing 'edu.ntnu.unipoker.fw.PokerAgent'.

```
public class CallAgent implements PlayerAgent{

    public double makePlay(Game game,PokerGameRunner gameRunner,
        Player player, Act theAct){
        return 0;
    }

}
```

Figure A.1: Example of a simple agent that always plays passively, e.g. check or call.

A.2 Setup 2: Integrated development

This setup involves checking out the complete source-code for UniPoker so that it can be viewed, modified or extended more easily.

1. Acuire a copy of the source for the UniPoker-project. An online mercurial-repository at <http://hg.agorait.no/repo/unipoker/> will be maintained for a period of time after the submission of this thesis.
2. Import the super-pom located in the "source"-folder as an 'Existing Maven Project' in Eclipse.

3. Create a package for your project either inside the unipoker-players-module or inside a new separate maven-module.
4. Start implementing your poker agent by subclassing 'edu.ntnu.unipoker.fw.PokerAgent'.

```
public class RaiseAgent implements PlayerAgent{  
  
    public double makePlay(Game game,PokerGameRunner gameRunner,  
        Player player, Act act){  
        return 2 * game.getCurrentDeal().getPot();  
    }  
  
}
```

Figure A.2: Example of a simple agent that always raises with two times the current size of the pot.

Setting up UpperCase

The following section presents how to set up and run UpperCase. The process require the same tools as the process of setting up UniPoker (see B)The following list shows how to set up UpperCase:

1. Setup UniPoker for *integrated development* as shown in section A.2 of the appendix.
2. Acquire a copy of the source for the UpperCase-project. An online mercurial-repository at <http://hg.agorait.no/repo/uppercase/> will be maintained for a period of time after the submission of this thesis.
3. Create a new folder that can contain the configuration of UpperCase. Copy the sample configuration-files found in the "config"-folder inside the source of the project.
4. Update path of "uppercaseDatabaseConfigPath" inside config.xml so that it correctly points to the hibernate.cfg.xml (the other configuration-file)
5. Configure the hibernate.cfg.xml file in order to select which SQL database that UpperCase will store its cases inside. This process is documented here <http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/session-configuration.html>. We recommend using the H2 Database¹, which does not require any configurations. A running database is needed for UpperCase to play.
6. Define a system environment-variable 'UPPERCASE_HOME' as the path pointing to the folder containing the configuration-files for UpperCase. You may have to restart Eclipse at this point.
7. Import the super-pom located in the "source"-folder as an 'Existing Maven Project' in Eclipse.

Be aware that UpperCase may require a large heap-space depending on the amount of training. When launching a test including the UpperCase agent for the first

¹H2Database is documented and freely available at <http://www.h2database.com/>.

time, a hand-rollout is performed by UniPoker. This results in a text-file including pre-generated pre-flop winning probabilities. Generating this file may take several minutes (but is only performed once).

The following list contains elements that can be executed:

- **UpperCase Tests**

The tests performed in this thesis is located inside the "uppercase-control"-module. Executing the class "AllTests" results in all tests being executed.

- **Using UpperCase**

The "uppercase-control"-module contains a set of examples illustrating how uppercase can be used to play with other agents in the UniPoker framework. Before playing, UpperCase should be trained. A minimal amount of training can be performed by executing the class "UpperCaseTraining".

- **UpperCase Web Interface**

The UpperCase web-interface is located inside the "uppercase-web"-module. This is a standard java web-application. It can be executed inside the Eclipse IDE or built using maven and deployed to a java web container.

Simulation Results EQ

The following figures present the results from the EQ tests in section 11.1.2. Each figure corresponds to the results shown in the tables in section 11.1.2. The graphs illustrate the total bankroll (money won or lost) for each player on the y-axis and the number of hands played on the x-axis.

C.1 Profit Evaluation Results

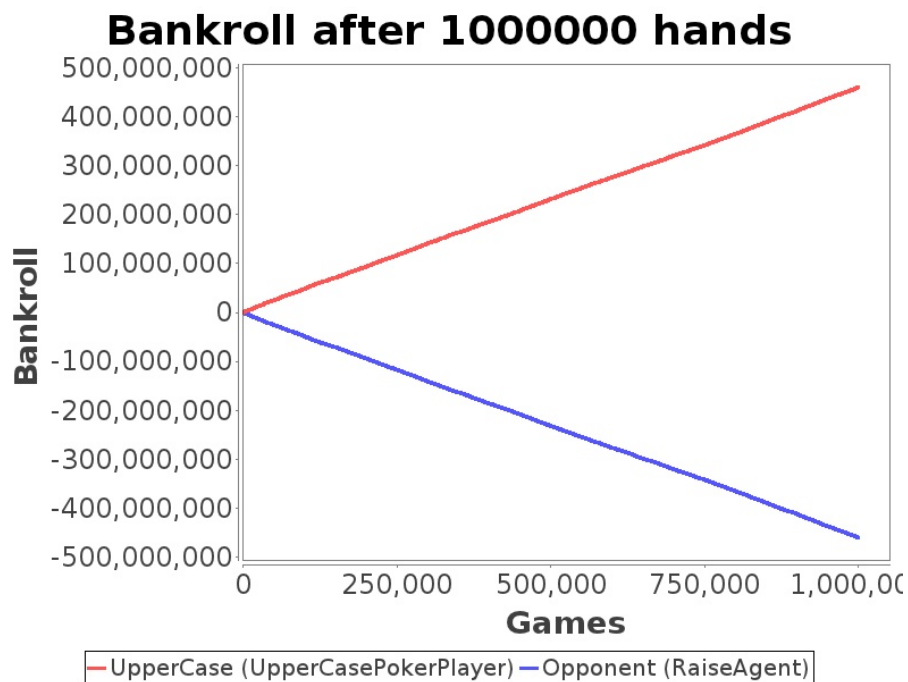


Figure C.1: EQ simulation using profit evaluation. Test 1.

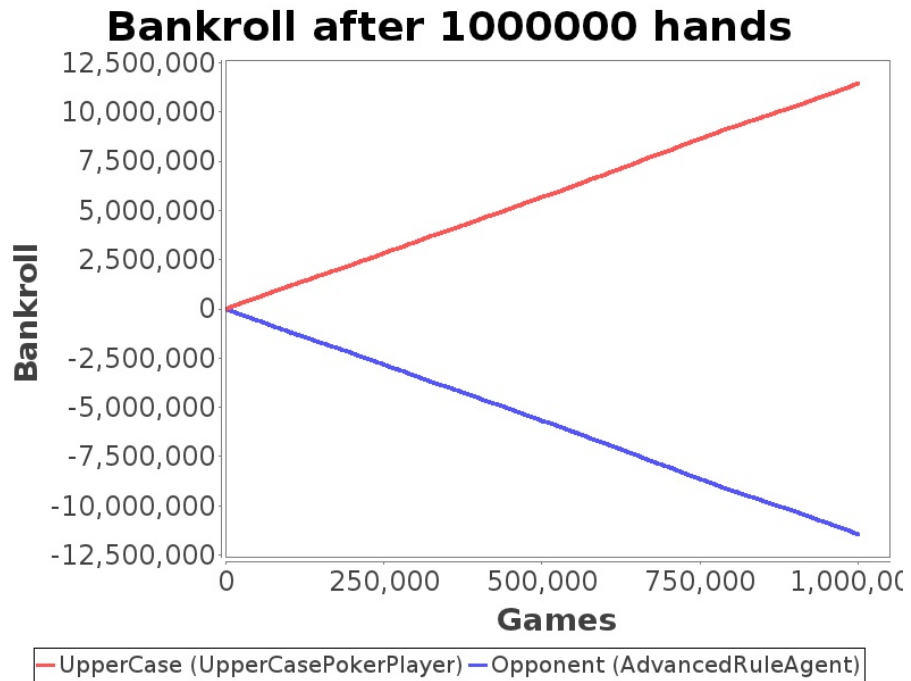


Figure C.2: EQ simulation using profit evaluation. Test 2.

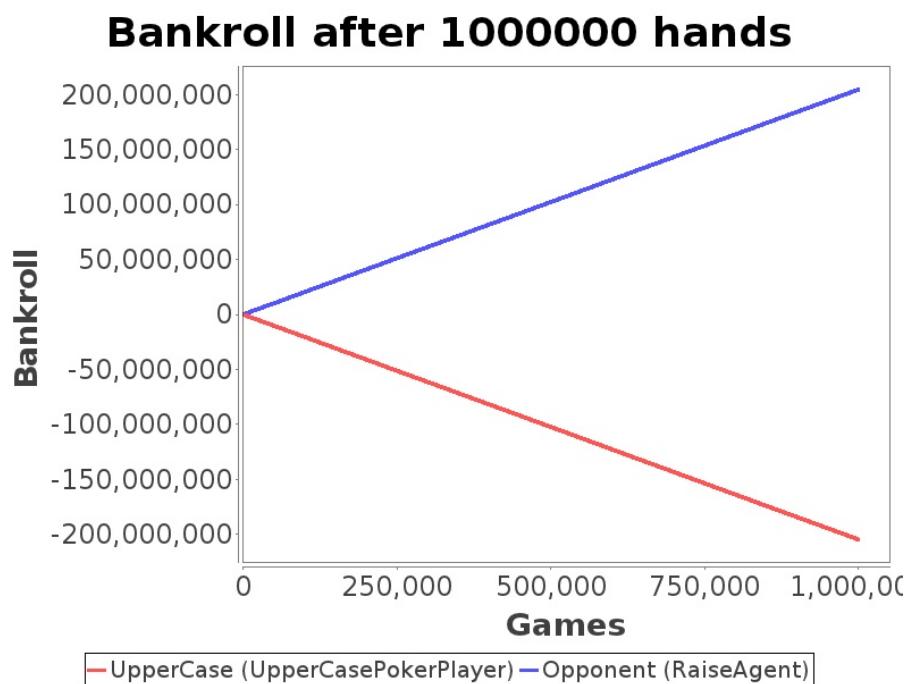


Figure C.3: EQ simulation using profit evaluation. Test 3.

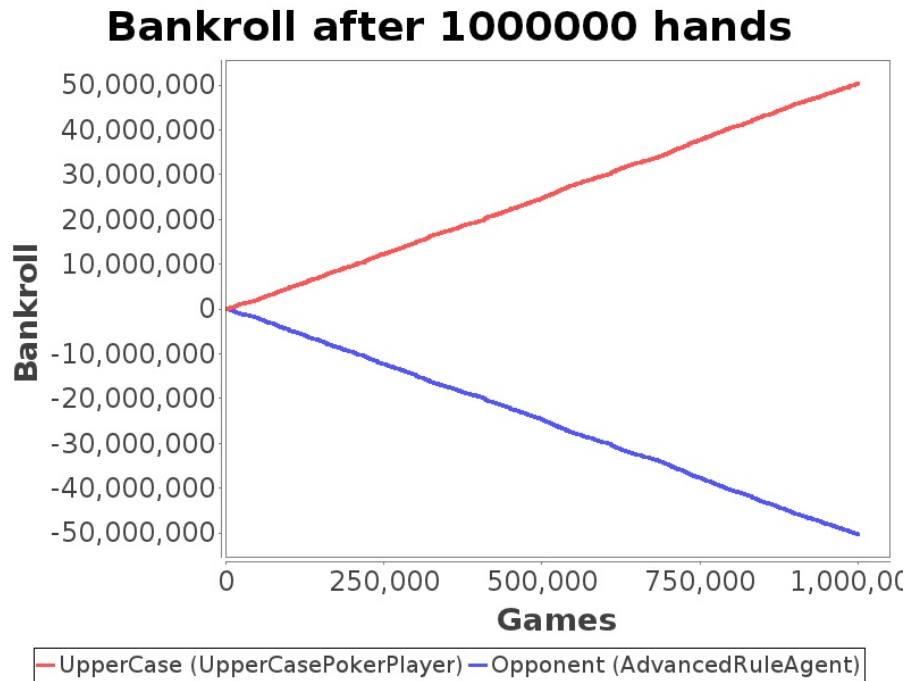


Figure C.4: EQ simulation using profit evaluation. Test 4.

C.2 AIVAT Evaluation Results

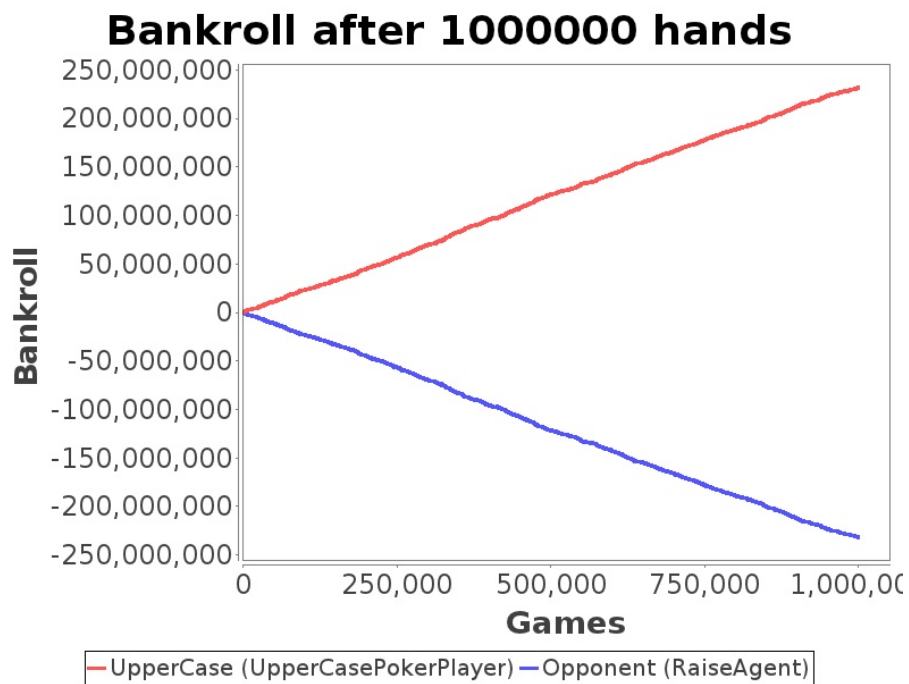


Figure C.5: EQ simulation using AIVAT evaluation. Test 5.

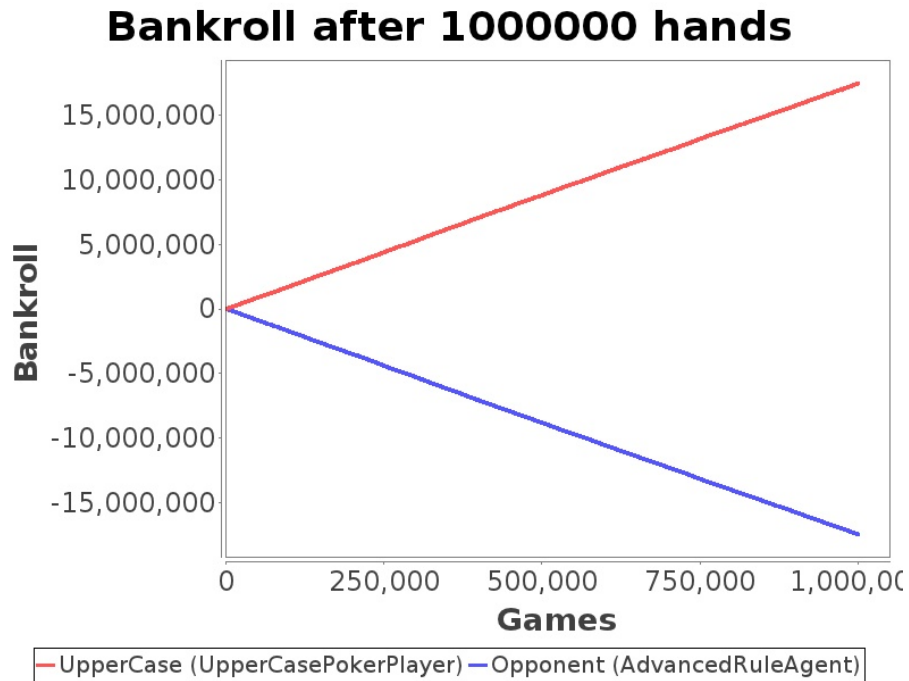


Figure C.6: EQ simulation using AIVAT evaluation. Test 6.

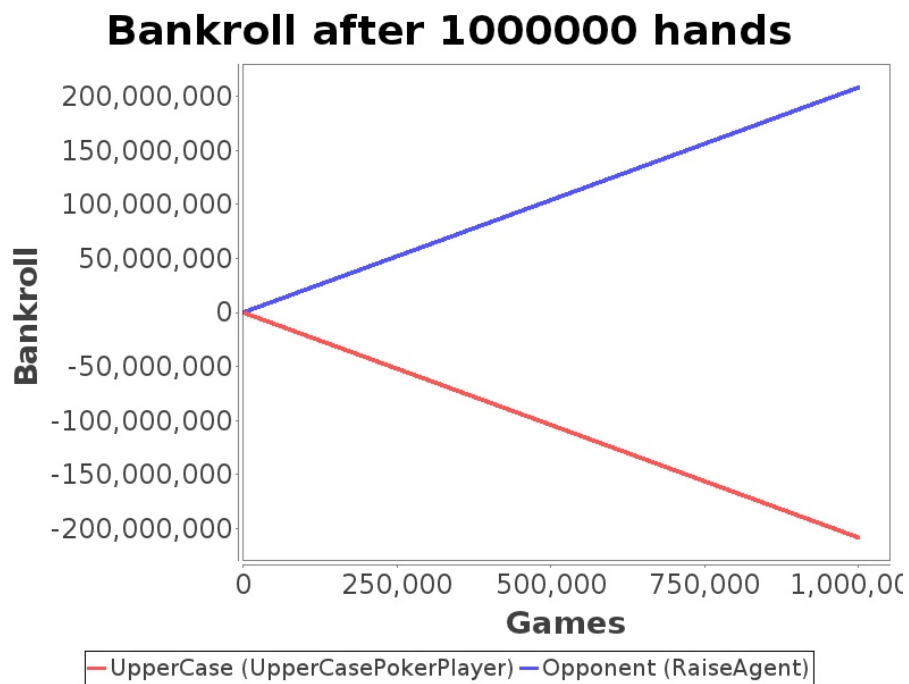


Figure C.7: EQ simulation using AIVAT evaluation. Test 7.

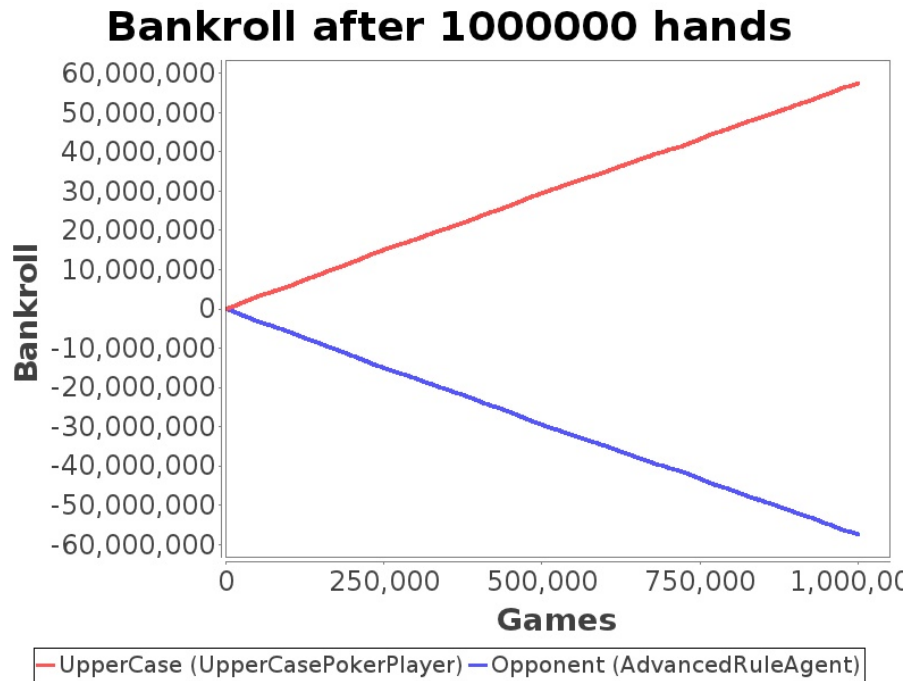


Figure C.8: EQ simulation using AIVAT evaluation. Test 8.

C.3 Equity Evaluation Results

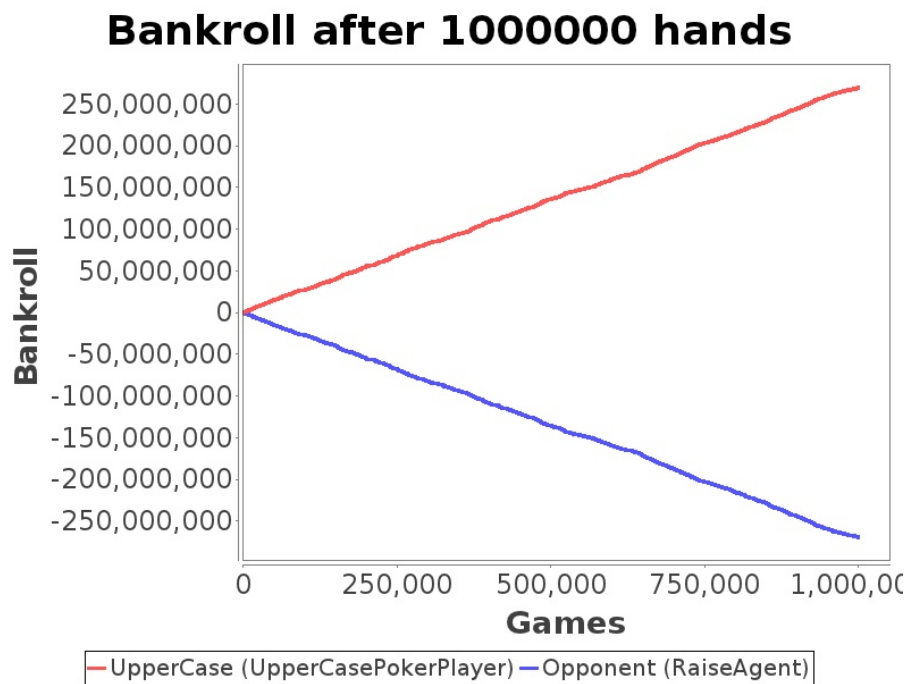


Figure C.9: EQ simulation using equity evaluation. Test 9.

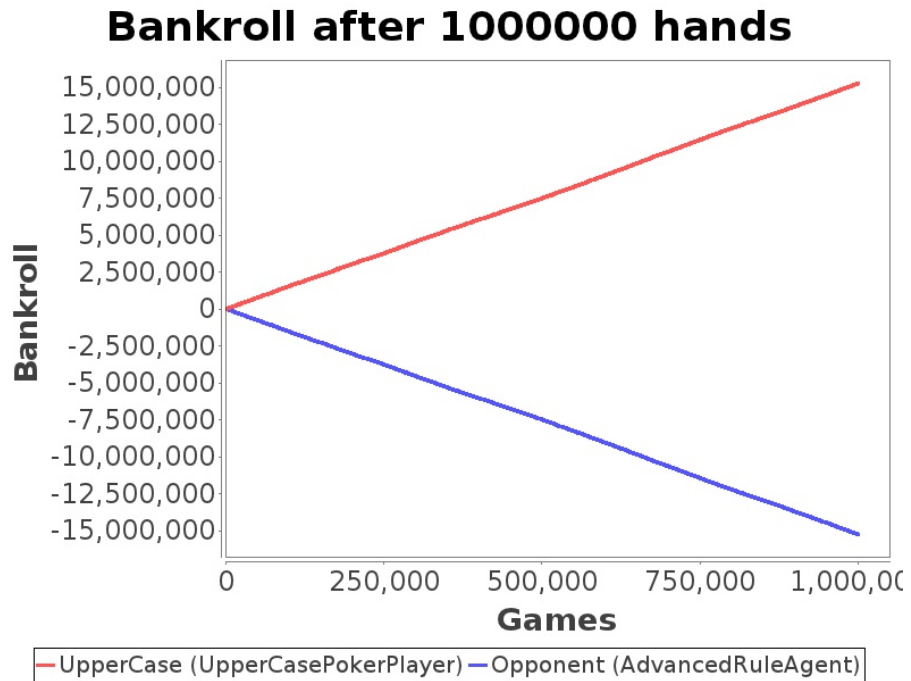


Figure C.10: EQ simulation using equity evaluation. Test 10.

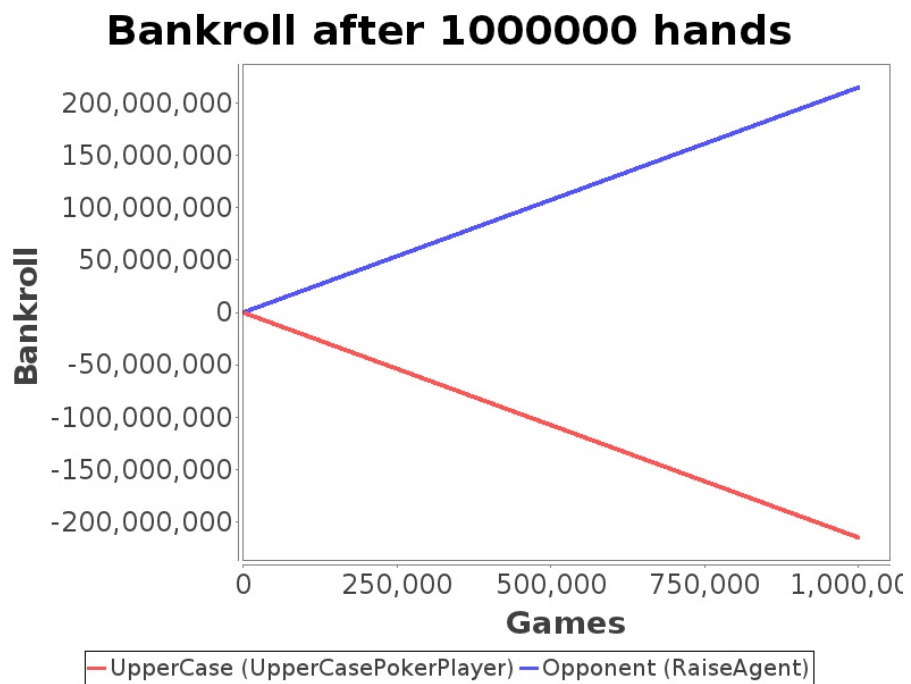


Figure C.11: EQ simulation using equity evaluation. Test 11.

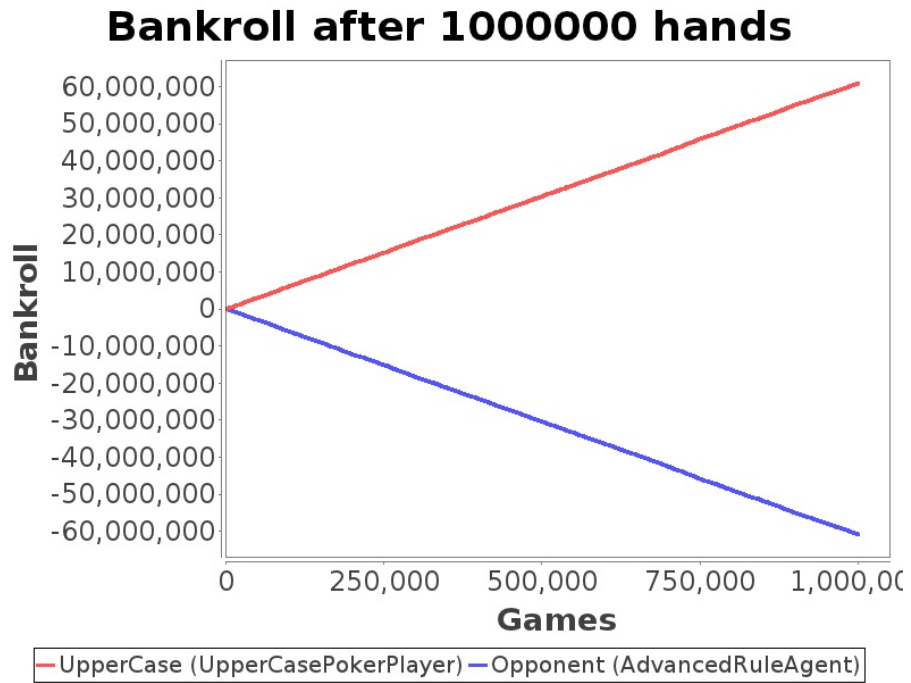


Figure C.12: EQ simulation using equity evaluation. Test 12.

Poker Hand Rankings

Figure D.1 from [31] presents the Texas Hold'em poker hand rankings from best (royal flush) to worst (high card).

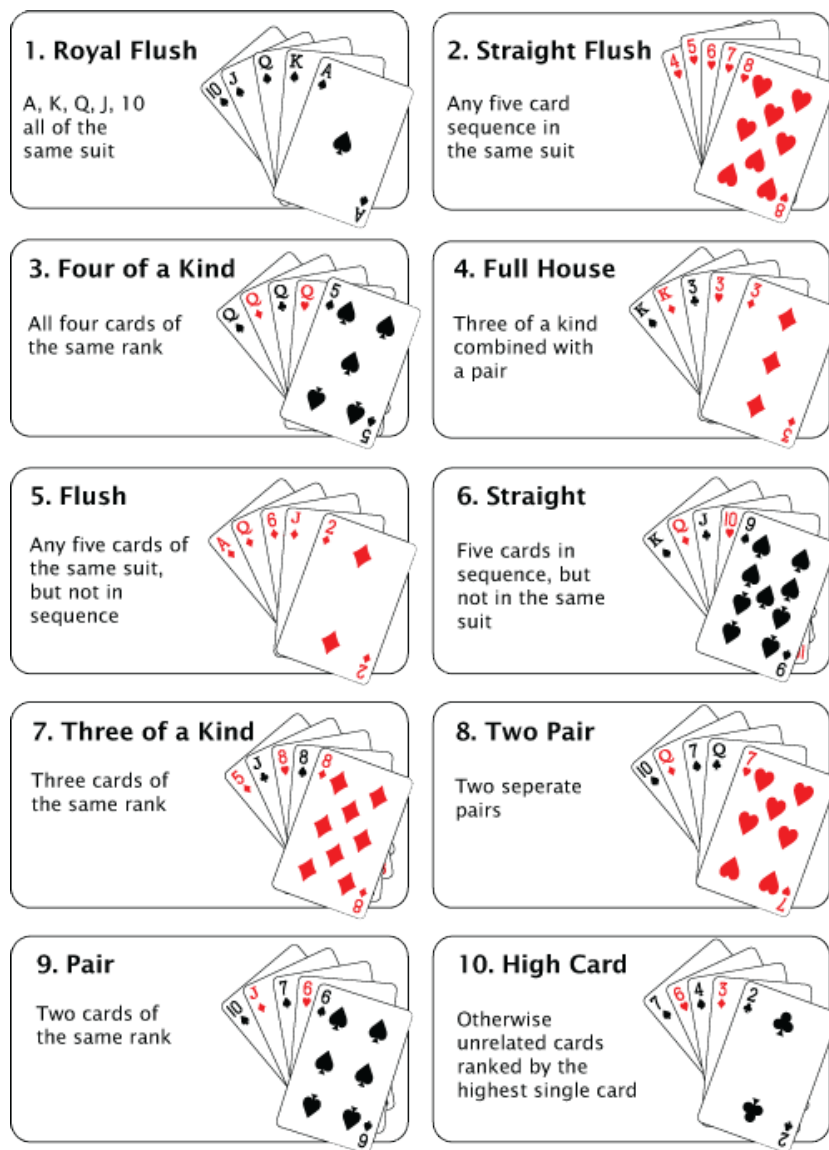


Figure D.1: Texas Hold'em hand rankings from best to worst.