|  |
| --- |
|  |
| Player Profiling and Unpredictability |
| Project Report |
|  |
| **Devin Denis** |
|  |

|  |
| --- |
|  |

Table of Contents

[Introduction 3](#_Toc385361585)

[Game Description 3](#_Toc385361586)

[Player Profiling 3](#_Toc385361587)

[Balancing Optimality and Unpredictability 3](#_Toc385361588)

[Motivation 3](#_Toc385361589)

[Choice of Game 3](#_Toc385361590)

[Techniques 3](#_Toc385361591)

[Profiling 4](#_Toc385361592)

[Genetic Programming 4](#_Toc385361593)

[Design Choices 4](#_Toc385361594)

[Language 4](#_Toc385361595)

[GEPSharp 4](#_Toc385361596)

[Profiling 5](#_Toc385361597)

[Results 5](#_Toc385361598)

[Profiling 6](#_Toc385361599)

[Genetic Programming 6](#_Toc385361600)

[Future Work 8](#_Toc385361601)

[Works Cited 8](#_Toc385361602)

[Appendix 9](#_Toc385361603)

[Running the Code 9](#_Toc385361604)

[Sample Test Data 9](#_Toc385361605)

# Introduction

In this project I have designed and implemented an AI for a very simple game. The focus was on the AI being able to learn the tendencies of its opponent and exploit these tendencies, while remaining unpredictable itself.

## Game Description

The two players each begin play with ten cards with the numbers one to ten on them in their hands. In each of ten turns, the players select a card from their hand and simultaneously reveal their choices. If the cards are not the same exact value, the player with the higher numbered card gets a point. Each card can only be used once. The player with the greater number of points after ten turns wins.

## Player Profiling

A major goal for this project was to get the AI to be able to predict the strategy of an opponent after observing a number of their previous games. I used a simple learning strategy for this, which will be detailed later. I tested the AI’s abilities against several simple deterministic and probabilistic AI patterns which do not include profiling, to see how well it could pick up their patterns.

## Balancing Optimality and Unpredictability

I also tested the AI against itself. To beat its own profiling, an unpredictable strategy is desirable. However, this must be balanced with the ability to exploit the opponent’s weaknesses. I used genetic programming to evolve a variety of strategies.

# Motivation

Player profiling is immensely useful in game AI for a variety of reasons. First, creating an AI that remains challenging when matched against skilled players is much easier if that AI can predict the player’s actions. Many multiplayer games, especially those with hidden information, essentially boil down to how well you can predict your opponent, given a high enough skill level. Secondly, profiling players is useful for games even when unrelated to the AI of the player’s opponents. Profiling can help the game understand what the player wants and enjoys, allowing the game to adapt itself to improve the experience. For much the same reason, this information may be very useful to game developers and designers when making changes to games or creating new ones.

## Choice of Game

The game I created is designed to strip away everything but predicting the other player’s moves. Given full knowledge of what your opponent is going to play, there is a clear optimal strategy – play one higher than your opponent if possible, otherwise play your lowest. However, due to the nature of simultaneous turns, to be successful a player must be able to accurately predict their opponent’s strategy.

# Techniques

## Profiling

My profiling technique is not designed around any specific established algorithm, but the basic idea is a form of reinforcement learning. The more the AI observes a specific pattern of behaviour, the more it expects that pattern to reappear in the future. This type of learning works well against simple opponents, but can be quite exploitable- a human opponent, for example, could establish a pattern and then play a strategy that will beat an attempt to beat the pattern.

## Genetic Programming

I used Gene Expression Programming (GEP), a variation on the standard Genetic Programming, to evolve strategies for the AI. While I could (and did when first testing the profiling part) give the AI the straightforward optimal strategy, having a simple set strategy like that makes the AI more predictable. It also only works well if the AI’s profiling is completely accurate, which it generally only is against simple strategies such as ‘always play highest’.

I will not describe the whole gene expression algorithm here. It differs from standard Genetic Programming mainly in the representation, which involves working with strings which generate trees, rather than with trees themselves. In general it runs faster than the standard genetic programming but is slightly less effective in finding optimal solutions.

# Design Choices

## Language

I wrote my project using C#. The most important reason for this choice is that the library I wrote which implements GEP is in C#. Additionally, features such as straightforward parallelization support, excellent IDE and debugging tools, SQL-style list manipulation, lamba expression support, and several others contribute to creating a language which is both reasonably powerful and very low hassle. That, combined with my own familiarity with C# means that I can spend my time most efficiently- the language never gets in the way.

## GEPSharp

My honors project is an implementation of Genetic Programming in a couple of different varieties. I used the libraries from that work for this project to speed development. The code is designed to be highly customizable and simple to work with. To use it I simply define the fitness function and the functions and terminals that will make up the tree, then give it population sizes and a number of generations. One downside to using this library in its current form is a lack of support for tournament selection. Selection is randomized and weighted by fitness score.

I used the terminals *My Highest, My Lowest,* and *My Closest*. Highest and lowest are obvious. *My Closest* will be the available card with the closest value to one higher than what the profiling section predicts will be the opponent’s next move, favoring the higher if two are tied for closest. Note that *My Closest* is not quite equivalent to the optimal full-knowledge strategy, as if the closest will not win optimal play is the lowest.

I used the functions *Coin, Plus, Minus, Highest Wins, Closest Wins,* and *High Certainty.* *Coin* takes two inputs and randomly returns one or the other. *Plus* returns one higher than the input, and *Minus* one lower than the input. *Highest Wins* takes two inputs and return the first if the AI’s highest card will beat its prediction for the opponent’s play. *Closest Wins* does the same, but with the card returned by *My Closest* instead of its highest card. *High Certainty* takes two inputs and return the first if the profiling section is confident in its answer (probability > 70%), and the second if the profiling section is not confident.

In each test run I generate a number of AI individuals using these functions and terminals, and test their fitness by having them play a large number of games against each other, and in some cases also against a set of simple deterministic strategies. I have run tests with a variety of parameters, as will be described in more detail later. After each round of fitness tests the most successful strategies are more likely to survive to the next round, and strategies are modified randomly- standard genetic programming behaviour.

## Profiling

I originally envisioned the AI storing profiling information for each player it had met, but during implementation I realized this was unnecessary. Instead, each player simply keeps a record of all their past plays and the factors present during those plays. Specifically, when the player plays a card, it increases a counter for that card, for each of the other cards it and its opponent are holding. The table below illustrates the idea.

|  |  |  |  |
| --- | --- | --- | --- |
| Total Plays | Card | While I Hold | While They Hold |
| 1000 | 1 | 1 – 100 | 1 - 60 |
| 2 – 49 | 2 – 30 |
| 3 – 72 | 3 – 46 |
| 4 – 80 | 4 – 43 |
| 5 – 44 | 5 – 68 |
| … | … |
| 10 - 32 | 10 – 82 |
| 2 | 1 - 53 | 1 – 68 |
| 2 – 100 | 2 – 59 |
| 3 – 51 | 3 – 57 |
| … | … |
| 10 - 79 | 10 - 32 |
| … | | |
| 10 | 1 - 90 | 1 - 31 |

To predict what its opponent will play next, the AI looks at their play history and adds up the probabilities based on what factors are present in the current game. It then makes its prediction for the card which has the highest probability of being played. This prediction is passed to the strategy algorithm, which decides what card to play.

# Results

## Profiling

The AI’s profiling abilities are extremely effective against simple strategies such as ‘play lowest to highest’, or any other unchanging order of cards. After a game or two to grasp the pattern, profiling predictions passed to the optimal full-knowledge strategy have a 100% win rate against these strategies. Even against strategies with some randomization, if there is an overall pattern to the play, profiling generally does a very good job of picking up on it. An example of a randomized strategy which profiling is very effective against is ‘random widening’, in which the first card played is random, and subsequent cards played are randomly selected one higher or lower than a previously played card.

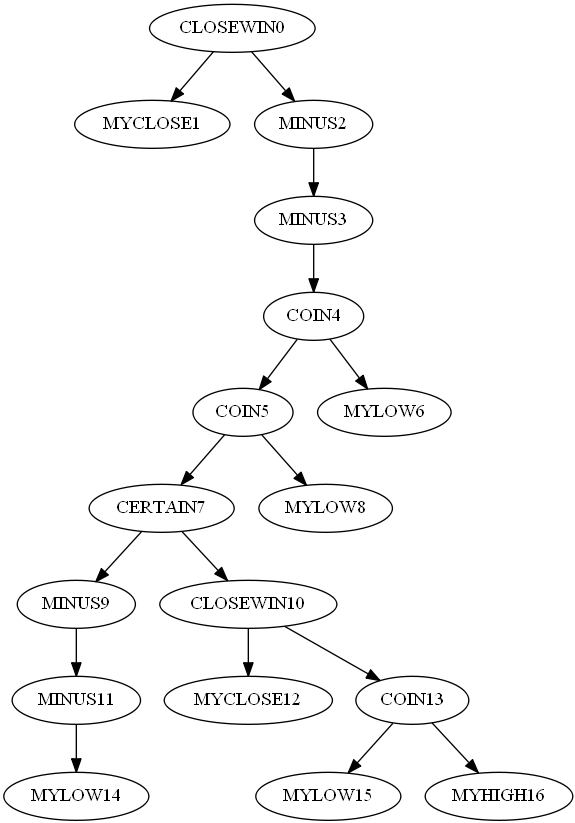
Against more unpredictable strategies, or those which make and use their own predictions, profiling is not as powerful, but still plays a very important role in victory. The node *My Closest,* which requires high accuracy in profiling to be effective, is common in high fitness individuals from GEP testing, especially in runs that also evaluate fitness against static strategies.

As predicted, a human player can very easily exploit the profiling system, especially if they know what the AI’s strategy is. Achieving a 100% win rate against the optimal full-knowledge strategy is trivial, for example.

## Genetic Programming

I tested evolving strategies with a population of fifty for twenty generations. In each generation, every individual played a certain number of games against every other individual. I ran tests with the number of games at one-hundred and one-thousand. In some cases they also played the same number of games against a set of static strategies, including the optimal full-knowledge strategy. An individual’s fitness was calculated as the ratio of won games to total games played. When static strategies were included in the fitness, the win ratios were calculated separately for vs. evolved strategies and vs. static strategies, and then averaged.

Evolving strategies through GEP was overall very effective. The highest fitness individuals nearly always have >80% win rates. Some high fitness individuals are identical or nearly identical to the optimal full-knowledge strategy. This is more common with the tests where one-thousand games are played between each set of opponents. Presumably the additional games allow the profiling to become more accurate, which allows the full-knowledge strategy to be more effective. Another common thread was for high-fitness individuals in hundred-game tests to have a *Coin* node at or near the root. This makes the individual more difficult to profile, and the lower number of games does not allow the volume of data to make up for that, resulting in highly unpredictable play.



Sample tree for 1000 games with statics

The sample tree above shows the most fit individual after twenty generations for a test with one-thousand games per encounter and statics included. This individual can nearly be reduced to the optimal full-knowledge strategy. The right branch from the root can be simplified to “If certain, play lowest. Otherwise play highest minus two with 12.5% chance and lowest otherwise.” The optimal full-knowledge strategy would simply replace that branch with “play lowest”- not much change, given the chances of getting something else in that branch.

To gain some information about how much use each of the functions and terminals see in high fitness individuals, I ran a short test (population 10, generations 15, games 100) fifty times and recorded for each type the percentages of nodes from the highest-fitness individuals that were of that type. The data is in the table below. It is important to note that the shortness of the test undoubtedly affected these numbers, but the data could still be useful.

Obviously, the terminals were the most common nodes, with *My Lowest* being the most common of those across the board. The most common function was *Coin*, which is in line with my earlier observation of *Coin* being common for one-hundred game tests.Overall though, the differences in the percentages are quite small, and I doubt they represent a statistically significant observation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Node | Terminal? | Appearance Rate with Statics | Appearance Rate without Statics | Average Appearance Rate |
| *My Highest* | Yes | 21.2% | 22.3% | 21.8% |
| *My Lowest* | Yes | 24.8% | 23.5% | 24.2% |
| *My Closest* | Yes | 24.2% | 21.8% | 23.0% |
| *Coin* | No | 8.3% | 6.6% | 7.5% |
| *Highest Wins* | No | 4.1% | 6.3% | 5.2% |
| *Closest Wins* | No | 3.6% | 5.2% | 4.4% |
| *Certain* | No | 4.0% | 4.6% | 4.3% |
| *Plus* | No | 4.9% | 5.2% | 5.1% |
| *Minus* | No | 4.9% | 4.5% | 4.7% |

# Future Work

Given more time or more powerful equipment, I would’ve like to see more and longer tests, especially in regards to determining which nodes are most commonly used in high fitness individuals. I’d also like to investigate the relationship between number of games played against each opponent and similarity to the optimal full-knowledge strategy.

Another area to expand on is the profiling. The current profiling, though situationally effective, is easy to abuse and requires a large number of observed games for good accuracy. I would like to explore other possible options.

# Bibliography

Ferreira, C. (2002). *Gene Expression Programming.* Angra do Heroismo, Portugal.

Koza, J. R. (1992). *Genetic Programming.* Cambridge: MIT Press.

# Appendix

## Running the Code

There are two options to run the code. First, the solution can be opened in Visual Studio (2012 or later). From there one can browse or run the code as normal. Alternatively, the files “WithoutStatics.exe” and “WithStatics.exe” can be found in the main folder for the project. These files run a fairly long test (population 50, generations 20, 1000 games per opponent), and as the names suggest, one includes static opponents in the fitness calculation and the other does not. After the test completes the best individual will be pitted against the optimal full-knowledge strategy for 1000 games, and following that will play observed games against the optimal strategy, allowing the user to view the profiler’s predictions and player choices at each step.

## Sample Test Data

A number of sample test runs are included in the main project folder. They are in folders named in the format “population-generations-gamesPerOpponent-statics/nostatics-highestFitness”. The folders contain twenty graphviz files showing the best individual from each generation. For each test run, the tree for the best individual in the final generation has been screen-capped and put in the folder “Tree Images”, in case the marker does not have access to graphviz. These images are labeled in the same way as the folders.