UNIVERSITY OF CALIFORNIA SAN DIEGO

Online MindReader Game Utilizing Weighted Hedging Trees

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Matthew Elliott

2018

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Thanks to Yoav Freund for his wisdom, guidance, and a thrilling few years. I also thank Aaron Schurger and Robert Schapire for their thoughts on extending the MindReader project.

ABSTRACT OF THE THESIS

Online MindReader Game Utilizing Weighted Hedging Trees

by

Matthew Elliott
Master of Science in Computer Science

University of California San Diego, 2018

Professor Yoav Freund, Chair

The MindReader web app is an online freely accessible version of the "matching pennies" game. Matching pennies is an old game mentioned in the works of Edgar Allen Poe and analyzed by Claude Shannon, but our implementation is very modern. By visiting, "www.mindreaderpro2.appspot.com", a user can instantly play on their computer, tablet, or phone and can save their game results by signing into Facebook. The website is part of the larger MindReader project, which collects game data from the app and provides a data analysis platform. We present the inner workings of the project so that academic researchers can easily run and contribute to MindReader.

Chapter 1

Introduction

1.1 Matching Pennies

The MindReader website is a modern recreation of the very simple “matching pennies” game. The matching pennies game can be thought of as a binary form of “rock, paper, scissors” where one player guesses a number, either 1 or 0, and then the other player tries to guess what number the first player chose. A point is awarded to one of the players depending on if the second player could correctly guess the first player’s choice. Two opponents can play this game over and over again stopping when one of player reaches a a certain score, in our case 100 points.

From playing just one or two rounds of this game it seems like the ability to guess a player’s move would be purely random chance. However, after playing many rounds patterns almost certainly begin to appear in people’s moves even when they try their best to be random. The fact that humans are not able to make random moves has many interesting implications in behavioural psychology research [5]. For our online video game, MindReader, this also allows us to create an AI opponent (a bot) that can take advantage of the implicit patterns in people’s moves. We find that when players compete against our AI algorithm the bot wins the vast majority of the time.

1.2 Pre-Computer Implementations

The matching pennies game has an incredibly rich history dating back to nearly 200 years ago. The first mention of the game was in a short story written by Edgar Allan Poe,“The Purloined Letter,” where 2 children use the matching pennies game as a way to gamble coins between each other [17]. The first person to try to build an AI algorithm for the game was D. W. Hagelbarger, who built a deterministic logic based algorithm [11]. Hagelbarger’s algorithm was created using electronic relays, as shown in the electrical diagram in Figure 1.1 on the next page. The famous Claude Shannon took and interest in the matching pennies game and Hagelbarger’s algorithm. Shannon built his own algorithm that used randomness in its implementation [12]. It is said that Hagelbarger and Shannon had their algorithms compete against each other where in the end, Shannon was the victor.

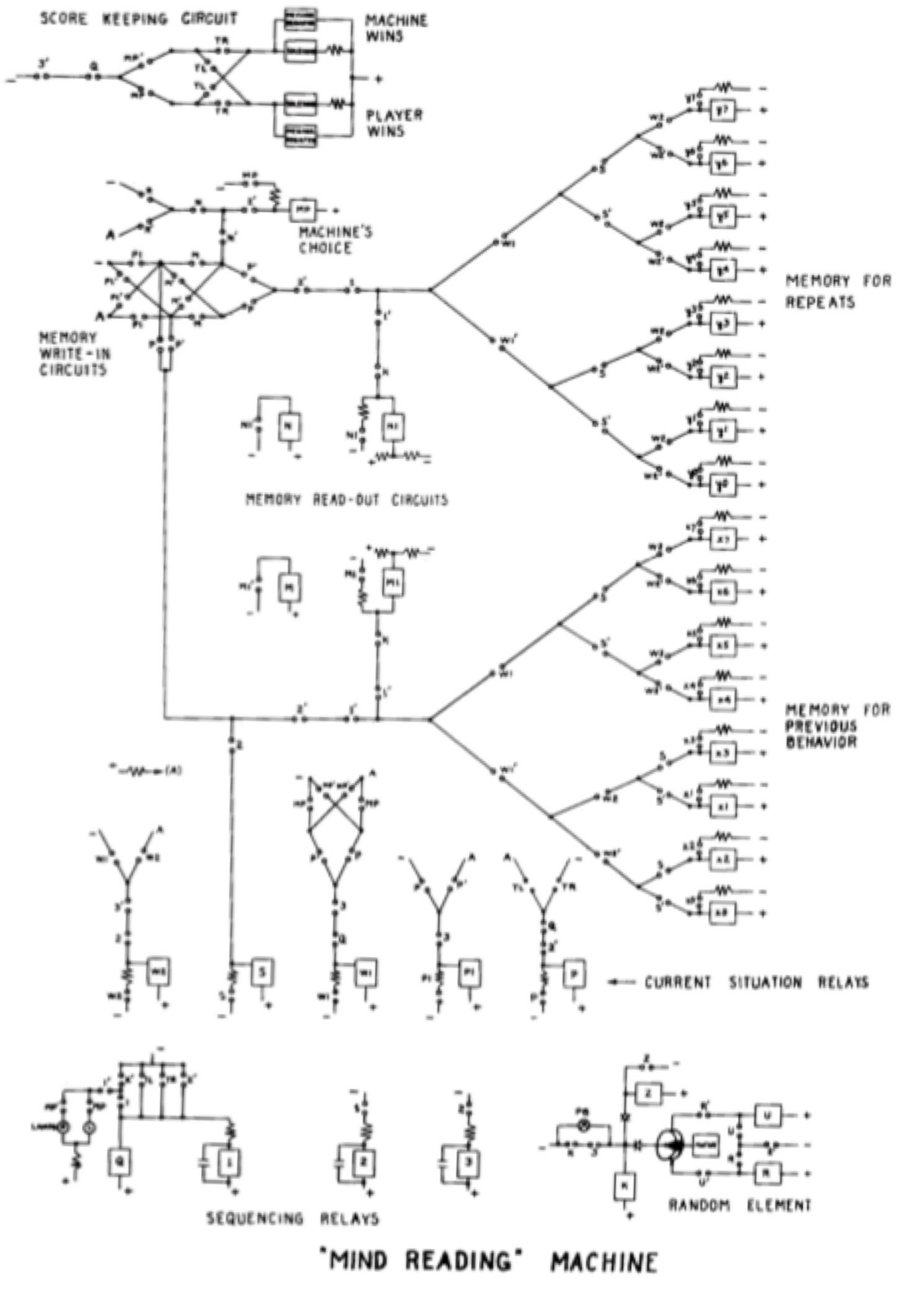
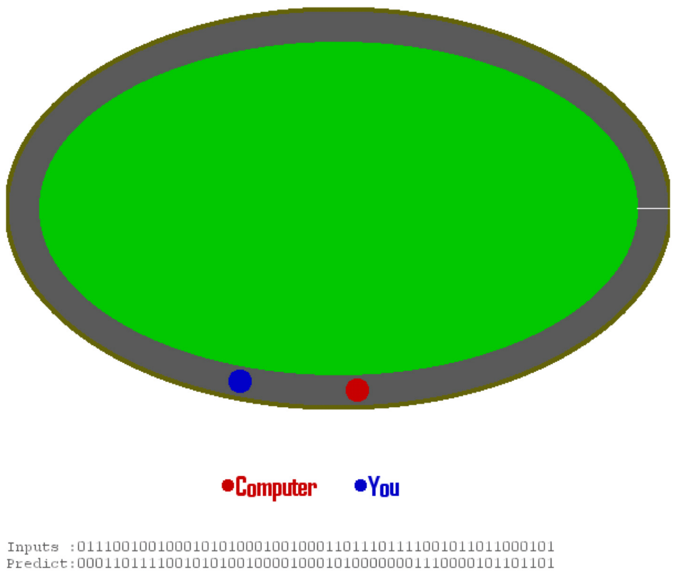


Figure 1.1: Electrical Diagram of the first MindReader AI algorithm

1.3 Previous MindReader Work

Freund's lab created MindReader, an online version of the matching pennies game, allowing anyone in the world to play the game using their computer. Unlike the AI algorithm created by Hagelbarger MindReader is probabilistic, utilizing a context weighted hedging tree (see algorithm chapter). The first version of MindReader was created by Anup Doshi. Figure 1.2 shows what the original MindReader game looked like. This version of MindReader was created using Java code embedded inside of a web browser (a Java applet). Later Andrea Biaggi extended the MindReader applet's functionality while also doing research on different variations of the AI algorithm [18].

Using Java within an applet made it possible to embed class based structures within a website. At the time this made it possible to easily create an architecture for the MindReader website (relative to JavaScript). Unfortunately on modern web browsers it has been shown that Java applets contain critical security flaws which made them all but disappear from web programming. For the safety of the user web browsers prohibit the user's computer from running a Java applet. Thus it was necessary to create a new MindReader that used JavaScript.



```
Inputs :011100100100010101000100100011011101111001011011000101
Predict:0001101111001010100100001000101000000001110000101101101
```

Figure 1.2: First Implementation of MindReader by Anup Doshi

1.4 The Modern MindReader

Since the goal of this project was to create a modernized version of MindReader we did not just stop at replacing the Java code with JavaScript. Instead we decided to build a truly modern web application which uses things like Facebook to manage players' accounts and the power of Google's cloud computing environment to host the website. We also used the HTML framework, Bootstrap, to ensure that MindReader can run on any device, even a phone.

These features create both a better experience for the player and make it much easier to analyze data as it is collected. The ability to collect and analyze data on the behaviour of people playing the MindReader game has attracted both computer science researchers from Microsoft and Cognitive Science Professors. The rest of this paper outlines the most important aspects of the MindReader project so to make it easier for future scientists to understand and contribute their own research. This is broken up into 4 major sections. First, we explain the various web functionalities of the front end from the perspective of a player. Second, we explain how the back end of the website is hosted and how its data is managed. Third, we give a in depth description of the AI algorithm used within MindReader as well as how it was coded. Finally, we explain how researchers can replicate the complete MindReader project locally on their computer and how to contribute to the project.

Chapter 2

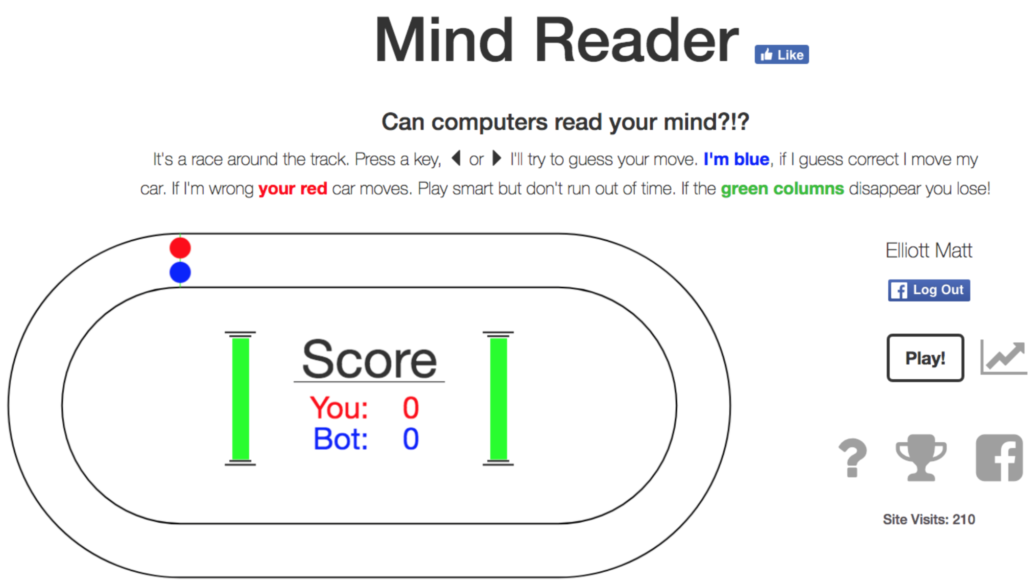
Front End: Web Interface

The MindReader website started as a single HTML document, but as the complexity and features of the game increased it has turned into a complex network of interconnected HTML pages. Additionally there is also a Facebook group that has it own page on Facebook.com . In this chapter we describe the functionalities of the most important webpages: the site's main menu, all the pages necessary for a user to play a game, and the user's profile page.

2.1 Main Menu

The main menu, or home page, is the first page that a player sees when they go to the MindReader website at "www.mindreaderpro2.appspot.com." From the main menu, a player can access all the functionalities of the website. There are two versions of the main menu, one for users who already have an account on the website and another simplified web page for users who don't have an account.

When a user visits the home page cookies are used to see whether or not the user has already created a user profile with their Facebook ID. If the user does not have an account the website displays the simplified version of the home page that contains only two buttons. One button allows the player to instantly start playing a game. A second smaller button allows the user to access the more complicated main menu for signed in users. The simplified web page makes it so that when a player first visits MindReader they are not overwhelmed by an overly confusing


Mind Reader

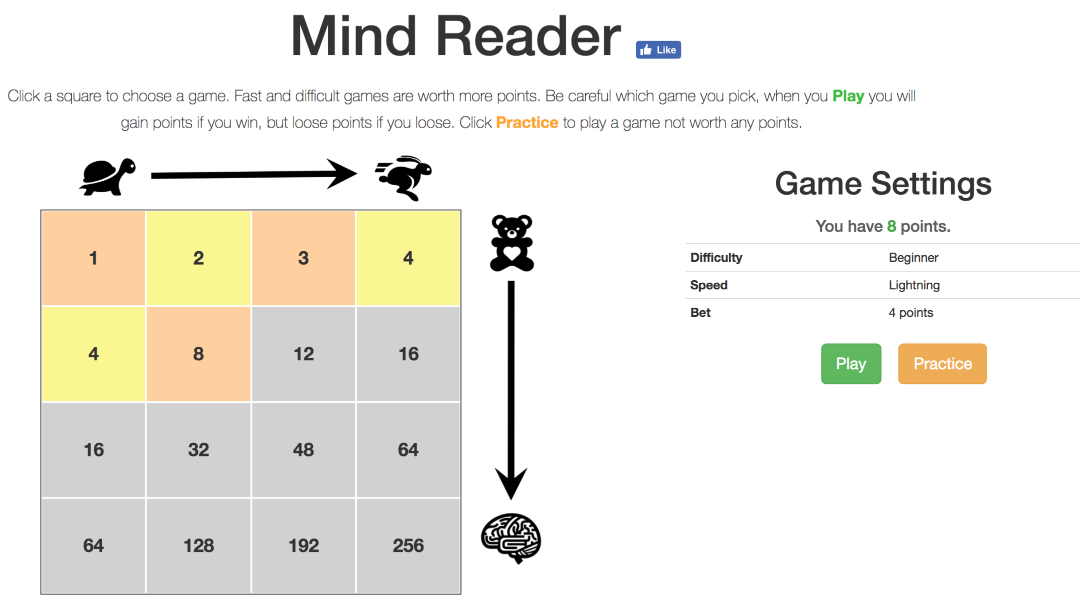
Can computers read your mind?!?

It's a race around the track. Press a key, ◀ or ▶ I'll try to guess your move. **I'm blue**, if I guess correct I move my car. If I'm wrong **your red** car moves. Play smart but don't run out of time. If the **green columns** disappear you lose!

Figure 2.1: Main Webpage of MindReader

interface. All they have to do is click one button and then they instantly start playing a game.

After a player has created an account, the website presents the bigger, fully functional, main menu. There are 7 buttons in total on this main menu. The most important button, “play,” is what a user clicks to start playing a game. Next to the “play” button is another button that brings up the player profile. From the player’s profile a user can view statistics about their previously played games and manage their account. Another button, the trophy icon, brings the user to a high scoreboard. There is an “about” button which give a more thorough explanation of the game as well as it’s history. Finally, there are 3 buttons that control the Facebook features of the website, a “like” button, an account logout button, and button that brings players to the game’s Facebook page.



Mind Reader

Click a square to choose a game. Fast and difficult games are worth more points. Be careful which game you pick, when you **Play** you will gain points if you win, but loose points if you loose. Click **Practice** to play a game not worth any points.

1	2	3	4
4	8	12	16
16	32	48	64
64	128	192	256

Game Settings

You have 8 points.

Difficulty	Beginner
Speed	Lightning
Bet	4 points

Figure 2.2: Interface for Game Setup

2.2 Playing a Game

2.2.1 Game Setup

When a user creates an account they are given a certain amount of points which they can use to play games. Before each game the player makes a bet against the computer using their points. If the player wins the game they earn the amount of points they bet, however if they lose the game they lose that amount of points. A game's difficulty is based on the amount that the player bets. The more points a player bets, the harder it is to beat the AI (called bot). The reason why we increase the difficulty for higher bets is because we want individuals who can consistently win against bots with high difficulty to be given more points.

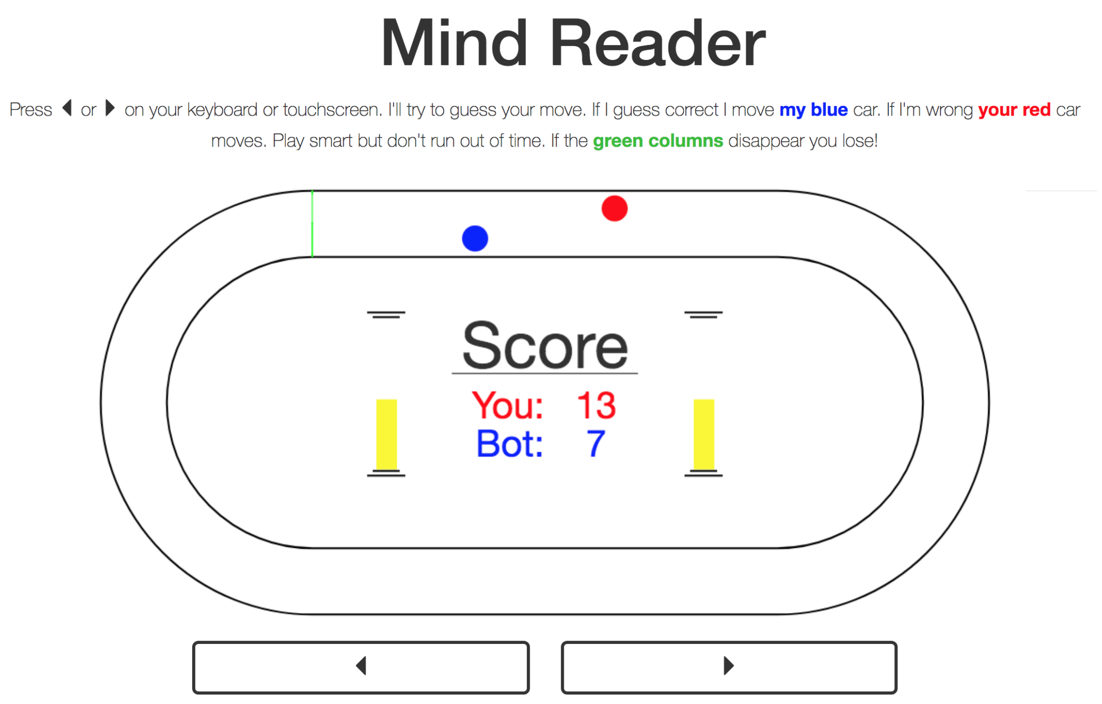
There are 2 different ways in which we make it more difficult to defeat a bot. The first is by making the bot "smarter" through the use of a more complex algorithm. This is done my making the tree depth of the algorithm larger (see algorithm section). The second way to make the game harder is to force the player to move faster. This is done through the use of ever decreasing "timer columns" during the game (See "Competing Against the AI"). Empirically we have found that forcing a player to move fast causes them to be less random and thus more likely to loose.

All of the various game difficulties are expressed on the "game setup" screen using a matrix (see Figure 2.2). The number inside of each box is the bet size, where a large bet corresponds to a more difficult game. As we move across across columns the game's speed becomes perpetually faster. As we move down rows the bot's algorithm becomes smarter.

How much a player can bet depends on how many points they have. We don't want a player's number of points to ever become negative. Thus we make it so that a player can never bet more points than what they already have. This is handled on the interface by "graying out" the bets that a player cannot make, which means that these types of games cannot be played for money. A user can play a game in one of two modes the "play" mode and the "practice" mode, which are indicated by the green and orange buttons respectively. When the "play" mode is selected that means that that results from the game are recorded and that the player either gains or loses points depending on the game outcome. If a player chooses a "practice" game the results from the game are not recorded. A player can even play games which are "grayed out" in practice mode, because the game outcome does not affect their final score. The practice mode allows any player to see what it is like to play against a bot at harder difficulties.

2.2.2 Competing Against the AI

After a player chooses a game and clicks either "play" or "practice" the game instantly starts. We subtract some points from a user's profile when they start a game so that players don't simply restart the webpage when they think they might lose. A countdown occurs so that the player has a little bit of time to get ready and then the game starts. The game consists of a racetrack where the user is a red dot and the bot is a blue dot. The two dots race around to track and whoever makes it around the track first wins. As this interface is just an implementation of the matching pennies game, the movement of the cars is dependent on whether or not the bot can guess a player's move. A player chooses a move in one of two

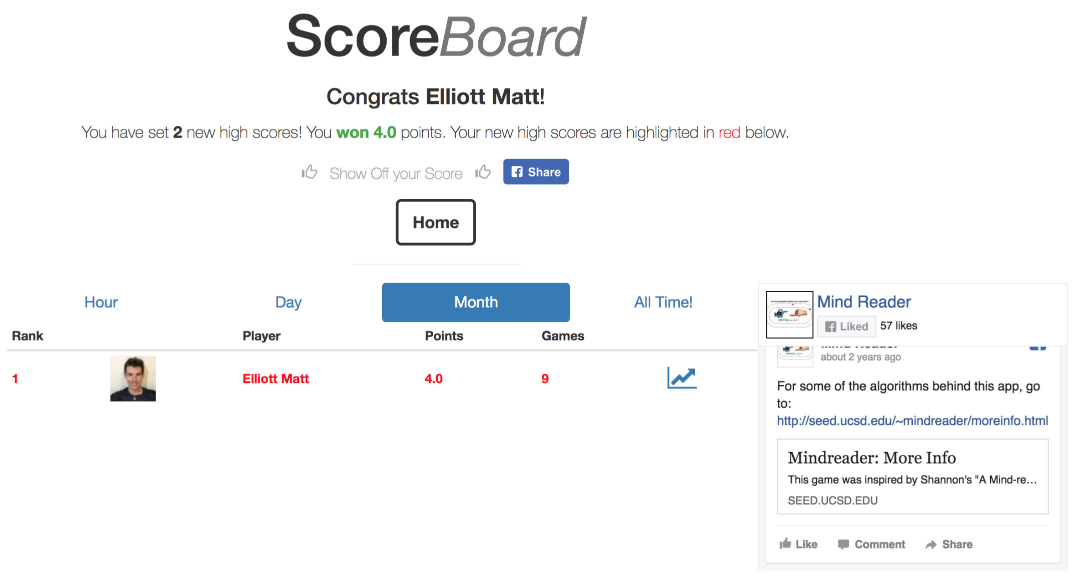

Mind Reader

Press ◀ or ▶ on your keyboard or touchscreen. I'll try to guess your move. If I guess correct I move **my blue** car. If I'm wrong **your red** car moves. Play smart but don't run out of time. If the **green columns** disappear you lose!

Figure 2.3: Interface when Playing Game

ways, by either using the ← and → keys on their computer, or if they are using their tablet or phone, by using the ← and → buttons provided on the interface.

One can think of the ← and → keys as corresponding to 1 and 0 in the matching pennies game. A point is rewarded to the player if the bot incorrectly guesses the player's move, otherwise a point is given to the bot. For the race to finish either the player or bot must score 100 points. To give the game a sense of excitement and increase its difficulty there is a timer that is perpetualy loosing time. If the times goes to zero the player loses. The timer is indicated by the two columns on the interface. The columns change color depending on how much time is left: green means plenty of time, yellow means caution, and red means the player is about to run out of time and loose. The player's car also starts to blink when they are about to run out of time.


ScoreBoard

Congrats Elliott Matt!

You have set **2** new high scores! You **won 4.0** points. Your new high scores are highlighted in red below.

Show Off your Score Share

Home

Hour Day Month All Time!

Rank		Player	Points	Games
1		Elliott Matt	4.0	9

Mind Reader
Liked 57 likes
about 2 years ago
For some of the algorithms behind this app, go to:
http://seed.ucsd.edu/~mindreader/moreinfo.html

Mindreader: More Info
This game was inspired by Shannon's "A Mind-re…
SEED.UCSD.EDU

Like Comment Share

Figure 2.4: Scoreboard Showing Results for a Winning Game

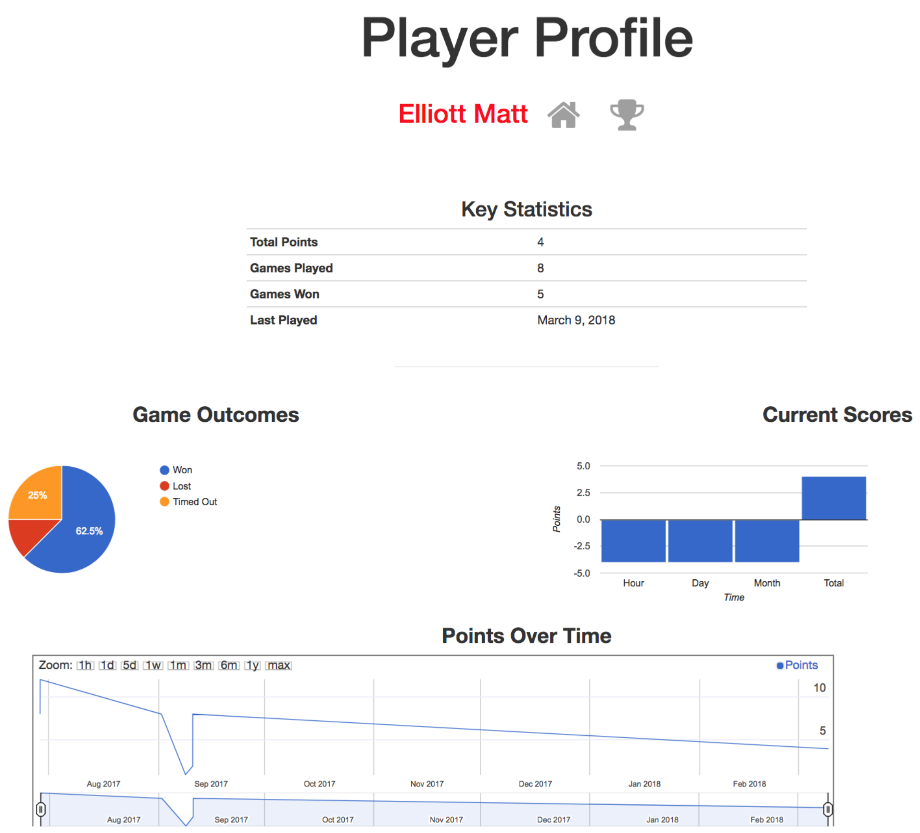
2.2.3 Game Results and Scoreboard

When a game is finished the player is brought to a new page where they are told the results of the game and are also shown the current high scores. This is the scoreboard page. If someone wants to view high scores without playing a game, there is also a button on the main menu that links to the scoreboard. We added the scoreboard to give the game a competitive edge that incentivizes individuals to keep playing the game. To the right of the high scores there is a news feed from the MindReader Facebook group that displays the group's latest messages.

If a player has just finished a game the scoreboard page tells them the results from the game. Specifically it tells them whether or not they won or lost; if they lost, then the reason why; the number of points that are subtracted/added to their account; and finally whether or not they set any new high scores, and if so, how many. If a user accesses the scoreboard from the main menu, these results are not displayed.

Underneath the player's results is all of the games high scores. The high scores are updated in real-time and show the 10 highest scores for each month/day/hour as well as the highest scores of all time. A player's Facebook page is linked to the high score board to deter cheating in the game. For each player on the scoreboard it is possible to view the player's game profile as well as visit their Facebook page. If a player is uncomfortable with having this information publicly available on the "player profile" page they the can delete their profile to remove themselves from the scoreboard.

2.3 Player Profile



Player Profile

Elliott Matt

Key Statistics

Total Points	4
Games Played	8
Games Won	5
Last Played	March 9, 2018

Figure 2.5: Game Statistics from a Player's Profile

The player profile is where people can view game statistics about themselves. It is also where a user can delete their player profile. To delete their account a player need only to click the "delete account" button and then confirm that they truly want to remove their profile and all data.

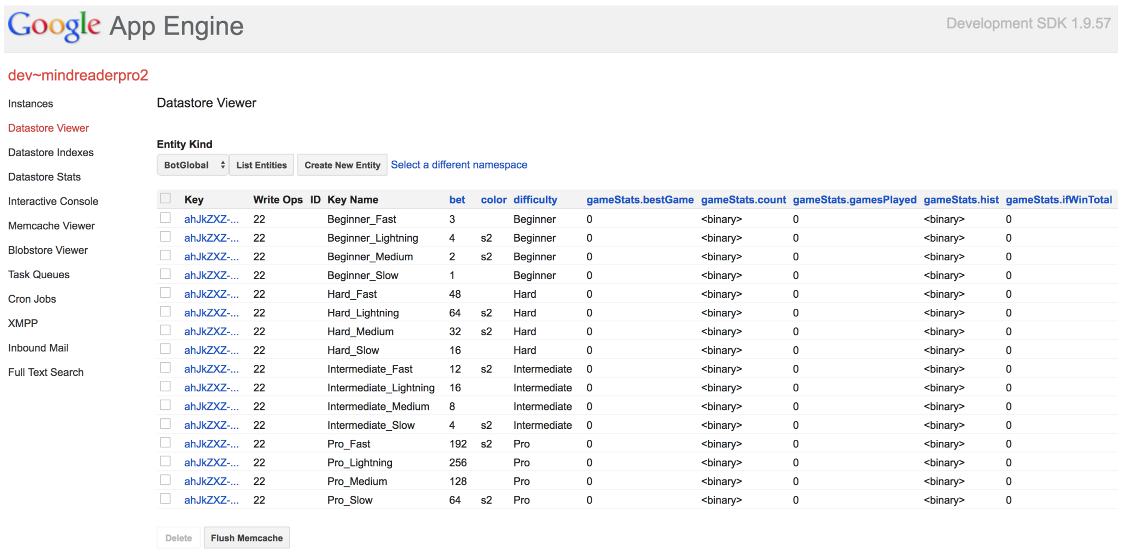
Information about a player's previous games is currently displayed in 4 different sections of the player profile. Key statistics is a table that provides the player with basic information like their current number of points or how many games they have played. Game outcomes has a pie chart that shows the percentages of various game outcomes, like “won”, “lost”, or “ran out of time”. Current scores displays the user’s total points for each hour/day/month and all time. Points over time is an interactive time series graph of the players points. All of the graphical displays were created using google’s “charts” api for websites.

Chapter 3

Back End: Data Management

In the previous chapter we saw how game data is used to provide many features that enhance the user's experience through the use of a scoreboard and a personal player profile. In this chapter we describe how data is stored, managed, and analyzed on the back end.

3.1 The gCloud Environment



Key	Write Ops	ID	Key Name	bet	color	difficulty	gameStats.bestGame	gameStats.count	gameStats.gamesPlayed	gameStats.hist	gameStats.ifWinTotal
ahJkZXZ-...	22		Beginner_Fast	3		Beginner	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Beginner_Lightning	4	s2	Beginner	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Beginner_Medium	2	s2	Beginner	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Beginner_Slow	1		Beginner	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Hard_Fast	48		Hard	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Hard_Lightning	64	s2	Hard	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Hard_Medium	32	s2	Hard	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Hard_Slow	16		Hard	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Intermediate_Fast	12	s2	Intermediate	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Intermediate_Lightning	16		Intermediate	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Intermediate_Medium	8		Intermediate	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Intermediate_Slow	4	s2	Intermediate	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Pro_Fast	192	s2	Pro	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Pro_Lightning	256		Pro	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Pro_Medium	128		Pro	0	<binary>	0	<binary>	0
ahJkZXZ-...	22		Pro_Slow	64	s2	Pro	0	<binary>	0	<binary>	0

Figure 3.1: The App Engine Local Host Admin Console

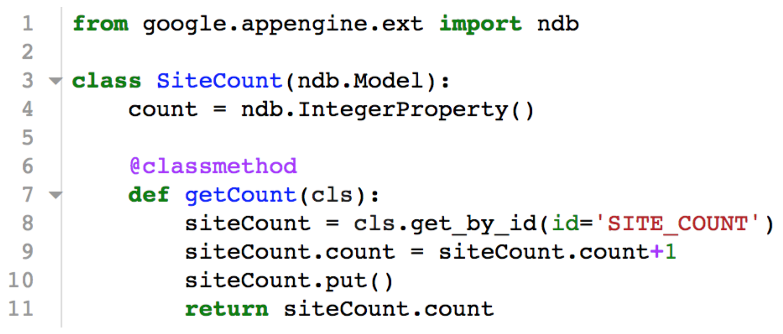
The MindReader website was built using Google's cloud computing environment called App Engine. App Engine has a very user friendly environment that makes it possible to rapidly build, test, and deploy web based apps. A key feature of App Engine is that the computational resources of your application can grow or shrink in real time demanding on how many people use the website. This "automatic scaling" feature occurs by default and is never something that has to be managed by the developer. Another benefit is the amount of money you pay each month depends only on the computational resources that are used. For this reason the website can be hosted for free during the development phase since this requires minimal computational resources.

Another key feature of App Engine is that you can run the MindReader website locally on your computer for testing and development. When running MindReader locally there is also an admin website that is automatically created by App Engine which is used to view, create, and edit data structures (see Figure 3.1). Using a single command in the terminal, the MindReader app is pushed to the cloud and available publicly at "www.mindreaderpro2.appspot.com". Google provides an administrative website called the developer console which is used to monitor and edit various aspects of the the publicly available MindReader site. The developer console is where data can be viewed or downloaded, where different versions of the website can be launched, and where things like payment settings are handled.

3.2 Data Structures

Since we are are using Google's App Engine framework, all the data is stored within Google's Cloud Datastore which is accessed by using Google's "new database library" (ndb). After importing the ndb package into our python code we can write and retrieve data from Cloud Datastore. The ndb framework can basically be thought of as Google's version of MongoDB. It is a NoSQL framework that stores data as JSON objects and, when retrieved, returns python objects with the same structure as the JSON object. The python object that is retrieved from the JSON object can also have prespecified functions that utilize the data. The process of coding an ndb object in python is almost identical to the steps used in

MongoDB. In a python file you import the ndb library and then to create a new data structure you extend the ndb.Model class with the data features that you would like to have. An example of this is shown below in Figure 3.2 .



```python
from google.appengine.ext import ndb

class SiteCount(ndb.Model):
    count = ndb.IntegerProperty()

    @classmethod
    def getCount(cls):
        siteCount = cls.get_by_id(id='SITE_COUNT')
        siteCount.count = siteCount.count+1
        siteCount.put()
        return siteCount.count
```

Figure 3.2: Example of Data Structure Written with NDB Library

3.3 Data Analysis

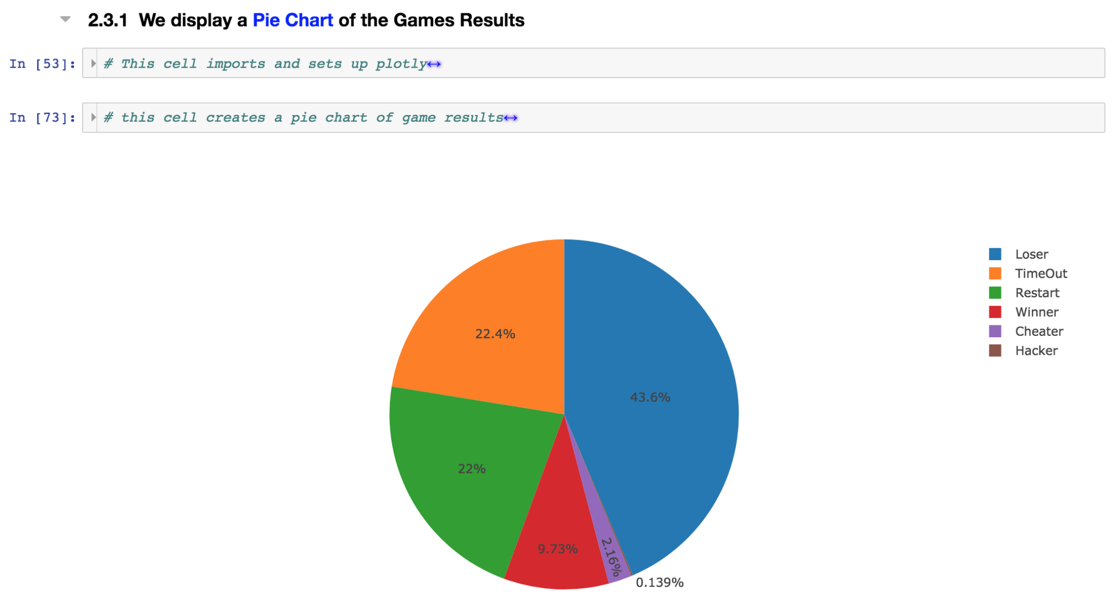
3.3.1 The Jupyter Framework

In MindReader's current form data analysis was done offline by downloading data to a local computer from Google's Cloud environment. The data was formatted into a CSV file and then analyzed using Jupyter Notebooks. The benefit of using Jupyter notebooks for data analysis is that we can use markdown text cells to clearly describe what the code does and then have cells that run code and display it's output.

The steps for downloading data from Google's cloud platform and then formatting the data offline is rather involved. Luckily Google now provides a much easier way to access data, through Jupyer's PyLab live on Google Cloud. The latest version of Google Cloud allows you to create a Jupyter environment inside the project of the MindReader app. From the online Jupyter environment the website's data can be accessed using the exact same commands as those used on the back end of the website itself. Another benefit of the online environment is that since we are accessing the cloud's live database, it is possible to instantly retrieve and analyze data from the most recently played games.

3.3.2 Simple Data Summary

After completing our first version of the MindReader website. We marketed it out the public using UCSD's CSE mailing list as well as YouTube. From this we collected data on 7925 games that people played. We use the analysis from these games for the following basic summary analysis of the MindReader app. From the pie chart below we see that the vast majority of people were not able to beat MindReader's AI algorithm. Players were able to beat the AI in only 10% of the games. While MindReader's AI defeated players in roughly 45% of the games, it is safe to assume that most of the games where the player ran out of time or decided to restart (likely gave up) are also instances where the AI defeated the player.



2.3.1 We display a Pie Chart of the Games Results

```
In [53]: # This cell imports and sets up plotly↔

In [73]: # this cell creates a pie chart of game results↔
```

Figure 3.3: Pie Chart of Game Results

Another interesting observation from the data is that people tend to play better against the AI when they choose to move more slowly. This is seen in Figure 3.4 on the next page, where we see an upward trend between game score and average time per move. This has interesting implications in cognitive psychology in that people are able to play more randomly when they give their mind time to forget the previous moves they have played [10].

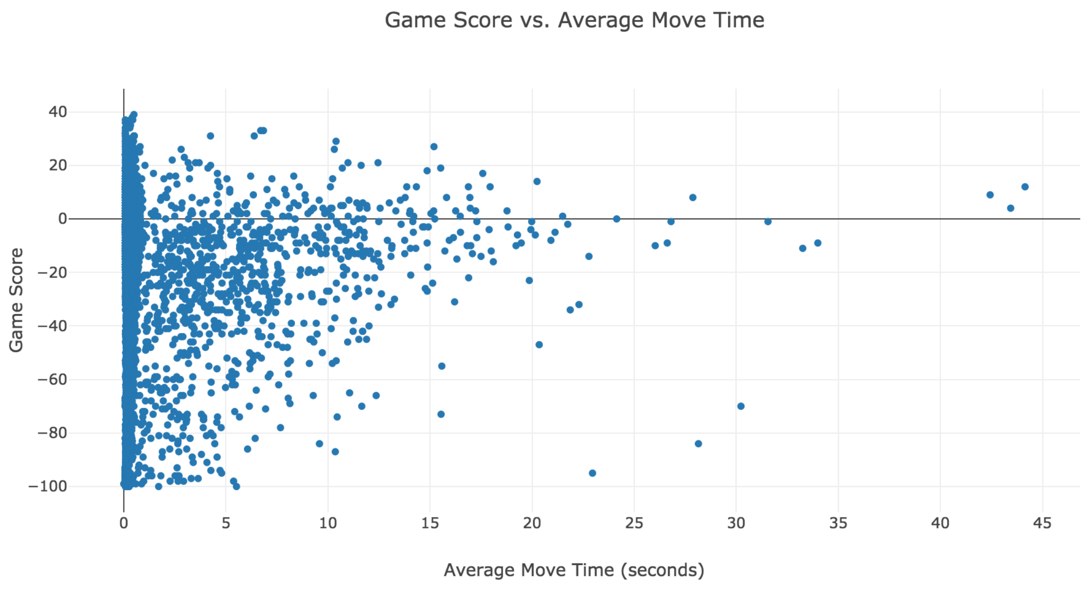


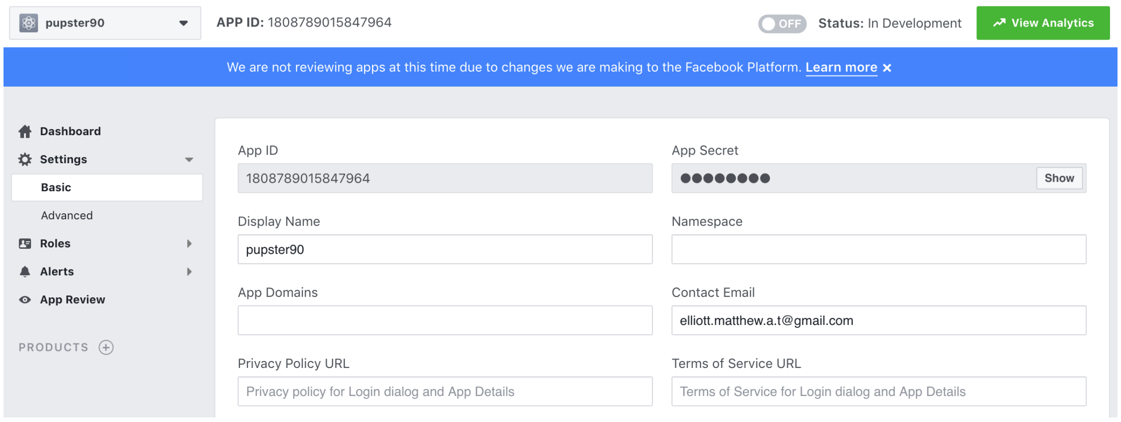
Figure 3.4: Game Score vs. Average Time per Move

3.4 Facebook API

In order for the Facebook functionality to work on the front end of the MindReader website it is important to register the website on Facebook's interface for developers, which is aptly called, "Facebook for Developers." By registering MindReader's domain name on Facebook for Developers, we satisfy the requirements necessary to use the Facebook api. The Facebook api is what allows people to "like" the MindReader site and also allows MindReader to collect data from people's Facebook accounts. MindReader uses this information to enhance the user's experience through things like the high scoreboard. We also use each player's unique identifier from Facebook when performing data analysis for research purposes. The MindReader website has a consent waiver where we clearly explain to players that their data is being using for academic purposes and that is not being used for marketing.

To register the MindReader site as a Facebook app go to the Facebook for Developers website and create an account, then create a new app within the account. Inside the settings section of the app you can then enter the domain name of the MindReader site in order to have it officially recognized by Facebook. On the

Facebook for Developers site it is also possible to manage settings and permissions for the MindReader app, however, for the version of MindReader that we create the default Facebook app settings was all that was needed. Figure 3.5 below shows what it looks like to be be logged into the Facebook for Developers interface.



APP ID: 1808789015847964

OFF Status: In Development

We are not reviewing apps at this time due to changes we are making to the Facebook Platform. Learn more

App ID: 1808789015847964

App Secret: ●●●●●●●●

Display Name: pupster90

Namespace:

App Domains:

Contact Email: elliott.matthew.a.t@gmail.com

Privacy Policy URL: Privacy policy for Login dialog and App Details

Terms of Service URL: Terms of Service for Login dialog and App Details

Figure 3.5: The Facebook for Developers Console

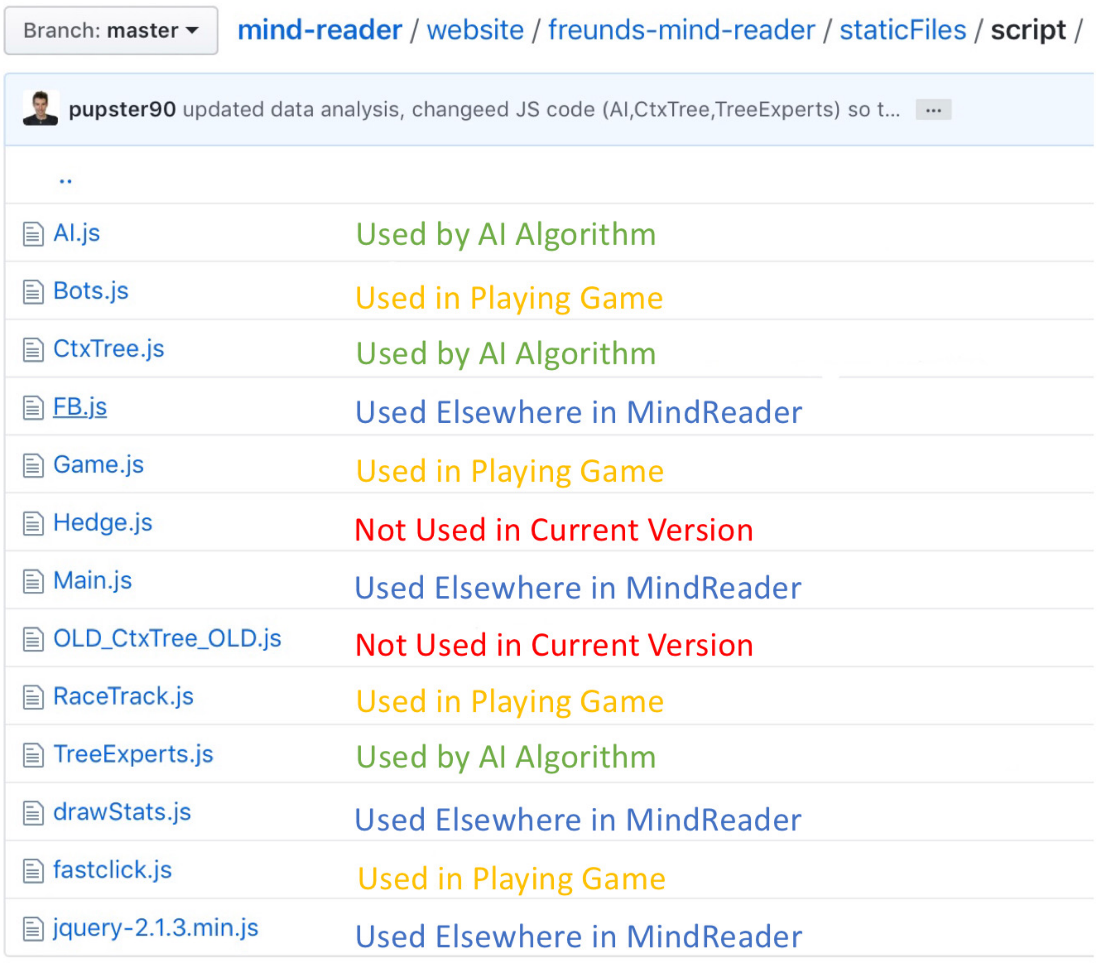
Chapter 4

AI Algorithm

In this chapter we give an in depth description of the AI algorithm that is used to predict players' moves and the code that was used to implement it. The goal of this chapter is not just to give a theoretical overview of the model, but also to provide the knowledge needed to edit the algorithm for personal research. All of the code for the AI algorithm is written in JavaScript and runs on the front end of the MindReader website during the game. The algorithm has a complicated structure so it's code was broken up across multiple JavaScript files. Each files represents a distinct class, when combined together these files create the complete MindReader game. For this reason a natural approach for understanding both the code and theory is to explain each JavaScript class one file at a time. Each following section represents (mostly) a JavaScript class that is a piece of the larger MindReader game. Together they are MindReader's AI.

4.1 The JavaScript Files

The JavaScript files for MindReader are located inside the website's folder called "script". All of these files are used to control the MindReader game except for "FB.js" (Facebook functionality), "drawStats.js" (player profile), and "Main.js" (main webpage). Of the remaining 7 files, 3 of them are used to create the AI algorithm and the other 4 are used to manage other aspects of the video game. For the 4 files not related to the AI algorithm: "Bots.js" controls the game setup page, “Racetrack.js” creates the visualization of the game, “Game.js” manages game processes unrelated to the AI (player’s score, updating the timer, posting results), and “fastClick.js” is a helper script that makes the game run efficiently on mobile devices. Below is a figure that summarizes the functionality of each JavaScript file. The rest of this chapter is devoted to the JavaScript files that relate to the AI algorithm.



Branch: master ▾ mind-reader / website / freunds-mind-reader / staticFiles / script /

pupster90 updated data analysis, changeed JS code (AI,CtxTree,TreeExperts) so t... ···

..

AI.js	Used by AI Algorithm
Bots.js	Used in Playing Game
CtxTree.js	Used by AI Algorithm
FB.js	Used Elsewhere in MindReader
Game.js	Used in Playing Game
Hedge.js	Not Used in Current Version
Main.js	Used Elsewhere in MindReader
OLD_CtxTree_OLD.js	Not Used in Current Version
RaceTrack.js	Used in Playing Game
TreeExperts.js	Used by AI Algorithm
drawStats.js	Used Elsewhere in MindReader
fastclick.js	Used in Playing Game
jquery-2.1.3.min.js	Used Elsewhere in MindReader

Figure 4.1: Summary of How each JavaScript File Relates to AI Algorithm

4.2 Algorithm Overview: AI.js

To better understand MindReader's AI algorithm let's start by considering what the algorithm's input is and what is its intended goal. The goal of the algorithm is to return a number, 1 or 0, which is the AI's best guess of the player's next move. In order to make a guess the algorithm can utilize all of the information that is being collected by the game. Specifically the AI algorithm uses the player's previous moves to make its best guess for what the player will do. This is the input to our algorithm.

The player's previous moves are stored in a linked list, with the beginning of the list being the player's most recent move and the end of the list being their last move. The algorithm first looks at the player's most recent move as this is intuitively most closely related to their next move. It then iteratively goes back in time looking at the next most recent move. This procedure is shown in the Figure 4.2 below.

...	x_{t-4}	x_{t-3}	x_{t-2}	x_{t-1}	x_t
...	0	1	1	0	?

Figure 4.2: AI Iteratively Considers the Most Recent Last Move

A key aspect of this algorithm is that it is probabilistic. The algorithm return a probability between $[0, 1]$ of the player's next move being 1. A higher probability means the that the algorithm believes the player is more likely to guess one. We could have the algorithm always return 0 or 1 depending on whichever element is more likely, however, if a player finds just 1 sequence that beats the AI the player could then play that exact same sequence to win over and over again. To avoid this we make the algorithm *nondeterministic* by randomly selecting 0 or 1 based on the Bernoulli probability that the algorithm outputs. This can be seen in the code by looking at the AI.js class's return output for its "predict" function (line 43). When doing data analysis we remove the nondeterministic feature of the algorithm to insure consistency in results.

4.3 The Context Weighted Tree: TreeExperts.js

In the previous section we said that the AI algorithm uses the list of a player's previous moves to predict the next move, but how is this done? In this section we introduce the Context Weighted Tree, a data structure that intuitively leads to a prediction algorithm for the next move [16]. Suppose a player's last move was 0 and the move before that was 1, we write this as $[0, 1]$. First, we could ask ourselves, "Given a player's current move is 1, what is the probability that their last move was 0". Second, we could ask, "Given a player's current move is 1, and the move before that is 0, what's the probability that the move before that is 1?". We can continue to ask this question going through all the player's previous moves. Notice that the probability in question, can easily be solved by considering the the number of occurrences of the given statement.

There is an intuitive visual structure that we can use to represent the probabilities from asking this question for each previous move. This structure is a binary tree, where each node represents the probability of asking that question for the path that leads to that node. For instance, the red path in the figure below represents the case where the player's most recent two moves were $[0, 1]$. The red leaf node would then contain a probability value that would answer the second question asked above.

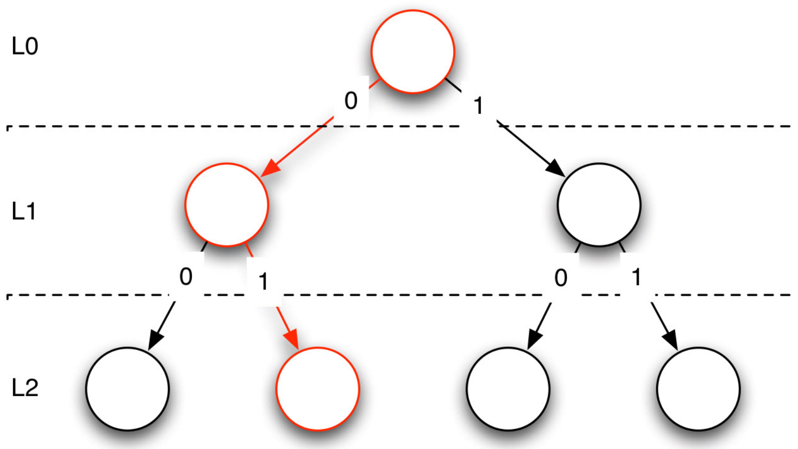


Figure 4.3: Binary Tree, First Move is 1, Second Move is 0

One way to think about the probability of a given node is to give each child a weight relative to its probability. As data is obtained we then update the weights of that node's children so that the more probable node is weighted higher. We then have a weighted tree whose weight depend on the context of previous data, hence a Context Weighted Tree. But as we are given new data, how do we update these weights? The Hedge algorithm provides an intuitive system for both determining the probability of nodes using weights and for updating these weights. The Hedge algorithm is provided in the figure below.

Algorithm 1 Hedge(η)

```
$L_{i,0} \leftarrow 0, \forall i$ {cumulative loss}
initialize $w_{i,1}, \forall i$ {weight distribution, $\sum_{i=1}^{N} w_{i,1} = 1$}
**for** $i = 1, 2, \ldots$ **do**
  Each action $i$ incurs a loss $l_{i,t}$
  $L_{i,t} \leftarrow L_{i,t-1} + l_{i,t}$ {expert loss}
  $w_{i,t} = w_{i,1} \cdot e^{-\eta L_{i,t}}$
  $p_{i,t+1} \leftarrow \frac{w_{i,t}}{\sum_{j=1}^{N} w_{j,t}}$ {update probability distribution}
**end for**
```

Figure 4.4: The Hedge Algorithm used at Each Node

Hedging is a general ensemble algorithm for combining expert opinions. In this case each node contains its own hedging algorithm, each of its children (and their correspond paths) is considered an expert. The loss is determined by the expert's ability in determining the correct outcome (0 or 1) and then the weights are adjusted according to the losses through exponential weighting. The new weights are then used to calculate the new probability of each child path.

4.4 Updating the Context Tree: CtxTree.js

In the MindReader game a player can have up to 199 moves, expressing all the node probabilities (or weights) of a tree of that depth would require too much computation. However, for any given particular game only a tiny amount of that tree is ever explored. Instead of expressing the entire tree we could start the tree with just one root node. Each time a path that was never seen before is explored we can create new children nodes from the last (previously explored) leaf. Creating new child nodes means the algorithm is trailblazing along a new path that has never been seen in the data. Since there is no data to suggest which of the two new child nodes is more likely we initialize each of them with the same weight of 1. This efficiently creates a Context Weighted Tree of only the paths that have been seen within the game.

We can develop a little more intuition about how this works using a Bayesian perspective. In the Bayesian mindset we can imagine that we have an infinitely large binary tree where each node contains probabilities as described in the previous section. Initially when a path is untraveled the nodes have no data to update their values and thus they are simply equal to their initially defined prior distribution. If we use a diffuse prior where each child node has the same odds, say $\text{Beta}(1, 1)$, we know that leaves of any unexplored branch will have 50/50 probability. Since we already know the probabilities of unexplored paths, we thus do not need to waste data expanding out that (infinite) portion of the tree. Instead we focus on the parts of the context tree where the priors have been updated by data. It can actually be shown that by just changing a few constants in the algorithm we have a consistent Bayesian framework for representing the context tree.

While the Bayesian framework yields an intuitive representation of the model, the actual methodology used for updating nodes and making predictions is derived from a loss based model across multiple decisions trees. In this framework we find a tree which is "not much worse" than the best decision tree. One could do this by applying a hedging algorithm across all possible prunings, however, such a procedure is not computationally feasible [20]. Hembold and Schapire found an efficient implementation of the procedure, which is what our algorithm uses [15].

In our implementation the JavaScript's predict function, "get_pred_wt", and the update function, "update_wt" are both implemented recursively. The recursion is over the path of all the player's previous moves, starting from the most recent (the tree's root). The next page has the prediction and updating algorithm.

Algorithm 2 predict(node,$x_1, x_2, \ldots, x_n$)

```
if $node.firstVisit$ then
  return 1/2
else if $node.isLeaf$ then
  return $w_0 \cdot 0 + w_1 \cdot 1$
else
  if $x_n = 0$ then
    next node ← child at 0
  else
    next node ← child at 1
  end if
  return $w_0 \cdot 0 + w_1 \cdot 1 + w_2 \cdot$ predict(next node,$x_1, x_2, \ldots, x_{n-1}$)
end if
```

Figure 4.5: The Recursive Algorithm the AI uses to Predict a Move

Algorithm 3 update(node, outcome, $x_1, x_2, \ldots, x_n$)

```
$l_i \leftarrow |\text{outcome} - p_i|, \forall i$
$l_A \leftarrow w_0 \cdot l_0 + w_1 \cdot l_1 + w_2 \cdot l_2$
$R_i \leftarrow R_i + (L_a - l_i), \forall i$
$p_i \leftarrow \frac{\frac{[R_i]_+}{c} \exp\left(\frac{([R_i]_+)^2}{2c}\right)}{\sum_{j=0}^{2} \frac{[R_j]_+}{c} \exp\left(\frac{([R_j]_+)^2}{2c}\right)}$
if $node.isLeaf = false$ then
  if $x_n = 0$ then
    next node ← child at 0
  else
    next node ← child at 1
  end if
  update(next node, outcome, $x_1, x_2, \ldots, x_{n-1}$)
end if
```

Figure 4.6: The Recursive Algorithm the AI uses to Update Itself

Chapter 5

Conclusion

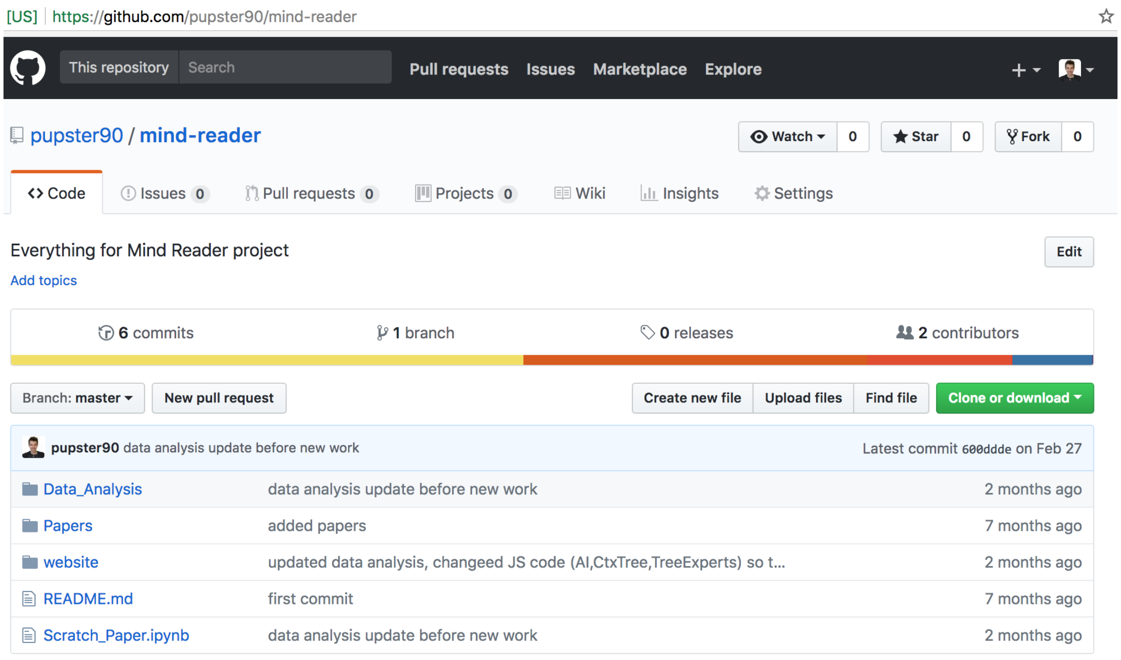
In this paper we explained the inner workings of everything that went into making our finished version of the MindReader project. The most prominent piece of MindReader is the website that people use to play the game, however, it is not the only piece. We described how the the data from MindReader is handled and analyzed. We also showed how Google and Facebook's cloud based developer consoles are instrumental in running/managing the MindReader site. Now that MindReader has been created it is up to future work to utilize the technology to construct meaningful research.

5.1 Future Work

There is considerable interest in using the MindReader site to analyze cognitive behaviour. Prof. Aaron Schurger at the French Institute of Health and Medical Research believes MindReader could be used in neuroscience to research the temporal uncertainty in perception and action [19]. Furthermore, Microsoft's Principal Researcher, Robert Schapire, is also interested in how MindReader's AI algorithm performs against human combatants. With the correct team of dedicated researchers the MindReader project can lead to a publication with significant impact across interdisciplinary fields.

5.2 Contributing to MindReader

We have created the MindReader project with the intention that it will later be used by future researchers. For this reason we tried to make it as easy as possible to start running and contributing to the MindReader project through the use of Github and Docker. All the source code and supplementary material for MindReader has been posted publicly on Github at the URL, "github.com/pupster90/mind-reader". The MindReader project can be downloaded from there and contributions can be posted to the Github repository.



[US] https://github.com/pupster90/mind-reader

pupster90 / mind-reader

Everything for Mind Reader project

6 commits | 1 branch | 0 releases | 2 contributors

pupster90 data analysis update before new work — Latest commit 600ddde on Feb 27

Data_Analysis	data analysis update before new work	2 months ago
Papers	added papers	7 months ago
website	updated data analysis, changeed JS code (AI,CtxTree,TreeExperts) so t...	2 months ago
README.md	first commit	7 months ago
Scratch_Paper.ipynb	data analysis update before new work	2 months ago

Figure 5.1: The Github page, "github/pupster90/mindreader", for MindReader

While MindReader's source code is informative, running the project locally on a computer requires downloading many different packages and frameworks like Jupyter and Google's gCloud. Rather than having people struggle for days trying to setup their computer appropriately (if they figure it out at all), we have instead built a virtual machine using docker that can be used to run MindReader. A link to the docker image's URL can be found in the README.md file on the Github page. When we were working on MindReader we made sure to always use the exact same docker image as the one that is provided on docker hub. This guarantees that the virtual machine will run appropriately on anybody's computer after it is downloaded.

If Docker is already installed one can get the MindReader website and the data analysis back end running on their computer within a matter of minutes by typing just 3 lines of code in their computer terminal. The docker hub pages give very straightforward step by step instructions of exactly which lines to type in the terminal and explains what they mean. We are excited to see what new research and contributions will spring from from the MindReader project!

Bibliography

[1] Nash, J.F., The Bargaining Problem. Econometrica, 1950. 19(2): p. 155-162.

[2] Wagenaar, W.A., Generation of Random Sequences by Human Subjects: A Critical Survey of the Literature. Psychol Bull, 1972. 77(1): p. 65-72.

[3] Tune, G.S., A brief survey of variables that influence random generation. Percept Mot Skills, 1964. 18: p. 705-710.

[4] Eliaz, K. and A. Rubinstein, Edgar Allan Poe's riddle: Framing effects in repeated matching pennies games. Games and Economic Behavior, 2011. 71(1): p. 88-99.

[5] Persaud, N., Humans can consciously generate random number sequences: A possible test for artificial intelligence. Medical Hypotheses, 2005. 65(2): p. 211-214.

[6] Tervo, Dougal G.R., Behavioral Variability through Stochastic Choice and Its Gating by Anterior Cingulate Cortex. Cell, 2014. 159(1): p. 21-32.

[7] Baddeley, A.D., The capacity for generating information by randomization. Quarterly Journal of Experimental Psychology, 1966. 49A(1): p. 5-28.

[8] Jahanshahi, M., Random number generation as an index of controlled processing. Neuropsychology, 2006. 20: p. 391-399.

[9] Baddeley, A.D., Random Generation and the Executive Control of Working Memory. Quarterly Journal of Experimental Psychology, 1998. 51A(4): p. 819-852.

[10] Schurger, A., J. Sitt, and S. Dehaene, An accumulator model for spontaneous neural activity prior to self-initiated movement. PNAS, 2012. 109(42): p. E2904-E2913.

[11] Hagelbarger, D.W., SEER, A SEquence Extrapolating Robot. IRE Trans Elec Comput, 1956. 5(1): p. 1-7.

[12] Shannon, C.E., A mind-reading (?) machine. Bell Laboratories technical report, 1953.

[13] Freund, Y. and R. Schapire, A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. Journal of Computer and System Sciences, 1997. 55(1): p. 119-139.

[14] Littlestone, N. and M.K. Warmuth, The weighted majority algorithm. Information and Computation, 1994. 108(2): p. 212-261.

[15] Helmbold, D.P. and R. Schapire, Predicting nearly as well as the best pruning of a decision tree. Machine Learning, 1997. 27(1): p. 51-68.

[16] Willems, F.M.J., Y.M. Shtarkov, and T.J. Tjalkens, The context tree weighting method: Basic properties. IEEE Trans Info Theory, 1995. 41(3): p. 653-664.

[17] Edgar Allan Poe. The Purloined Letter (Tale Blazers). Perfection Learning, 1980.

[18] Biaggi, Andrea. Normal Hedge in weighted trees. University of California, San Diego, MS Thesis. 2010.

[19] Schurger, Aaron, Neural Antecedents of Spontaneous Voluntary Movement: A New Perspective. Trends in Cognitive Sciences , Volume 20 , Issue 2 , 77 - 79.

[20] Cesa-Bianchi, Nicol'o , Freund, Yoav, Helmbold, David P., Haussler, David, Schapire, Robert E., Warmuth, Manfred K. 1993. How to use expert advice. In Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, pages 382391, 1993. To appear, Journal of the Association for Computing Machinery.