FIFTH AEON - A.I COMPETITION AND BALANCER

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

William Ritson

June 2019

COMMITTEE MEMBERSHIP

TITLE: Fifth Aeon - A.I Competition and Balancer

AUTHOR: William Ritson

DATE SUBMITTED: June 2019

COMMITTEE CHAIR: Foaad Kosmood, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz J. Kurfess, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Alexander M. Dekhtyar, Ph.D.
Professor of Computer Science

ABSTRACT

Fifth Aeon - A.I Competition and Balancer

William Ritson

Collectible Card Games (CCG) are one of the most popular types of games in both digital and physical space. Despite their popularity, there is a great deal of room for exploration into the application of artificial intelligence in order to enhance CCG gameplay and development. This paper presents Fifth Aeon a novel and open source CCG built to run in browsers and two A.I applications built upon Fifth Aeon. The first application is an artificial intelligence competition run on the Fifth Aeon game. The second is an automatic balancing system capable of helping a designer create new cards that do not upset the balance of an existing collectible card game. The submissions to the A.I competition include one that plays substantially better than the existing Fifth Aeon A.I with a higher winrate across multiple game formats. The balancer system also demonstrates an ability to automatically balance several types of cards against a wide variety of parameters. These results help pave the way to cheaper CCG development with more compelling A.I opponents.

ACKNOWLEDGMENTS

Thanks to:

- My parents Lark and Marc Ritson for their unfaltering love and support.
- My advisor Foaad Khosmood for guiding me along each step in the process of writing this thesis.
- Andrew Guenther, for uploading this template

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This paper concerns the application of Artificial Intelligence to electronic card games, using the collectible card game Fifth Aeon for the research. Fifth Aeon is an open-source, multiplayer, browser-based, collectible card game written in Typescript by the author. It is intended to be both an enjoyable game and an open platform for researches to utilize to study games, A.I., and any other subject to which it might be applicable. The full source code for the game is available under the Apache-2 license at https://github.com/Fifth-Aeon. The games rules are similar to those found in popular games like Magic The Gathering and Hearthstone, but Fifth Aeon rules have been adjusted to address popular complaints about the overuse of randomness.

The first major application of Fifth Aeon is providing a platform for a game A.I. competition. The Fifth Aeon game was used as a platform for hosting an open competition where contestants could submit various A.I. agents that would play the game against each other in a tournament environment. It serves as an open, non-IP-laden platform to compete at one of the most popular types of games. In doing so, Fifth Aeon hopes to contribute to the goals of Game A.I. tournaments, which are discussed later in the paper.

The second major application of the platform is to balance cards procedurally. Collectible card games are very content driven and usually receive regular batches of new content referred to as sets. The process of creating new sets is very expensive and each new release threatens to dramatically upset the balance of the game, which can render it unappealing and cause users to leave. Thus, it is important to make

sure new cards fit well with existing ones. This may require extensive play-testing by experts and be very costly in terms of time and money. In order to expedite the process, this paper puts forth a mechanism for using A.I. to automatically balance a card against a set of existing cards.

1.1 Introduction to Automatic Balancing

This paper proposes a system that attempts to make the process of introducing balanced cards to a collectible card game easier. It does this by allowing designers to search for the most promising version of a card's potential parameters to ensure it has the desired amount of impact on the metagame.

The term balance can refer to several different properties of a game. This paper is primarily concerned with one specific meaning, the balance between asymmetric elements of a game. An asymmetric element is any part of the game that makes the game different for one player than another prior to the start of the game. For example, what race a player plays as in StarCraft will have a major effect on how they play the game even though it is chosen before the game starts. This definition of asymmetric game elements does not include differing game stats that result from the actions of players within the game itself, such as a different board state in chess after playing for several turns. Asymmetric elements are important because they may give one player an advantage over another before the game starts making it unfair. If the game is too unfair, it can hinder players ability to enjoy it, so designers must control for this.

Many classical games have little to no asymmetry. In Chess, both players have exactly the same pieces and options, however, a small amount of asymmetry arises from one player getting to go first. Newer games often have far more asymmetry. For example,

letting a player choose their faction, character, or starting location before a game begins. These additional differences between a player's initial game stats are added because they can make the game more fun. By allowing players to more choices, each of which creates a unique way of playing the game, game designers can create additional replay value and appeal to more players individual play-styles. The trade-off is that it makes the game harder to balance.

This balancing of asymmetric content becomes more and more complicated as the number of different configurations available to players increases. For example, Chess provides only two possible configurations, with players having either the first or second turn. Beyond this difference, all other options available to the players are the same. In a game like Starcraft, the designer must worry about three different factions giving six distinct match-ups [19]. In League of Legends, there are more than 100 characters and teams are composed of up to five characters each, giving billions of combinations [20] of characters. Ideally, a game designer would want to be able to show that none of these configurations give an advantage to one team so that the game remains balanced. But this becomes more and more difficult the more different configurations they have to test are balanced.

The size of the configuration space is most prominent in collectible card games. In these games, players form decks that typically contain somewhere between 30 and 100 cards, selected from a pool of hundreds or thousands of options. This results in mind-boggling vast numbers of possible combinations. The huge configuration space makes balancing individual cards, while accounting for all of the potential combinations, a huge amount of work for designers.

Understandably then, most collectible card games do not aim to be entirely balanced. Rather, they aim to reward a player's ability to strategically pick powerful combinations of cards through the process of "deck-building." This, in turn, encourages

players to collect additional cards to complete their decks. However, even when the goal is not to have all cards or decks be equally powerful, it is important that there are no "God tier" decks, which are so strong that they prevent any other decks from being competitively viable. Instead, designers seek to create a dynamic meta-game in which there are several viable deck archetypes, and in which each of the archetypes might have several versions. This meta-game is generally discovered by the game's community of high-level players experimenting with available decks to find the best combinations available. However, most collectible card games frequently release new cards in order to encourage purchases and remain fresh. It is therefore important to be able to verify that adding new cards to an existing game won't result in any deck becoming too powerful and creating an unhealthy meta.

The task of determining the effect of new cards is typically accomplished through a combination of applied designer knowledge and a large amount of play-testing. However, this play-testing may be prohibitively expensive for a small team. Due to the cost requirement, there is a need for an automated system that can help designers balance new cards that they plan to introduce into a collectible card game. This system must be able to work with any of the different cards a designer might create. For instance, it should be able to balance a card containing a newly designed mechanic. Therefore, the A.I. must be limited in what types of changes it can make to a card. The goal is to balance the card without erasing the essential design elements the designer created to accomplish ludic goals.

Chapter 2

BACKGROUND

2.1 Artificial Intelligence Competitions

An Artificial Intelligence (A.I.) competition is an event in which competitors submit computer programs referred to as "bots" or "A.I.s" which will play a digital game. These bots play a game and are ranked based on performance. Sometimes, as with Chess, the game involves direct competition between the A.I.s. Other times, as with Mario, the bots compete independently to see which one can get the highest score on a shared task.

2.1.1 Purpose and Goals

Game A.I. competitions provide several benefits to the A.I community. These include providing a fair means to benchmark algorithms, providing an opportunity to educate students, and making results easily understood by those outside of the industry [38]. The advantage of a competition for benchmarking is that it provides an impartial, easily replicated test by which multiple teams can compare their results. The benefit for education is that being able to build on top of a "real" game may make the prospect of A.I. more fun and palatable for students. Finally, the advantage to understanding is that, unlike academic benchmarks, laymen, who likely have played them, can easily understand and assess the games' difficulty.

2.1.2 Organization

When hosting a Game A.I. tournament, there are multiple questions about how the tournament will be organized that must be answered. These include, how will participants interface with the game, how will the competition be scored, and how the participants will report what they have accomplished. [38]

The first step in participating in any competition is for the contestant to understand the rules of the game. There are several ways to learn the rules of a game, including reading the rule book. However, games are often quite complicated, and certain rules may not be intuitive until they are experienced. Therefore, it is a good idea to make sure that the game can be easily played by humans. It is a best practice for A.I competitions to ensure their games can easily be played by human contestants [38]. This means the competitions must have human-usable user interface and graphics.

Once contestants understand the game, they need to be able to interface with it through some programming language. Previous competitions have provided interfaces for C++ [13], Java [39] [27] [10], and other programming languages. In addition to interfaces for specific languages, some systems can also be interfaced through standard I.O. [10] or networking [39]. This allows the construction of agents in arbitrary languages but requires programmers do extra work to implement the communication in their chosen language.

Competition organizers must also specify how they will run their tournament and how it will be scored. For example, a tournament may be organized into a single round-robin pool or many brackets. The type of organization is a trade-off between the time it takes to run the tournament and how much data the organizers collect. A bracketed elimination system reduces the number of games needed to determine which A.I. is best. However, if victory is non-deterministic, it may be more prone to

random fluctuation. It also gives less information about which A.I.s and techniques are strongest against each other.

Finally, once the contestants have submitted their A.I.s, and the tournament has been run and scored, the techniques that have been used need to be written into a paper. Togelius suggests four categories of how this might be done, which he names Sophist, Dictator, Anarchist, and Big Hippie Family approaches. In the Sophist approach, nothing about the tournament is documented. The Sophist approach is strongly discouraged as it negates much of the scholarly value of running the tournament in the first place. In the Dictator approach, the tournament organizer writes a paper which documents all of the techniques used by the contestants. In the Anarchist approach, the organizers write a paper on the tournament's organization and encourage the contestants to write papers on their submissions. Finally, in the Big Hippie Family approach, the tournament organizer works together with the authors of each submission to write a single large paper covering the organization of the competition and the techniques of the submissions [38].

2.1.3 Games

One of the major advantages of Game A.I. tournaments is that we can use different games to simulate different challenges for A.I. systems. Each game can encapsulate a different part of the complexity of the real world while ignoring other parts.

- Adversarial: Is there one or more other agents trying to foil the A.I.s plans?
- Symmetry: Do both players have the same abilities and resources?
- Observability: Is some information hidden from one or more of the players?

Table 1: Properties of Various Games

Name	*Timing*	*Observability*	*Randomness*	*Asymmetry*	*Complexity*	*Strength*
Chess	Turn Based	Full	Deterministic	First Turn	Medium	Superhuman
Go	Turn Based	Full	Deterministic	First Turn	High	Superhuman
StarCraft	Real Time	Partial	Deterministic	3 Factions	Very High	Subhuman [1]
Mario	Real Time	Partial	Deterministic	PvE [2]	Medium	Superhuman
Fifth Aeon	Turn Based	Partial	Stochastic	Decks, Shuffling	High	Subhuman
Angry Birds	Turn Based [3]	Full	Deterministic	PvE	High	Subhuman
Ms Pac Man	Real Time	Partial	Deterministic	PvE	Medium	Unknown

- Determinism: Is the game fully deterministic, or are some parts random (stochastic)?
- Turn Based/Real Time: Must players make decisions continuously or at discrete time points?
- Complexity: How large is the state space of the game?

2.1.4 Brief Survey of Tournaments

A.I. competitions have been run with a wide variety of games and formats. A subsection of these games and formats will be presented in this paper.

[1] AlphaStar may have superhuman StarCraft 2 performance but no bot has done the same for StarCraft Brood war. In addition AlphaStar is currently limited to one match up and one map.

[2] Player Vs Environment. A game in which the player does not compete against other agents

[3] Angry Birds runs its physics simulations in real time but players have an unlimited amount of time to decide where to fire.

2.1.5 Board Games

Some of the first games to be addressed by A.I.s were traditional board games. These include Chess, Checkers, Go, and many others. Board games have several advantages for A.I. competitions. First, the rules are relatively simple and often in the public domain. This means it's easy for the organizers and the participants to implement them in code. Second, these games are often relatively simple, as they must be playable by humans without digital assistance. This has allowed A.I.s to more easily achieve a higher level of play and more impressive results. However, this can also limit their usefulness as analogs for the real world.

Chess

Chess is an ancient and popular board game. Two players take turns sequentially moving pieces across the board. When one player's piece moves on top of their opponent's piece, that piece is captured. When a player's king piece is threatened (in danger of being captured) and cannot escape, that player loses the game [4].

Figure 1: Chess Pieces

[4]Full Rules `http://www.uschess.org/content/view/7324/`

Interest in computer Chess has been around since at least 1951 when Claude Shannon published the paper "Programming a Computer for Playing Chess." A great deal of interest focused on the problem of playing Chess at a human level. This was eventually attained by the Deep Blue Chess system, which defeated Garry Kasperov in 1996 [14]

Despite Chess A.I achieving superhuman results over 20 years ago, Chess is not a solved game. Solved games are games where the optimal strategy has been computed, such as Tic Tac Toe or Checkers. There is still room for better computer vs. computer play. The World Computer Chess Championship is run periodically to help find the current state of the art Chess engine, with the latest competition being run in 2018 in Stockholm.

Table 2: World Chess Competition Results [7]

Event #	*Year*	*Location*	*Participants*	*Winner*
1	1974	Stockholm	13	Kaissa
2	1977	Toronto	16	Chess 4.6
3	1980	Linz	18	Belle
4	1983	New York	22	Cray Blitz
5	1986	Cologne	22	Cray Blitz
6	1989	Edmonton	24	Deep Thought
7	1992	Madrid	22	ChessMachine
8	1995	Hong Kong	24	Fritz
9	1999	Paderborn	30	Shredder
10	2002	Maastricht	18	Deep Junior
11	2003	Graz	16	Shredder
12	2004	Bar-Ilan University, Ramat Gan	14	Deep Junior
13	2005	Reykjavk	12	Zappa
14	2006	Torino	18	Junior
15	2007	Amsterdam	12	Zappa
16	2008	Beijing	10	HIARCS
17	2009	Pamplona	10	Junior
18	2010	Kanazawa	10	Rondo, Thinker
19	2011	Tilburg	9	Junior, Shredder, Sjeng
20	2013	Yokohama	6	Junior
21	2015	Leiden	9	Jonny
22	2016	Leiden	6	Komodo
23	2017	Leiden	4	Komodo
24	2018	Stockholm	8	Komodo

Go

Go is another ancient and popular game. Two players alternate placing stones on a grid. When one player surrounds another players stones, they are captured, granting the capturing player points. The player with the most points at the end of the game wins. The player who goes second is usually awarded some points at the start of the game to make up for the disadvantage of getting the second turn [5].

Figure 2: A Game of Go in Progress

Go is harder than Chess for computers due to its larger search space. A.I. systems did not achieve professional-level play until 2016 when the AlphaGo system defeated Lee Sedol a 9 dan professional Go player (9 dan is the highest rank) [35].

[5]Full rules `https://www.britgo.org/intro/intro2.html`

Table 3: Go Competition Results [3]

Event	*Date*	*Place*	*Entrants*	*Winner*
AI Ryusei	December 9th-10th 2017	Akihabara, Tokyo	18	FineArt
132nd KGS tournament	November 5th 2017	KGS	6	Zen
Slow KGS tournament	September 3rd-6th 2017	KGS	2	Zen
1st World AI Go Open	August 16 - 17	Ordos City, China	12	Zen
131st KGS tournament	July 9th 2017	KGS	4	Zen
130th KGS tournament	May 7th 2017	KGS	6	AyaMC
10th UEC Cup	March 18th-19th 2017	Tokyo	30	Fine Art
Slow KGS	March 5th-8th	KGS	4	Zen
129th KGS tournament	January 14th 2017	KGS	4	Zen
128th KGS tournament	December 4th 2016	KGS	2	AyaMC
127th KGS tournament	November 13th 2016	KGS	4	AyaMC
126th KGS tournament	October 9th 2016	KGS	4	AyaMC
2016 Slow KGS	September 3rd-7th	KGS	4	AyaMC
125th KGS tournament	7-Aug 2016	KGS	4	Zen
124th KGS tournament	10-Jul 2016	KGS	4	Zen

2.1.6 Real Time Strategy Games

A Real-Time Strategy game or RTS is a game in which players control a group of entities, often representing an army or civilization, in real time. RTS are useful games for researchers because they allow us to study essential problems common in dynamic systems, such as decision making with uncertainty and real-time adversarial planning [33].

Real Time Strategy (RTS) A.I. competitions have mainly focused on two games. The first game, OpenRTS, is an open-source RTS game built for research. However, following the release of an API for StarCraft: Brood War [19], a popular commercial RTS, StarCraft has received much of the community's attention [13].

StarCraft

StarCraft is a game in which players select one of three species: the Terran, Protoss or Zerg. They then use the unique abilities of their faction to gather resources, build armies and eventually destroy their opponent. The game is played in real time over a variety of maps with different terrain. Players can only see areas of the map within a certain distance of units that they control, making the game only partially observable. The rest of the map is hidden by "fog of war."

Figure 3: StarCraft Screenshot

Starcraft has consistently been one of the most popular games. Researchers have attempted to solve the problem of creating high-quality StarCraft agents using a variety of strategies. The first type of strategies were hard-coded ones. These were often organized around finite state machines that organize the A.I. into a number of states, such as gathering resources, building, attacking, and defending. Each of these behaviors was hard-coded, based on human knowledge. [33].

Another approach to developing StarCraft A.I.s is to use search methods to explore the game's state space to find good strategies. This approach has been employed very successfully with board games like Chess and Checkers; however, its application to StarCraft is far more limited. Researchers have constructed planners that use Hierarchical Task Networks (HTN) to cut the search space down to a more tractable level. HTN divide complex tasks into subtasks, which are then independently planned. HTNs reduces the complexity of the search problem, but may also reduce the quality of the results. [33].

Researchers have also applied machine learning based techniques to StarCraft. Early researchers applied ML techniques to specific parts of the game, such as determining

Table 4: StarCraft Brood War Competition Results [33]

Name	*1st place*	*2nd place*	*3ed place*	*Winrate 1*	*Winrate 2*	*Winrate 3*
AIIDE 2010	Overmind	Karsi0	Chronos	Unknown	Unknown	Unknown
AIIDE 2011	Skynet	Ualbera	AIUR	88.9	79.4	70.3
AIIDE 2012	Skynet	AIUR	Ualbera	84.4	72.2	68.6
AIIDE 2018	SAIDA	CherryPi	CSE	96.15	90.84	88.37
2011 CIG	Skynet	Ualbera	Xelnaga	86.7	73.3	36.7
2012 CIG	Skynet	Ualbera	AIUR	78.3	65.2	60.4

build orders and openings.

2.1.7 Classic Computer Games

A.I. researchers have adopted several classic computer games. These games differ from board games in that they often incorporate physics and real-time simulation. However, they are much simpler than modern computer games. In addition, they are all very popular and well known within the gaming community, which helps draw interest to them.

Mario

Mario is a platforming game in which the player guides a character (the titular Mario) across a series of platforms. Players must avoid enemies, obstacles, and pits while also trying to collect power-ups and coins.

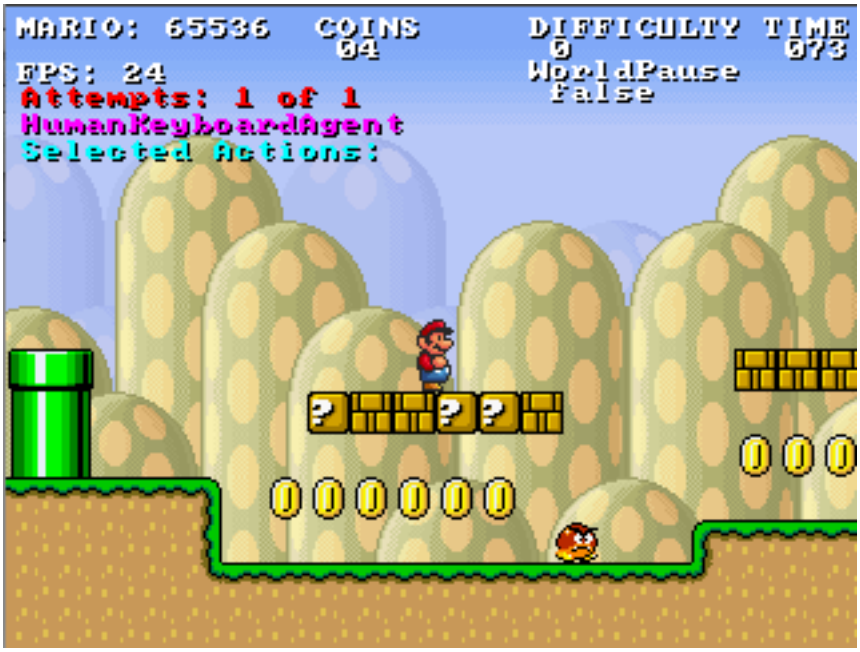

Figure 4: Infinite Mario Screenshot

In 2009 Julian Togelius, Sergey Karakovskiy and Robin Baumgarten ran a tournament using the Infinite Mario platformer [39]. The goal was to see which agent could make it the furthest through a procedurally generated level before dying. The most successful agent was Robin Baumgarten's A* agent.

Ms. Pac-Man

Ms. Pac-Man is a classic arcade game in which the player navigates a maze collecting pellets and avoiding ghosts.

Table 5: 2009 Mario Competition Results [39]

Competitor	*Progress*	*ms/step*
Robin Baumgarten	17264	5.62
Peter Lawford	17261	6.99
Andy Sloane	16219	15.19
Sergio Lopez	12439	0.04
Mario Perez	8952	0.03
Rafael Oliveira	8251	?
Michael Tulacek	6668	0.03
Erek Speed	2896	0.03
Glenn Hartmann	1170	0.06
Evolved neural net	7805	0.04
ForwardJumpingAgent	9361	0.0007

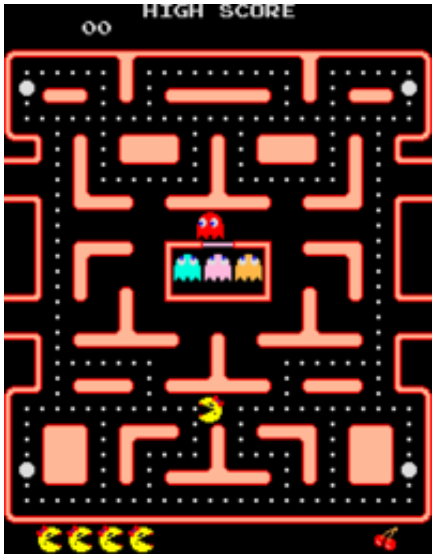

Figure 5: Ms. Pac-Man Initial Setup

In the Ms. Pac-Man tournament, agents are given a limited view of the environment. They know the layout of the maze, but can only see other agents when they are within eyesight; they cannot see through walls. User-submitted agents control both the ghosts and the player. The goal of the player is to survive and collect as many pellets as possible. The goal of the ghost is to eliminate the player as quickly as possible. [5]

2.1.8 Physics Games

Physics games are games in which the player must interact with a simulation of real-world physics in order to play the game. They play strongly into human intuition about physics, which makes these games challenging for A.I.s to surpass humans.

Table 6: Mrs Pac-Man Competition Results [4]

Player Agents	
Agent	*Average Score*
Squillyprice01	7736.63
GiangCao	7516.63
thunder	6733.13
PacMaas	6275
Starter PacMan	5865.5
StarterPacManOneJunction	1134.25
StarterNNPacMan	535
user76	120
Ghosts Agents	
StarterGhostComm	3859.13
StarterGhost	4288.25
thunder	4864.81
user76	4948.88

Angry Birds

Angry Birds is a physics-based game in which players launch birds out of a slingshot to destroy pigs. Levels consist of targets (pigs) placed among a variety of physics-based obstacles such as boxes and explosives. Players must calibrate their shot and predict the interactions of the physics system to destroy all the pigs with as few shots as possible. Angry Birds has become immensely popular since its inception in 2009 and has garnered interest in the A.I community.

Figure 6: A.I Birds Screenshot

As of the 2018 A.I. Birds Competition, humans are still considerably better than A.I. players at Angry Birds. [1]

2.2 Collectible Card Games

A collectible card game (CCG) is a card game in which players do not start with all of the available cards. Instead, they collect more over time. In conventional CCG these cards are then used to form decks, which are used to compete against other players. The decks are shuffled each time the game is played; then players draw cards without knowing the order of their deck or the contents of their opponent's deck. This means

Table 7: 2018 Angry Birds Competition Results [1]

Bot	*Round Reached*
Eagle's Wing 2017	Final (won)
BamBirds	Final
Eagle's Wing 2018	Semi-Final
IHSEV	Semi-Final
PlanA+	Quarter Final
DQ-Birds	Quarter Final
MetaBirds	Quarter Final
AngryHex	Quarter Final
MYTBirds	Quarter Final

CCGs have a number of interesting properties, including a large degree of asymmetry, variance, and uncertainty.

The first generally recognized CCG is Magic the Gathering, which was released in 1993 [6]. In Magic the Gathering's standard format, players form decks of 60 cards out of a pool of thousands of cards. A deck may repeat any individual card up to four times, except for land cards, which may have any number of repetitions. This means that there are a vast number of possible decks that can be played, potentially resulting in a very high degree of asymmetry between players.

Table 8: Tournament Results

Name	*Game*	*Participants*	*Winner*
AIIDE 2010	StarCraft		Overmind
AIIDE 2011	StarCraft	15	Skynet
AIIDE 2012	StarCraft	10	Skynet
AIIDE 2018	StarCraft	25	SAIDA
Mario 2009	Mario	9	Robin Baumgarten
CIG 2018	Pac-Man	8	Eagle's Wing 2017
AI Ryusei 2017	Go	18	FineArt
132nd KGS tournament	Go	6	Zen
Autumn 2017 Slow KGS	Go	2	Zen
1st World AI Go Open	Go	12	Zen
131st KGS tournament	Go	4	Zen
130th KGS tournament	Go	6	AyaMC
10th UEC Cup	Go	30	Fine Art
Spring 2017 Slow KGS	Go	4	Zen
129th KGS tournament	Go	4	Zen
128th KGS tournament	Go	2	AyaMC
127th KGS tournament	Go	4	AyaMC
2010 WCCC	Chess	10	Rondo, Thinker
2011 WCCC	Chess	9	Junior
2013 WCCC	Chess	6	Junior
2015 WCCC	Chess	9	Jonny
2016 WCCC	Chess	6	Komodo
2017 WCCC	Chess	4	Komodo
2018 WCCC	Chess	8	Komodo

Figure 7: A Screenshot from MTG Arena, a Digital Adaption of MTG

Following Magic, a wide variety of collectible card games have been released. One of the most popular ones is Blizzard's digital CCG, Hearthstone [18]. Hearthstone uses many of the same rules as Magic but simplifies the game by streamlining resource costs and reducing the number of cards in a deck from 60 to 30.

Figure 8: A Screenshot of Hearthstone a popular digital CCG

2.3 Automated Balancing

In recent years, a significant amount of work has been put into procedural content generation. A large amount of this effort has been directed towards the generation of peripheral assets, such as trees, buildings, race-tracks, and other structures [23]. Comparatively, the area of content generation for card games has remained somewhat underdeveloped. The research that currently exists is largely centered around the implementation of balancing systems. This push for good automatic balancing is supported by the research of Andrade et al., the research shows that game balance has a significant impact on player satisfaction [8]. In most of these systems, the focus is put on generating new balanced cards from scratch, rather than working with a designer to manipulate specific aspects of a design.

In the paper, 'Evolving Card Sets Towards Balancing Dominion", Mahlmann, Togelius, and Yannakakis lay out a method for evolving sets of cards for the deckbuilder game Dominion [31]. That work differs from this one in three important ways. The first difference is that it focuses on a deckbuilder game, rather than a collectible card game. This lowers the amount of asymmetry between players, as they are unable to enter into the game with pre-made decks. Second, the system works in a vacuum, generating an entirely new set that is not expected to interact with any previously existing cards. The system aims to focus on integrating new cards into an existing set of cards while being able to control how much of an impact the new cards are likely to have on the games' metagame. Finally, the system works purely autonomously. While this can be very beneficial in saving labor, it limits a designer's ability to interact with the game to inject fun or thematic elements.

The paper 'Evolving Maps and Decks for Ticket to Ride" details the authors' attempts to generate two kinds of content for the game Ticket to Ride: cards and maps [17].

The authors' goal is to generate maps that resemble real-world locations but are also balanced. However, similar to Dominion, Ticket to Ride does not involve a deck-building phase before starting the game. As such, certain aspects that apply to this paper, such as a competitive metagame formed of top-tier decks, are not relevant.

Another effort towards the study of automatic game balancing was put forth by Volz et al. [43]. Using a comparison of manual and automated deck balancing techniques, these researchers demonstrated the feasibility of automatic balancing in the card game Top Trumps. A similar example of this strategy is shown in the research done by Bhatt et al. towards evolving decks in the collectible card game, Hearthstone, developed by Blizzard [18, 12]. Through the use of a consistent aggro-style, utilizing differing decks, Bhatt et al. demonstrated the possibility of improving win-rate by modifying deck composition. They also noted a distinct difference in win-rate for each of the decks, depending upon the deck style of their opponent. Prior research was done by Mahlmann et. al. also demonstrated the separation between card balance and player skill [31]. Their tests set three agents of varying skill levels against each other using a series of different decks. In this manner, they were able to determine that regardless of the player's skill, individual decks lent themselves more favorably towards game balance.

Automatic balancing systems have seen implementation in other genres of games as well. The paper "Automatic Design of Balanced Board Games' generates entire games based on game-definition language and balances them using a general game player [24]. However, the complexity of these games is limited, and as such, the system is not very useful for balancing complex existing games such as collectible card games. In Automatic Playtesting for Game Parameter Tuning via Active Learning, Zook et. al. show a method for tuning the parameters of a space shooter game [47]. This work is similar but creates a system that can deal with the unique complexities of

collectible card games, such as a multitude of different decks.

Additionally, in the realm of fighting games, Zuin et. al. explore an evolutionary method for attempting to find unbalanced combinations of abilities [26]. Similar work could eventually be applied to finding combinations of cards in CCG that might be overpowered. However, the search space of CCG is much larger due to the number of cards and card combinations in existence.

Philipp Beau and Sander Bakkes examined a more general view of game balancing. In their work, they attempt to create a general method for balancing asymmetric games using Monte Carlo simulation [11]. However, such a general solution cannot be trivially applied to all games in a practical manner. This is especially true of collectible card games, which have a vast number of possible match-ups due to the nature of deckbuilding. It is, therefore, necessary to apply some specialized technique to reduce this vast number of to a manageable level.

In addition to competitive balance, Lankveld et. al. explore the theory of incongruity as it applies to balancing games [42]. According to the authors, 'Incongruity is defined as is the difference between the complexity of a context and the complexity of the internal human model of the context." Using that theory, they attempt to use an automatic balancing system to make the level of incongruity within their game constant, adapting to the player's changing mental model. This shows the potential of automatic balancing systems not only to be used to attain fairness in multiplayer games, as this paper works towards but also in obtaining specific emotional goals.

Chapter 3

SYSTEM DESIGN

3.1 The Fifth Aeon Collectible Card Game

Fifth Aeon is a collectible card game in which two players attempt to defeat each other by reducing the opponent's health pool to zero, using customizable decks of cards. The players may create any deck they like out of the 140 existing cards in the game, with no limits placed upon the availability of cards from differing factions. The only constraint that must be followed to obtain a legal deck is to include 40 cards, with no more than four copies of any single card present.

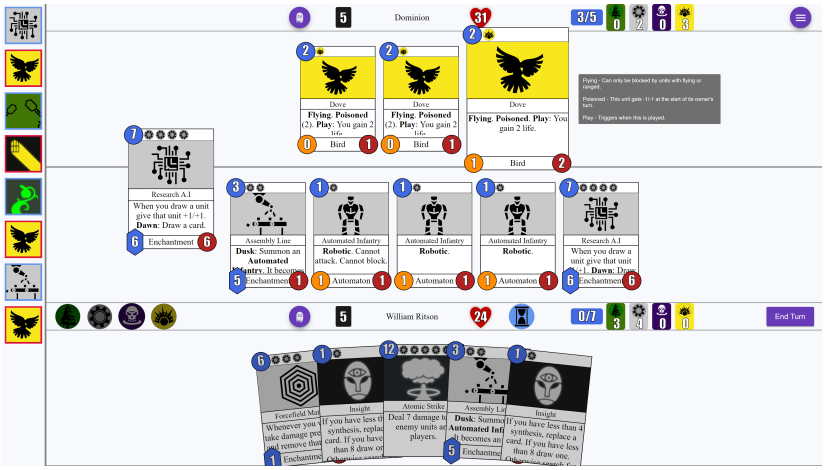

Figure 9: An Example of Fifth Aeon Gameplay

Cards are split into four types, spells, units, enchantments, and items, each with varying costs, special abilities, and statistics. For example, spells have a cost and an arbitrary number of effects, such as destroying a target unit or drawing more cards for their owner. Units carry all of the functionality present in spells, but also have attack

and defense values. These allow them to interact with other units and players in the combat phase. Cards may also be customized through the addition of mechanics, which alter how they behave. For example, the Immortal mechanic causes a unit that dies to return to the battlefield at the end of the turn. Other mechanics, such as the deal damage mechanic, can be further customized by specifying parameters including options such as: how much damage should be dealt, what should be targeted, what the trigger will be, and when the effect should be applied. This system allows a very large number of cards to be specified in data without any additional programming.

In order to play a card, a player must have enough energy remaining to meet its energy cost. Any energy spent to play a card will be depleted until that player's next turn. The card's owner must also have the prerequisite amount of Fifth Aeon's four resources, synthesis, growth, renewal, and decay. Every card specifies its energy cost, as well as its resource prerequisites. At the start of each turn, a player must choose one of the four resources, giving them one more maximum energy and one more resource of that type. Thus, as in many other collectible card games, every card may be played in any deck and the player will eventually have enough resources to play it. However, playing cards of different factions has the potential to make a deck less efficient and cause it to take longer to assemble its resources than a deck that places a focus on one faction.

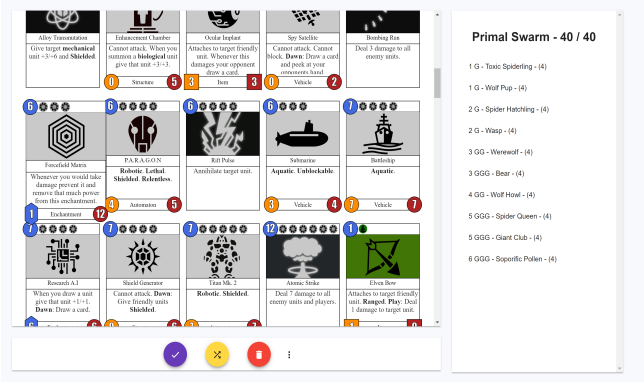

Figure 10: Fifth Aeon Deck Building

3.1.1 Rules Engine

The implementation of Fifth Aeon's core rules, aka the rule engine, is kept separate from any client or server code in the repository `https://github.com/Fifth-Aeon/CCG-Model`. The rules engine is only responsible for computing the internal model of a Fifth Aeon game and does not control user interface (U.I.), rendering, etc. The rule engine is the model in the Model View Controller design pattern.

3.1.2 Networking

When writing a networking layer for a collectible card game, there are several issues to consider. First, we must keep both players in sync with one another. If one player takes an action, the other should be alerted of it. If either client's game falls out of sync, they may unintentionally take actions which are not legal, causing the game to become unplayable. Second, we must prevent players from cheating. A player should not be able to make illegal moves, even if they modify their client. Third, some information must be hidden from players. Players should not know the order of cards in their deck, what cards are in their opponents deck, or what cards are in

their opponent's hand. Finally, we want to make the game fast and responsive, even on slow connections. That requires reducing the amount of information we send over the network and doing operations locally as often as possible.

There are several different paradigms to achieve these goals of maintaining synchronization, preventing cheating, reducing latency, and hiding secret information. One method is to keep only one game model on the Server, which would compute all the game rules. The clients would only have dumb models that could display information, but would not be able to compute the results of any actions. Whenever an action was taken, the Server would send all the changes to the model as well as a list of what actions are legal after the action resolves. This model is referred to as Server Simulation.

Another model is to have both the client and the Server understand the full set of game rules. Whenever an action is taken, the clients and the Server communicate only the minimum amount of information needed to synchronize the changes. Because all parties understand the rules, they only need to share what actions were taken, not the consequences of those actions. This results in less network traffic and a more responsive game over poor connections.

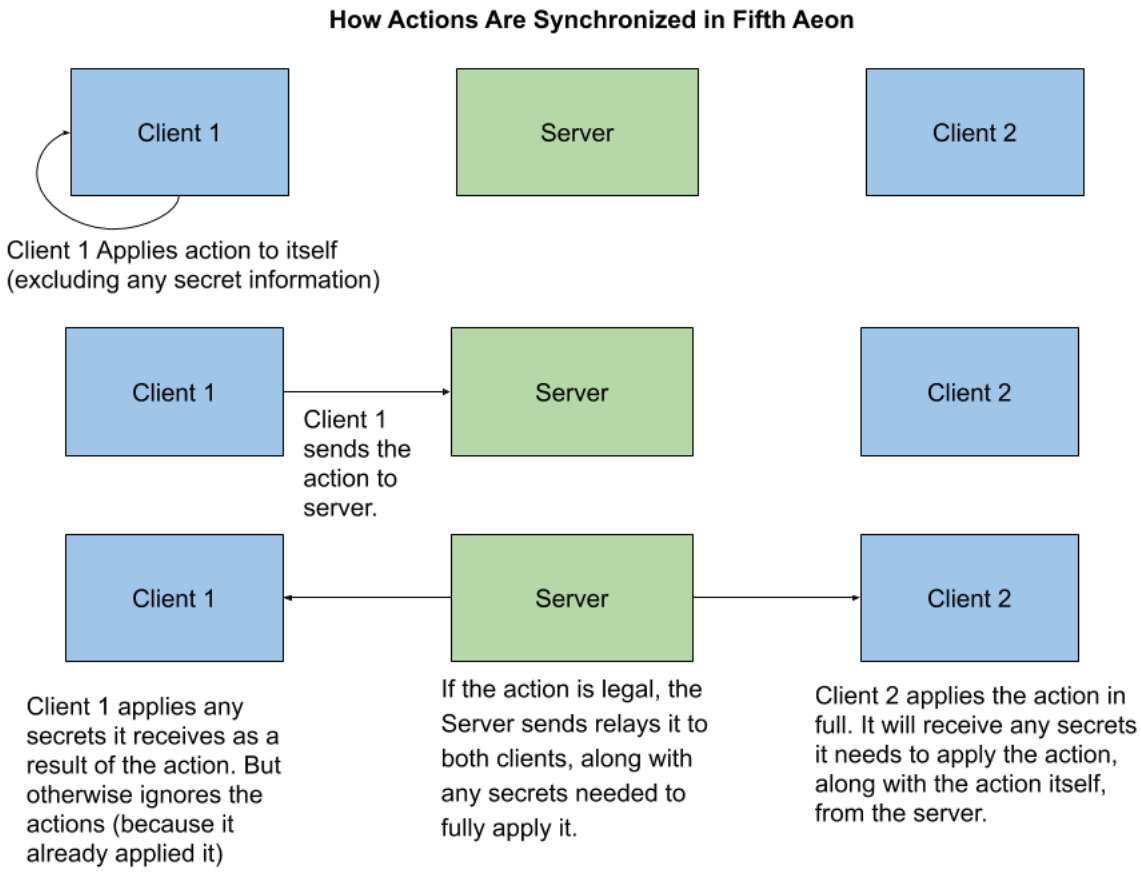

Figure 11: How Actions are Synchronized in Fifth Aeon

Fifth Aeon uses the synchronized simulation paradigm to reduce network traffic. For each game, three models of the game are kept. The Server has the canonical model, and each client has their model. Whenever a client takes an action, it first attempts to apply that action to its local model. If the action is entirely deterministic, and its outcome does not rely on any hidden information, this will succeed. If it relies on secret information, such as what cards will be drawn in response to a card draw effect, the client will have to wait for more information from the Server before being able to update its model fully. The result of this is that deterministic actions will appear instantaneous to the user, even over very slow connections. Once the client

has applied the action, it will send it to the Server. The Server will check if its legal, which would imply the client had been modified. If it is legal, it will apply it to its canonical model, and relay it to the other client. It will also release any secret information revealed by the action such as what cards were drawn. The second client will receive this information and use it to update its model, so all three models are in sync.

There are downsides to the synchronized simulation paradigm. The largest one is that its harder to keep all three models in sync. All three models need to be able to deterministically apply each action with the information they are given. The programmer must figure out the minimum set of information needed for this to happen. If the programmer makes a mistake and send insufficient information, or apply that information incorrectly, it will result in a desynchronization. These kinds of bugs can be particularly tricky to find and debug, because they often do not cause an immediate failure, but rather, cause the game to fail at some point in the future when the client attempts to do something illegal. The second major issue is that it is more challenging to hide secret information. In the dumb client model, only the Server ever needs to do any computation on secrets; thus the only additional logic needed to handle secrets is to know how to limit what information is sent to a client. In the synchronized simulation model, a client must know that it can compute some actions, but in other cases, it will have to request additional information from the Server.

3.1.3 Server

Fifth Aeon's Server is written in TypeScript and runs on the Node.JS platform. It provides several services to the client, including data storage, authentication, and server-authoritative multiplayer gaming. The Server is responsible for storing the majority of the data associated with an account. It stores a player's email, password,

and collection of cards in a PostgreSQL database. The passwords are salted with 32-character random strings, then hashed with pbkdf2 in order to make it more difficult to convert passwords back into plain text.

The Server also provides the API and storage for the tournament system. Regular Fifth Aeon accounts can be used to sign up for the A.I. competition. The backend can then store teams and submissions. It enforces permissions to ensure that only the owners of a submission or an admin can see it.

One of the primary purposes of the Server is to allow players to play multiplayer games against each other via the internet. Unlike most of the Server's other functions, this is done via WebSockets rather than with standard HTTP requests. This is because WebSockets, unlike HTTP requests, allow two-way communication, meaning that both the client and the Server can send information to one another without first having to receive a request. During a multiplayer game, the Server holds the canonical server-model of a game. Whenever a client takes an action, it must send it to the Server, which validates it. If the action is legal, it will be run and passed on to the other client, otherwise, the action will be rejected and an error message will be sent to the client attempting to take the illegal action.

Notably, in Fifth Aeon, the Server is only required for online games. The client is capable of running offline games (vs. the A.I.) without connecting to the Server. This greatly reduces the load on the Server but means the Server cannot verify who won a player vs. A.I. game. That would be a problem if the game were built on a Free to Play model and offline games granted rewards. In that case, the game would first have to be altered to run all games on the Server, which would not require any major architectural changes, but would use more resources.

3.1.4 Client

Fifth Aeons client is built to run in a web browser. It is written in TypeScript using the Angular framework. The web client provides access to all the functionality of the base game, but not the Bot Tool Kit, which is only accessible as a command line tool. The web client runs on modern browsers and is tested on Chrome, Firefox, and Edge.

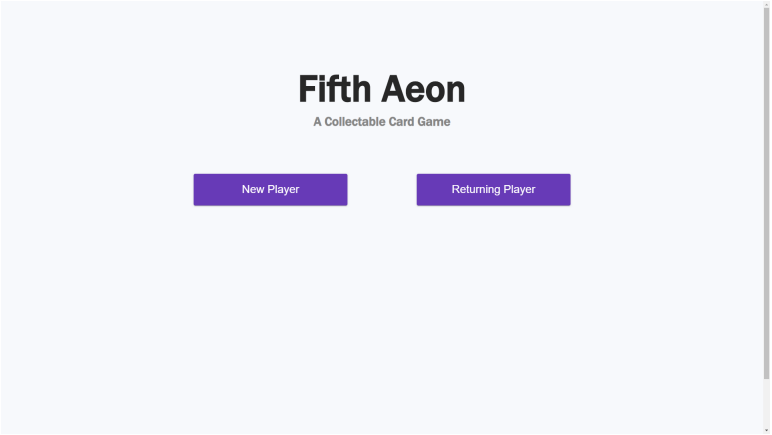

Figure 12: Front Page of the Fifth Aeon Client

When a player opens the game, the client checks to determine if it has a valid session. Sessions are stored as JSON Web Tokens (JWTs) so the Server can efficiently check them without having to access the database. If the user does not have a valid session they will get the new/returning player screen. From there, the user can either log into an account, register a new account, or play a guest. These options are intended to minimize the amount of time it takes to resume playing the game for both existing and new players.

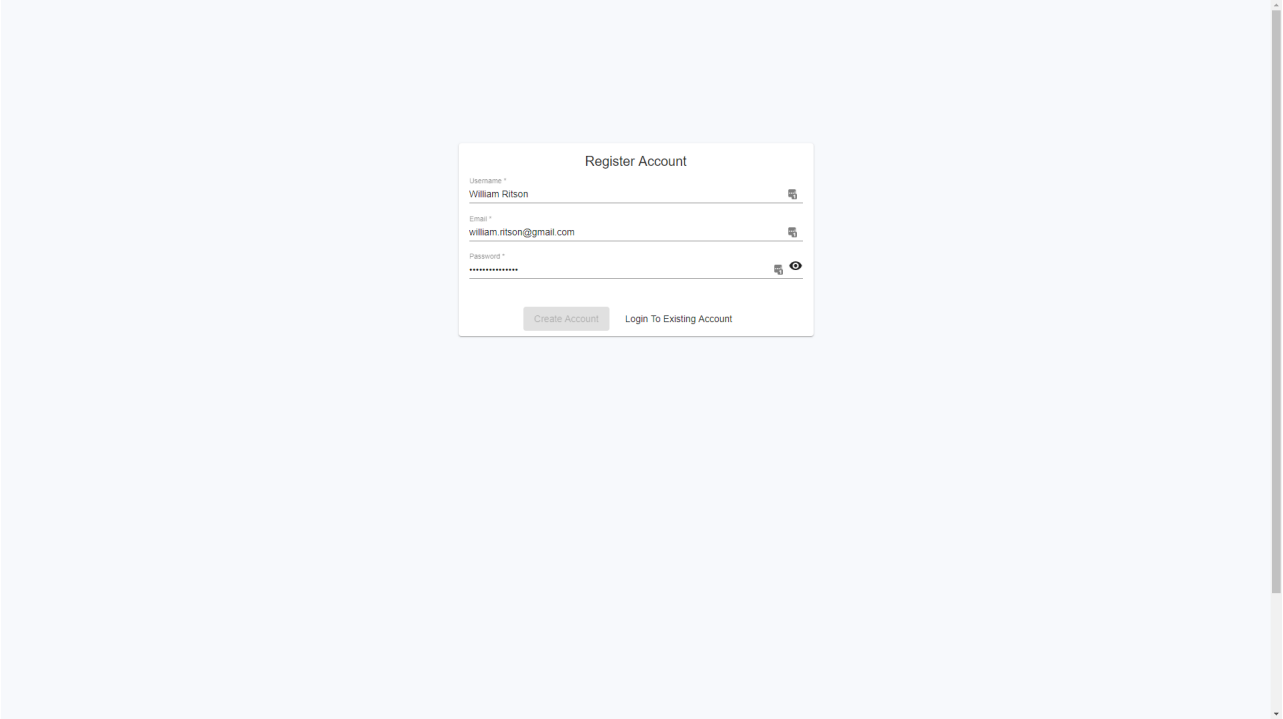

Figure 13: Creating a New Account

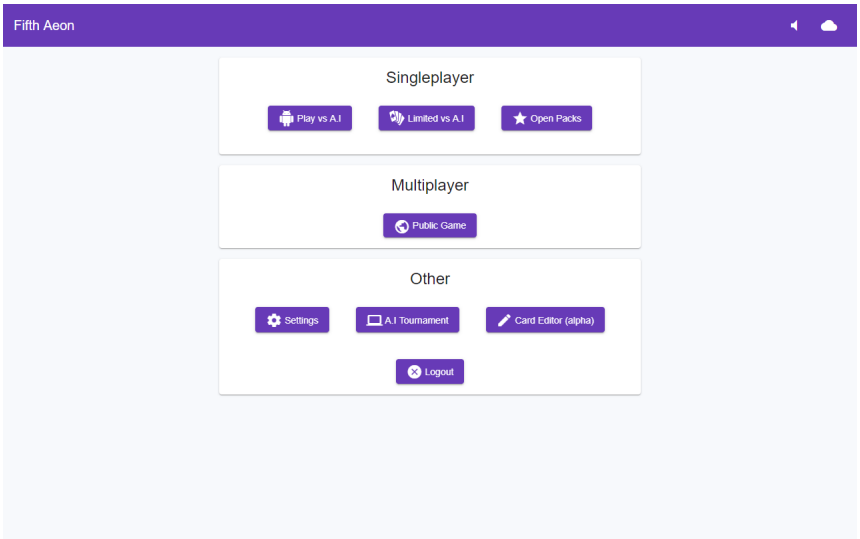

Figure 14: The Main Lobby Where a Player Decides What Mode to Play.

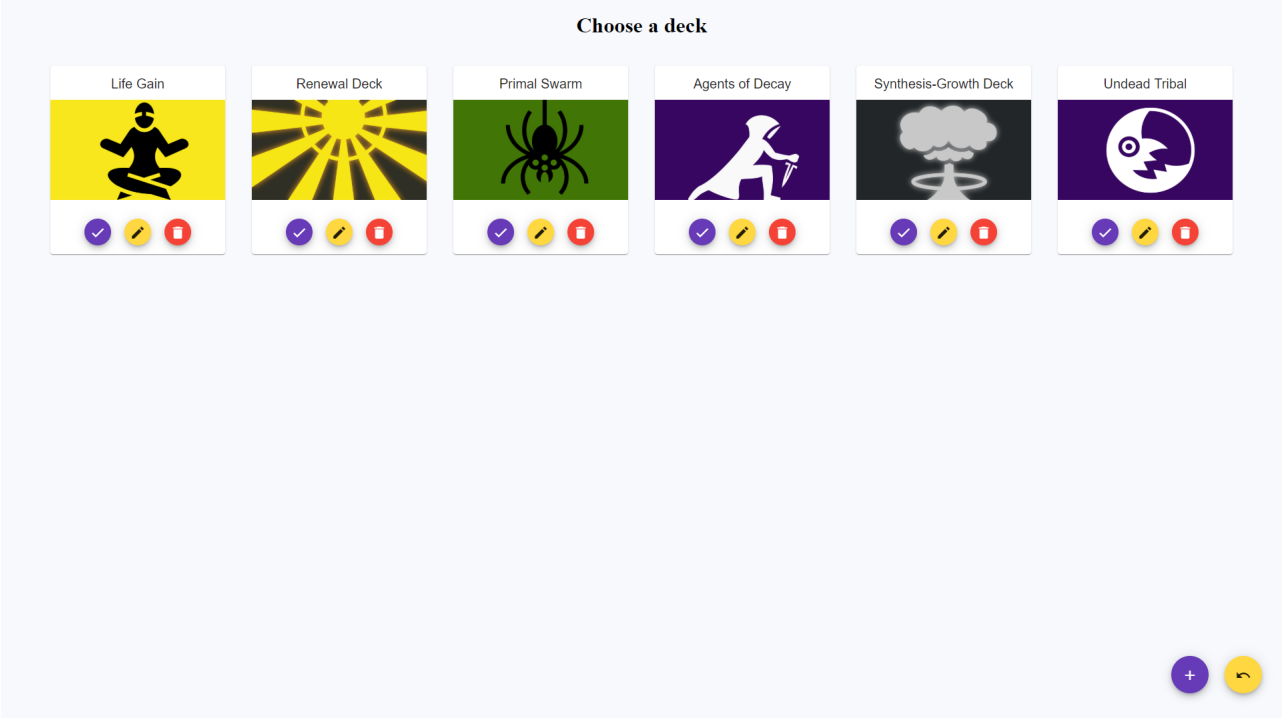

Figure 15: Selecting a Deck to Play With

In addition to the U.I. to play the game, the client also contains a card editor where users can create custom cards. Users use any of the existing mechanics, targeters, and triggers to create a vast number of potential custom cards.

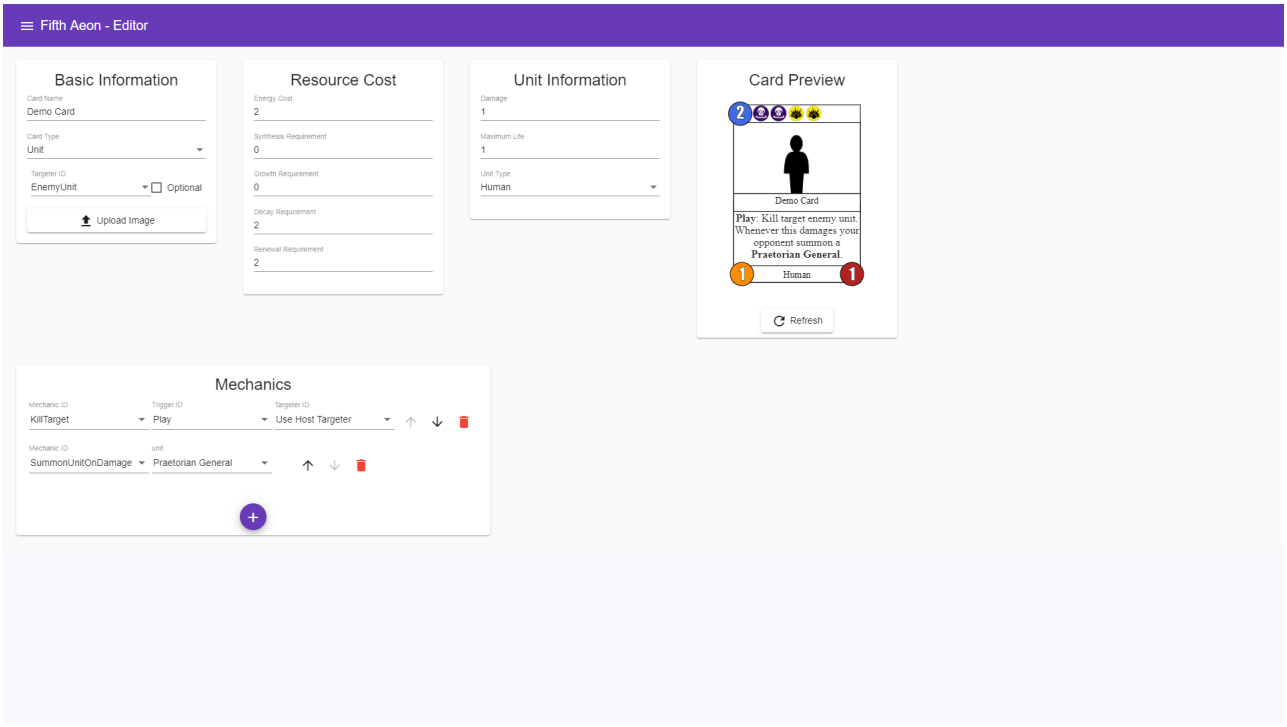

Figure 16: Creating a Card in the Editor

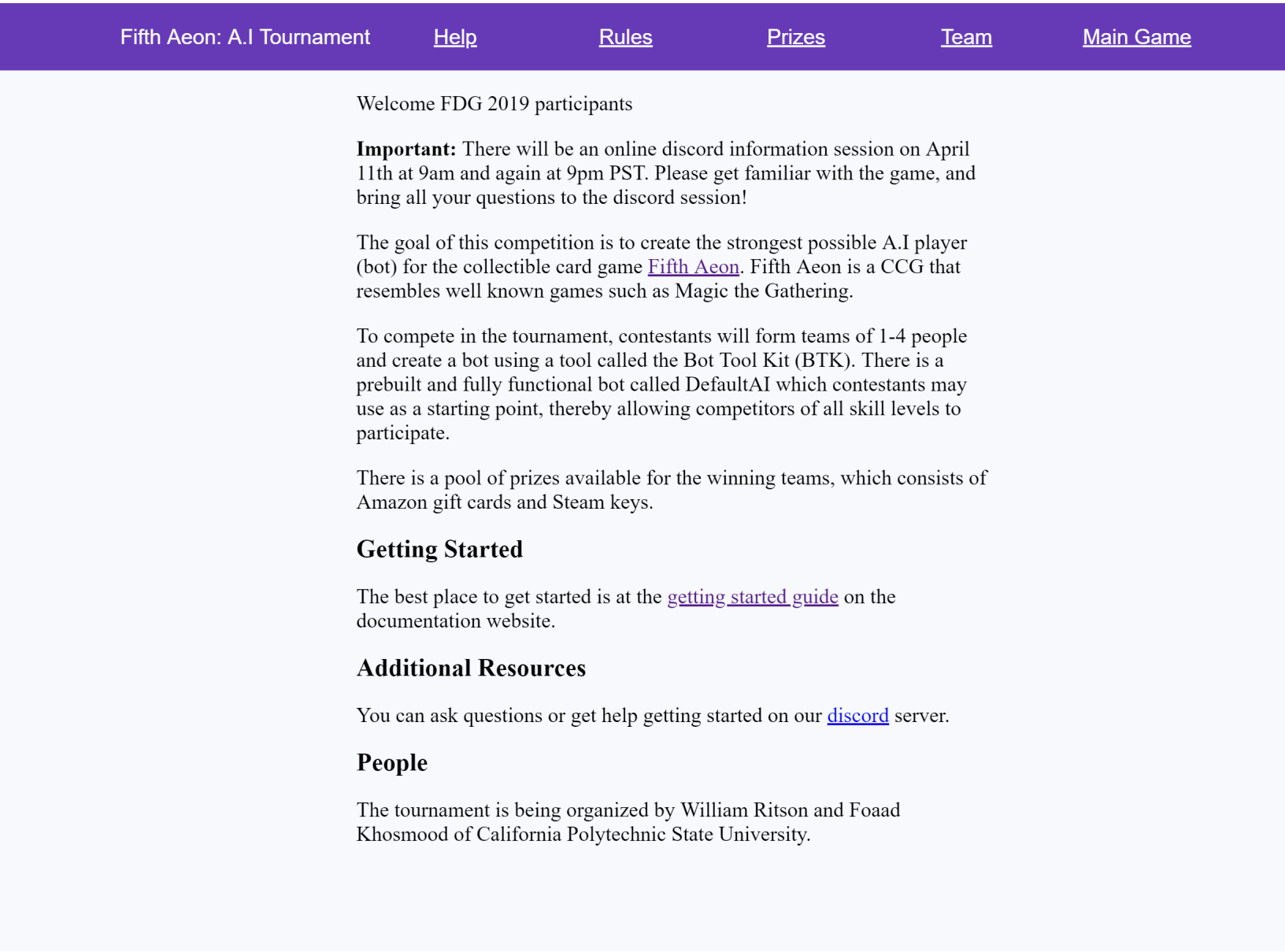

Figure 17: Information about the Tournament Available through the Client

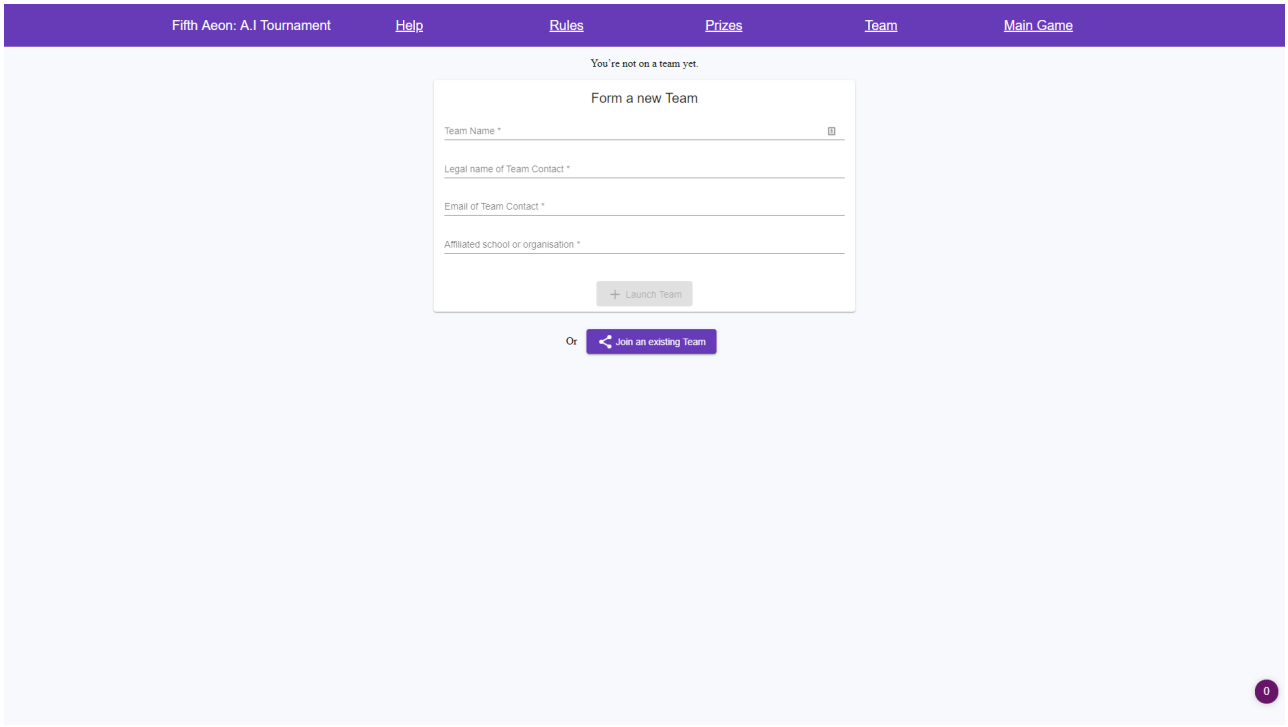

Figure 18: UI to Create a Team to Submit to the Tournament

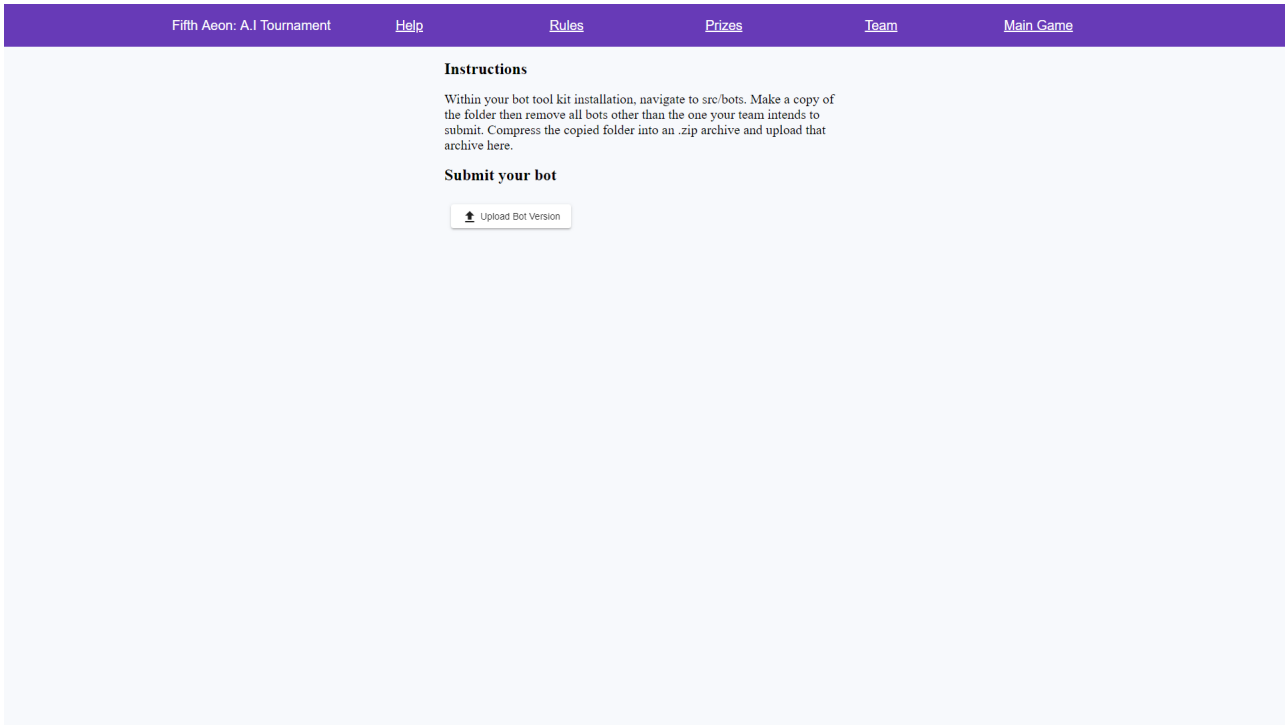

Figure 19: UI to Submit a Bot

3.2 The Bot Toolkit

The Fifth Aeon Bot Toolkit is a command line tool used for running games or tournaments between A.I. bots. It can be given various parameters, such as what decks to use, what bots should participate, and how many games should be played. The system is then able to run the specified games in parallel using multiple system processes. It also contains code for recovering from errors and restarting failed processes.

3.2.1 Creating Bots

The Bot Tool Kit has the ability to create a template bot for a user. The user enters in the name of the bot then the BTK will create a source code file for it and automatically import it. From there, the user can edit their bots source code in an editor of their choice.

3.2.2 Running Tournaments

The main function of the bot tool kit is to run tournaments. This functionality is used both by competitors to test their bots against others and to run the final tournament and decide the winner.

The BTK can run different variations of tournaments based on JSON configuration files. These files are stored in the data/tournaments file. By default, the BTK has configuration files to run the three standard tournaments that are used to evaluate the winner of the tournament. Users may also create their own configurations for testing. For example, they could create a configuration with a minimal number of trials to make sure their bot runs in a short amount of time. They could customize what bots will be included in a tournament to see what matchups their bot is or is

not good at.

A user will be given a list of all the configurations that are known to the BTK and may select one to run. After that, the system will run the tournament using process per logical CPU core. The system will print out the results when the tournament runs, or if the user terminates the process early, it will display the results it has already finished running.

Bot Tool Kit tournaments are run in using multiple processes in order to promote isolation and efficiently utilize all a machines hardware.

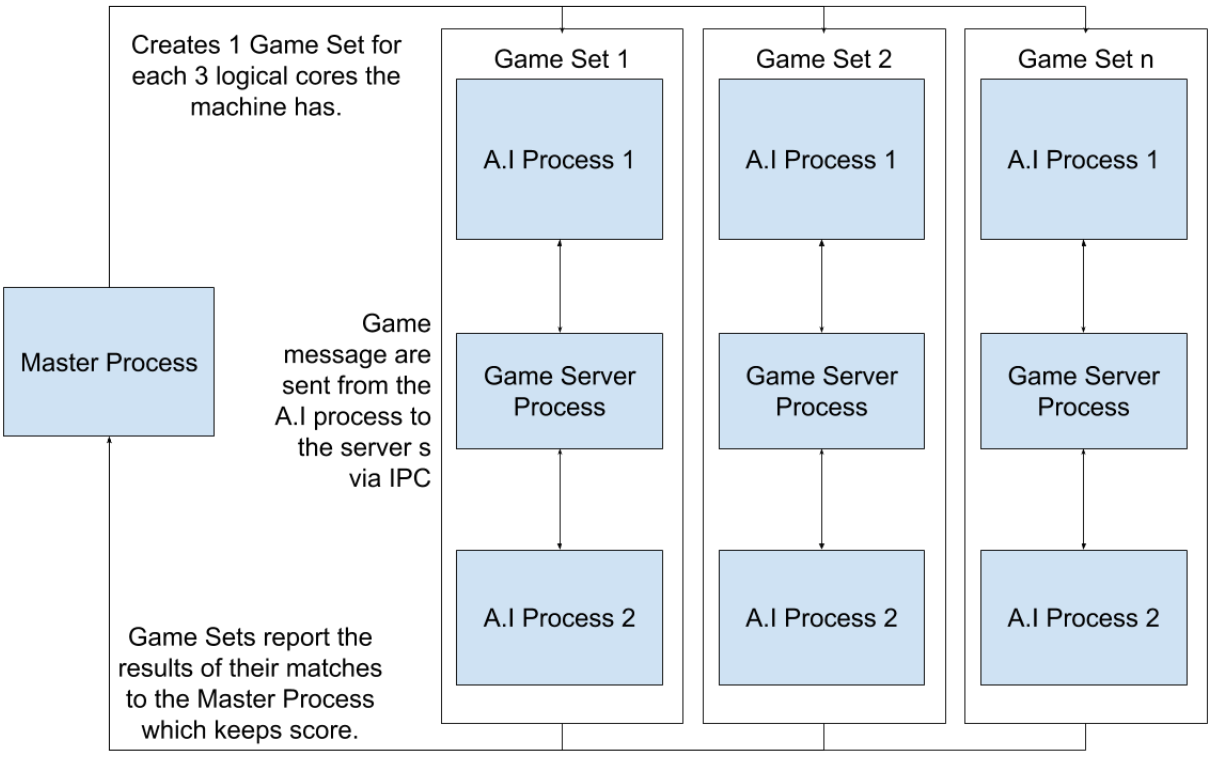

Figure 20: Bot Tool Kit Tournament Multiprocessing

3.2.3 A.I. Server

When creating an A.I., it is often helpful to be able to play against the A.I. using the client's GUI. This allows users to get more intuition about how their bot plays as well as to trigger specific scenarios that might help them find issues. Ideally, they should be able to do this without having to build and run a copy of the client on their local machine. In order to accommodate this feature, the BTK supports the A.I server mode.

An A.I server is a WebSockets-based server that the game client can connect to in order to play against a bot. From the BTKs menu, a user may select the A.I. server mode. They then have to select which A.I. they want to the server to run and the deck it should use. Once that is done, the server will start and run on localhost port 4236. The user can then open the client at fifthaeon.com (a local copy is not necessary). The client will detect the A.I. server and connect to it, allowing the user to play against their A.I. using the standard WebClient GUI.

3.3 DefaultAI

Fifth Aeon includes an A.I. player called DefaultAI which serves as both the opponent when playing in single-player mode and as a platform for other bots to be built off of during the A.I. competition.

3.3.1 Architecture

DefaultAI is based on a series of heuristics and utility calculations. It employs a standard plan for a turn and customizes it based on what actions give it the best

utility.

The source code for DefualtAI is available at `https://github.com/Fifth-Aeon/CCG-Model/blob/master/ai/defaultAi.ts` with full comments.

DefaultAI's standard plan is as follows.

1. Play a resource.
2. Play cards and/or empower enchantments.
3. Attack.
4. Play cards and/or empower enchantments.

In addition to its plan for a turn, DefaultAI must also act when the opponent attacks it or when it has a choice. These do not require plans as in either case it must immediately block or make a choice without being able to do anything else first.

Deciding What Card to Play

The most significant problem for DefaultAI is deciding what cards to play in a turn. Every card in Fifth Aeon has a non-zero energy cost which limits how many cards may be played in a turn. For example, if a player has 6 energy and cards which cost 2, 4, and 5 energy, then the player could play the 2 cost card and the 4 cost card, or it could play the 5 cost card. However, it could not play all three. Thus, DefaultAI must decide some subset of the cards in it's hand it wants to play. In addition, the value of playing a certain card will change significantly depending on the current state of the game. It would not be a good idea for DefaultAI to play a board clear (a spell that destroys all the units currently in play) when it has many units in play and its

opponent has none. Therefore, DefaultAI's card evaluation strategy needs at least some level of situations awareness to be competent.

DefaultAI achieves this by having a combination of evaluator functions built into all the games mechanics, triggers, and targeters. An evaluator is a function that the A.I. can use to gain an estimation of the value of some mechanic based on the current state of the game. The basic unit of evaluation is 1 evaluation point, which represents a single stat point of a unit. So, Minotaur, a 4/5 unit with no abilities is worth 9 evaluation points. The evaluation of units and items take into account both the stats the unit will add to the battlefield and the effects of the unit's ability.

Figure 21: Minotaur a Simple Unit With no Abilities

Some abilities such as Flying multiply the value of the unit they are attached to. Other abilities add to it. Abilities that have targets must consider what targets they will currently be applied to. For example, Flame Ifrit is an 8/4 unit with the ability 'Play: Deal 3 damage to all other units". Flame Ifrit's stats give it a base evaluation score of 12, but its ability must be taken into account. The ability is created by combining the OnPlay trigger with the DealDamge mechanic and the AllUnits targeter, all of which contribute to how it is evaluated, The DealDamage mechanic's evaluator checks if it will do enough damage to kill a target. If it does, then it returns that units own evaluation score as its value, or if that unit is allied to the mechanic's owner, minus

that score. It does this for all its targets, which, when combined with AllUnits, will be all the units currently in play. Finally, this has to be multiplied by the triggers value, which estimates how many times the effect will occur. With OnPlay this is simply 1, but with more complex triggers it might be more or less. The final result of all of these factors is that DefaultAI will consider Flame Ifrit more valuable to play if its effect will kill the A.I.'s opponent's units, but less valuable if it will kill its own units. If it will kill both, it will consider the relative value of the units lost.

Figure 22: Flame Ifrit a Unit With a Situational Ability

If DefaultAI has a choice of targets, it will enumerate over all of them and check which one gives it the highest score. It will then consider the score of the card to be equal to the score of the card given its best target.

Once DefaultAI has assigned numeric scores to each of the cards in its hand, it will use the Dynamic programming in-advance Knapsack algorithm to create a subset of those cards that have total cost less than or equal to its energy and maximize total utility. It will then cast one of the cards in that subset, then reevaluate the board and do it again. It might seem natural to cast all the cards in the set instead of reevaluating, but in some cases, casting a card causes new information to become available, for example, casting a spell might cause DefaultAI to draw new cards which are even better than what DefaultAI had previously planned to play.

3.3.2 Deciding What to Block With

When its opponent attacks it, DefaultAI must decide how to block. To do this, it enumerates every possible block that each of its units could make. It then categorizes these blocks into four cases. Those cases are only the attacking unit will die, only the blocking unit will die, both units will die, or neither unit will die. DefaultAI always makes blocks in the first or third cases, and in the fourth case, where both units die, it evaluates the relative power of both units (using the card evaluation method described above) and only makes the block if its unit is less valuable than the one it will be trading for. DefaultAI will only make blocks where its unit dies, and the attacker survives if it would otherwise take lethal damage. Notably, DefaultAI does not consider blocking a single unit with multiple units, despite the fact that this is legal. This is a known flaw and has been left as an exercise for A.I. tournament contestants.

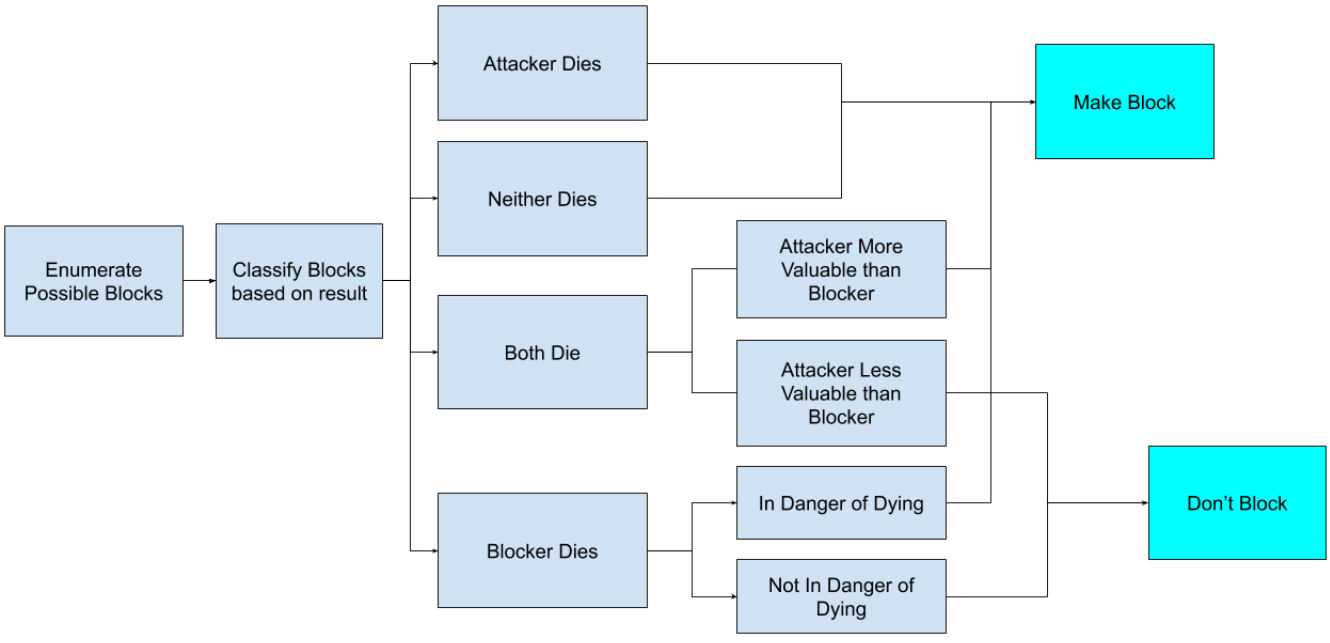

Figure 23: The Process of Deciding Whether to Make a Block

3.3.3 Deciding What to Attack With

Fifth Aeon's attack logic is the reciprocal of its defense logic. When deciding if it should attack with a particular unit, it enumerates all the enemy units that could block that unit. If it would not choose to make any of those blocks, then it makes the attack. This strategy is rather conservative and does not allow DefaultAI to strategically sacrifice attackers in order to damage its opponent, even if they are low in health.

3.3.4 Deciding What Resource to Play

DefaultAI decides which of the four resources to play based on which will unlock the most possible plays. To do this, it calculates how many of each resource it would need to play any cards in its hand that it does not already meet the requirements for. It then chooses the card that will get it closest to being able to play the largest number of them, favoring cards which it can play soon. If DefaultAI can already play every card in its hand, it instead uses the same algorithm to maximize how many cards it can play in its deck based on its decklist. Finally, if it already has the right resources to play every card in its hand and deck, it plays whatever resource its deck has the highest average total of.

3.3.5 Deciding How to Make a Choice

There are several situations in which a player in Fifth Aeon has to select a number of card from a set. This is referred to by the engine as a choice. Choices are used in several different situations. For example, at the beginning of the game, a player gets the choice to replace any of the cards in their starting hand. If they choose to

replace a card, that card is shuffled into their deck then they draw a new card. A different type of choice is a deck searching ability. These abilities let a player search their entire deck for any card they want.

DefaultAI handles these diffident kinds of choices by having a heuristic for each type. The main heuristic it uses is the draw-heuristic which estimates how much it would like to draw a given card. It estimates this by calculating the distance between its current energy pool and the cost of the given card. Cards which are dramatically higher or lower than DefaultAI's current energy are unlikely to be good because they are probably will either take too long to become playable or are already irrelevant. DefaultAI uses the draw-heuristic to handle several kinds of choices. It will mulligan a card if that draw-heuristic is less than the draw-heuristic of the average card in its deck. When DefaultAI is forced to discard cards, it will discard ones with the lowest draw-heuristic value. Finally, when DefaultAI gets to search for a card, it will choose the one with the highest draw-heuristic value.

3.3.6 Building on Top of DefaultAI

DefaultAI can be used as a based for creating new A.I. players to submit to the competition. The cleanest way to do this is by creating a new class which extends DefaultAI. Then, override the functions of DefaultAI to customize its behavior. For example, BerserkerAI is a simple modification of DefaultAI that always attacks with everything it can, and never blocks. It does this by overriding DefaultAI's standard attack and block functions.

```
import { DefaultAI } from '../game_model/ai/defaultAi';

/**
```

```
 * Berserker is a version of the default A.I. that always makes every av
 *
 * As a consequence, this A.I. is unlikely to be very competent.
 *
 * Otherwise it is identical to DefaultAI
 */
export class BerserkerAI extends DefaultAI {

    /** Attack with all legal units */
    protected attack() {
        let potentialAttackers = this.game.getBoard()
            .getPlayerUnits(this.playerNumber)
            .filter(unit => unit.canAttack());

        for (let attacker of potentialAttackers) {
            this.game.declareAttacker(attacker);
        }

        return true;
    }

    /** Never block anything */
    protected block() {}
}
```

3.4 The Auto Balancer System

The auto balancer is a system that attempts to balance a new proposed "Target" card against an existing "Goal" card, modifying the selected cared until it becomes equally as powerful as the Goal card. In order to achieve this, the balancer is given both the Target and Goal cards as input, along with a list of parameters for the Target card that it is allowed to modify.

For example, depending on designated parameters, the balancer may be permitted to modify the cost of a unit, but not its attack or defense. Conversely, it could be asked to modify the attack and defense, but not the cost. This allows the designer to adapt the system to fit their needs. If they notice that a particular faction lacks sufficient six cost creatures, they could ask the balancer to fit a card into that slot.

Once the system has been given these inputs, it attempts to search for the set of parameters which best balances the target card against the goal card. In order to do this, the system applies a search method to determine what combination of parameters to try next. It then applies the scoring method to try to determine how balanced that particular combination of cards is, with lower scores being judged as better. It does this until the search method tells it to terminate, at which point, it returns the n-best parameter sets as well as their scores. Currently, the system has one search method, a comprehensive search, which tests every possible combination of parameters. Comprehensive search is simple and works well, but can become intractable when there are a large number of parameters.

The current scoring method is full injection. This scoring method takes a list of decks as its input. It then proceeds to inject four copies of the target card, with the given parameters, into a clone of each of the decks it receives. After this, the same process

is replicated using the goal card. Then, a tournament is run between the decks with the target card and the goal card. The score is the difference between the target card's win rate and 50%. This method is good for testing how strong the target card is in a wide range of decks and circumstances. This method is especially useful for indicating how strong the constructed card might be in a limited environment where players don't have full control over their decklists.

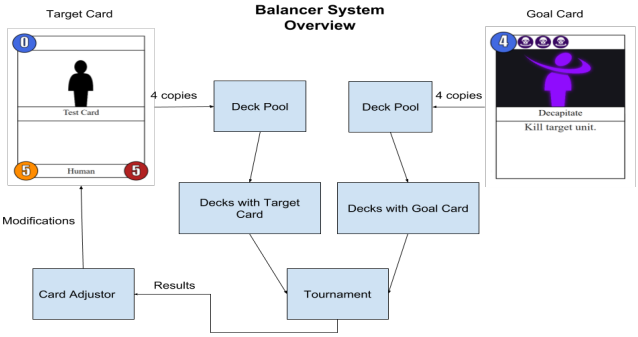

Figure 24: Balancer System

Chapter 4

USAGE

4.1 Playing the Fifth Aeon Game

Fifth Aeon is a browser-based game and does not require any installation. It can be played on a modern browser at https://fifthaeon.com (Chrome is recommended). The game features a tip system that explains each element of the game as it appears.

4.1.1 Winning the Game

The standard Fifth Aeon game's goal is to defeat your opponent before they defeat you. The normal way to do this is to reduce their life total from its starting point of 25 health, to zero or less. You must also keep your life total above zero as your opponent's goal is the same as yours. There are a few additional ways to win via special card effects, but these are much rarer.

4.1.2 Resources and Cards

In order to progress towards victory, you will need to play cards. Cards do many different things, but they are all designed to help you win the game.

In order to play a card, you must first have the right resources. There are four faction resources synthesis, growth, renewal, and decay as well as one additional resource, energy. During each turn, you may choose one of the four faction resources. This will increase your resource of that type by one. It will also increase your maximum

energy pool by 1. All cards require energy to play, but most require only one of the four faction resources. The faction resources act as prerequisites to playing a card and are not used up by playing cards. energy, on the other hand, is consumed when you play a card and will recharge during the next turn.

Cards that are currently playable are brighter than those you cannot play. You can play them by clicking on them. If the card requires targets, you will also be required to select a valid target. Once this is done, the card will be played and any energy you used to play it will be depleted until your next turn.

There are four types of cards in the game, spells, units, items, and enchantments.

4.1.3 Spells

The simplest type of card is a spell. When a spell is cast it has an immediate effect based on the rules in its text box. As soon as its effect is done, it goes to the crypt and cannot be used again.

4.1.4 Units

The next type of cards is units. Units, like spells, may have immediate effects upon being played. In addition, they remain on the board where they can attack, block, and have reoccurring triggered effects. Attacking is the primary means of dealing damage to your opponent, and will be covered on the next section.

In addition to their special abilities, all units have an amount of attack and life, which are displayed at the bottom of the card. These statistics affect how effective they are at attacking and blocking.

4.1.5 Items

Items are a type of card that must be attached to a unit. In order to play an item, you must have a valid unit that it may attach to. When an item is attached to a unit, it will increase its attack and life by the item's attack and life. It will also grant its host unit any special abilities it has.

4.1.6 Enchantments

Enchantments are cards that remain on the board like units. However, they cannot attack or block. Instead, they have lasting effects that modify the game over time. Enchantments cannot be killed in the same way units can be. Instead every enchantment has a level of power, as well as an empower cost. During their turn, the owner of the enchantment may pay its empower cost to increase its power by one. Similarly, the owner's opponent may pay the empower cost to diminish its power by one. When an enchantment has no power left, it is dispelled and goes to the crypt.

You can empower or diminish an enchantment on the board by clicking on it during your turn if you have enough energy to pay the empower cost.

4.1.7 Attacking and Blocking

The primary way to damage your opponent and thus move closer to winning the game is by attacking them with units. The primary way to avoid attack damage is by assigning units to block your opponent's attackers. Both have specific prerequisites.

4.1.8 Attacking

In order to attack, a unit must be ready and not be exhausted. Units do not become ready to attack until the turn after they are played (units cannot attack the unit they are played). When a unit attacks, it becomes exhausted, and cannot attack or block until it refreshes, at the start of its owner's next turn.

You cannot choose what target a unit attacks. That choice is up to your opponent when they assign blockers. If a unit is not blocked it will deal damage equal to its attack power to your opponent's life total.

4.1.9 Blocking

When a player is attacked, that player may use their own units to block the attackers. In order to block, a unit must not be exhausted or under a special ability that prevents it from blocking. Each valid unit may block one attacker, however, a player can also use multiple units to block a single attacker.

When all blockers are declared, the defending player must pass. Then the attack will be resolved. Any attackers that have been blocked will fight with the unit or units blocking them. Both attackers and defenders will deal damage to each other equal to their attack. Any units reduced to, or below zero, life will die. Non-lethal damage will remain on units until they refresh at the beginning of their owner's turn. At that point, the Units will regenerate to maximum life.

If a unit is not blocked it will deal damage the defending player instead. If the defending player's life total is reduced to zero or less, they will lose the game.

Some units have evasion abilities, and cannot be blocked normally.

4.1.10 Stages of a Turn

At the start of a player's turn, that player's units refresh and they draw a card.

Then they enter their first play stage. During this stage, they may gain resources, play cards, or declare attackers. When they are finished they can pass the phase.

If the active player declared any attackers, and their opponent has valid blockers, then the block stage will begin. During the block phase, the defending player can assign blockers. Once they are done, combat resolves, and the second play stage begins. If the active player did not declare any attackers, their turn will end immediately.

The second play stage is identical to the first play phase, except that attackers cannot be declared. There is only one combat stage per turn. After the active player is done, they can pass and their opponent's turn will begin.

A player can only play one resource per turn and must do so during either the first or second play phase. Players are not allowed to end their turn until they have played a resource.

4.2 Using the Fifth Aeon Bot Toolkit

The BTK can be installed from its Github repository at https://github.com/Fifth-Aeon/Bot-Tool-Kit. Full instructions are available in the readme.

4.3 Running a Fifth Aeon Competition

Most of the tools needed to run a Fifth Aeon Competition are in the BTK. The same tools that are used to do a test run as a contestant can be used to run the

final tournament. In addition, the Fifth Aeon client and server contain a tournament admin panel which allows an organizer to see all the current teams and download their latest submissions.

4.4 Using the Automatic Balancer

The automatic balancer is deployed as a component of the Bot Tool Kit repository, as they share a large portion of code. It can be used by running the command ‘npm run balance‘. Configurations to test can be edited in the testBalancer.ts file.

Chapter 5

EXPERIMENTS AND RESULTS

5.1 A.I Competition

5.1.1 Experiment Design

The Fifth Aeon tournament results were decided by running the three standard tournaments as defined in the Bot Tool Kit on all of the A.I. agents submitted through the website.

1. All of the latest submissions from each team were downloaded through the admin page.
2. All the zip files were extracted.
3. All deck.json files were moved to the data/decks folder.
4. All bots.ts were moved to the src/bots folder.
5. The importBots.ts file was modified to include all the submitted bots.
6. The Bot Tool Kit was run three times, each time with a different one of the standard configurations. The results were copied into a text file.
7. The average winrate for each bot was computed across all three types of tournaments to decide the overall winner.

5.1.2 Results

The final round of the 2019 Fifth Aeon A.I. competition was run on May 24th and included submissions from two participants known by the pseudonyms Tiggy and N8. They submitted bots known as VII and Sunpraiser respectively. The A.I. packaged with the game (DefaultAI) was also included in the competition for comparison. As you can see from Table 5.1, VII came in first, followed by Sunpraiser in second and DefaultAI in third place. Both VII and Sunpraiser were built on top of DefaultAI and both were able to improve it, but VII was stronger in all three formats.

Table 9: Fifth Aeon Tournament Winrates

	Constructed	*Limited*	*Preconstructed*	*Total*
VII	55.86%	70.75%	59.50%	62.04%
Sunpraiser	47.70%	41.00%	39.00%	46.73%
DefaultAI	46.44%	38.25%	51.50%	41.23%

5.1.3 Analysis of Sunpraiser

Sunpraiser is a modification of DefaultAI that attempts to fine tune it to be more skilled at using its two constructed decks, 'Praise the Sun" and 'Secondary Praise," Both of these decks employ the same basic strategy: to play defensively, gain a large amount of life, and then use a card called named "Overwhelming Radiance" to win the game. Sunpraiser mostly uses the same code as DefaultAI adding about 100 extra lines to optimize its unique strategy.

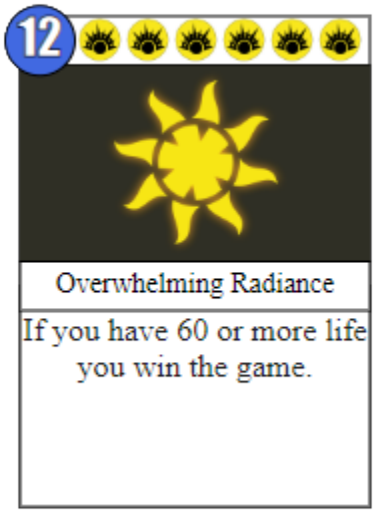

Figure 25: Overwhelming Radiance the core of Sunpraiser's strategy

Sunpraiser was optimized towards the goal of attaining a high life total in several ways. It avoids attacking with units that are key to its strategy, even if it might be advantageous to do so. It is generally more conservative about attacking because it does not aim to win by damaging the opponent. Sunpraiser will only attack if it has many attackers or one very powerful one.

Sunpraiser was more successful than the DefaultAI in the constructed tournament, showing that its strategy and optimizations were beneficial. However, it did not do better than DefaultAI in preconstructed as it was not built to do so.

5.1.4 Analysis of VII

VII is more of a generalist than Sunpraiser. It aims to improve upon DefaultAI in a variety of ways by fixing various flaws, dealing with edge cases, and making improvements based on empirical testing against other bots. VII is a substantial modification of DefaultAI and approximately doubles the number of lines of code.

Here is a list of changes from the author's comments with my comments in brackets:

1. Improves limited deck building by considering the faction types and resource costs of cards to build a proper curve.

2. Values attack more highly than health in trades. [Presumably, this was better in empirical tests.]

3. Changes the value of unit based on enemy or allied lichs. [Liches are units that get stronger when a unit dies, the DefaultAI does not consider this when making trades.]

4. Takes more global effects into play such as the effect of Death's Ascendancy. [Death's Ascendancy is an enchantment that makes some units stronger and others weaker. Again the DefaultAI doesn't consider it.]

5. Attacks if the A.I. can guarantee lethal damage regardless of trades.

6. Considers whether it is best to leave a unit on defense, even if its a good attacker.

7. Considers chump attacks. (These are attack where the player will lose a unit in a bad trade, but some damage is guaranteed.)

8. Considers chump blocking if its health is more valuable than the unit it would sacrifice.

Overall these changes made the A.I better across all modes, but especially in the limited format where VII performed dramatically better than its competitors.

5.1.5 User Feedback

The contestants were asked to fill out a survey about their experience with the competition. The questions and their responses are listed below.

- Please describe your goals in entering the competition.
  - Try to make an ai that plays a deck based around not attacking.
  - 1: Learn a new language. 2. Win. 3. Have fun.
- The Bot Tool Kit (BTK) was easy to use.
  - Agree
  - Neither agree nor disagree
- The Bot Tool Kit had sufficient features for the competition.
  - Agree
  - Strongly Agree
- Please describe any aspect of the BTK that you especially liked or disliked.
  - I liked figuring out how to make a ai that plays a bad deck better than other ais can.
  - Like: How easy it was to set up different kinds of tournaments against the default bot. Disliked: How sometimes the tournament would error out and I would lose hours worth of testing time without showing me what the results were so far.
- If you could add one feature to the BTK what would it be?

  - Being able to physically see the hands and board of each player in some form of GUI then being able to watch and replay parts of the game.

- I was successful in fulfilling my goals for the competition
  - Agree
  - Agree

- Please describe what goals you were not able to fulfill.
  - I was unable to figure out how to change the general turn actions outside of attacking.
  - My bot uses way to many arbitrary constants set specifically by me and not mathematically or experimentally the best. It also has a lot of ‘ “jerry-rigged” portions that are meant to solve one particular edge case. It also has next to zero comments aside from what features I have completed.

- What aspects of the competition contributed to your successes?
  - Nearly instant answering of my questions on discord, how well documented and commented the default bot is, and tournament testing.

- I was provided with plenty of educational materials (tutorials, discord assistance, etc) to help me write my bot.
  - Agree
  - Strongly agree

- If you could have one additional educational resource, what would it be?
  - A 15min video or website with pictures that details the entire process from downloading, to installing, to creating a new bot, to making one small

change to the bot, to setting up a new tournament, to actually running the tournament.

5.2 Automatic Balancer

5.2.1 Automatic Balancer Experiment Design

In order to test the Automatic Balancer system, we first construct a target card with specific traits or mechanics. For example, our initial tests featured a unit with ten life, ten attack, and no abilities, but with an unspecified cost. For the sake of simplicity, we chose various cards where we had a reasonable idea of what they should cost and asked the balancer to choose that cost. The system is also capable of other balancing tasks, such as changing a unit's attack power.

Once the target card has been declared, we must supply the system with a set of decks to test it in. Ideally, this would be the set of tier-one decks, determined by a competitive gaming community. Because Fifth Aeon does not have a competitive community, the game's list of standard A.I. opponent decks is used instead. These decks encompass all four factions, as well as a number of strategies such as aggression (to try to win quickly) and control (to avoiding losing in the early game, then win using powerful-late game cards).

Following the construction of the template decks, we select the goal card to balance the target against. By balancing against an existing card with a relatively known strength, we are able to tailor the power of the generated card. One of the more powerful cards in the game may be selected if a large impact on the meta is the desired outcome, or alternatively, an average or weak card may be selected for the purpose of adding simple diversity.

Next, we designate the search parameters. These must be adjusted to ensure the system runs in a reasonable amount of time. Ideally, the system would do a comprehensive search over all parameters with a large number of trials. However, this approach often takes requires too much time to be practical. Thus, it is useful to be able to limit the search space when using this method. This can be done using designer knowledge, or by running the search with a small number of trials to get an initial impression of what ranges of values are likely to be correct. The system can then be rerun on that range with a larger number of trials to ensure that any results obtained are not the result of chance. For our experiments, we ran experiments with ten decks, and ten repetitions. This lead to a total of 2,000 games per tournament (10^2 [from decks] $\times 2$ [from first or second] $\times 10$ [from repetitions]), which allowed the balancer to run on an I7-4720HQ in approximately an hour and a half.

Once the system has been run we then see if it converges to a low score. If it does, this indicates that the target card has been successfully balanced. If it does not, then this suggests that the balancer was unable to find any parameters to make the card appropriately powerful. This result, in turn indicates that the designer may need to give the balancer more parameters to work with. The system then outputs a final card design, which has the lowest score among the tested options. We then judge if this card seems balanced according to our qualitative knowledge as designers to see if the score assigned by the system is reasonable.

5.2.2 Automatic Balancer Results

While the system is capable of adjusting any numeric parameter of a card, we focused on trying to find the best energy cost for a card among the common energy costs, which range from zero to ten energy. In every case, the test card used was ‘Decapitate”. This card is a four energy, three decay cost card with the effect ‘kill

target unit." We did this for several cards, including: a ten attack, ten life unit with no abilities; a three attack, three life unit with no abilities; a five attack, five life unit with the "flying" ability which makes it harder for the opponent to block it; a spell that has the effect "kill target unit" and also requires three decay, making it exactly the same as the goal card; and finally a spell that draws its controller three cards.

For the 10/10 unit the balancer was able to determine that, at low costs, the unit was very overpowered with a starting win-rate of 80%. As the energy cost is increased, the win-rate became close to 50%, at seven cost, and remained close at eight cost then dipped below it at nine cost. The best score for this card was attained at eight cost, so that was the final cost the balancer assigned.

The next test was the 5/5 flying unit, which had a very similar pattern. While it has significantly lower stats, the flying ability is very powerful and considerably increases the usefulness of the unit. The balancer actually considered it to be more powerful than the 10/10 unit, assigning the flyer a final cost of nine energy.

The 3/3 unit gave a $> 50\%$ win-rate at zero, one, and two cost before dropping below 50 at three, which was the final cost it was assigned. This unit continued a general trend of getting worse as the cost was increased above three. However, notably that the win-rate doesn't decrease that much as it becomes highly over-costed. This is probably because, at some point, the A.I. stops playing it most of the time. Therefore, the difference between being over-costed by three or four energy is not very significant. Similarly, this result explains why its easier to get significantly above 50% win-rates than it is to get significantly below them. Having four copies of a terrible card does not hurt a deck as much as having four copies of an overpowered card helps it. Due to this effect, a weak card at worst is equivalent to having one less draw.

The next test was run using the kill spell. It had exactly the same ability and decay prerequisite as the goal card. Thus, we expected the system to give it the same energy cost as that card (4). It did eventually reach this value, achieving a 50% winrate (score of 0) at cost zero. This result served to verify the sanity of the system.

Finally, we tested a spell with the effect 'draw three cards". The balancer thought this was only slightly overpowered at zero energy cost and close to balanced at one energy cost. This assignment made it similar to the Magic the Gathering card Ancestral Recall, one of the strongest cards ever printed. It then believed that for any cost above two the card was too weak. Unlike the other tests, which mirrored our insight as designers, this result seemed very off. We would have expected it to be assigned a cost of at least 4 energy.

5.2.3 Varying Attack Power

In the next set of experiments, we assigned each of four Units a fixed cost of seven energy, then changed their attack power. We saw a general trend of increased winrate increasing along with attack power, as was expected. However, there was some noise in this trend, with some higher attack powers giving units lower win-rates. This might indicate that the balancer system needs more trials to give the most accurate results.

The most significant changes occurred within the range of 3-5 attack power. This trend may be indicative of the relative value of the attack stat within Fifth Aeon. While dealing damage to the opponent's cards and health are necessary to achieve victory, most of the combat in the game relies upon board-clearing and defensive mechanics. Since these mechanics often function independently of a card's statistics, the ability to remove cards through unit combat is devalued. This interaction would seem to be supported by the win-rate of the flying unit tested. Since flying units are

well suited to bypassing player defenses, the card would most likely not be engaging in direct combat with other units. As player damage is a less common mechanic than those used for board control, this would grant attack placed on a flying unit much higher value within the game space. In a different game, where a higher value is placed on aggressive unit interactions, a card's attack value may show a higher impact on game results.

The final results were the X/10 unit was assigned to be at 6/10 (with a 7/10 being considered equally good). The X/3 unit was assigned to be a 9/3. The X/5 flying unit was assigned to be a 3/5 and the 5/X relentless unit a 5/8. The two most suppressing results are the X/3 where it never managed to accomplish even a 50% win-rate but nonetheless chose nine instead of ten attack. This is probably due to some noise in the system causing higher attack powers not to always result in higher win-rates. The X/5 flier is also somewhat surprising in that it ended up with very low stats for a 7 cost unit at only X/5. This might indicate the "Flying' keyword in Fifth Aeon is very powerful.

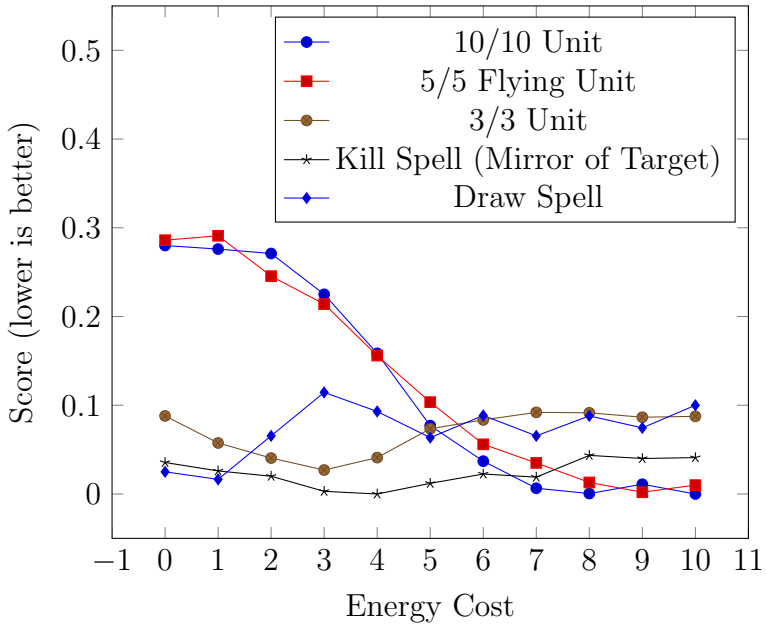

Figure 26: The Change in Score as Energy Cost is Changed.

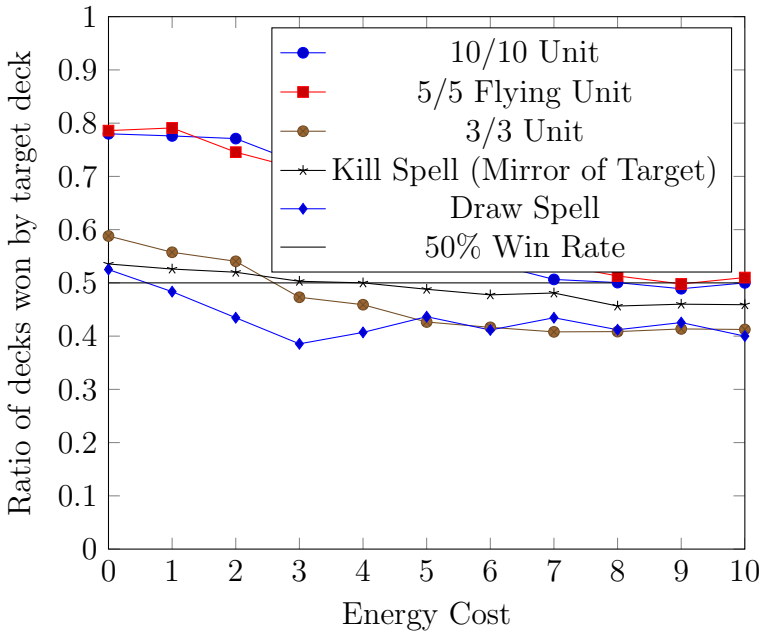

Figure 27: The Change in Winrate as Energy Cost is Changed.

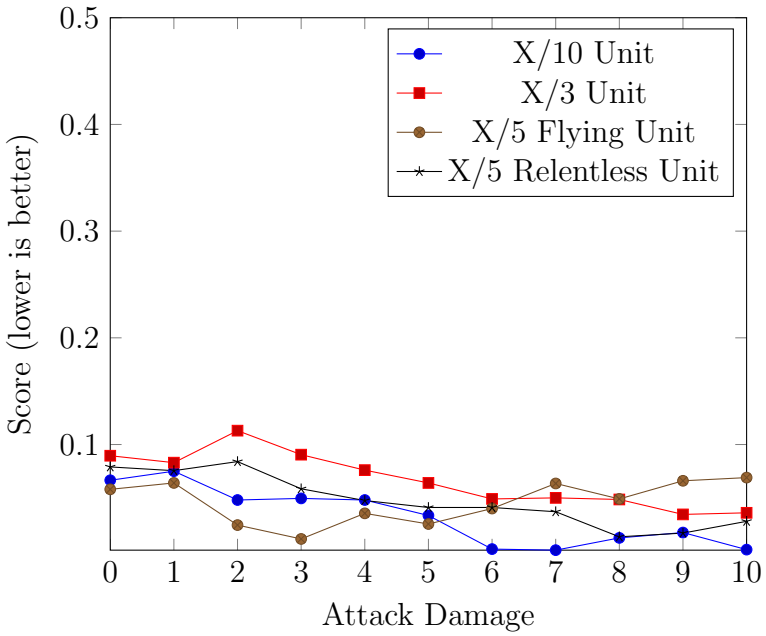

Figure 28: The Change in Score as Attack Damage is Changed.

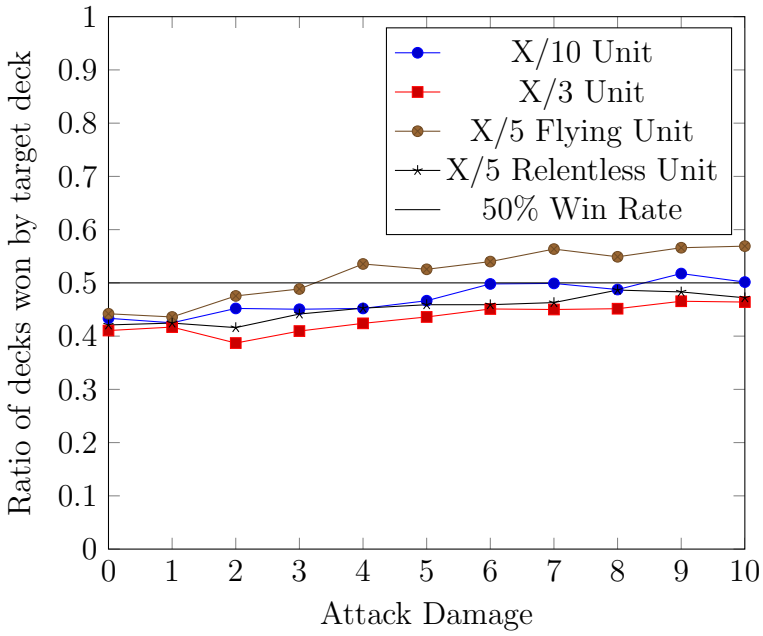

Figure 29: The Change in Winrate as Attack Power is Changed.

Chapter 6

FUTURE WORK

6.1 Future Work in Automatic Balancing

The Automatic Balancing System put forth in this paper has several limitations on which future papers could improve. The first is the A.I. player used to run games known as DefaultAI. That A.I. is based on a series of heuristics that attempt to model the game's mechanics mathematically. However, in practice, it is weaker than skilled human players. Thus any future work on creating stronger, and potentially superhuman, Fifth Aeon A.I.'s could be applied to make the balancing System stronger.

The System could also be improved by adding additional methods of scoring (deciding when a card is balanced) and searching (strategically exploring the parameter space). The current scoring method places the target and goal cards into a wide variety of decks and measures their average strength across them. But in some cases, the average strength of a card across many decks is less relevant than its strength in the most powerful deck that can use it. Therefore, another scoring metric that measured this strength would be useful. Similarly, the comprehensive parameter search used by this paper works well as long as the parameter space is small, but becomes computationally intractable as it grows too large. Therefore, more advanced methods, such as evolutionary search, could be applied to make the system work with larger inputs.

Another limitation is the requirement to have a preexisting set of decks to test with. Future work into the automatic balancing of collectible card games could look into trying to automatically determine a deck-list that would be a compliment to a target card. Automatically determining deck lists could be very difficult because the space

of potential deck-lists is vast. However, if this problem could be solved, it would eliminate the need to have expert players determine the most powerful decks. This would also mean that there would be no chance of being blindsided by a novel deck that was previously not competitive but became very strong with the introduction of a new card. This would make the System far more practical to apply to collectible card games that release entire sets of cards at a time, which is currently the paradigm for most commercial games.

Finally, there is room for the System to be applied to procedural generation. It could be used to balance the output of any system which outputs valid Fifth Aeon cards to make them balanced. It is important to note that while the System can detect imbalances in card power using win-rate, it does not account for the novelty of the cards. Since no testing was done employing user studies, it was not possible to verify whether the cards generated might be considered interesting or useful by players. Under this metric, it may be that the cards generated have no good position within the game, regardless of balance. However, if a system was able to produce cards that lead to enjoyable gameplay, it could be combined with this System to produce fun and balanced cards. As noted in our experiment design, far more opportunity for card modification is available. The intent behind this project was to provide a proof of concept for the viability of procedural generation in card design.

6.2 Future Work in Fifth Aeon Competitions

Consistency is a crucial aspect of successful competitions, and thus running annual Fifth Aeon competitions might lead to greater success [38]. A great deal of groundwork has been laid for this in the creation of the Bot Tool Kit and the tournament section of the client and server. However, additional work could be done to improve

future tournaments.

First, an end to end video should be created that explains how to install all required software and create a contestant's first Bot could be created, as requested by the users.

Second, new GUI integration should be added to the BTK to allow users to see Bot vs. Bot matches in real time. Currently, they can only use the GUI in A.I vs. human mode.

Third, the speed and reliability of the BTK should be improved to make testing less tedious. Ideally, each completed match would be saved into a database so that an aborted tournament can quickly be restarted, and the results of each game can be reviewed at any time.

BIBLIOGRAPHY

[1] Angry birds 2018 results. https://aibirds.org/angry-birds-ai-competition/previous-results.html.

[2] Cal Poly Github. http://www.github.com/CalPoly.

[3] Computer go results. http://www.computer-go.info/events/.

[4] Ms. Pac-Man results. http://www.pacmanvghosts.co.uk/results.html.

[5] Ms. Pac-Man rules. http://www.pacmanvghosts.co.uk/problem.html.

[6] The history of magic. https://magic.wizards.com/en/content/history.

[7] Wccc results. https://icga.org/?page_id=2469/.

[8] G. Andrade, G. Ramalho, A. Sandro Gomes, and V. Corruble. Dynamic game balancing: An evaluation of user satisfaction. In *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*, AIIDE '06, Palo Alto, California, USA, 2006. AAAI.

[9] G. Andrade, G. Ramalho, H. Santana, and V. Corruble. Automatic computer game balancing: A reinforcement learning approach. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1111–1112, New York, NY, USA, 2005. ACM.

[10] T. Atkinson, H. Baier, T. Copplestone, S. Devlin, and J. Swan. The text-based adventure ai competition. *IEEE Transactions on Games*, 2019.

[11] P. Beau and S. Bakkes. Automated game balancing of asymmetric video games. In *Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games*, CIG '16, Piscataway, NJ, USA, 2016. IEEE.

[12] A. Bhatt, S. Lee, F. de Mesentier Silva, C. W. Watson, J. Togelius, and A. K. Hoover. Exploring the hearthstone deck space. In *Proceedings of the 13th International Conference on the Foundations of Digital Games Article No. 18*, FDG '18, New York, NY, USA, 2018. ACM.

[13] M. Buro and D. Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106–106, 2012.

[14] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[15] P. I. Cowlin, C. D. Ward, and E. J. Powley. Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. Piscataway, NJ, USA. IEEE.

[16] P. I. Cowlin, C. D. Ward, and E. J. Powley. Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):241–257, Dec. 2012.

[17] F. de Mesentier Silva, S. Lee, J. Togelius, and Andy Nealen. Evolving maps and decks for ticket to ride. In *Proceedings of the 13th International Conference on the Foundations of Digital Games Article No. 48*, FDG '18, New York, NY, USA, 2018. ACM.

[18] B. Entertainment. Hearthstone, 2018.

[19] B. Entertainment. Starcraft remastered, 2018.

[20] R. Games. League of legends, 2017.

[21] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62–62, 2005.

[22] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[23] M. Hendrikx, S. Meijer, J. van der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):22, February 2013.

[24] V. Hom and J. Marks. Automatic design of balanced board games. In *Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE '07, pages 25–30, Palo Alto, California, USA, 2007. AAAI.

[25] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic. Evaluating competitive game balance with restricted play. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE '12, pages 26–31, Palo Alto, California, USA, 2012. AAAI.

[26] G. L. Zuin, Y. P. A. Macedo, L. Chaimowicz, and G. L. Pappa. Discovering combos in fighting games with evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 277–284, New York, NY, USA, 2016. ACM.

[27] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas. Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 320–323. IEEE, 2013.

[28] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas.

Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 320–323, Oct 2013.

[29] S. M. Lucas. Ms Pac-Man competition. *ACM SIGEVOlution*, 2(4):37–38, 2007.

[30] J. Ludwig and A. Farley. A learning infrastructure for improving agent performance and game balance. *Proceedings of the AIIDE*, 7:7–12, 2007.

[31] T. Mahlmann, J. Togelius, and G. N. Yannakakis. Evolving card sets towards balancing dominion. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, IEEE '12, Piscataway, NJ, USA, 2012. IEEE.

[32] J. McCarthy. Ai as sport. *Science*, 276(5318):1518–1519, 1997.

[33] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[34] J. Renz. Aibirds: The angry birds artificial intelligence competition. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[35] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[36] A. Stiegler, K. P. Dahal, J. Maucher, and D. Livingstone. Symbolic reasoning for hearthstone. *IEEE Transactions on Games*, 10(2):113–127, June 2018.

[37] G. Sutcliffe. The cade atp system competitioncasc. *AI Magazine*, 37(2):99–101, 2016.

[38] J. Togelius. How to run a successful game-based ai competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(1):95–100, 2016.

[39] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[40] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[41] J. Togelius, S. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, et al. The 2007 ieee cec simulated car racing competition. *Genetic Programming and Evolvable Machines*, 9(4):295–329, 2008.

[42] G. van Lankveld, P. Spronck, H. J. van den Herik, and M. Rauterberg. Incongruity-based adaptive game balancing. In *Advances in Computer Games, 12th International Conference, ACG 2009, Pamplona, Spain, May 11-13, 2009. Revised Papers*, pages 208–220, 2009.

[43] V. Volz, G. Rudolph, and B. Naujoks. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 269–276, New York, NY, USA, 2016. ACM.

[44] C. D. Ward and P. I. Cowling. Monte carlo search applied to card selection in magic: The gathering. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, CIG '09, Piscataway, NJ, USA, 2009. IEEE.

[45] G. N. Yannakakis and J. Togelius. *Artificial Intelligence and Games*. Springer, 2018. `http://gameaibook.org`.

[46] S. Zhang and M. Buro. Improving hearthstone ai by learning high-level rollout policies and bucketing chance node events. In *Proceedings of the 2017 IEEE Conference on Computational Intelligence and Games*, CIG '17, Piscataway, NJ, USA, 2017. IEEE.

[47] A. Zook, E. Fruchter, and M. O. Riedl. Automatic playtesting for game parameter tuning via active learning. In *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3-7, 2014.*, 2014.