The Use of Genetic AI in the Balancing of Competitive Multiplayer Games

Student Name: Max Clayton Clowes

Supervisor Name: Tom Friedetzky

Submitted as part of the degree of BSc Computer Science with Foundation to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Context/Background - In the increasingly profitable competitive multiplayer games industry, ensuring products are fun, challenging and fair can be key to a games success. Traditionally, measuring and improving the balance of games has been costly in terms of both time and money, requiring many man-hours and large teams. Automated approaches to the process could improve game quality whilst offering ongoing time and money savings - a huge advantage to an ever-growing pool of game developers.

Aims - This project aims to highlight the potential for improving gameplay balance within multiplayer competitive games by automating the highlighting of potential balance issues. More specifically, the project explores how imbalances can be highlighted with data gathered by playing Artificial Intelligence agents against one another iteratively.

Method - This project tests the proposed approach on a two-player card game typical of the competitive multiplayer genre, developed for the purposes of this research. A Genetic Algorithm-improved Artificial Intelligence was developed to play this game. Through pitching pools of the AI against one another and recording metricised data about that gameplay, the information needed to analyse in-game cards was gathered. This data was used to draw conclusions about the games mechanics, enabling accurate manual game design changes.

Results - Gathered data presents clear issues with the initial in-game tools, allowing adjustment of the mechanics accordingly. Over several rounds of this balancing process, a quantifiably more balanced version of the game had been achieved.

Conclusions - The success of the balancing process suggests that the proposed new approach to game balancing is viable. The clarity with which test data highlighted issues, combined with the speed in which this data was gathered, make the technique particularly effective.

Keywords — Genetic AI, Genetic Algorithms, Games, Game Balancing, Multiplayer Games, Competitive Games, Meta-game

I INTRODUCTION

A Introduction to Competitive Multiplayer Games

Multiplayer competitive game development is an ever-growing multi-billion-dollar industry (*The games industry in numbers* 2016). In many of these games two or more players are pitted against one another, defeating their opponent(s) through strategy and an arsenal of tools. Many

factors determine a games success - creativity, artistic value, addictiveness - but ultimately a game must strike a balance between providing a challenging yet fun experience for all players involved. In an increasingly crowded industry, any tool that helps achieve this balance can be of great help.

B Introduction to Game Balancing

Balancing is a key consideration in any asymmetric game play experience. There are two key elements that require balancing in a typical game - the tools the players have access to, and the scenario in which they compete. Typically, these elements should not dictate the outcome of a given game instance, with the outcome instead being decided by player skill.

Balancing is particularly important in successful complex competitive decision making games. One example of this is *Hearthstone: Heroes of Warcraft*, a *Trading Card Game* (TCG) developed by Blizzard Entertainment, where players do battle utilising an array of 30 cards which each perform a unique action or effect, from casting spells to summoning magical creatures. With 877 cards at the players disposal, each interacting in complex ways, balancing *Hearthstone* is a difficult, complicated, and time consuming process but one that is key to maintaining the game's playability. Indeed, each time new cards are added to *Hearthstone* it is the play-testing and subsequent balancing that consumes most of the developer's time and resources (*Building the AI for Hearthstone* 2014).

Imbalances lead to homogenous gameplay styles and frustration with the experience, ultimately leading to fewer people playing a given game. This means that testing a game's balance *before* the release of a game or game update is crucial. Balancing is a difficult process; a tool's theoretical potential may be very different to its performance in action, and so it is only through contextual testing that a card's actual value can be seen. Balancing techniques vary from anecdotal analysis and *Quality Assurance* (QA) testing - a time intensive and high cost approach - to more automated methodologies, reliant on various heuristic evaluation techniques.

Storing information about in game events is standard practice in the industry and is key to understanding a game and players' behaviour. The problem is gathering meaningful volumes of data without rolling a game out to millions of real users; automation of gameplay, through *Artificial Intelligence* (AI), offers a way of gathering the necessary data. The high economic and time costs of traditional balancing mean the potential advantages of automation are clear, and the final results may be significantly more balanced (Desurvire et al. 2004) given the scalability of automated processes. Two key industry trends make efficient, accurate and rapid balancing increasingly important: smaller teams with stretched resources due to the fragmentation of developer organisations, and the move towards constant game update deployment through *patches*.

It is worth noting that a completely balanced game may actually be less fun, with less variety in gameplay and fewer dramatic moments. For this reason, it is likely that any automated solution should be employed as a complementary process alongside existing, human-lead, play-testing and balancing approaches.

C Introduction to Genetic Algorithms

A *Genetic Algorithm* (GA) is an algorithm that simulates the naturally occurring processes by which biological evolution occurs (Goldberg & Holland 1988). Genetic solutions to problems work by randomly generating an initial pool of solutions to that problem, evaluating the perfor-

mance of those solutions, and combining the best to create new solutions. Small mutations are also added, giving GAs the ability to find globally optimal solutions beyond any locally optimal solutions they might find.

This principle can be applied in the context of games as a tool for optimising AI. Where an AI entity's decision making is based on a set of input variables, randomising these variables and evolving the AI over many games based on performance can create improved AI. For example, in *Hearthstone* an AI would accumulate card preference based on a combination of several heuristics, including the card's standalone value based on the effect it will have to the state of the game, and the card's strategic value in the context of a *meta-game* amongst others.

This approach is particularly relevant for competitive multiplayer games because of their dynamic nature and the transient qualities of what is known as the meta-game. Because an in-game tool's utility is dependent on the tools used by opponents, tools become more or less powerful as others fall or rise in popularity. To play a game correctly an AI's decision making variables must be non-static - just as with a human player's analysis - and must adapt to the overall context of the game.

D Project Aims

This project aims to highlight the potential for automated-assisted game balancing by modelling the approach on a real example. The first aim of the project was the development of a competitive multiplayer TCG, similar is mechanics to the aforementioned *Hearthstone* and possessing many of the qualities of a typical competitive multiplayer game. This game, designed for the purposes of the project, was designed with intuitive consideration towards game balance, but no rigorous analysis beyond this.

The next step was to create and employ a GA-supported AI, able to play the game to a medium or high level of skill. Pools of these AI instances were to be faced against one another repeatedly, iteratively improving their AI and in doing so, gathering information on the balance of the cards at its disposal

Information on the value of each card and large quantities of other gameplay metrics - such as the percentage of times the player who started first won - was gathered to enable analysis of tools and any situational imbalances the game scenario might have. Through multiple subsequent rounds of data analysis and manual balancing the goal was a well-balanced game.

The goal was to also offer a single player experience within the game using the AI as an effective opponent to any player, making the efficiency of the AI an important consideration. An additional outcome of the project was the production of a well-balanced cross-browser-compatible web application offering single and multiplayer gaming experiences.

E Project Deliverables

The project aims were encapsulated in the following deliverables:

1. Minimum Objective:
    (a) Implement a playable competitive multiplayer game.
2. Intermediate Objectives:
    (a) Implement a basic AI to play the game.

(b) Implement a playable single player game mode that utilises a basic AI.

(c) Networking support providing players a remote multiplayer experience or delivering the AI necessary for a single player experience.

3. Advanced Objectives:

(a) Implement an effective GA-based AI.

(b) Undertake multiple balancing rounds utilising the GA AI and its analysis as a tool for highlighting imbalances.

(c) Ensure that the competitive multiplayer game is well-balanced in nature.

II RELATED WORK

In this section existing work across a variety of disciplines relating to the work of this project is explored.

A Genetic Algorithms in Games

The use of genetic algorithms in game AI is nothing new, and previous work has seen success in developing high performing AI through GA techniques. However, previous work has been oriented solely around the development of human-level AI (Laird & van Lent 2001) with the focus on providing competition to real human players.

Cole *et al.* (2004) succeeded in creating a GA-driven tuning algorithm that evolved AI players for the popular game *Counterstrike*, a competitive multiplayer *First-Person Shooter* (FPS). The authors compared a manually tuned AI implementation to the results of a GA seeded with a randomised AI population. Over many evolutions this GA was able to create AI that exceeded the level of proficiency achieved by the authors' manual work. Given past success in generating human-level AI, this work hypothesises that such AI could equally pass for human players for other purposes - specifically, gathering game balancing data.

B Algorithmic Approaches to Balancing

Given that the importance of successful balancing of games is well documented (Rollings & Morris 2004), it is perhaps surprising how little research has been conducted in this field, likely due to the commercialised nature of the industry.

Most existing published thought centres around dynamic balancing of game difficulty. One existing work attempts to lay a foundation for the study, examining the balancing of ability levels in *Role-Playing Games* (RPGs), and the opportunity for balancing these fluctuating ability modifiers using a genetic algorithm (Chen et al. 2012). The study showed that a genetic algorithm *was* able to balance the ability levels in the small prototype RPG game implemented; however, the simplistic tested model and its case study do little to convince of the approach's potential ability in balancing content in a commercial games.

C Design

Jacob Nielsen's outlines a set of usability principles (Nielsen 1994) that any user oriented project should endeavour to adhere to:

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

These principles were taken into consideration at all stages of the design process.

III SOLUTION

In this section the design and implementation of the solution is described in detail. Firstly, the implemented game is discussed, followed by design and architecture of the system and the algorithms supporting the AI. Finally, challenges encountered during the project are explored.

A The Game

Proving the efficacy of the proposed balancing process first required the construction of a two-player game to balance, designed specifically for the project so as to be representative of games the technique might be applied to.

The implemented game consists of two distinct but interacting gameplay elements. The core gameplay element - and the section which ultimately decides the victor - is a game of Tic-Tac-Toe played on a 4x4 grid. Each turn, each player places a piece on the grid, with the first to place four pieces along a single row, column, or diagonal of the grid winning the game.

The second gameplay element is a turn-based TCG, with each card affecting the state of the 4x4 game board. Each player has a deck of 20 cards, drawing one card from their respective decks each turn and using one card per turn. Each card has a distinct effect, including removing played pieces and blocking board positions, turning the solved game of Tic-Tac-Toe into an exciting and dynamic gameplay experience. Examples of these cards and their varied effects can be seen in Table 1.

Card Name	**Card Effect**	**Card Class**
Fire Blast	Damage 1 piece	Basic
Frost	Freeze all squares	Rare
Sabotage	Destroy 5 shields	Elite

Table 1: Example cards and their effects.

The class of each card determines how many of that card can be included in a deck: three of each *basic* card can be included, with two of each *rare* and only one of each *elite* card. Players

are limited to 5 elite cards (25% of their deck). This limitation is a tool for ensuring powerful cards cannot be used too many times, and it is therefore a useful balancing tool.

Each card performs one of a variety of unique effects, including:

- Damage dealing to pieces
- Shielding, preventing damage to pieces
- De-shielding
- Destroying pieces outright
- Blocking squares
- Drawing cards
- Discarding cards

Fig. 1 shows the gameplay flow for each turn. The game begins with each player drawing three cards, before the first player's turn begins. Each turn, each player can play a card and then a piece. Following each turn, the win conditions are checked for, before the next player's turn begins.

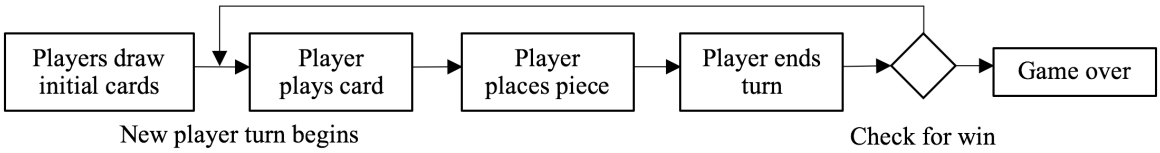

Figure 1: The flow of gameplay within the game.

Fig. 2 shows the implemented user interface. In the centre of the image a 4x4 game board is visible, where the Tic Tac Toe game is played. Directly above and below the board are the cards of the opposing players.

The game is designed to adhere to Nielsen's usability heuristics (Nielsen 1994), as outlined in the related works section, where possible. For example, as shown in Fig. 3, labelling clearly conveys the effect of a given action, supported by intuitive visuals and animations and minimising the need for recall - a key design principle. Consistency between the card descriptions and their effects is achieved as processing of effects is based on natural language analysis through regular expressions; therefore cards do exactly what their effect says they will. Some principles are less appropriate; Nielsen propones the ability for users to undo mistakes, which would potentially ruin the game.

As shown in Fig. 2, the game interface resembles a real life board and cards, almost as if a physical game were being played. This *skeuomorphy* (replication of real-world cues) ensures the interface and interactions are intuitive to users, particularly those who have played card games in the past. These intuitive cues are supported by the game rendering objects the player can currently act upon as glowing, further clarifying available interactions.

Figure 2: Screenshot of the implemented game. Note: all game artwork was created by the author specifically for the purposes of this project.

B System Architecture

As a multiplayer game, the system is built upon a server architecture that enables remote player matchmaking. The core element of the system is a server facilitating game sessions, provisioning both the front and back end, running multiple game instances at any one time, and also maintaining AI player instances. Human players can interface with this game server through a web browser on non-mobile devices.

The server runs the game manager and all AI instances, and stores data gathered on an ongoing basis. The game manager runs each of the games, interfacing with human and AI players indiscriminately. It is able to create and remove game instances dynamically based on demand; upon receiving a connection request from a player, the server queries the game manager. If there is a game waiting for players, the new player is added to it, and if no empty active games exist, one is created.

This approach means that as many games as needed may be run at any one time. With the

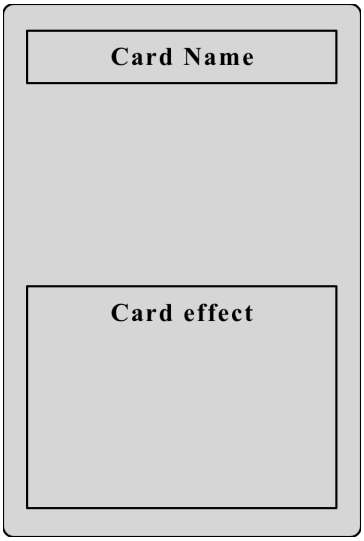

Figure 3: Card UI layout.

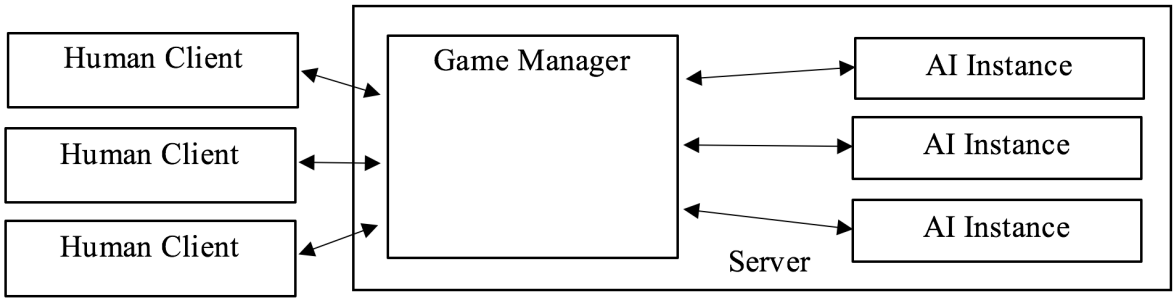

Figure 4: System architecture diagram.

number of games required for this project capped, the capacity of the server is able to predictably cope with the load, though different approaches to server architecture would be needed to support large scale use. In order to deal larger numbers of players, distributed computing techniques would be required; however, this is outside of the scope of this project.

C Tools

Developing the game and architecture previously described required the use of several tools and technologies. Because the game is played by both human and AI players, ensuring that the User Interface (UI) is detached from the core gameplay experience is important, following *Model-View-Control* (MVC) paradigms. This makes web technologies a great choice for the project, especially as networking is crucial and particularly simple to implement.

As a web-based game with many visual elements, *Canvas* - a graphics-containing HTML element - is a natural choice of technology for the visual interface (Garaizar et al. 2012). The game canvas is controlled by client-side *JavaScript*, manipulating the canvas to represent the game state and handling and mouse interactions. Note: the game is only tested for compatibility for Google Chrome on Mac OS X computers, though minor adjustments would ensure full cross-

browser compatibility.

JavaScript is an interpreted programming language typically utilised in web-development. Its flexibility and rapid deployment capabilities, as well as the cross-platform nature of web technologies, make it an appropriate candidate for the development of this project.

A particular advantage of JavaScript is its native operability with *Node.js*, a popular server-side web application environment. Developing servers with Node.js is simple, produces great results, and its compatibility with the chosen game development environment makes it a perfect choice for managing networked games, as well as running the actual AI and GA processes. In order to maintain connections with clients over Node.js, *Socket.io.js*, a JavaScript framework implementing TCP networking protocols, is used with the support of Express.

Cards and their effects, AI input variables and gathered data are stored and loaded using *JSON*, a light-weight data structure perfectly suited to the JavaScript environment.

D Balancing

There are two key factors to balance analysis: the scenario balance and the balance of each card. The scenario balance includes potential issues with the structure of the game. Traditionally in TCGs, or more generally all turn-based games, the player that goes first is at an advantage. The game's mechanics counteract this randomised inherent advantage - for example, the player going second might receive extra tools such as an extra card. This kind of issue can be found through analysing general play data, which means measuring wins, losses, which player went first, and more.

There are two aspects of a card's value to consider: the upper bound of its value (its *potential* value), and the card's average value when played. This distinction is important - for example, the card Sabotage (as shown in Table 1) destroys 5 shields. Although destroying 5 shields appears useful in principle, if in reality there are rarely 5 shields in play then the card's average contribution is typically far lower than its potential value.

Measuring the potential and average actual values of a card can be achieved by looking at its potential and average effect on the state of a game. Evaluation compares the initial game state to the game state after playing a card and placing a piece. Conveniently, this is the same calculation performed by the AI each turn (as outlined in Algorithm 2).

The game state analysis is a sum of the board state, modified by the following variables:

- The value of having a card in hand
- The value of the opponent having a card in hand
- The value of possessing a piece, modified by:

  The strategic value of having a piece in the centre of the board

  The value of a piece being shielded
- The value of the opponent possessing a piece, also modified
- The value of a square being blocked through freezing
- The value of a square being blocked through boulders

This set of variables is able to accurately score the effect of any card in the card pool. However, the value of each variable is difficult to decide upon or measure, hence the need to find the optimal variables via a GA. It is these variables that seed and define each AI instance, as the cards they choose to play is based on this combination of variables.

The balancing process followed the following structure:

1. A pool of AI instances was created with randomly assigned values for the aforementioned variables, and evolved as previously explained over many games.
2. An optimal set of input variables was found based on this evolution.
3. A set of optimal AI instances were created, which played one another, with the AI's evaluations and accumulated game data gathered and examined.
4. Based on gathered data, manual card design and game changes were implemented.
5. The process began again.

Each round ends with manual changes to cards being made to rectify highlighted issues. Changes to tools come in the forms of *buffs* - positive changes - and *nerfs* - changes that make a given card worse. Although balancing is ultimately at the discretion of the game designer, the ethos behind the implemented process is that changes should be a direct response to the metrics presented by testing.

E AI Approach

Algorithm 1 shows how the AI instances were created, managed and made to evolve over a series of rounds of games via a simple GA implementation. This evolutionary process continued until variance between solutions became small.

The crossover mechanism generated new members of the population by taking input from two existing members, with selection weighted towards higher scoring AI instances. New AI instances inherited the first random x input variables, where $1 < x < 7$, from their first parent, and the rest from the second. Additionally, there was a 20% chance of the child 'mutating' with all variables being randomly seeded instead.

Crucial to this process is a heuristic for ranking the optimality of each AI instance. The *Elo Rating System* is a system for rating and relatively ranking player proficiency (Glickman & Jones 1999) - in this case the AI. It works by awarding points for a win based on the assumed probability of a win. It is famously used in the game of Chess, but variations on the approach are used in many popular competitive multiplayer games such as *Hearthstone*, and previous research has examined its use as a evaluative tool in genetic algorithms (Cole et al. 2004). Because of the randomness involved in the game, it is important to rank the AI in a manner that accounts for the likelihood of each player having a few poor games, irrespective of their proficiency.

For a game between *players* A and B, with current ratings of R_A and R_B respectively, the probability P_{player} that *player* A will win is (Yliniemi & Tumer 2013):

$$P_A = \frac{1}{1 + 10^{-(R_A - R_B)/400}} \tag{1}$$

Algorithm 1: Genetic Algorithm for evolving AI test instances.

```
input  : N, the number of AI instances to create, I, the number of games to play
output: Optimal AI seed variables
begin
    Initialise N AI instances with randomised game state variables;
    while AI solution pool is varied do
        for each AI do
            for I times do
                Matchmake via the Elo Rating System, N times per AI instance;
                Play game, using Algorithm 2;
                Rank each AI using the Elo Rating System;
            end
        end
        Eliminate the lowest 50 instances;
        Create 50 new AI from the remaining AI, via genetic crossover method;
    end
    return seed variables of the highest rated AI instance;
end
```

Each win or loss alters the rating for each player by:

$$R_{player} \leftarrow R_{player} + K(S - P_{player}) \tag{2}$$

K takes an arbitrary initial value of 55, reduced by 1 for each game played until it reaches a value of 25, S is 1 for a win and 0 for a loss, and E_{player} is the previous probability of victory for that player, as defined above.

One risk of any optimisation algorithm is the over-proliferation of initially successful solutions resulting in the algorithm becoming trapped within local minima. The Elo system produces convergent ratings because as a player's rating increases with a win, the magnitude by which it does so is reduced, reducing the reinforcement of that solution and allowing solutions to new optimal solutions to emerge. This, combined with the mutations introduced at the crossover stage, ensures the fitness of the resulting AI.

The AI plays each game using the approach outlined in Algorithm 2. Its evaluation function works by analysing the current board state, and calculates all potential board state changes based on card choices and piece placing, with all combinations compared; if no card will produce a positive outcome, no card is played. The evaluative calculation is a simple function of the pieces in each row, column and diagonal summed, with +1 for the AI's own pieces and -1 for the AI's opponent's, with the AI's variables modifying the calculation accordingly. The value of a chosen card is then be recorded and stored for game balance analysis.

Although this AI approach is inefficient and somewhat "brute force" in approach, the algorithm is adequate for the implemented game and operates extremely swiftly - a necessity given the real-time requirements of gaming AI. With only a small number of possible decisions each turn the AI only need play locally optimum moves, rather than strategising over many turns or attempting to predict the potential plays of the opponent - tasks that are more computationally

Algorithm 2: In-game AI algorithm.

```
input : player card value, enemy card value, center mod, enemy mod, shield mod, freeze
        mod, rock mod
begin
    while no player has won do
        Evaluate game state;
        for card in hand do
            Test card value;
            Track highest card value ;
        end
        if maximum card value > 0 then
            Evaluate hand;
            Play card of highest value;
        end
        else
            Do not play a card;
        end
        Resolve card effect for maximum resultant game state;
        Place piece to maximise game state;
        Store change in game state;
        End turn;
    end
end
```

demanding. The implemented solution is able to compute decisions and make plays almost instantaneously from a human perspective.

F Implementation and Testing Issues

The project was undertaken with an Agile methodology. Although not a team project, meaning the collaborative ethos of the Agile Manifesto was not relevant, many of its other guiding 10 principles (Beck et al. 2001) still were - for example, Agiles consideration of changing requirements was particularly relevant given the nature of the Genetic Algorithms, which will require ongoing tuning and maintenance post initial development. Because of the potential scale of the project, executing the deliverables as efficiently as possible was key to its success.

In-keeping with the Agile approach, work was based on rapid lightweight prototypes at all stages, from the physical initial prototype used to clarify the rules of the game pre-implementation to the iterative development of the implemented system.

A secondary advantage of the use of AI is that it functions as a software verification tool, testing the core game's programming rigorously and highlighting even edge case bugs.

Typically a project like this would benefit from rigorous unit testing, to ensure implementation integrity. However, the nature of Javascript - specificly its asynchronicity - precluded this, making the development and testing approaches very much trial-and-error driven. Getting to grips with the peculiarities of Socket.io.js was one particular issue, as throwing the correct errors

meant running the function and waiting for niche error throwing scenarios to occur.

IV RESULTS

A Test Conditions

For the purposes of this project, three consecutive balancing rounds were undertaken. All elements of this testing adhered to the processes outlined in the previous section. Each AI player was equipped with the same deck of cards, though these decks changed between rounds based of card class changes.

As previously explained, the values associated with each card are based on the evaluated value of playing that card, resolving its effect, and then - if relevant - placing a piece. Inclusion of the piece placed is important as some cards are balanced by not allowing you to place a piece that turn. Values were only recorded when the card was actually played, meaning the data represents times when the AI deemed the situation optimal for use of that card. The absolute values are fairly arbitrary; it is their relative value that is key.

B Balancing Round 1

The optimal set of input variables for the initial pool of cards, as returned by the genetic algorithm, can be found in Table 2. The validity of the variable set was tested informally, where the AI was able to hold its own against human players.

Player Card	Enemy Card	Centre	Enemy	Shield	Freeze	Rock
88	40	1.6	1.1	1.5	2.1	4.5

Table 2: Balancing round 1 optimal AI input.

The implications of these variables are that the AI valued its own cards relatively highly, especially compared to the value of their opponent having cards in hand. This indicates successful AI were wary of the potential disadvantage discarding cards can create, but unafraid of making plays that gave cards to their opponent. This suggests that discarding cards is a viable method of balancing powerful cards.

The average and maximum values for cards played in this balancing round can be seen in Table 3.

The most obvious balance issues within this first round of data were *Floods* and *Barrage*. Both of these cards had extremely high maximum values, which is to be expected as they are both able to clear whole boards. Indeed, it is important to have tools which can massively change the state of the game in *edge case scenarios*. However, the fact that the average values for these cards was also so high - approximately double the average value of most cards - indicated a level of consistency that may be toxic to the enjoyment of the game. Because cards that allow players an ability to make a comeback a important to the game, it was decided that their effects should not be removed

Another imbalanced card was *Sabotage*. Sabotage had both the lowest average and maximum values. Although it has the potential to remove 5 shields - and a shield's protection adding around 50% to the value of a piece - the data suggested that not enough shields were ever placed to

Card Name	Card Class	Card Effect	Avg. Value	Max. Value
Armour Up	Basic	Shield a piece, Draw a card	2,928	14,770
Barrage	Basic	Damage all pieces, Discard 2 cards	5,382	23,041
Bezerker	Rare	Discard a card, Deal 1 damage, If you have the least pieces, return this card to your hand	3,181	16,370
Boulder	Rare	Discard a card, Block a square	2,275	12,880
Fire Blast	Basic	Deal 1 damage	3,412	20,923
Floods	Rare	Destroy all pieces, End your turn	4,921	23,642
Flurry	Rare	Deal 2 damage to your pieces, Deal 2 damage to enemy pieces	2,620	13,152
Frost	Basic	Freeze all squares	2,421	15,878
Ice Blast	Basic	Freeze a square	2,197	12,861
Reckless	Rare	Your opponent draws 2 cards, Destroy a piece	3,038	14,574
Sabotage	Elite	Remove 5 shields	1,267	10,170
Sacrifice	Rare	Destroy a piece of yours, Draw 3 cards	2,397	11,676
Summer	Basic	Thaw 1 square, Draw a card	2,192	17,791
Taxes	Rare	Discard 2 cards, Shield 3 pieces	2,180	14,518

Table 3: Balancing round 1 cards, effects, and values.

maximise this card's potential value. This could be rectified by adding more cards with shielding abilities to the game, but in order to maintain consistency between testing rounds, the effect was altered. Otherwise, all cards seemed to be relatively in-line with one another.

Finally, players who went first won with approximately 60% certainty. In order to counteract this the opponent was allowed to draw an extra card at the beginning of the game - a standard solution in similar games (*Magic: The Gathering Basic Rulebook* 2013).

Changes made after round one:

- **Floods** - Class: Rare $\rightarrow$ Epic.
- **Barrage** - Class: Basic $\rightarrow$ Rare, Effect: $\rightarrow$ "Damage all pieces, Discard 3 cards"
- **Sabotage** - Effect: $\rightarrow$ "Remove 5 shields, Draw a card."

C Balancing Round 2

The optimal set of input variables used in the second balancing round, can be found in *Table 4*. The validity of the variable set was once again verified informally.

Player Card	Enemy Card	Centre	Enemy	Shield	Freeze	Rock
77	53	2.3	1	1.9	0.9	1.3

Table 4: Balancing round 2 optimal AI input.

The first clear difference between rounds is that the weighted values of blocking squares (through Ice and Rock) dropped. Because of the previous value of board clears, cards able to block squares - unaffected by board clears - were particularly useful, though their evaluate values were surprisingly high. Having curbed the effect of these cards, the value of rock and ice cards dropped significantly.

The average and maximum values for cards played in the balancing round can be seen in *Table 5*. The increase in the value of the centre square and shielding lead to notably higher maximum values for the cards.

Card Name	Card Class	Card Effect	Avg. Value	Max. Value
Armour Up	Basic	Shield a piece, Draw a card	4,558	30,082
Barrage	Rare	Damage all pieces, Discard 3 cards	2,398	18,520
Bezerker	Rare	Discard a card, Deal 1 damage, If you have the least pieces, return this card to your hand	2,418	33,399
Boulder	Rare	Discard a card, Block a square	3,075	18,880
Fire Blast	Basic	Deal 1 damage	2,468	32,350
Floods	Epic	Destroy all pieces, End your turn	3,385	47,098
Flurry	Rare	Deal 2 damage to your pieces, Deal 2 damage to enemy pieces	5,278	31,190
Frost	Basic	Freeze all squares	4,349	24,260
Ice Blast	Basic	Freeze a square	3,784	14,796
Reckless	Rare	Your opponent draws 2 cards, Destroy a piece	4,042	18,104
Sabotage	Elite	Remove 5 shields, Draw a card	4,238	29,544
Sacrifice	Rare	Destroy a piece of yours, Draw 3 cards	2,986	23,583
Summer	Basic	Thaw 1 square, Draw a card	7,121	37,151
Taxes	Rare	Discard 2 cards, Shield 3 pieces	3,608	24,376

Table 5: Balancing round 2 cards, effects, and values.

This second round of balancing showed that cards that are able to damage (rather than destroy) pieces were over powered. This was probably due to the lack of protective cards available. To counteract this, *Taxes* was improved, giving players more tools for shielding their pieces.

Summer is a card that is able to free up blocked squares, making it a great tools for winning games resulting in its high maximum value. Being able to remove freeze effects is an important ability, but the ability to do so whilst also drawing a card was clearly too valuable.

Finally, *Sacrifice* displayed a low average and maximum value, suggesting that the disadvantage of having to remove one of your own pieces was not worth the ability to draw cards. Indeed, this is supported by the AI input variables which attribute more importance to pieces.

Changes made after round 2:

- **Taxes** - Class: Rare $\rightarrow$ Basic, Effect: $\rightarrow$ "Discard a card, Shield 3 pieces."
- **Summer** - Effect: $\rightarrow$ "Thaw 1 square."
- **Sacrifice** - Effect: $\rightarrow$ "Destroy a piece of yours, Draw 4 cards."

D Balancing Round 3

The optimal set of input variables for the final pool of cards, as returned by the genetic algorithm, can be found in 6.

Player Card	Enemy Card	Centre	Enemy	Shield	Freeze	Rock
97	70	1.9	1.4	1.8	0.6	0.8

Table 6: Balancing round 3 optimal AI input.

The average and maximum values for cards played in the balancing round can be seen in Table 7.

Card Name	Card Class	Card Effect	Avg. Value	Max. Value
Armour Up	Basic	Shield a piece, Draw a card	4,175	41,718
Barrage	Rare	Damage all pieces, Discard 3 cards	5,382	23,041
Bezerker	Rare	Discard a card, Deal 1 damage, If you have the least pieces, return this card to your hand	3,671	28,401
Boulder	Rare	Discard a card, Block a square	2,307	22,429
Fire Blast	Basic	Deal 1 damage	4,064	42,503
Floods	Epic	Destroy all pieces, End your turn	6,542	43,913
Flurry	Rare	Deal 2 damage to your pieces, Deal 2 damage to enemy pieces	4,482	53,652
Frost	Basic	Freeze all squares	6,159	48,727
Ice Blast	Basic	Freeze a square	2,197	25,878
Reckless	Rare	Your opponent draws 2 cards, Destroy a piece	4,746	27,197
Sabotage	Elite	Remove 5 shields, Draw a card	3,397	34,151
Sacrifice	Rare	Destroy a piece of yours, Draw 4 cards	2,397	11,676
Summer	Basic	Thaw 1 square	5,917	62,688
Taxes	Basic	Discard a card, Shield 3 pieces	3,799	38,699

Table 7: Balancing round 3 cards, effects, and values.

The data highlighted that *Frost* seemed to have become more powerful with the worsening of *Summer* - the only card able to negate freeze effects. Anecdotally, Frost is able to stagnate the game, stopping players playing any pieces for several turns, making it potentially imbalanced. To counteract this restrictive effect the card was made 'Elite', meaning one one of it can be included in each deck.

Changes made after round 3:

- **Frost** - Class: Basic $\rightarrow$ Elite, Effect: $\rightarrow$ "Freeze all squares, End your turn."

E Summary

After three rounds of balancing, the data indicated a more balanced game had been achieved. This balance was validated with informal playtesting (though human validation was out of the scope of this project). This improved balancing evidences the potential of the proposed balancing approach.

Although the changes to the cards may seem like simple adjustments, the affects of implementing any change are unpredictable and complex and can have large impacts on the player's experience. The value of having such design changes supported by data in this manner cannot be understated.

V EVALUATION

A Strengths

The project was a success in terms of deliverables. A playable competitive multiplayer game was developed, with networking allowing players to compete remotely. Matchmaking functionality supports an arbitrary number of players, making the game scalable. The game was implemented with a variety of gameplay elements and card effects, with pleasing custom visuals. Informal testing suggested the game was fun.

A single player experience was successfully created, supported by artificial intelligence. Notably, the implemented AI player algorithm is a marked success; not only is it very competitive able to reliably defeat human players, but it makes these extremely rapid decisions on a game with over 6^{16} possible board states and a huge number of possible card interactions. This consistent competency can be attributed to its evolution via the applied genetic algorithm.

The strong AI lead to success in the balancing rounds, with several subtle but important card changes being implemented based on the collected data. Gathered data highlighted not only individual cards that were over and underpowered, but common themes around imbalances, such as the bias towards the player going first.

The fine-tuned AI makes the implemented game particularly extensible going forward. Modern games are no longer finished products at launch, and undergo significant changes over time through the release of update *patches*, requiring testing capability on an ongoing basis. As an adaptive system, beyond the initial development time the approach requires no maintenance or further work. The project demonstrated potential for these techniques to be applied for the first time on an industrial scale in the Games sector, with promising operational and commercial benefits.

B Limitations

One of the clear issues with the process is that actually acting on the data and changing the effects of the cards requires human input, and any adjustments made are at the discretion of the designer. This adds a level of uncertainty and inconsistency to the process.

Another issue is that the foundation of the approach lies in tuned AI offering viable substitute to real players. Although the AI is strong, repeatedly besting human players in informal playtesting, there is certainly room for improvement. Extensions of the AI could allow strategising over many turns allowing card combinations, or even the prediction of opponents' moves allowing counter play. An awareness of how much value could potentially be extracted from each

card, and holding out for situations that maximise that value - rather than just playing local best - could also improve the AI's performance.

Another area to automate would be deck selection, changing the choice of deck and cards used based on how good those decks are. It is the other cards you come up against that primarily define a meta-game (cards are good relative to one another), but building another Artificial Intelligence system that is able to intelligently build decks was beyond the scope of this project.

Deck building is a particularly important part of typical TCGs as cards are often used in combination during a single turn and it is through these combinations that the real value of a card becomes apparent. However, in this project's game only one card can be used per turn, meaning combinations have a reduced impact on card value and do not need to be tested for.

The current system architecture is currently the primary limitation and issue with the implementation. Running all the games on a single server and through a single socket resulted in a computational bottleneck, slowing games that would otherwise finish within several seconds to several minutes. Through extending the implementation to take advantage of distributed techniques, parallelisation and a scalable number of sockets, notable speed increases could be achieved. This speed up would have a massive effect on the logistics and reliability of the testing process; more AI could be run simultaneously, resulting in more evolutions, a more finely tuned AI, and as a result better data.

C Approach

The software engineering approach to this project was very successful, with all of the deliverables being met ahead of time and to a high standard, despite the scale of the project. Rigorous specification and planning at the design stage ensured that the a potential painful development process was mitigated; indeed, the project followed the initial plan almost to the letter. A

The Agile "fail fast" ethos behind the work meant that set backs happened early but were easily recovered from, despite the used of new tools and technologies and the associated learning curve. This was particularly notable where - having built a working two player game - the majority of the code base was scrapped as it emerged that achieved the required decoupling of view and controller wasn't feasible with the use of jQuery.

The games industry is notorious for software engineering failure, with the proliferation of "crunch time" development sprints is perhaps the most telling sign of software engineering failures. This is due to their scope combined with their multi-disciplinary nature. By this comparison the delivery of a working multiplayer networked game complete with matchmaking, strong AI and automated quality assurance, under such strained resources, is a great success.

VI CONCLUSIONS

In summary, this project set out to prove by example that automated approaches could be used as a substitute for, or at the very least in tandem with, traditional game balancing approaches to improve games.

This project successfully and fully implemented a multiplayer networked game, complete with human and AI player support. AI was developed to play the game, achieving a high level of competitive success. This AI was then tracked as AI instances played against themselves repeatedly, accumulating insightful data. The initial pool of cards were improved iteratively over

several balancing rounds until at the end of the balancing process a fairer set of cards was settled upon.

The success and importance of the work lies in 2 key points:

1. The balance of the implemented game was successfully improved using information gathered from AI agents.

2. A successful game balancing process was undertaken without the need for QA testing or other traditional resources.

Further development and wide spread adoption of this approach could have notable impact on the games industry, particularly in the Indie (independent) sector, where small developer teams often lack the tools and resources to employ traditional balancing techniques.

A next step in quantifying the feasibility of this approach would be to seek to specify correlation between game balance and player enjoyment in the competitive multiplayer genre. This is key as "player enjoyment is the most important goal for digital games" (Blashki & Isaias 2013). Although previous study into the relationship between balance and fun has been done (Andrade et al. 2006), most of this relates to the difficulty of single-player experiences, rather than facilitating balanced two-player experiences.

An exciting and natural area to subsequently examine would be to support the automation dynamic difficulty adjustment. The proposed approach could be potentially be combined with some heuristic that is able to draw the kind of conclusions that were manually made about the data, and make appropriate changes before starting the balancing again, repeating until the desired balance was achieved. This could easily be done by creating a pool of possible effects and rating their degrees of positivity or negativity, and assigning them procedurally; such work have previously been explored (*Hearthstone Cards As Created By A Neural Network* 2015). However, this fully automated process, runs the risk of compromising fun for the sake of balance or becoming homogenous.

References

Andrade, G., Ramalho, G., Gomes, A. & Corruble, V. (2006), 'Dynamic game balancing: an evaluation of user satisfaction', *AIIDE 06* .

Beck, K., Beedle, M., van A. Bennekum, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001), 'Manifesto for agile software development', `http://agilemanifesto.org/iso/en/`. Accessed: 24th January 2016.

Blashki, K. & Isaias, P. (2013), *Emerging Research and Trends in Interactivity and the Human-Computer Interface*, IGI Global.

Building the AI for Hearthstone (2014), `http://www.gdcvault.com/play/1020775/Hearthstone-10-Bits-of-Design`. Accessed: 29th December 2015.

Chen, H., Mori, Y. & Matsuba, I. (2012).

Cole, N., Louis, S. & Miles, C. (2004), 'Using a genetic algorithm to tune first-person shooter bots'.

Desurvire, H., Caplan, M. & Toth, J. (2004), 'Using heuristics to evaluate the playability of games', *CHI'04 extended abstracts on Human factors in computing systems* .

Garaizar, P., Vadillo, M. & de Ipia, D. L. (2012), 'Benefits and pitfalls of using html5 apis for online experiments and simulations', *iJOE* **8**. Special Issue 3.

Glickman, M. & Jones, A. (1999), 'Rating the chess rating system', *Chance* **12**.

Goldberg, D. & Holland, J. (1988), 'Genetic algorithms and machine learning', *Machine Learning* **3**.

Hearthstone Cards As Created By A Neural Network (2015), `https://www.reddit.com/r/hearthstone/comments/3cyi15/hearthstone_cards_as_created_by_a_neural_network/`. Accessed: 21st January 2016.

Laird, J. & van Lent, M. (2001), 'Human-level ais killer application interactive computer games'.

Magic: The Gathering Basic Rulebook (2013). Accessed: 27th February 2016.

Nielsen, J. (1994), *Usability Inspection Methods*, John Wiley and Sons.

Rollings, A. & Morris, D. (2004), 'Game architecture and design: A new edition"', *New Readers* .

The games industry in numbers (2016), `http://ukie.org.uk/research`. Accessed: 25th January 2016.

Yliniemi, L. & Tumer, K. (2013), 'Elo ratings for structural credit assignment in multiagent systems'.