STEP: A Framework for Automated Point Cost Estimation

George E.M. Long, Diego Perez-Liebana, Spyridon Samothrakis

***Abstract*—In Miniature Wargames such as Warhammer 40k, players control asymmetrical armies which include multiple units of different types and strengths. These games often use point costs to balance the armies. Each unit is assigned a point cost, and players have a budget they can spend on units. Calculating accurate point costs can be a tedious manual process, with iterative playtests required. If these point costs do not represent a units true power, the game can get unbalanced as overpowered units can have low point costs. In our previous paper we proposed an automated way of estimating the point costs using a linear regression approach. We used a turn-based asymmetrical wargame called Wizard Wars to test our methods. Players were simulated using Monte Carlo Tree Search, using different heuristics to represent playstyles. We presented six variants of our method, and show that one method was able to reduce the unbalanced nature of the game by almost half. For this paper, we introduce a framework called Simple Testing and Evaluation of Points (STEP), which allows for further and more granular analysis of point cost estimating methods, by providing a fast, simple, and configurable framework to test methods with. Finally, we compare how our methods do in Wizard Wars against expertly chosen point costs.**

***Index Terms*—Wargames, Automatic Game Balancing, Game testing**

I. Introduction

In game design, the process of balancing a game can be described as modifying the rules that govern a game (also known as mechanics) to achieve a desired goal. Game designers see this as an important part of game development, even if there is no agreed upon definition of what game balancing is [1].

Additionally, this process is often long and tedious, done through estimating the mechanical values (e.g., how much a property is worth in *Monopoly* [2]) after an initial play-test, and then tuning these values through dozens of more tests. This manual process can lead to elements of the game being over- or under- powered. Perhaps as a result of these problems, Artificial Intelligence has been proposed to automated balancing, with some promising results [3]

Wargames are a game genre where players control armies (which can be asymmetric) to act out a battle or war, usually played on a detailed map under a set of rules to see if their strategies influence the game's outcome. [4]. Due to the inherent competitive nature of these games, good balance is considered an essential feature. There are several points of view on ideal balance goals for these games. One common goal is having a simple 50%-50% win-rate, where every army would have an equal chance of winning. Another viewpoint is *intransitive superiority*, a rock-paper-scissors style in which each army counters another (while itself being countered by a different army) [5].

A particularly popular sub-genre of wargames is that of miniature wargames. In these games, such as *Warhammer 40k*, [6] armies are comprised of miniatures which represent individual units on the battlefield such as infantry or tanks, with the battles being skirmishes between these units. In these games, a common way of achieving balance goals is through the use of point costs. Wargames often operate with an element of scarcity, with unit costs being one way to enforce this. For example, a common army budget in the aforementioned Warhammer 40k is one thousand points, and each unit included in the army (and the optional equipment added to them) has a point cost. For example adding a Space Marine 'hunter' tank to an army will cost 100 points. This ensures that armies fighting against each other will be of a similar strength and also adds an element of strategy to the army creation process, as players can optimise the point costs of their armies to create the strongest army at the budget level.

Unit costs are considered one of the hardest parts of balancing in wargames. As described in Tabletop Wargames: A Designers' and Writers' Handbook: "There are essentially three things to grasp about points values - (i) they don't work, (ii) nevertheless we have to have them and (iii) even so they can't really be reduced to a mathematical formula." [7].

Some wargames, such as *Song of Blade and Heroes* [7] [8], use a mathematical formula to calculate unit costs. However, this is often impractical for wargames with more complexity, due to their being too many mechanical values per unit to be accurately calculated. Instead, the previously mentioned manual playtests are used, however, these can cause overpowered units to appear which dominate other units.

One approach used in contemporary automated balancing is using an optimiser such as an Evolutionary Algorithm to find good sets of parameters for the mechanical values, using a fitness function based on desired balance goals [9]. One possible disadvantage with this kind of approach is that developers may feel a loss of control, and work has been done on creating an integrated approach of both automatic and manual balancing [10]. Point costs do not directly modify mechanical values, therefore it enables developers to decide how to modify the parameters to get desired point costs.

Our motivation for this paper involve how we can both estimate unit point costs for wargames, and the possibility of adapting them for general use. Using point costs generally could allow for easy patching of games after they are released (this is especially a problem for board games). In addition, it could be used to determine alternative rule-sets for games in scenarios such as tournaments, where highly balanced rules or corrections on these are often required.

In this paper we present a foundational approach on how artificial intelligence can be used to automatically estimate these point costs, without any domain knowledge required. Our approach uses linear regression in combination with linear programming to estimate the point costs of units in a wargame using data from played matches.

In or previous work we created a bespoke asymmetric wargame called Wizard Wars [11], which implemented in the Stratega framework [12]. The framework allows us to use advanced agents to play simulated games. In our case, MCTS-based agents were used to play Wizard Wars with custom heuristics to represent different play-styles, in addition to an opponent model to more accurately predict an opponent's move. The results of these Wizard Wars games were used by the method's to estimate unit costs.

Finally we created a metric to inform us of how balanced the game is. The Balance Loss (BL). Using the methods derived from our approach and data from the Wizard Wars simulations we were able to estimate point costs for units which when used, managed to reduce the Balance Loss (e.g., the imbalance of the game) by approximately half.

In this paper we extend our previous work by introducing a new framework called Simple Testing and Evaluation of Points (STEP). This framework is designed to reduce games to their most abstract elements for fast evaluation and controllable testing. We use this framework to do a further analysis of the methods used on Wizard Wars. We hope that this framework will allow for more vigorous testing of point cost estimating methods before using them on actual games. In addition to creating this framework, we conduct two more experiments using it, one checks how the performance of the baseline varies based on number of games in the training data. The other experiment shows the effect of stochastic elements on the costs estimated by the methods.

Section II introduces the background of this work, with regards to wargames and automated game balancing. Section III introduces our proposed methods. Section IV describes the Wizard Wars game and how we simulated it, while Section V introduces the STEP framework used for further analysis of the methods. Section VI goes into the results of the Wizard Wars experiments, and Section VII outlines the results of experiments using STEP. Finally, we conclude with a summary of our findings and possibilities for future work in Section VIII.

II. Background

This background section will give an overview of topics relevant to this work. First aspects of Wargames important to our research are outlined. Secondly examples of automated game balancing will be explored.

A. Wargames

As mentioned before, our research is focused on Miniature Wargames, recognized for their aforementioned use of miniature figures to depict individual units on the battlefield. These units are assigned point values that ideally reflect their effectiveness in the game. Units with greater strength wield more influence in battles and consequently possess higher point values. It is important to note that these point values are determined independently of other factors. While a unit's effectiveness may vary depending on the battle's circumstances (for example, a ranged unit with low health might become more formidable when paired with a support unit for reinforcement), its assigned point value remains constant.

As discussed in the Introduction, armies are allocated specific point budgets to ensure a relative balance in power between two opposing forces. Within the wargaming community, the assembly of units that compose an army is referred to as the "army list" [7]. If players wish to get an advantage over their opponent they can optimise their army list to include units more powerful then their point costs suggests.

B. Automated Game Balancing

Artificial intelligence has been applied to the problem of balancing games in academia, this can be called automated game balancing. Some examples of these work include: Mahlmann et al. using Evolutionary Algorithms (EAs) to generate balanced sets of cards in Dominion [9]. Morosan and Poli using EA's to balance both Mr's Pac-Man and StarCraft, by trying to achieve a desired win-rate while minimising mechanical changes made [13]. Finally Silva et al. used EAs to balance decks from Hearthstone [14]. They used similar balance goals as Morosan and Poli, in addition they introduced some metrics to see which cards may require modifications.

One potential problem with using Evolutionary Algorithms, is selecting which game parameters to balance. For example, when outlining their integrated balance framework, Beyer et al. identified which parameters to select based on expert knowledge from playing the game [10]. For Wargames, with lots of units and asymmetry, choosing which units and parameters to balance might be difficult, therefore an automated approach for identifying unit point costs may be advantageous.

Tomašev et al. explored alternative rulesets in chess [15]. As part of their work, they look at how the value of each chess piece changes based on which variant used. They do this by creating a win predictor using the difference in units of each player as weights. This is encouraging as it presents a way to estimate point costs, however chess is a symmetrical game. For wargames there is no guarantee that both players will have the exact same unit composition therefore a different approach will be required.

Stanescu et al. use the Lanchester Attrition Laws to predict combat outcome in Starcraft [16]. As part of their work they use Maximum Likelihood Estimation to estimate the strength of each unit, which shares some similarities with unit costs.

Hind and Harvey [17] used a NEAT neural network to calculate spawn probabilities for enemy units in Tower Defence games. This is an interesting approach, however these probabilities depend on which towers the player has got on the map, which may prove difficult for adapting to point-costs due to the previously mentioned fact that they have to be independent of any other factor.

III. Method

Point costs can be seen as a measure of how influential a unit is towards victory, with more influential units having higher point costs. Assuming this relationship is linear, we can formulate it as a linear regression problem for each army matchup in the game $a, b \in M$ as described in Equation 1:

$$\begin{gathered}\left(w_0^1 x_{i0}^1 + w_1^1 x_{i1}^1 + w_2^1 x_{i2}^1 + ... + w_n^1 x_{in}^1\right) - \\ \left(w_0^2 x_{i0}^2 + w_1^2 x_{i1}^2 + w_2^2 x_{i2}^2 + ... + w_m^2 x_{im}^2\right) = \hat{y_{in}}\end{gathered} \tag{1}$$

In this context, the weights w_l^j signify the point cost associated with unit l in army j. The inputs x denote the count of units of each type (up to n types for Army a and up to m types for Army b). Each battle yields a distinct data sample labeled as i. The sets $X = \{X_0, ..., X_{n-1}\}$ and $Y = \{y_0, .., y_{n-1}\}$ serve as our inputs and outcomes, respectively, collectively constituting our dataset $\mathcal{D}$. It's worth noting that game outcomes can take on values of 1 (win), 0 (loss), and 0.5 (draw). Wargames are usually strictly competitive, with games being either won, lost, or ending in a draw with no other values possible. This leads us to use these values, however it is important to emphasise that we are not binding to any particular linear equation.

We can estimate a unit's point cost as the average value of its weights when pitted against all possible armies in M, denoted as $\mathbb{E}_M[w]$. To determine this value, we compute the average point cost per unit across all the regressions conducted for each matchup. It's important to note that even though the elements in the set Y are strictly 0, 0.5, or 1 (representing loss, draw, or win), we opt for different forms of linear regression rather than logistic regression. This choice is due to the logistic regression coefficients being somewhat more challenging to interpret, given their log-odds nature. It is arguable that logistic regression might prove more accurate/correct for the problem we are setting to solve, but we use linear regression because of ease of interpretation. We will now go over the different implementations of linear regression our method can use.

1) Least squares / bounded optimisation: Our initial approach, which we refer to as the *bounded least squares* method, involves the analysis of all games played between two armies. We optimize the weights in a straightforward linear optimization framework, making use of the SciPy library (https://scipy.org/). In this method, the loss function adheres to the conventional mean squared error, expressed as $mse(X, Y) = \sum_{i=1}^{|X|}(\hat{y}_i - y_i)^2$. The optimization process follows the procedure outlined in Equation 2. Notably, we impose constraints on the optimization bounds to ensure that no unit from the first army can have a negative point cost, and similarly, no unit in the opposing army should be assigned a positive point cost. This constraint essentially restricts the solution to yield exclusively positive unit costs for each army.

$$\begin{gathered}minimise\ mse(X, Y) \\ subject\ to\ w^1 \geq 0, w^2 \leq 0,\end{gathered} \tag{2}$$

In Equation 2, w^1 and w^2 refer to the weights of the first / second army respectively.

2) Elastic net CV: In our second approach, we tackle a constrained linear optimization problem, this time without imposing constraints on the weights w. Instead, we apply penalties to discourage high L1 and L2 norms. This approach is commonly referred to as an *elastic net*, and its goal is to attain sparsity in the weight vector w while simultaneously keeping the values of w relatively low. We determine the optimal values for α and $l1_{ratio}$ in Equation 3 through a process of cross-validation, using the implementation provided by scikit-learn [18].

$$\begin{aligned}minimise\ 1/(2 \cdot n_{samples}) \cdot mse(X, Y) \\ +\alpha \cdot l1_{ratio} \cdot ||w||_1 \\ +0.5 \cdot \alpha \cdot (1 - l1_{ratio}) \cdot ||w||_2^2\end{aligned} \tag{3}$$

Similar to the "bounded least squares" method, there's a potential issue where some units may acquire negative weights, denoted as w. To address this, we employ two methods: either we artificially establish a minimum value of 0 for each w within every army list combination ("bounded elastic net"), or we implement this constraint after averaging the values across all armies ("elastic net").

3) Normal form game / linear programming: The problem, as delineated previously, assumes that both players will essentially select their army lists randomly. However, this is likely an oversimplification. In reality, both players will make an effort to choose an army list that maximizes their chances of winning while causing their opponent to lose. Consequently, the entire process of selecting army lists can be modeled as a zero-sum, normal form, asymmetric game. In this normal form game, every conceivable army list is considered an action or strategy, with each army corresponding to one of the players.

The formal game is subsequently resolved using a standard linear programming solver. For this paper we used OpenSpiel's [19] implementation. This results in a mixed strategy profile for each player. Considering that not all potential games have necessarily been played, particularly for the more complex wargames, it becomes computationally prohibitive to evaluate all conceivable army lists. To address this, we pruned out any army list pairs which had not been played against.

Subsequently, we selected, from the mixed policy, all the army lists that were not dominated, meaning they performed as well as or better than another army list. These selected army lists were used to generate hypothetical games based on the army lists that players might have employed. It is important to note that since not all possible army list combinations may have been employed in the actual data, our approach should be regarded as an approximation. We refer to the army lists resulting from this process as *equilibrated.*

4) Final Methods: The weights w produced were then used as the point values for the units. Six different methods are implemented, *least squares* (LS), *elastic net* (E), and *bounded elastic net* (BE) are the three basic methods. We also used *equilibrated* data on these three methods to create new methods (ELS, EE and EBE, respectively).

IV. Wizard Wars Experimental Setup

Our first approach to testing our methods was to run them on an actual game. In our case we created a bespoke wargame called Wizard Wars which is implemented using the Stratega framework. This section will go into how Wizard Wars works, and then how experiments using it were set up.

A. Stratega

Stratega[1] is a framework designed for the creation of both turn-based and real-time strategy games. Games can be created in the framework using the YAML format. Various elements of strategy games are modeled, including: intricate rules, diverse victory conditions, terrain, unit customisation, complex actions, technology progression, and build orders [12]. It contains an integrated forward model enabling advancement of the game state by providing actions during the decision-making process of in-games agents. This allows for the implementation of algorithms such as Monte Carlo Tree Search and Rolling Horizon Evolution [20].

B. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [21] is a highly selective and efficient best-first search algorithm. It constructs an asymmetric search tree over multiple iterations, carefully balancing the exploration and exploitation of moves. This equilibrium is achieved through its tree policy, such as UCB1 [22], which weighs both the exploitation term $Q(s, a)$ (the average of rewards after taking action a in state s) and the exploration term $C \times \sqrt{\frac{\ln N}{N(s,a)}}$, where $N(s)$ is the number of times state s has been visited and $N(s, a)$ is the number of times action a has been applied to state s. The parameter C, known as the exploration constant, determines the emphasis placed on the exploration component. In the case of Wizard Wars, each action will be an action a unit can do (such as moving, or attacking). The state will contain the position and status of each unit on the battlefield, and which player is next to act.

In its default configuration, the MCTS algorithm extends the tree with a new node during each iteration. Subsequently, it conducts a random Monte Carlo simulation until the conclusion of the game or when a predetermined depth is reached. MCTS has found applications in various domains [21] [23], including strategy games [24], demonstrating its adaptability in complex and dynamic environments by rapidly re-planning and adapting to unforeseen states.

C. Wizard Wars

Wizard Wars is a simple wargame built in Stratega to test the methods proposed. MCTS agents act as players. It is a turn-based game with no stochasticity. Units use actions to affect the game. All units can move and attack, some units have additional actions. Units are allocated action points, each non-movement action uses one point, and these points replenish at the start of each turn.

[1]https://github.com/GAIGResearch/Stratega

Fig. 1. Wizard Wars being simulated in Stratega.

TABLE I
An enumeration of the attributes of each army's units.

Race and Attributes	Unit Types			
Dwarf	**Melee**	**Ranged**	**Support**	**Elite**
Health	4	1	4	10
Action Points	2	1	2	3
Movement Points	2	1	2	2
Attack Damage	3	5	2	4
Attack Range	1	4	1	1
Armour	2	0	2	4
Abilities			Rf	Rf
Elf	**Melee**	**Ranged**	**Support**	**Elite**
Health	4	4	3	7
Action Points	3	2	2	3
Movement Points	3	5	2	4
Attack Damage	3	3	2	6
Attack Range	2	2	2	1
Armour	1	0	0	2
Abilities	AP		Hl	AP
Orc	**Melee**	**Ranged**	**Support**	**Elite**
Health	6	4	3	8
Action Points	2	2	2	3
Movement Points	3	2	2	3
Attack Damage	4	3	2	5
Attack Range	1	2	2	1
Armour	0	0	0	2
Abilities	WC	Da	Cs	WC

Each team has one king unit, a player loses if their king dies. Figure 1 shows a game of Wizard Wars in play. The game has a grid layout, with no unit stacking possible. The move actions changes a units position on the grid. They can travel one tile at a time non-diagonally, going a total distance equal to their movement points, which are refreshed each turn.

To use the attack action, a unit chooses an opposing target to hit. The damage inflicted is determined by the formula D - A, where D is the unit's attack damage, and A is the targets armour. The resulting value, if positive, is deducted from the target's health. If the target's health reaches < 0 it is removed from play. There are three distinct armies in the game: Dwarves, Elves, and Orcs. Each army has four distinct unit type:

- *Melee units*: Units at the front of the battle.
- *Ranged units*: Units that attack from a safe distance.
- *Support units*: Units that support other units with abilities
- *Elite units*: The most powerful units in each army

Table I, shows the attributes and abilities of each army's units. Each army is designed to have a unique playstyle: Dwarves prioritise defence, having slower units but more

TABLE II
THE WEIGHTING OF EACH FACTOR PER HEURISTIC (FROM 0-1)

Factor	Defensive	Balanced	Aggressive
Player Army Size	1	0.5	0.25
Player Army HP	1	0.5	0.25
Player King HP	1	0.5	0.25
Opponent Army Size	0.25	0.5	1
Opponent Army Health	0.25	0.5	1
Opponent Kings HP	0.25	0.5	1
Units in Range of Support	0.5	0.5	0.5
Mean Distance from Opp King	0.25	0.5	1

health and armour. Elves feature agile units designed for maneuverability in battle. Orcs have high attack damage and offensive abilities to attack the opponent directly. These values were manually selected / tweaked with no formal playtesting conducted. The abilities in the game are outlined below, most abilities have a cooldown, which is the number of turns after using an ability before it can be used again.

- Rf - Reinforce an allied unit increasing their armour by 2 (3 tile range) (3 turn cooldown).
- AP - Unit's attacks ignore armour (passive ability)
- Hl - Heals an allied unit, increasing their health by 3 (3 tile range) (3 turn cooldown).
- WC - Gives the caster +1 movement and +1 damage for 2 turns (3 turn cooldown).
- DA - Reduces the attack damage and range of an enemy unit by -1 for 2 turns. Units with no attack range are unable to attack (2 tile range) (3 turn cooldown).
- Cs - Curses an enemy making them lose 1 health per turn for 4 turns (3 tile range) (3 turn cooldown).

These armies make the game asymmetric, as different armies facing each other will have unique units, unlike in chess where both players have exactly the same units. This could lead to diverse play styles, with no clear way of identifying dominant strategies. This suggests suitability for our methods.

D. Heuristics

Our methods rely on data derived from Wizard Wars simulations, therefore it is important that the dataset includes a variety of playstyles. Otherwise a single playstyle may be overemphasised. To mitigate this, we incorporated three heuristics to represent different playstyles. These were Agressive, Balanced, and Defensive heuristics. The outcomes of these heuristics are normalised to a range of 0 - 1, with increasing value meaning a more favourable state for the agent.

The factors included in the heuristic evaluation were:

- The current size of the player / opponent army.
- The current HP of the player's / opponent's army.
- The current HP of the player / opponent's king.
- How many allied units are covered by a support.
- The mean distance of the player's offensive units from the opponent's king.

Table II shows the weightings assigned to each factor for every heuristic. These factors are all normalized to fall within the range of $[0, 1]$ and are then combined linearly by multiplying them with their respective weights to calculate the reward assigned to the MCTS agent.

In addition to the heuristics, the MCTS agents employed an opponent model designed to anticipate the actions of the opposing player. This model was provided with Stratega. It operates under the assumption that the opponent's strategy involves targeting the player's unit with the lowest health. Accordingly, the opponent's units move towards the player's low-health unit and attack if within range.

E. Experimental Configuration

For each Wizard Wars experiment, two datasets were created: One for training the methods, the other to validate them. Every match took place on a grid of 5 x 13 squares. There was no special terrain, and the initial positions were reasonably randomised, with melee units place in the front and ranged towards the rear. The kings were placed behind each player's army. For training, army lists were generated randomly, bounded between 0 and 3 units per type.

For the validation run, the point costs estimated by each methods were used to generate army lists. Each army had a budget of 60 (due to configuration of the methods, this budget was scaled down by a factor of 100, to 0.6), and units point costs were rounded to 2 decimal places. Each army list could have a maximum of 5 units per type. To generate the list, sampling with replacement was used. Any unit with a cost of 0 or less was considered ineffective and not included for selection. The generator tried to use its entire budget, if infeasible as much as possible was used instead.

Stratega's MCTS agent was used for both players, using the aforementioned heuristics and opponent model. A time budget of 300ms per move was used, a rollout length of 20, and the exploration constant set at $\sqrt{2}$. These values were chosen through trial-and-error. Each turn had a limit of 10 seconds, and a turn limit of 50, is this turn limit is reached the game will end as a draw.

We introduced a metric to measure the balance of the game, we termed it the "Balance Loss" (BL). This metric is standard deviation of win rates of each matchup. A value of 0 represents a perfectly balanced game (according to our goal), with each army having a win-rate of 50% overall. If win-rates deviate from 50%, this value will increase, indicating greater imbalance in the game. While we are using a 50-50 balance goal currently, we aim to try other targets in future work.

1) Training Run Configuration: To create the training dataset for our methods, we generated a total of 1,200 army list pairs (2,400 in cases of mirror matchups, like Orc vs. Orc) , generated randomly with constraints defined previously, for each possible battle combination (including Dwarf vs. Elf, Elf vs. Dwarf, Orc vs. Dwarf, Dwarf vs. Orc, Orc vs. Elf, Elf vs. Orc, Dwarf vs. Dwarf, Elf vs. Elf, Orc vs. Orc). This process yielded a total of 14,400 army lists for each run. For each of these list pairs, games were played involving all possible heuristic combinations (for example, Aggressive vs. Balanced). Consequently, this resulted in 172,800 games for each run. Each run required approximately 24 hours of time on a High-Performance Computer.

2) Validation Run Configuration: In the validation run, the total number of games played remained constant. However, this time, the games were divided based on which method's point costs were utilized to generate the army list pairs. This division resulted in 1200 games being played per matchup (200 pairs for each combination of armies, heuristics, and methods).

V. STEP Framework

Testing automatic balancing solutions on games is important to do as it provides a way to test the real world effectiveness of methods. However, because of the complex nature of games, testing on them can give uncertainty to the results. A good instance of this is the stochastic nature of many games, which can add noise to the results. Another is due to the fact that games require players, and uncertainty can be generated from their game play (whether agent-based or human.)

To account for this, in addition to testing our method on a wargame, we also introduce the STEP (*Simple Testing and Evaluation of Points*) framework. Our objective for STEP is to provide a fast way to evaluate balancing solutions with minimal uncertainty, by breaking down a game to its most abstract elements.

The structure of STEP is similar to a Blackbox, which have been used in many fields to model complex problems in a way for methods to try to solve them [25]. Our framework is most similar to stochastic Blackboxes due to the element of random chance in many games.

This abstraction involves modelling core concepts in games such as stochasticity, and quantifying winning (i.e., can players draw) into easily controllable variables. Due to the influence of wargames in our work, some concepts in STEP are named using wargaming terminology, however, these can be used for a variety of games. As an example, for cards games, the army list could be turned into decks, and point costs could be the cost to play a card (e.g. Mana in *Magic The Gathering* [26]).

Central to STEP are army lists. Similar to wargames, these are a list of which units that are brought into the game. In our case there are two army lists, one for each player. In addition to these, a ruleset is required that contains the true strength of each unit. In an ideal world, the point costs estimated will be the same as these true strengths, as the point costs represent the influence of a unit on victory, which thanks to the abstract nature of STEP can be easily measured. It is important to note that STEP can be used to model any game, not just Wizard Wars.

A. Blackbox

These Blackboxes in STEP can be defined as a game with an outcome that can be formulated as a function:

$$w = BB(U_1, U_2, S, f, d, x)$$

From this function the outcome of the game, $w \in (0, 0.5, 1)$ where w is 1 if player 1 won the game, 0 if player 2 won the game, and 0.5 being a draw. The winner depends on the input parameters described below, which model aspects talked about in the previous paragraphs.

1) Units (U_1, U_2): A map containing the army lists for each player. Each key will be the name of a unit, and each value the number of that unit in the army list.

2) Ruleset (S): A map containing the true strengths of each unit. Each key will be the name of a unit, and each value the strength of the unit (as a number).

3) Outcome Function (f): The function chosen to evaluate which team wins the game (e.g., Linear Outcome). The value returned by this function will be the outcome (o) of the game, which will be positive if player 1 won, negative if player 2 won, and 0 if a draw.

4) Draw Range (d): A value d that sets the range to be considered a draw. Having an exact draw is highly unlikely due to the abstractions made in STEP, therefore including some bounds for error, ensures draws can be achieved. Draws are determined when o is between the upper (UB) and lower (LB) bounds. These bounds are be calculated by $0 \pm d$ and will then be used to evaluate $LB \leq o \leq UB$, which if true, the game will be a draw ($w = 0.5$).

5) Stochastic Strength (x): A value x used to add stochasticity to the game, if desired. If $x > 0$ the outcome of the game (o) will be modified by a value from -x to x, sampled uniformly at random.

B. Outcome Functions

As mentioned previously, these Blackboxes use *Outcome Functions* to determine the outcome of the game. These are essentially mathematical functions, such as a linear combination, which compare the strength of each player's army to artificially determine which player wins.

As an example, if we were to use the linear outcome function, the strengths of the armies would be computed by summing up each units strength with how many are in the army list, to then observing which one has a higher value.

To enumerate this example, imagine a ruleset S in which the red player has units *Red 1* and *Red 2* with strengths 2 and 3 respectively. The blue player has only unit *Blue 1* with a strength of 5 . In an example match, the red player has two Red 1 units and one Red 2 unit, while the blue player has three Blue 1 units. These units would be multiplied by their strengths in the ruleset to find the army strengths for each player. In this case the strength of the red (rs) and blue (bs) players will be $rs = (2 * 2) + (1 * 3) = 7$ and $s = (3 * 5) = 15$. These values would then be normalised, so that: $rs = \frac{rs}{(rs + bs)}, bs = \frac{bs}{(rs + bs)}$ Finally the outcome of the linear function can be calculated by taking player 2's army strength away from player 1's army strength. In the case of this example $o = rs - bs = -0.36$, so player 2 won.

C. Winner Calculation

Once the outcome function has returned a value the winner of the game can be calculated. The stochasticity value will be applied if it greater then 0. By default it has a value of 0.1, which means o can be modified at most by ± 0.1.

Now the draw range is used to check if there is a draw. By default it is 0.05 so in our example even if o is modified by 0.1 by the stochasticity value, it will still not be within the draw range of 0 ± 0.05. Therefore since this is a negative value the Blackbox will return a value of 0 as player 2 won.

D. Testing Methods

To test methods games can be played using the Blackbox. In order to generate army lists for these games generators are provided. These create army lists using approaches such as sampling with replacement (used with a point cost budget).

Once these generators have been chosen a specified number of games will be played for each matchup. Once these games have been concluded, various statistics will be saved, for example: the army list pair per game, the winner per game, win rates, and Balance Loss.

E. STEP Experimental Setup

As mentioned in the framework section, we are using the Blackboxes in the STEP framework to further analyse the methods proposed in a more controllable setting then wizard wars. Our first experiment was to see the consistency of the proposed methods, and how many games are required to estimate sufficient point costs. This was possible due to the fast computation time of a Blackbox run in the framework, and the fact that the ground truth is known (the true unit strengths).

1) Variance Experiment: This experiment varied the number of games played per run. The initial number of games was 100, this increased by 1000 until reaching 10100 games per run. 20 runs were done per number of games to measure variance in results. The purpose was to see how increasing the number of games played effects results, and to see if the results vary on a per-run basis. The Blackbox used the default stochastic strength of 0.1 and draw range of 0.05.

A simple rule set of 2 teams with 4 units each was used, this rule set was designed to be deliberately unbalanced. The previously mentioned linear outcome function was used. And similarly to Wizard Wars, a random unit generator with a lower bound of 1, and an upper bound of 3 was used to generate teams for the training set. A generator utilising sampling with replacement was used to generate teams for the validation set.

Compared to Wizard Wars the lower bound was changed from 0 to 1, this was due to testing showing better results with this configuration in the Blackbox. In addition a budget of 100 was used. It is important to note that due to way the methods work, this budget is divided by a factor of 100, so it is the same scale as the points estimated by linear regression. Unit point costs were not rounded for the STEP experiments. (Unit costs were rounded in Wizard Wars for neat presentation in Table III and Table VI.)

2) Stochastic Experiment: As mentioned before, the Blackbox has a "Stochastic Strength" option. Increasing this value will escalate the effect of random chance on the game. This experiment was used to see how stochasticty may effect the performance of the methods. For this experiment the stochastic strength was increased from 0 to 10, with an increment of 0.5. 5 runs were performed per level of stochastic strength. There were 1000 games per run. The rest of the configuration was the same as the variance experiment.

TABLE III
THE POINT COSTS OF EACH UNIT FOR THE THREE ARMIES AS ESTIMATED BY THE SIX LINEAR REGRESSION METHODS (EQUIL. = EQUILIBRATED).

Race / Method	**Unit Types**			
Dwarf	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	7	12	5	13
Elastic-Net (E)	6	12	4	13
Bounded Elastic-Net (BE)	6	12	4	13
Equil. Least Squares (ELS)	2	7	3	10
Equil. Elastic-Net (EE)	2	5	3	7
Equil. Bounded Elastic-Net (EBE)	2	5	3	7
Elf	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	10	10	6	17
Elastic-Net (E)	10	9	6	17
Bounded Elastic-Net (BE)	10	9	6	17
Equil. Least Squares (ELS)	5	4	4	11
Equil. Elastic-Net (EE)	5	4	3	9
Equil. Bounded Elastic-Net (EBE)	5	4	3	9
Orc	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	10	12	6	16
Elastic-Net (E)	9	12	6	16
Bounded Elastic-Net (BE)	9	12	6	16
Equil. Least Squares (ELS)	4	6	6	11
Equil. Elastic-Net (EE)	3	5	4	9
Equil. Bounded Elastic-Net (EBE)	3	5	4	9

VI. WIZARD WARS RESULTS

A. Estimated Point Costs

Table III displays the point costs estimated by each method, revealing some patterns. The point costs make intuitive sense. Elite units have the highest costs, and are the most powerful. The melee and ranged units follow the elites, with the stronger unit differing between armies. Support units have the lowest estimated costs, this could suggest that their in-game abilities are not as important as the raw stats of other units.

Three clusters of estimated point costs emerge. The non-equilibrated methods have estimated similar values. This could be because of the large amount of training data negating the different approaches used. Additionally it could be an upper bound for more complex methods to supersede.

The elastic-net method pairs estimated the same point costs. This suggests that none of the units had an estimation less than zero, meaning all units were found to be somewhat useful. The equilibrated least-squares and elastic-net methods estimated different costs, possibly because of the equilibration process.

For the Dwarven army there is only a 1 point difference between its elite and ranged unit. The other armies have a larger difference between elite / non-elite units. This could imply that the Dwarven elite unit is less powerful then the others. Looking at the unit attributes we can see it has an attack damage of 4, which is less then the other elites and the Dwarven ranged unit. This could mean that attack damage is an important attribute. Additionally, excluding ELS, the Dwarves have lower point costs compared to the other armies for every unit except the ranged. This may suggest that per unit they are less powerful compared to other armies. The Elves and Orcs have quite similar estimated costs, differing at most by a few points. We can infer this to mean that the methods perceive them as close in power.

TABLE IV
THE FIRST ARMY'S WIN-RATE AND STANDARD ERROR FOR EACH METHOD AND THE BALANCE LOSS (BL).
E VS D - ELF VS DWARF E VS O - ELF VS ORC O VS D - ORC VS ELF

Method	EvD	EvO	OvD	BL
Uncosted	53% (0.003)	39% (0.003)	59% (0.003)	9.03
LS	46% (0.006)	35% (0.006)	54% (0.006)	10.13
E	44% (0.006)	34% (0.006)	52% (0.006)	10.58
BE	43% (0.006)	35% (0.006)	52% (0.006)	10.18
ELS	48% (0.006)	48% (0.006)	43% (0.005)	4.86
EE	40% (0.005)	42% (0.006)	41% (0.005)	9.65
EBE	40% (0.005)	41% (0.006)	42% (0.005)	9.84

TABLE V
THE WIN-RATE'S AND STANDARD ERROR VS. THE SAME FACTION

Method	DvD	EvE	OvO
Uncosted	61% (0.002)	56% (0.003)	55% (0.003)
LS	62% (0.006)	60% (0.006)	54% (0.007)
E	62% (0.005)	61% (0.006)	54% (0.007)
BE	63% (0.005)	62% (0.006)	54% (0.007)
ELS	58% (0.005)	60% (0.006)	58% (0.006)
EE	55% (0.006)	57% (0.006)	57% (0.005)
EBE	54% (0.006)	59% (0.006)	57% (0.005)

B. Win-Rates

Table IV highlights that the balance loss of the uncosted simulations (i.e., training run) stands at 9.03, serving as a baseline for result comparison. This is a relatively low balance loss, which suggests the game is reasonably balanced.

Equilibrated Least Squares manages to achieve a more balanced game than the baseline. It reduces the balance loss by almost half, to 4.86. The Elf vs. Dwarf and Elf vs. Orc matchups are close to the goal win-rate of 50%, the Orc vs Dwarf is less balanced at 43%, yet it is still an improvement compared to the baseline matchup.

The non-equilibrated methods come close to the baseline without surpassing it. In these methods the Elves underperform compared to the other armies. These methods estimated similar points for the Elves and Orcs, implying that they overestimated the strength of the Elven units. This is further supported by them losing the Elf vs Dwarf matchup, one in which they previously performed well. However, the Orc vs. Dwarf matchup is more balanced compared to the baseline, suggesting the estimated points for these armies are more accurate.

The Equilibrated Elastic-Net pair get a slightly better balance loss compared to the basic methods, but still higher than the baseline. For these methods the Dwarves now win both matchups they previously lost. This suggests that they underestimated the point costs of Dwarven units. The Elf vs Orc matchup is slightly better but still not at the goal of 50%.

Additionally, we also ran mirror matches against the same army (with varying choices of units). Table V outlines these win-rates. In every case, Player 1 outperformed Player 2, indicating a probable first-turn bias in the game. None of the methods managed to mitigate this advantage. Implying that point-costs are not suitable to fix this, and that modifications to the game rules may be needed.

TABLE VI
THE ESTIMATED POINT COSTS FOR THE BIASED EXPERIMENTS.

Race / Method	Unit Types			
Dwarf	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	22	9	4	11
Elastic-Net (E)	21	8	3	10
Bounded Elastic-Net (BE)	21	8	3	10
Equil. Least Squares (ELS)	18	8	5	8
Equil. Elastic-Net (EE)	15	6	4	7
Equil. Bounded Elastic-Net (EBE)	15	6	4	7
Elf	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	10	4	5	16
Elastic-Net (E)	9	4	5	15
Bounded Elastic-Net (BE)	9	4	5	15
Equil. Least Squares (ELS)	6	3	4	8
Equil. Elastic-Net (EE)	5	3	3	8
Equil. Bounded Elastic-Net (EBE)	5	3	3	8
Orc	**Melee**	**Ranged**	**Support**	**Elite**
Least Squares (LS)	8	11	6	14
Elastic-Net (E)	8	10	5	14
Bounded Elastic-Net (BE)	8	10	5	14
Equil. Least Squares (ELS)	6	8	3	9
Equil. Elastic-Net (EE)	4	6	2	7
Equil. Bounded Elastic-Net (EBE)	4	6	2	7

TABLE VII
THE WIN-RATES AND STANDARD ERRORS FOR THE BIASED EXPERIMENTS.

Method	EvD	EvO	OvD	BL
Uncosted	22% (0.008)	34% (0.008)	28% (0.008)	24.55
LS	21% (0.016)	34% (0.019)	30% (0.02)	24.51
E	18% (0.016)	30% (0.019)	24% (0.018)	28.92
BE	19% (0.017)	32% (0.02)	23% (0.018)	28.42
ELS	26% (0.018)	43% (0.02)	32% (0.02)	19.74
EE	21% (0.018)	33% (0.018)	31% (0.02)	24.28
EBE	22% (0.018)	35% (0.018)	32% (0.02)	23.17

C. Biased Units Experiment

We also ran an experiment to test if our methods can detect unbalanced units, where the Dwarven Melee unit was made overpowered, and the Elven Ranged unit was made underpowered. In this experiment we exclusively used the Balanced Heuristic, and 600 training and 100 validation army list pairs per matchup, resulting in 7200 pairs per run.

Table VI displays the estimated point costs per method. The Dwarven Melee unit has a much higher cost then before, even surpassing the Elite units. This suggests that the method's identified it as being the most powerful unit in the game. The Elven Ranged unit also has a lower point costs, being similar to the support units. However, it is objectively less powerful then them (while still being playable), suggesting a floor to how low point costs will be estimated with these methods.

Table VII shows the win-rates with these unbalanced units. Again, ELS is the most effective, reducing the balance loss to 19.74, and improving the balance of each matchup. The game is still unbalanced however, suggesting that while intentionally unbalanced units can be identified, changes to the rules of the game are required to properly balance them.

TABLE VIII
THE POINT COSTS CHOSEN BY EACH EXPERT.

Race / Expert	**Unit Types**			
Dwarf	**Melee**	**Ranged**	**Support**	**Elite**
Expert 1	6	10	8	14
Expert 2	3	10	5	15
Expert 3	6	6	6	14
Elf	**Melee**	**Ranged**	**Support**	**Elite**
Expert 1	10	8	8	15
Expert 2	7	4	8	12
Expert 3	6	10	8	14
Orc	**Melee**	**Ranged**	**Support**	**Elite**
Expert 1	10	8	4	14
Expert 2	5	5	5	10
Expert 3	10	8	8	12

TABLE IX
THE FIRST ARMY'S WIN-RATE AND STANDARD ERROR FOR EACH EXPERT'S COSTS, AND THE BALANCE LOSS (BL).

Method	EvD	EvO	OvD	BL
Uncosted	53% (0.003)	39% (0.003)	59% (0.003)	9.03
Expert 1	48.08% (0.002)	27.19% (0.003)	62.17% (0.003)	16.40
Expert 2	54.13% (0.002)	30.09% (0.003)	64% (0.002)	15.62
Expert 3	43.25% (0.003)	36.38% (0.003)	50.16% (0.003)	9.61

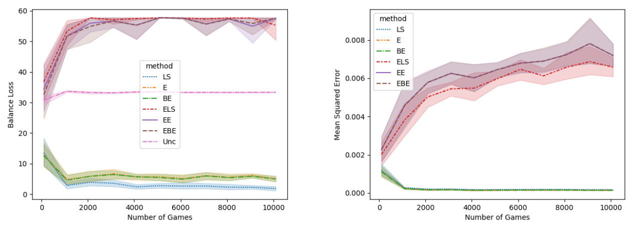

Fig. 2. Graph's showing the change of Balance Loss and Point Costs MSE over number of games played. The hue represents the 95% confidence interval.

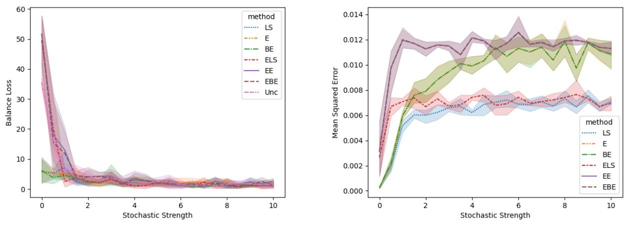

Fig. 3. Graph's showing the change of Balance Loss and Point Costs MSE over the level of stochasticity. The hue represents the 95% confidence interval.

D. Comparison with Game Designer Point Costs

One motivation of our work was to see it applied into the game design process. To this end it is important to see how our methods would compare to human balancing. We asked three experts (self-identified as being amateur or professional game-designers), to come up with point costs for Wizard Wars.

They were given a document detailing how the game works, each unit in the game, and how the maps have been setup for our experiments. They were then asked to choose a point value for each unit. Table VIII shows us the point costs chosen by each expert. It is interesting to see that for most units there is a variation in the costs chosen.

Games were then played with army lists generated from these point costs. In total 100,000 games were with each set of expertly chosen point costs, divided equally between matchup and heuristic. Table IX displays the win-rates achieved with each set of point costs. We can see none of the experts managed to beat the baseline (getting a lower balance loss than the uncosted run), and 2 experts got a higher balance loss then any of our methods.

VII. STEP Experimental results

A. Variance Experiment

Figure 2 shows us the results of this experiment. The first graph shows the performance of each method as more games are played. We can see that the uncosted games have a balance loss of around 30, showing us this game is unbalanced.

The non-equilibrated methods quickly manage to reduce the balance loss to the single digits, with the results plateauing by 2100 games. In addition we can also see how the Mean Square Error (MSE) of the estimated point costs change. This represents how close the estimated costs are to the true strengths of each unit. It is noted here that these values are normalised before comparison to ensure correct scaling. For the non-equilibrated methods the error decreases as the number of games played increases. This suggests that the points estimated are close to the true strengths, which makes sense considering the low balance loss. Conversely, the equilibrated methods do not perform well in this scenario, with the balance loss actually increasing as the number of games played increases.

This is surprising, as for Wizard Wars the equilibrated least squares method performs better than all other methods. We outline two possible reasons for this. Firstly, looking at the win-rates themselves, in the uncosted games blue is heavily favoured. However, the equilibrated methods flip this, making red win almost all of the time. So perhaps the equilibrated methods have over corrected the point costs and given red the advantage instead. Secondly, the equilibration process was created to account for the fact that there may be lots of nonsensical unit compositions. It may be possible that due to the simplicity of the linear outcome function these interactions may not be modelled, damaging the equilibration process.

B. Stochastic Experiment

We can see the outcomes of the experiment from Figure 3. The first graph shows the change of balance loss with stochasticity. As to be expected the balance loss quickly decreases for all methods with increasing stochasticity, this makes sense as the game essentially becomes random chance. The second plot supports this conclusion, as the MSE increases with stochasticity, likely due to the fact that there is too much noise to allow learning sensible point costs.

VIII. Conclusion

It is our understanding that before our research, there was no automated method for estimating unit costs in intricate wargames. We introduce a game-agnostic algorithm designed to automatically calculate unit costs using only data derived from game results. We have tested the methods both using a framework and an actual game. The non equilibrated methods performed well in both sets of experiments. In the STEP framework they achieved a low balance loss, and got point costs close to the ground truth. In Wizard Wars they did not manage to improve on the baseline but came close to it, and the costs themselves made intuitive sense.

The equilibrated methods are more inconclusive. In Wizard Wars, Equilibrated Least Squares achieved the lowest balance loss, reducing it by almost half compared to not using costs. In addition it successfully identified the unbalanced unit in the biased units experiment. However, in the STEP frameworks they did not fare well. We outlined that they may perform better in complex environments.

We have identified two pathways to enhance our approach for estimating unit point costs. For our current methods, we assume a linear relationship between win-rate and units, however, exploring the impact of nonlinear regression—such as non-linear least squares or other approaches like genetic programming could be insightful. This could be tested by making a non-linear outcome function in the STEP framework. Secondly, we could expand the scope of the game data we model. Currently, the algorithm exclusively estimates independent unit costs. However, as previously mentioned, numerous factors influence a unit's effectiveness in battle. Exploring how incorporating these factors (e.g., recognizing that a support unit works well when paired with a powerful unit) may contribute to estimating more accurate point costs.

Another aspect of improvement is the generation of army lists. Both methods used in this paper were essentially random. It may be worth creating intelligent army list generators which can identify and exploit under costed units, to further test our algorithms. Finally, another interesting idea is the altering the balancing objective. In this study, we aimed to balance for a 50% win-rate for each army. However, it would be worth exploring an intransitive superiority approach, where the strategic selection of an army based on various game factors becomes crucial for achieving victory.

We have identified potential applications for this algorithm in the future. One avenue is incorporating complex terrain into Wizard Wars as it would add another layer of complexity which could affect a units performance. Another is adapting our methods for use in other game genres. While Wizard Wars is a wargame, assessing the algorithm's performance in different game types, such as Trading Card Games, could be valuable. Additionally, the estimated points generated by our algorithm can serve as a surrogate for a unit's power level. Consequently, it may be feasible to employ it as a diagnostic tool to identify unbalanced units, and then feed this information to an automatic game balancing algorithm, like N-Tuple Bandit Evolutionary Algorithm [27], to allow balancing with no human input.

IX. Acknowledgments

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising. This work was supported by the EPSRC IGGI CDT (EP/S022325/1) and utilised QMUL's Apocrita HPC facility. We used ChatGPT [28] for assisting in rephrasing each section and titling the paper, and for some code generation in the STEP framework.

References

[1] A. Becker and D. Görlich, “What is game balancing?-an examination of concepts,” *ParadigmPlus*, vol. 1, no. 1, pp. 22–41, 2020.
[2] *Monopoly*. Parker Brothers, 1935.
[3] V. Volz, G. Rudolph, and B. Naujoks, “Demonstrating the feasibility of automatic game balancing,” in *GECCO*, 2016.
[4] J. F. Dunnigan, *The Complete Wargames Handbook*. 1997.
[5] N. Beume, T. Hein, B. Naujoks, N. Piatkowski, M. Preuss, and S. Wessing, “Intelligent anti-grouping in real-time strategy games,” in *IEEE Symposium On Computational Intelligence and Games*, pp. 63–70, 2008.
[6] *Core Book*. Warhammer 40,000, Games Workshop, Limited, 2020.
[7] R. Priestley and J. Lambshead, *Tabletop Wargames: A Designers' and Writers' Handbook*. Pen & Sword Books Limited, 2016.
[8] A. Sfiligoi, J. Hartman, J. Hartman, and J. McBride, *Song of Blades and Heroes - Revised Edition*. CreateSpace Ind. Publishing Platform, 2012.
[9] T. Mahlmann, J. Togelius, and G. N. Yannakakis, “Evolving card sets towards balancing dominion,” in *IEEE CEC*, pp. 1–8, 2012.
[10] M. Beyer, A. Agureikin, A. Anokhin, C. Laenger, F. Nolte, J. Winterberg, M. Renka, M. Rieger, N. Pflanzl, M. Preuss, and V. Volz, “An integrated process for game balancing,” in *IEEE CIG*, 2016.
[11] G. E. Long, D.-P. Liebana, and S. Samothrakis, “Balancing wargames through predicting unit point costs,” in *IEEE Conf. on Games*, 2023.
[12] A. Dockhorn, J. H. Grueso, D. Jeurissen, and D. P. Liebana, “Stratega: A general strategy games framework.,” in *AIIDE Workshops*, 2020.
[13] M. Moroşan and R. Poli, “Automated game balancing in ms pacman and starcraft using evolutionary algorithms,” pp. 377–392, 03 2017.
[14] F. de Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, “Evolving the hearthstone meta,” *CoRR*, vol. abs/1907.01623, 2019.
[15] N. Tomašev, U. Paquet, D. Hassabis, and V. Kramnik, “Assessing game balance with alphazero: Exploring alternative rule sets in chess,” *arXiv preprint arXiv:2009.04374*, 2020.
[16] M. Stanescu, N. Barriga, and M. Buro, “Using lanchester attrition laws for combat prediction in starcraft,” 2015.
[17] D. Hind and C. Harvey, “A neat approach to wave generation in tower defense games,” in *IMET*, pp. 1–8, 2022.
[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, “Scikit-learn: Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
[19] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, *et al.*, “Openspiel: A framework for reinforcement learning in games,” *arXiv preprint arXiv:1908.09453*, 2019.
[20] A. Dockhorn, D. Perez-Liebana, *et al.*, “Game state and action abstracting monte carlo tree search for general strategy game-playing,” in *2021 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2021.
[21] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE T-CIAG*, 2012.
[22] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European Conf. on machine learning*, pp. 282–293, Springer, 2006.
[23] M. Swiechowski, K. Godlewski, B. Sawicki, and J. Mandziuk, “Monte carlo tree search: a review of recent modifications and applications,” *Artificial Intelligence Review*, 2022.
[24] L. Xu, J. Hurtado-Grueso, D. Jeurissen, D. P. Liebana, and A. Dockhorn, “Elastic monte carlo tree search with state abstraction for strategy game playing,” in *IEEE Conference on Games (CoG)*, 2022.
[25] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, “Two decades of blackbox optimization applications,” *EURO Journal on Computational Optimization*, vol. 9, p. 100011, 2021.
[26] R. Garfield, *Magic: The Gathering*. Wizards of the Coast, 1993.
[27] K. Kunanusont, R. D. Gaina, J. Liu, D. Perez-Liebana, and S. M. Lucas, “The n-tuple bandit evolutionary algorithm for automatic game improvement,” in *IEEE Congress on Evolutionary Computation*, 2017.
[28] *ChatGPT*. OpenAI, 2022.