Automated Game Balancing of Asymmetric Video Games

Philipp Beau
University of Amsterdam
Intelligent Systems Laboratory
Amsterdam, The Netherlands
Email: philipp.beau@student.uva.nl

Sander Bakkes
Tilburg University
Tilburg centre for Cognition and Communication
Tilburg, The Netherlands
Email: s.c.j.bakkes@uvt.nl

***Abstract*—Designing a (video) game such that it is balanced - i.e. fair for all players - is a prevailing challenge in game design. Perhaps counter-intuitively, games that are symmetric with respect to (board) design, starting conditions, and the employed action set, are not necessarily fair games. Indeed, perfect play from all players does not automatically lead to a draw, but may probabilistically favour e.g., the first player to move. Even more so, *asymmetric games* – in which the action set of one player is typically highly distinct from that of another player – are generally unbalanced unless meticulous care has been taken to ensure that the asymmetry in the design does not skew win probabilities. In this context, the present paper contributes a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation. To assess the effectiveness of the proposed method, experiments were performed with automatically balancing a set of tower-defence games. Preliminary experimental results revealed that the proposed method (1) is able to identify the principal component of a game's imbalance, and (2) can automatically adjust the game design till it is balanced by approximation.**

I. Introduction

It is generally acknowledged that a (video) game needs to be balanced in order to be enjoyable [1]. Informally speaking, this entails that every player, given they possess equal skill, should have the same probability of winning the game. In symmetrical games, where each player can always choose from the same action set and can always start from a position analogous to that of the opponent player(s), this is generally assumed by default (*cf.*, rock-paper-scissors). As such, derived from Herik et al. [2], we consider a game a balanced (i.e. fair) game if it is a game-theoretical draw, and both players have roughly an equal probability on making a mistake. In games containing asymmetrical choices however – like most multiplayer strategy games – each player typically starts with a set of actions that is highly distinct from that of other players, making balancing the game design particularly challenging [3]. Foremost, the challenge follows from attributing the relative impact of an action on the win probability. That is, the effectiveness of an individual action is not directly apparent, as it is dependent on the context in which it is executed.

As current-day video games typically contain dozens to hundreds of different types of (unit / building) actions, it has become highly challenging for a human designer to identify the precise action which causes the game design to be imbalanced. Currently game designers rely on extensive and expensive human testing [4], requiring a considerable amount of time and effort which can even go far past the public release of a game. As such, developing a method which can automatically identify (and correct) the cause of an unbalanced game would accelerate the design process, and can be assumed to positively impact game design practise.

The contribution of the present paper, therefore is a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation.

II. Related work

In related work, Jaffe [5] investigates the restricted-play balance framework, arguing for a mathematical formulation of game balance in which carefully restricted agents are played against standard agents. The work foremost contributes to the field of quantitative balance analysis, and is related to that of Nelson [6], who conceptually explores strategies for automatically extracting balance information from games. Also, Mahlmann et al. [7] have previously investigated evolving card sets to automatically balance the game of Dominion. Van Rozen et al. [8] have performed work on generating balanced tower defence games using *MicroMachinations*.

Indeed, the topic of automated game balancing is of general interest to the gaming community (cf. e.g., Elias et el. [9], Chapter 4.4). As such, Kim et al. [10] investigated a system to collect and visualise data from user studies, called TRUE. Their system analyses player deaths to find the cause of unintended difficulty artefacts introduced during development. Debeauvais et al. [11] uses aggregated data from the racing game *Forza Motorsport 4* to analyse how players use and customise driving assists; enabling them to balance the difficulty level of the game. Also, Lewis and Wardrip-Fruin [12] collected and analysed large quantities of game data from the popular MMORPG *World of Warcraft* which they used to investigate common player assumptions, such believed imbalances in specific game classes being more efficient for reaching the maximum character level. Indeed, while a plethora of research exists on dynamic difficulty adjustment (DDA), deep analysis of the balance of a game design – and its principal components – and the automated balancing thereupon, is still a relatively under-explored field.

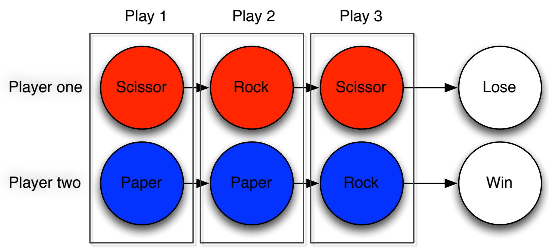


Fig. 1: Action history of a best of three rock, paper, scissor game.

III. Method

Here we describe our method to automated game balancing of asymmetric video games. It consists of maintaining an administration of action histories (III.A) in the form of an action history tree (III.B). We give an example of an (im)balanced game, (III.C), and describe our procedure to identifying actions of high impact on a game's win probability (III.D); the method builds upon Monte Carlo tree search (MCTS) techniques.

A. Action history

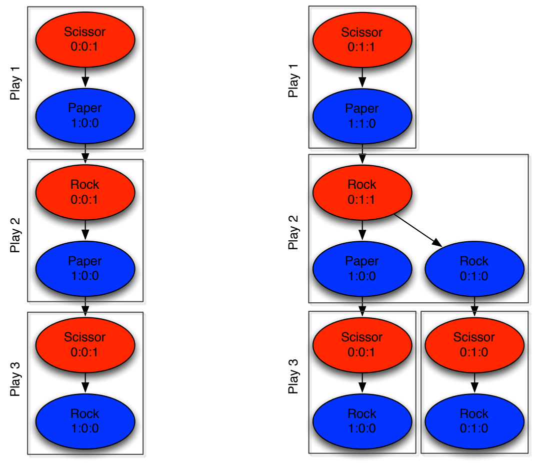
We consider the action history in a game to be the concatenation of actions taken by a player within the game, leading to a certain outcome. For instance, looking at a game of best of three rock-paper-scissors (RPS). If player one plays scissor, rock, scissors and player two plays paper, paper, rock this leads to player two winning the game and player one losing. The action history of this RPS game is thus $(scissor \times paper), (rock \times paper), (scissor \times rock)$ which leads to a loss for player one and a win for player two.

B. The action history tree

The simulated action histories are stored in a tree structure. The root node of the tree is the first action of player one, and is succeeded by the the first action of player two, then the second action of player one, etc. In every node the observed number of wins, draws, and loses subsequent to that action is stored; the metrics are derived from all simulated playouts from this node onward. Figure 2a shows the action history tree for the action history shown in Subsection III-A. Adding another action history to the tree, player one (Scissor, Rock, Scissor) vs. player two (Paper, Rock, Rock), results in Figure 2b.

C. Example of an (im)balanced game

Consider the – balanced – rock-paper-scissors (RPS) game; a zero-sum hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. The action set for both players is identical, and the game is designed such that each player has an identical probability of winning the game. To create an imbalanced game on the basis of RPS, a fourth action called *Spock* can be introduced. The Spock action only loses against paper, and replaces rock of player one in the second play of the best of three RPS game (Table Ia). As a result the win ratio of the game becomes skewed towards player one (Table Ib) making RPS+Spock an imbalanced game. As such, the Spock action can be identified as an action with high impact on the game's win probability, in this case skewing the win probability in favour of player one.



(a) One action history (b) Two action histories

Fig. 2: Action history tree.

TABLE I: Results of RPS+Spock in the second play of the game (a), and the outcome of 10000 best of three RPS+Spock games (b).

(a)

		P2		
		Rock	Paper	Scissor
P1	Rock	Tie	P2	P1
	Paper	P1	Tie	P2
	Scissor	P2	P1	Tie
	Spock	P1	P2	P1

(b)

Winner	Ratio
Player one	42%
Tie	25%
Player two	33%

D. Discovering high impact actions in a game

In complex, actual video games, play-out information such as that in Table Ia and Ib is typically unavailable, among other reasons because game actions generally can not be directly pitted against each other in isolation. What is feasible, however, is observing the win ratio of simulated games, and monitoring the actions played to construct a means of analysis – such as building an action history tree.

Having an imbalanced game, the goal is to identify high impact actions which are leading to a win disproportionately more often than a loss. In an effort to calculate the impact of an action on a game one first has to look at the average win/draw/lose ratio (AWR, ADR, ALR) following an action of a player. In other words, does an action in general lead to a win more often than other actions of the same player.

$$\begin{aligned} AWR_{action} &= \textstyle\sum_a^A a.win / \sum_b^A (b.win + b.draw + b.lose) \\ ADR_{action} &= \textstyle\sum_a^A a.draw / \sum_b^A (b.win + b.draw + b.lose) \\ ALR_{action} &= \textstyle\sum_a^A a.lose / \sum_b^A (b.win + b.draw + b.lose) \\ &\text{where } A = \text{all nodes of } action \text{ in the action history tree} \end{aligned}$$

In the example of RPS+Spock of Table IIb, spock outperforms all other actions of player one in AWR and ALR, with the AWR of spock (49%) being substantially above the overall win ratio of 42% (Table Ib). Second, we introduce AWR$_{a,w}$, ADR$_{a,w}$ and ALR$_{a,w}$ which is defined as the AWR, ADR, ALR of an action a given the absence of action w; it is calculated via Algorithm 1. That is, procedure $CalcWDLRatio$ calculates the average win/draw/loss ratio of an action a in

TABLE II: Average win, draw and lose ratio after a specific action got played.

(a) Rock-paper-scissors

		AWR	ADR	ALR
P1	Rock	0.37	0.26	0.37
	Paper	0.37	0.26	0.37
	Scissor	0.36	0.26	0.38
P2	Rock	0.38	0.25	0.37
	Paper	0.37	0.26	0.37
	Scissor	0.36	0.27	0.37

(b) Rock-paper-scissors-spock

		AWR	ADR	ALR
P1	Rock	0.42	0.24	0.33
	Paper	0.40	0.25	0.34
	Scissor	0.39	0.25	0.35
	Spock	*0.49*	*0.21*	*0.29*
P2	Rock	0.32	0.24	0.44
	Paper	0.34	0.25	0.40
	Scissor	0.35	0.24	0.41

TABLE III: The calculated impact for each action of player one in RPS+Spock.

Action	Impact
Rock	-0.06
Paper	-0.04
Scissors	-0.03
Spock	*+0.11*

the game if the action w would not exist. If the average win/draw/loss ratio is affected, a can be considered dependent on w. Procedure $CalcInternal$ is an internal subroutine of procedure $CalcWDLRatio$. The result of the algorithm is a vector containing

$$\begin{bmatrix} AWR_{a,w} \\ ADR_{a,w} \\ ALR_{a,w} \end{bmatrix}$$

Finally we define the *impact* I of an action b as how much does b affect other actions towards a higher win ratio when used in the same game. In other words, how big of a positive/negative impact does a specific action have on other actions and ultimately the game.

$$\mathrm{I}(b) = \sum_a^A (AWR_a - AWR_{a,b}) + 0.5 \times (ADR_a - ADR_{a,b}) - (ALR_a - ALR_{a,b})$$
where $A =$ all nodes of $action$ in the action history tree

We surmise that in an imbalanced game, where the win probability is skewed towards one player (race / class, etc.), the action with the highest impact of the winning player is the source of the imbalance. In our example, having calculated the impact I of every action b with Algorithm 1, one observes that of all actions that player one can use in RPS+Spock (Table III), the imbalanced action spock of player one can be identified as the action with the highest impact on the game.

Algorithm 1 Calculate AWR$_{a,w}$ ADR$_{a,w}$ and ALR$_{a,w}$

```
procedure CalcWDLWithout($a, w$)
    result = [0,0,0]
    for all node in actionHistoryTree.startingNodes do
        result += CalcInternal(a, w, node, false)
    end for
    total = result[0] +result[1] +result[2]
    return result /= total
end procedure

procedure CalcInternal($a, w, node, found$)
    result = [0,0,0]
    if node.player != a.player then
        for all child in node.children do
            result += CalcInternal(a, w, child, found)
        end for
        return result
    end if
    if node.action = a then
        result += node.result
        found = true
    else if node.action = w then
        if found then
            return -node.result
        else
            return result
        end if
    end if
    for all child in node.children do
        result += CalcInternal(a, w, child, found)
    end for
    return result
end procedure
```

IV. Simulator

For our experiments, we investigate a highly asymmetric type of strategy game, namely the popular tower defence (TD) games. A typical TD game (illustrated in Figure 3) is highly asymmetric, in that one player can spawn units that traverse a predetermined path (to attack the tower of the opponent), while the opponent player can construct buildings alongside the path that attack these units (to defend the tower). The offensive player wins if she succeeds in destroying the tower of the opponent, the defensive player wins if she can withstand these attacks. Indeed, the actions that players can take are highly distinct from each other, and balancing their effectiveness generally requires meticulous manual balancing.

We developed a simulator for real-time TD games, which – for rapid experimenting – provides the ability to decouple graphics from actual gameplay.[1] Indeed, TD is a genre which is popular within the gaming community and offers a variety of research opportunities such as dynamic difficulty adjustment, map generation, and player modelling [13], [14], [15]. Our hope is that the developed simulator may contribute to such related branches of research as well.

A. Experimental Implementation

The developed simulator contains two races, the *human race* and the *alien race*. Both races are distinguished by the towers they can build (IV-E3), while having the same selection of units and upgrades to choose from (IV-E). In this section we first give a brief overview of how the developed game works before explaining each design aspect in detail.

[1]The developed simulator is publicly available at Github https//github.com/philiiiiiipp/multiplayer-balancing

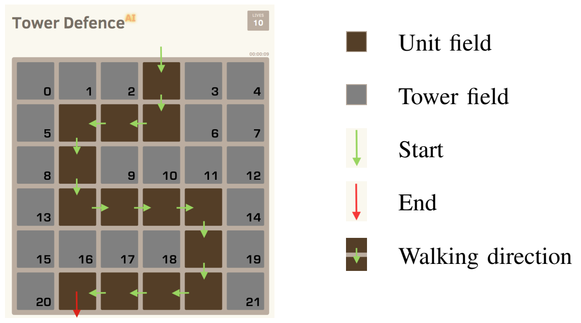


Fig. 3: Example 6×6 tower defence game field.

```
OOOSOO   O:  Tower field
OXXXOO   X:  Unit field
OXOOOO   S:  Start unit field
OXXXXO   E:  End unit field
OOOOXO
OEXXXO
```

Fig. 4: Representation of the example 6×6 game field in Figure 3.

In our simulated TD games, two players are playing against each other. The game is played simultaneously on two identical maps; one for the alien player, one for the human player. Each player builds towers on their own map. Also, each player sends creeps (units) to the other player's map. Each player can upgrade the units it will send to the other players map (once an upgrade was chosen, all consecutive sent units will have the improved attributes, units already sent stay the same). If a unit reaches the end of a player's map without getting 'killed' this player loses a live. If a player is at 0 or less lives, he/she loses. If that happens to both players at the same time it's a tie.

B. Game field

The game field or map consists of a $n \times n$ grid of fields. A field can be either a tower field or a unit field. The tower fields belong to the player playing on that map for placing towers, while the unit fields are where the opposing players units will be walking. Units get spawned at the start field and are walking towards the end field. Each tower field can be uniquely identified by its position starting with zero at the top left corner and ending at N-1 in the bottom right where N denotes the total amount of tower fields of the map (Figure 3).

New game maps can easily be created using the notation of Figure 4 inside a text file and placing it inside the root directory. The framework will automatically parse the file and convert it into a map if all of the following attributes are met. First, exactly one start field S and exactly one end field E must be present at the boundary regions of the map. Second, the start and the end fields must be connected via unit fields. Third, in every map, the way units walk must be uniquely identifiable. Meaning, a unit field must be adjacent to exactly two other unit fields with the exception of the start and end field which have to have exactly one adjacent unit field.

C. Towers

In the simulator, races differ from each other with respect to the towers that are available to them. Every tower in the game is exclusive to one race. Where the human race can build the fire, ice and archer tower, the alien race has the chain lightning, parasite and shock tower. To provide a realistic challenge to our method, every tower (action) is given two attributes, the damage dealt with each shot, and the distance that it can shoot. The distance or range of a tower is denoted in game fields reachable from the place the tower got build using horizontal, vertical or diagonal movement. In addition, most towers have a unique special ability, as detailed in Table V and Table VI.

TABLE IV: Possible unit upgrades that the players may select.

Id	Attribute	Initial	Increase per upgrade
H	Health	1.0	+0.7
M	Movement	1.0	+0.4
A	Amount	1.0	+0.75

D. Game cycle

At the start of every game both players enter with the same amount of lives and an instance of the same $n \times n$ map. Subsequently, the game cycles through the following three steps. (**1**) Both players choose one action which gets executed right away. An action can either be to send units, to upgrade all subsequent units in health, movement or amount or to build a tower on one of the free tower fields of the players map. (**2**) The already placed towers will pick a unit inside their range to shoot at. The decision which unit is picked at a given point is dependent on the type of tower choosing the target. Fire, archer and chain lightning towers will shoot at their last damaged target. If their last target is either dead or out of range, they choose a target at random. Ice, parasite and shock towers will preferably shoot at a unit currently not influenced by their special ability. If no such unit exists they will choose a target at random too. (**3**) All units walk appropriate to their movement attribute towards the end of the game field. If a unit walks out of the map the player playing on that map loses a life.

If after step 3 none of the three criteria are met, the game continues with step 1. The three end criteria are (1) a player has 0 or less lives (the other player wins), (2) both players have 0 or less lives (tie), or (3) game exceeded the maximum amount of cycles (tie).

E. Actions

In the simulated tower defence game, two types of actions are available, that is (1) actions which are available to both races like upgrading and sending units, called *global actions*, and (2) actions only available to players of a certain race, called *race specific actions*, such as placing a specific tower.

1) Upgrading units: At every step of the game, a player can choose to (1) upgrade one of the attributes of the units, or (2) upgrade the amount of units that is spawned at starting position. A unit has two attributes, *health* and *movement*. Health describes the amount of damage a unit can take until they get removed from the map. Movement describes the amount of fields a unit walks along its path at every step.

2) Sending units: If at any step a player chooses to send units, the game will create units using the attributes of Table IV. Every attribute a will be set to:

TABLE V: Towers available to the Human race.

Id	Tower	Damage	Range	Special
F	Fire	0.6	1	Damages all units on one field
I	Ice	1	1	Reduces movement speed by 90% for 3 steps
A	Archer	1	3	-

TABLE VI: Towers available to the Alien race.

Id	Tower	Damage	Range	Special
C	Chain lightning	0.4	1	Damages 3 units less than 3 fields apart
P	Parasite	0	1	Lets the hit unit walk backwards for 2 steps
S	Shock	1	1	Reduces movement speed to 0 for 3 steps

$$I_a + t_a * U_a \tag{1}$$

where I_a is the initial value of attribute a, t_a denotes the amount of times a was upgraded, and U_a denotes the increase per upgrade. While all three attributes hold floating point values, the amount attribute only takes the integer-part into account. E.g., if the amount is 2.5, then only 2 units will be spawned.

3) Placing towers: Next to actions available to both races, each race can build towers that are unique to its race (discussed in Section IV-C). Where the human race can build the fire, ice and archer tower, the alien race has the chain lightning, parasite and shock tower. Every tower has two attributes, the damage dealt with each shot and the distance it can shoot in game fields (Table V and Table VI).

V. Experiments

Here, we discuss two experiments that test our method to automated game balancing in an asymmetric tower defence video game. The first experiment evaluates to what extent our method can identify and automatically correct an imbalance in a reasonably well-balanced asymmetric game. The second experiment evaluates to what extent our method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game.

A. Data generation

The generation of player data was performed with two Monte Carlo tree search (MCTS) [16] agents playing against each other. One MCTS agent was playing the alien race, the other MCTS agent was playing the human race. The two MCTS agents learned from 100.000 playouts of playing against each other (backpropagating rewards to learn effective behaviour), and the last 5.000 action histories were used to populate the action history tree.

The specific MCTS algorithm used by the agents is referred to as Upper Confidence Tree (UCT) [16] which is extending the Upper Confidence Bound algorithm by Auer et al.[17] to trees. UCT combines the exploration and the building of the tree. The tree starts at the root node, after which the algorithm iterates through three phases: the bandit phase, the tree building phase, and the random walk phase.

The bandit phase starts in the root node where to agent continually chooses an action/child node until arriving in a leaf node. The decision which action is taken at every step is handled as a multi armed bandit problem. The set $\mathcal{A}_s$ of possible actions a in a node s defines the child nodes (s, a) of s. The selected action a^* maximises the upper confidence bound:

$$\hat{r}_{s,a} + \sqrt{c_e \log(n_s)/n_{s,a}} \tag{2}$$

over all a in $\mathcal{A}_s$ with $\hat{r}_{s,a}$ describing the average reward accumulated by selecting action a in state s, n_s the total number of times node s was visited and $n_{s,a}$ the amount of times action a was taken from node s. The term c_e handles the exploration vs. exploitation trade-off where a high c_e favours exploration and a low c_e exploitation.

The tree building phase is entered upon arrival in a leaf node. An action is selected uniformly at random and added as a child node of s.

The random walk phase begins after the new child node was added to the tree. At every step an action is taken (uniformly or heuristically) until the game ends. At this point the acquired reward r_u is back propagated towards the root node and all nodes in this tree run are updated:

$$\hat{r}_{s,a} \leftarrow \frac{1}{n_{s,a}+1}(n_{s,a} \times \hat{r}_{s,a} + r_u) \tag{3}$$

$$n_{s,a} \leftarrow n_{s,a} + 1; \; n_s \leftarrow n_s + 1 \tag{4}$$

Both MCTS agents where initialised with the same values for all experiments in the following sections. I.e., $\hat{r}_{s,a}$ initial is 0.5, and c_e is $3 * \sqrt{2}$. The terminal reward r_u for an agent is dependant on the outcome of the game:

$$r_u = \begin{cases} 10 & \text{if agent won} \\ L_{played} - L_{max} & \text{if draw} \\ -2 * L_{max} & \text{if agent lost} \end{cases}$$

with L_{max} denoting the maximum allowed game length and L_{played} the actual game length.

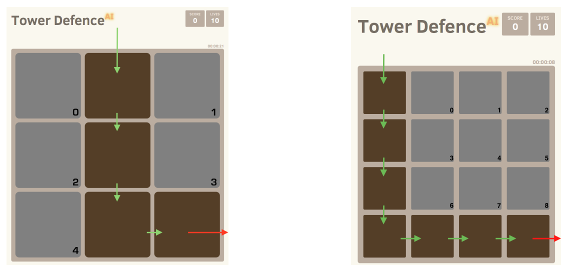
B. Experiment 1

This first experiment evaluates to what extent our method can identify and automatically correct an imbalance in a reasonably well-balanced asymmetric game.

1) Experimental setup: All tower attributes were initialised manually with the designer's best intention to create a balanced game (Table VII). The game runs on two different maps, a three by three map (Figure 5a) and a four by four map (Figure 5b) where each player starts with 10 lives and the maximum game length is set to 100 steps. Every run was repeated three times and all numbers in the following section depict the average of those. After the agents have played against each other, the resulting win ratio is calculated, the attributes of the tower with the highest impact of the winning

TABLE VII: Experiment 1 – Tower attributes.

Id	Tower	Damage	Range	Special
F	Fire	0.6	1	Damages all units on one field
I	Ice	1	1	Reduces movement speed by 90% for 3 steps
A	Archer	1	3	-
C	Chain lightning	0.5	1	Damages 3 units less than 3 fields apart
P	Parasite	0	1	Lets the hit unit walk backwards for 3 steps
S	Shock	0.5	1	Reduces movement speed to 0 for 3 steps



(a) Three by three map (3x3) (b) Four by four map (4x4)

Fig. 5: Maps used in the two experiments.

faction will be decreased and the experimental trial is repeated until the difference in win ratio is smaller than 1% and the game considered balanced.

2) Results on 3x3 map: Analysis of the provided game design reveals a considerable difference in win ratio between the alien and the human player, with the alien player being 34% more likely to win the game against the human player on the 3x3 map (the human player wins 38%, the alien player 51%, the game ties 11%). This observation leads to the conclusion that this game is imbalanced in favour of the alien player. Calculating the impact of each tower (Figure 6) reveals the chain lightning tower has the largest impact on the game of the alien player.

Following the hypothesis that in an imbalanced game the action with the largest impact, is the action that is presumably causing the imbalance (cf. Section III), the damage of the chain lightning tower was gradually lowered from 0.6 to 0.3 (Table VIII). Lowering the damage output of the chain lightning tower ultimately resulted in a balanced game (Table VIII). Interestingly a direct correlation between the decline of the chain lightning tower's impact (Figure 7) and the decline of the alien players win ratio could be observed, indicating that, indeed, the chain lightning tower was the action causing the imbalance.

To verify these results, indeed, one could argue that the same effect would have been achieved lowering the attributes of the shock and/or parasite tower. To investigate this, the experiment was repeated using three different settings (Table IX), where the attributes of the parasite and/or shock tower where lowered. Looking at the results of each setting in Table X, even substantially lowering the attributes of any alien tower other than the chain lightning tower does not have the same effect on the game as lowering the attributes of the chain lightning tower.

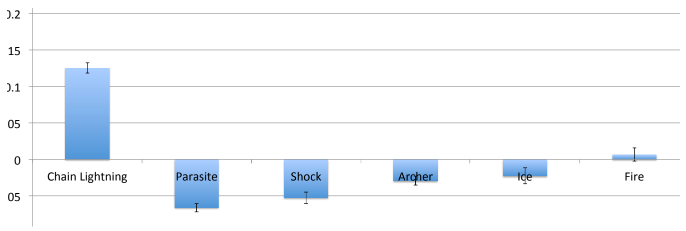


Fig. 6: Impact of every tower using the attributes in Table VII.

TABLE VIII: Win ratio human vs. alien agent depending on the damage of the chain lightning tower.

	0.6	0.5	0.4	0.3
Human	0.38%	40%	42%	44%
Tie	0.11%	11%	12%	12%
Alien	0.51%	49%	46%	44%

Calculating the impact of the towers over the course of the different settings (Figure 8) one can observe that neither settings has a significant impact on the values. The chain lightning tower remains the tower with the highest impact, supporting to the conclusion that it was in fact the chain lightning tower which caused the game to be imbalanced, and was as such corrected adjusted for establishing a balanced game design.

3) Results on 4x4 map: In contrast to the 3x3 map, the analysis of the proposed game design reveals a win ratio in favour of the human player, winning the game 17% more often than the alien player (the human player wins 48%, the alien player 41%, the game ties 11%). This indicates that this game is imbalanced in favour of the human player. To find the action responsible for the imbalance, an analysis of the impact of each tower was performed (Figure 9). It reveals that of all towers, the archer tower is the tower with the highest impact on the game.

To assert the correctness of the analysis, we employ the same process as in the previous section. First the damage of the archer tower was gradually lowered until a predictably balanced game was achieved (Table XI). When lowering the damage of the archer tower to 0.4, a balanced game is achieved. Again, a clear correlation between a lower damage and a decrease in impact of the archer tower can be observed (Figure 10).

To independently verify that the archer tower was indeed the source of the imbalance in the human race, we again try to achieve balance lowering the attributes of the other towers (fire and ice) (Table XII). As in the previous section, even substantially lowering the attributes of the other towers did not result in a balanced game (Table XIII). The win ratio changed slightly in favour of the alien player, but the human player still won the game around 7% more often. This leads to the conclusion, that the archer tower was in fact the source of the imbalance as suggested by its impact, and was correctly adjusted to achieve a balanced game design.

4) Discussion of the results: In this experiment, the proposed method was tested on two different maps using the same tower attributes. On both maps a balanced game was achieved automatically after adjustment of the correctly identified imbalanced action. On both maps a correlation between decrease

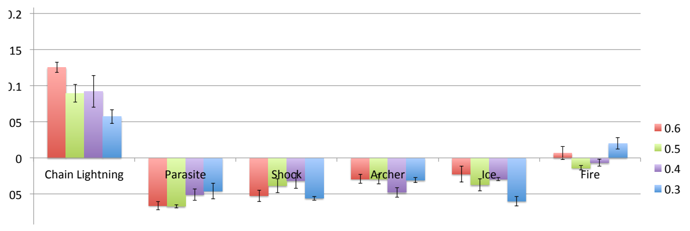


Fig. 7: Impact of the lightning tower, lowering its damage output from 0.6 to 0.3.

TABLE IX: The different settings the experiment was repeated with. Each setting uses Table VII as a foundation.

Setting	
A	Parasite effect reduced to 2 steps
B	Parasite and shock effect reduced to 1 step
C	Setting B and shock tower damage reduced to 0.25

of tower attributes and decrease of impact could be seen when balancing the highest impact tower (HIT), while the impact of it even increased if other towers attributes got lowered; this follows naturally, as the HIT will become more and more important as other towers get weaker.

C. Experiment 2

This second experiment evaluates to what extent our method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game, regardless of initial values of the action attributes.

1) Experimental setup: To test the proposed method, the following system was setup to automatically balance the tower defence game. This happens in two steps, first the tower attributes are initialised at random using the ranges described in Table XIV. Second, the thus created game is balanced using three different methods subsequently while recording the balancing steps applied by each of them.

A balancing step is defined as the decrease of one attributes value of a chosen tower using the arithmetic in Table XIV to slowly lower the attribute action. The attributes get selected in turns. For example, if the shock tower gets selected three times, the damage attribute will be lowered first, followed by the duration attribute, followed by the damage attribute. The experiment was repeated 10 times on the same 3×3 map used in experiment 1 where every player starts with 10 lives and the maximum game length is set to 100 steps. A game is considered balanced if the difference between the human and the alien win ratio is below 1%.

We employ three methods for automated balancing, (**1**) balance the tower with the highest impact of the winning faction, (**2**) balance a random tower of the winning faction, and (**3**) balance a random tower of the winning faction with the exception of the tower with the highest impact. If the impact attribute does accurately predict the unbalanced tower, then the first method should use considerably fewer balancing steps than the one choosing randomly while the method excluding the high impact action should use more.

2) Result: The experimental results reveal that using the impact to balance a game (method 1) – with an average of

TABLE X: Win ratio human vs. alien agent given setting A, B, or C.

	Setting A	Setting B	Setting C
Human	39%	40%	41%
Tie	11%	11%	11%
Alien	50%	49%	48%

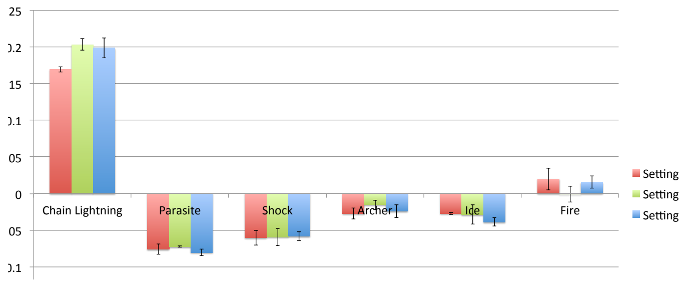


Fig. 8: Impact of every tower after applying setting A, B, or C.

8.1 balancing steps – it is by approximately 65% faster than choosing random towers (Table XV), outperforming method 2 in 10 out of 10 runs. Avoiding the high impact tower in method 3 is 35% worse than picking it randomly (excluding the runs it did not result in a balanced game) and over 2 times worse than using the impact to balance the game. Method 3 never outperforms method 1 but does outperform method 2 in 2 out of 10 times. Method 3 was the only method which did not result in a balanced game in 3 out of 10 times.

3) Discussion of the results: This second experiment indicated that the proposed method can identify and automatically correct an imbalance in a strictly imbalanced asymmetric game, regardless of initial values of the action attributes. Indeed, it may be argued that favourable initial parameter settings may render balancing the design a relatively straightforward task. As the present experiment showed, if this actually would have been the case, choosing random actions should have outperformed our method at least once in the course of this experiment. This however, was not the case. Balancing the action with the highest calculated impact consistently outperformed choosing randomly and was on average 65% faster in creating a balanced game. While the experiment should be considered a first step in our investigation, the experimental results indicated that the proposed automated balancing method is able to successfully balance a game design regardless of parameter values suggested by the game designer.

VI. Conclusion

Creating a well balanced multiplayer game is a challenging and tedious task requiring a large amount of of human player feedback. The challenge however can be significantly reduced by understanding which actions cause a game to be unbalanced. As such, the present paper contributed a method for automatically balancing the design of asymmetric games. It employs Monte Carlo simulation to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation. To assess the effectiveness of the proposed method, experiments were performed with automatically balancing a set of tower-defence games. Preliminary experimental results revealed that the proposed method (1) is able to identify the principal

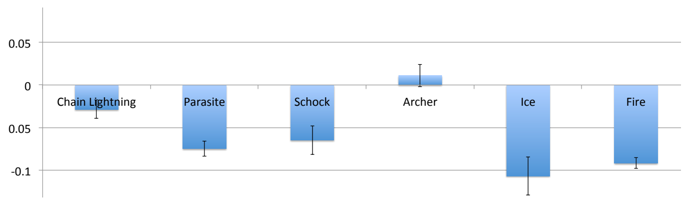


Fig. 9: Impact of every tower using the attributes in Table VII.

TABLE XI: Win ratio human vs. alien agent depending on the damage of the archer tower.

	1.0	0.8	0.6	0.4
Human	48%	47%	46%	44%
Tie	11%	11%	11%	11%
Alien	41%	42%	43%	45%

component of a game's imbalance, and (2) can automatically adjust the game design till it is balanced by approximation.

For future work, we will particularly investigate how the present linear computational effectiveness of the method may be enhanced further, and how the method may be embedded in mixed-initiative game-design toolkits.

References

[1] A. Rollings and D. Morris, "Game architecture and design: a new edition," 2003.

[2] H. J. Van den Herik, J. W. Uiterwijk, and J. Van Rijswijck, "Games solved: Now and in the future," *Artificial Intelligence*, vol. 134, no. 1, pp. 277–311, 2002.

[3] K. Hullett, N. Nagappan, E. Schuh, and J. Hopson, "Empirical analysis of user data in game software development," in *ESEM*. IEEE, 2012, pp. 89–98.

[4] R. Leigh, J. Schonfeld, and S. J. Louis, "Using coevolution to understand and validate game balance in continuous games," in *GECCO*, 2008, pp. 1563–1570.

[5] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic, "Evaluating competitive game balance with restricted play." in *AIIDE*, 2012.

[6] M. J. Nelson, "Game metrics without players: Strategies for understanding game artifacts," in *Artificial Intelligence in the Game Design Process*, 2011.

[7] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *IEEE CEC*, 2012, pp. 1–8.

[8] R. van Rozen and J. Dormans, "Adapting game mechanics with micro-machinations," in *FDG*, 2014.

[9] G. S. Elias, R. Garfield, K. R. Gutschera, and P. Whitley, *Characteristics of games*. MIT Press, 2012, 4:4.

[10] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, "Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems," in *CHI*, 2008, pp. 443–452.

[11] T. Debeauvais, T. Zimmermann, N. Nagappan, K. Carter, R. Cooper, D. Greenawalt, and T. Solberg, "Off with their assists: An empirical study of driving skill in Forza Motorsport 4," *FDG*, 2014.

[12] C. Lewis and N. Wardrip-Fruin, "Mining game statistics from web services: a world of warcraft armory case study," in *FDG*, 2010, pp. 100–107.

[13] S. K. Lo, H.-C. Keh, and C.-M. Chang, "A multi-agents coordination mechanism to improving real-time strategy on tower defense game," *Journal of Applied Sciences*, vol. 13, no. 5, p. 683, 2013.

[14] P. Avery, J. Togelius, E. Alistar, and R. P. Van Leeuwen, "Computational intelligence and tower defence games," in *CEC*, 2011, pp. 1084–1091.

[15] P. Rummell *et al.*, "Adaptive ai to play tower defense game," in *CGAMES*, 2011, pp. 38–40.

[16] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conference on Machine Learning*, 2006, pp. 282–293.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

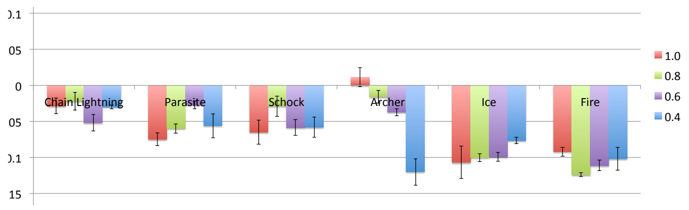


Fig. 10: Impact of the archer tower while lowering the damage from 1.0 to 0.4.

TABLE XII: The different settings the experiment was repeated with. Each setting uses Table VII as a foundation.

Setting	
A	Fire tower damage set to 0.4
B	Setting A and frozen effect reduced to 40%
C	Setting B and ice tower damage reduced to 0.5
D	Setting C and fire tower damage reduced to 0.2

TABLE XIII: Win ratio human vs. alien agent given setting A, B, C, or D.

	Setting A	Setting B	Setting C	Setting D
Human	47%	48%	46%	46%
Tie	11%	11%	11%	11%
Alien	42%	41%	43%	43%

TABLE XIV: Experiment 2 – Tower attributes.

Id	Tower	Attribute	Range	Balancing step
F	Fire	damage	0.0 - 1.0	$*0.9$
I	Ice	damage	0.0 - 1.0	$*0.9$
		hindrance	0.0 - 1.0	$*0.9$
		duration	1 - 9	-1
A	Archer	damage	0.0 - 1.0	$*0.9$
C	Chain lightning	damage	0.0 - 1.0	$*0.9$
		jumps	1 - 5	-1
		length	1 - 5	-1
P	Parasite	damage	0.0 - 1.0	$*0.9$
		duration	1 - 9	-1
S	Shock	damage	0.0 - 1.0	$*0.9$
		duration	1 - 9	- 1

TABLE XV: Experiment 2 – The amount of times a tower's attribute had to be adjusted in order to achieve a balanced game. ($*$) = Balance could not be achieved.

Method	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Avg.
1.	3	5	18	5	2	8	11	14	1	14	8.1
2.	4	9	19	6	9	18	15	26	3	25	13.4
3.	14	41	23	24	$*$	9	13	$*$	3	$*$	18.1