Balancing Turn-Based Games with Chained Strategy Generation

William Kavanagh, Alice Miller, Gethin Norman, Oana Andrei
School of Computer Science, University of Glasgow, United Kingdom

Probabilistic model checking can overcome much of the complexity inherent in balancing games. Game balancing is the careful maintenance of relationships between the ways in which a game can be played, to ensure no single way is strictly better than all others and that players are offered a wide variety of ways to play successfully. We introduce a novel approach towards automating game balancing using probabilistic model checking called chained strategy generation (CSG). This involves generating chains of adversarial strategies which mimic the way players adapt their approach during repeated plays of a game. We use CSG to map out the evolving metagame. The trends identified can allow game developers to identify strategies which will be too strong and ways of playing the game which a player may want to use, but are never viable for successful competitive play. We introduce a case study, a game called RPGLite, and use CSG to compare five candidate configurations for the game. We show how to determine which configurations of RPGLite lead to a more fair and interesting experience for players. We also identify unexpected trends in how the strategies evolve. Our approach introduces a new technique for improving game development and player experience.

***Index Terms*—Formal Verification, Automated Game Balancing, Game Design, Model Checking, Adversary Generation**

I. Introduction

GAME balancing is a crucial, but notoriously difficult part of game development. What constitutes a balanced game is varies widely because the term depends on the game genre. Beyer et al. [1] describe game balancing as:

"the process of systematically modifying parameters of game components and operational rules in order to determine satisfactory configurations regarding predefined goals."

The predefined goals are specific to the game under consideration with the aim of ensuring it is fair to all players and is interesting to play. In this paper we introduce a technique based on model checking, *chained strategy generation (CSG)*, that models a game being played over a period of time through evolving strategies. We focus here on turn-based stochastic games. We use a simple case study to show how this approach allows for comparison of several similar configurations of a game and the analysis of which are *satisfactorily balanced*.

Our approach displays the metagame [2], which is the evolving trend in what material and strategies players are most likely to play. The metagame must be considered when looking at game balance because games must be balanced for the duration that they are played and not just at a single point in time. *We focus on measures of fairness and of how interesting a game is to play, to show how balanced a game is*. Traditionally fairness refers to how fair a game is to the players, however this is often achieved through symmetry with players being offered the same choice of game materials. When we refer to fairness, we are considering fairness with respect to the materials rather than the players. We demonstrate that a game is interesting by verifying that the material used during competitive play is diverse, i.e. that differing material is used throughout the metagame. We argue that this is sufficient to show that a game is interesting, given that the material is orthogonally differentiated [3], [4]. This means that the material differs qualitatively rather than quantitatively, e.g. comparing a sword and a bow rather than comparing a longsword and a shortsword. We ensure fairness by showing that there are no dominant strategies and no material is dominated such that it is never employed during competitive play. We measure fairness by calculating the extent to which effective strategies can be countered.

The results of CSG can be used to predict developments in the metagame ahead of time. The intuition behind this approach is not that we are modelling the ways individual players learn to play a game, rather that we are modelling whole communities playing over long periods of time. We assume that communities will tend towards a locally optimal strategy against some currently known effective strategy. Players will look for the most effective way to win, which is likely to be the best way to win against the current most popular strategy. Our approach can be used to show how diverse the strategies found during the players' ongoing search for the best way of playing will be, with greater diversity suggesting a more balanced game.

Current industry methods for game balancing are reactive, using player data and feedback to adjust game mechanics and the configurations of game material to improve the player experience. The aim of our work is to introduce a proactive approach that can show that a game is fair and interesting before it is released. Our approach does not use player data, the gathering of which may require players to have a poor experience playing unbalanced games.

Approaches to aid competitive game balancing without using player data include using genetic algorithms [5] and artificial agents [1] to automate game play. However, these approaches do not consider strategies that develop in response

to repeated plays of a game. The comparative balance of specific material, rather than strategies, is analysed using autonomous agents and restricted game mechanics in [6]. The balance of the card game, Top Trumps, is optimised using both single-objective [7] and multi-objective optimisation [8], although the cards in Top Trumps are not orthogonally differentiated. Reinforcement learning (RL) techniques have been used for dynamic difficulty adjustment of single player games [9], [10], but RL has not yet been adopted as a tool to balance multiplayer games. Other approaches to automated game analysis include [11] which measures the *interestingness* of games as a function of several aspects such as game length, drawing quota and variability.

Model checking [12] is a widely used technique for automated verification of reactive systems. Model checking involves constructing a mathematical model that captures a systems behaviour. To verify that a requirement, usually specified in temporal logic, is satisfied, the model is systematically explored and analysed. CSG uses model checking to recursively define effective strategies. Specifically, we use the probabilistic model checker PRISM [13], which is employed in a wide variety of contexts, from analysing automated search-and-rescue efforts [14] to verification of cardiac pacemakers [15]. Model checking is increasingly being used to analyse games and game development, including identifying bugs in major videogame titles [16] and the model checking of simple multiplayer games [17]. Balance in board games has been studied [18], where *replayability* is achieved through maximising game duration, randomised play and there being various successful strategies. Our work uses a more realistic form of strategy description, obtained through maximising the probability of winning, rather than altering a static value of *aggressiveness*.

In summary, the contributions of this paper are: (i) the specification of CSG, a technique for analysing a game's balance; (ii) methods for analysing a predicted metagame to improve player experience; (iii) a case-study of a simple turn-based stochastic game analysed with CSG, and; (iv) the introduction of model checking as a tool for game balancing.

Next we describe the mathematical model of games we use and explain our CSG technique along with the statistical analysis that can be performed on the results. In Sect. III we introduce a case study, RPGLite, and use it to illustrate how our technique can both measure the variety with which a game can be played and identify dominant strategies. We then analyse the results in Sect. IV and discuss how CSG can be used to make judgements on game configurations. We conclude in Sect. V by reflecting on our approach, the challenges of implementation and outlining future work.

II. Methodology

A. Turn-based Stochastic Games

We now introduce the mathematical model of games that we use. For any finite set S, let $Dist(S)$ denote the set of discrete probability distributions over S.

Definition 1. *A turn-based multiplayer stochastic game (TSG) is a tuple* $\mathsf{G}=(\Pi, S, A, \langle S_i\rangle_{i\in\Pi}, \delta)$ *where:* Π *is a finite set of players;* S *is a finite set of states;* A *is a finite set of actions;* $\langle S_i\rangle_{i\in\Pi}$ *is a partition of* S*; and* $\delta : S\times A \to Dist(S)$ *is a partial transition function.*

We say that an action a is *available* in a state s of a TSG if $\delta(s,a)$ is defined. The choice of which available action is selected in state s is under the control of the player i such that $s \in S_i$ and, if action a is selected, then the probability of transitioning to state s' equals $\delta(s,a)(s')$.

A path of a TSG is a sequence $\omega = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ such that $\delta(s_k,a_k)(s_{k+1})>0$ for all $k\geq 0$. Let $\omega(k)$ denote the $(k+1)$th state of the path ω. A strategy for player i is a way of resolving the action choices of the player, based on the game's execution so far. More precisely, a strategy for player i is a function σ_i which maps finite paths whose last state is controlled by player i to distributions over actions available in this last state. The set of strategies of player i is denoted Σ_i. A strategy is deterministic if it always selects actions with probability 1, and memoryless if it makes the same choice for paths that end in the same state. A Markov decision process (MDP) is a single-player TSG; in this case we denote the set of strategies for the single player by Σ.

A strategy profile for a TSG has the form $\sigma=\langle\sigma_i\rangle_{i\in\Pi}$ listing a strategy for each player. We use $IPaths_s^\sigma$ for the set of infinite paths corresponding to the choices of the profile σ when starting in state s. For a profile σ and starting state s, the behaviour of a TSG is fully probabilistic and we can define a probability measure $Prob_s^\sigma$ over the set of infinite paths $IPaths_s^\sigma$ [19]. A fundamental property of TSGs is the probability of reaching a target set. For TSG G, profile σ and set of target states F, the probability of reaching F from the state s under profile σ is given by:

$$\mathbb{P}_{\mathsf{G}}^\sigma(s,F) = Prob_s^\sigma\{\omega \in IPaths_s^\sigma \mid \omega(k)\in F \text{ for some } k\in\mathbb{N}\}$$

For a two-player TSG G and target set F we assume the game is zero-sum, i.e. player 1 tries to maximise the probability of reaching F and player 2 tries to minimise it. In this zero-sum setting, an optimal strategy $\sigma_1^\star$ for player 1 in state s of G is a strategy that maximises the probability of reaching F no matter the strategy of player 2, formally we have:

$$\inf_{\sigma_2\in\Sigma_2}\mathbb{P}_{\mathsf{G}}^{\sigma_1^\star,\sigma_2}(s,F) = \sup_{\sigma_1\in\Sigma_1}\inf_{\sigma_2\in\Sigma_2}\mathbb{P}_{\mathsf{G}}^{\sigma_1,\sigma_2}(s,F)$$

An optimal strategy for an MDP M in state s for reaching a target set F, is a strategy $\sigma^\star$ that maximising the probability of reaching the target:

$$\mathbb{P}_{\mathsf{M}}^{\sigma^\star}(s,F) = \sup_{\sigma\in\Sigma}\mathbb{P}_{\mathsf{M}}^{\sigma}(s,F)\,.$$

For both two-player TSGs and MDPs there exists memoryless deterministic optimal strategies. Efficient algorithms exist for generating such optimal strategies and the corresponding

optimal probabilities [20], [12] and have been implemented in the model checkers PRISM-games [21] and PRISM [13].

For a two-player game G and memoryless strategy σ_2 for player 2, we can construct an MDP $\mathsf{M}^{\mathsf{G}}_{\sigma_2}$ where the choices of player 2 are resolved according to σ_2. The optimal strategy of $\mathsf{M}^{\mathsf{G}}_{\sigma_2}$ is the most effective *counter* to the strategy σ_2, and therefore we refer to this optimal strategy as the *adversarial strategy* to σ_2 as it is only locally optimal.

We refer to the game elements available to players as material. Material is frequently used in game development as a way of enforcing diversity in the way in which games are played, where players choose their material at the start of a game. Examples of game material include characters, cars, weapons and football teams. We consider games as a TSG only after material has been chosen by all players. Often the choice of material is considered part of a strategy, but separating the two stages allows us to discount the effects of different orderings on when players choose material. For example, a two-player game where players choose material simultaneously is very different to the same game where they choose consecutively. In the latter case the second player will be able to act with knowledge of what their opponent has chosen, gaining an advantage. We justify this separation of material and strategy by considering all possible allocations of material amongst players.

B. Chained Strategy Generation

We use PRISM to generate *effective strategies* by finding the counter to a strategy that is popular in the metagame. CSG is the process of generating an effective strategy, then generating the adversarial strategy against it. This lets us reason that, when finding adversaries of adversarial strategies, we are always finding effective strategies – a strategy that is best against some previous effective strategy must itself be effective. By identifying a small subset of effective strategies from all of those available, we overcome one of the major challenges with automated game balancing, the need to compare all strategies to each other – a task which is often intractable. We can use CSG to identify two possible issues with a game's balance, a dominant strategy existing for any material or all strategies being dominated for some material.

Identifying Dominant Strategies. As with game balancing, what constitutes a dominant strategy is different depending on the genre of game. Generally a dominant strategy can be thought of as a strategy so effective as to be superior to all others. We define a dominant strategy as a strategy which is best played against by itself, i.e. a strategy which is its own adversarial strategy when all possible material is considered.

In order to identify dominant strategies quickly we exploit the fact that a dominant strategy must be optimal against a player with the same material. There can only be one optimal strategy for each matching of material to players for the type of games we consider. We can identify potential dominant strategies by only investigating strategies which are optimal against their own material, limiting the number to the size of the material. We can then test these candidate strategies against other material to see if they are indeed dominant.

Identifying Dominated Material. Dominated player material is that which an informed player will never use, because no effective strategy exists for it. In order to identify dominated material we generate a series of effective strategies and measure how effective individual material is against them. Using CSG we can compare adversarial strategies for all material against known effective strategies. If any material is significantly worse than all others against all effective strategies we claim it is dominated. Whilst not exhaustive, this approach will give a good representation of how particular material could be used in the best case.

Algorithm 1: Chained Strategy Generation

```
output: Returns the adversarial probabilities for all
        material at each iteration
probs := [][]
strats := []
k := 0                                              // iteration
/* Start with a randomly generated strategy          */
strats[k] := seed_strategy
while strats[k] ≠ strats[j] for all j < k do
    best_m := null                                  // best material
    best_probability := 0
    for m ∈ material do
        /* Find best opponent                        */
        calculate adversarial probability against
          strats[k]
        probs[m][k] := probability
        if probability > best_probability then
            update best_probability
            update best_m
        end
    end
    k++
    /* store new strategy                            */
    strats[k] := strategy for best_m against strats[k−1]
end
return probs
```

The methodology of CSG is shown in Algorithm 1. First we generate a seed strategy for some material, a strategy with an action chosen at random from those available at every occasion where the player has a choice to make. We then calculate the adversarial probabilities, i.e. the maximum probabilities of winning, for all material against the seed strategy. We generate the adversarial strategy for the material with the greatest adversarial probability and then calculate adversarial probabilities against the newly generated strategy. We continue in this manner, calculating probabilities and gen-

erating the strategy which performs best, until we generate a strategy that we have generated before – i.e. we found a cycle of strategies. The algorithm always terminates because there are a finite number of strategies and PRISM's strategy generation is deterministic. This happens when either a dominant strategy or a cycle of effective, non-dominant strategies is identified. At termination, the adversarial probabilities for all material at each iteration are returned, however further analysis is required to contextualise the results.

This approach generates a representation of the metagame consisting of a series of adversarial strategies calculated in turn. These strategies may be globally optimal, in which case the algorithm will terminate having identified a dominant strategy. If a dominant strategy is not identified then the effect of performing CSG is the calculation of a series of strategies representative of the evolution of the metagame over time when played by real players. We will not always find dominant strategies by iteratively calculating adversarial strategies, which is why we first search for them separately. It is not our intention to find dominant strategies during the iterative phase of our approach.

By comparing the material used by the strategies generated at each step, we can analyse the comparative effectiveness of each material independent of strategy. Material used to generate adversarial strategies that consistently perform worse than strategies for other materials is considered dominated. By comparing the probabilities of winning for all strategies per material, game designers can make value judgements on comparative strength across all material.

C. Statistical Analysis of CSG

CSG outputs a sequence of probabilities for all material in a game at every iteration of the algorithm. Analysis of these results gives a greater understanding of how fair and interesting the game is. We introduce a series of metrics for objective comparison between similar games. Our definition of an effective strategy is recursive: *an effective strategy is a strategy which performs best against another effective strategy*. Because of this, in our analysis we do not consider all iterations of CSG, starting instead at a lower-bound to allow the strategies time to settle. This is the base case for our recursive definition of effective strategies. The delay will need to be configured by game developers as games have varying complexity and the length of time taken before strategy generation settles will differ. The process of using a delay in this way is equivalent to waiting for players to familiarise themselves with a game before assuming they are effective players.

In order to describe the formulae used, we introduce the following notation:

- M is the set of playing material (of size $|M|$) and m denotes some material in M;
- k^* is the first iteration of CSG where a strategy is identified to be effective, i.e. the value for the delay before players are using effective strategies in a single execution of CSG (this is the base case for our definition of effective strategies);
- K and $K^\star = K - k^*$ are the number of total iterations and the number of iterations generating effective strategies by CSG, respectively;
- $winProb(m, k)$ is the maximum probability of winning for any strategy using material m at iteration k;
- $winProb(M, k)$ is the maximum probability of winning for any strategy using any material in M at iteration k.

Material Robustness. First we study how effective specific material is over an extended period of time. We refer to this as a measure of material robustness. We calculate material robustness by taking the mean of the maximum probabilities of winning against effective strategies for some material:

$$\rho(m) \stackrel{\text{def}}{=} \left(\sum_{k=k^*}^{K} winProb(m,k)\right) \Big/ K^\star$$

This gives a measure of how viable material is over time. It can be used to compare strength between different material to suggest redesign. If the robustness for material m is lower than 0.5, then m is too weak as it loses more often than it wins, even when employed with the strategies that maximise the probability of m winning.

Mean Robustness. The mean of material robustness over all material M gives a measure of how strong the effective strategies are when compared to the best ways of playing against them. We call this the mean robustness of M, denoted $\mu_\rho(M)$ and define it by:

$$\mu_\rho(M) \stackrel{\text{def}}{=} \left(\sum_{m \in M} \rho(m)\right) \Big/ |M|$$

A game with a high mean robustness is one in which players can always find multiple ways of successfully playing against effective strategies. A game with low mean robustness, i.e. one that is close to or below 0.5, would be one in which the effective strategies identified were overly powerful. For this reason we claim that a higher mean robustness indicates a more interesting game, as effective strategies do not limit the choices of the opponents as much.

Win Delta and Loss Delta. The results of CSG can be used to measure the variability of potential effectiveness for material against effective strategies. We do this by calculating the win and loss deltas for all material. The win delta of material m, denoted by $\delta^{win}(m)$, is the average probability with which material can beat effective strategies. The loss delta of material m, denoted by $\delta^{loss}(m)$, is the average minimal probability with which material will lose to effective strategies. Formally, for $result \in \{win, loss\}$:

$$\delta^{result}(m) \stackrel{\text{def}}{=} \left(\sum_{k=k^*}^{K} \delta^{result}(m,k)\right) \Big/ K^\star$$

where

$$\delta^{win}(m,k) = \begin{cases} winProb(m,k) - 0.5 & \text{if } winProb(m,k) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$\delta^{loss}(m,k) = \begin{cases} 0.5 - winProb(m,k) & \text{if } winProb(m,k) < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

These values allow us to measure effectiveness of a particular material m without considering situations where material m is very unsuited to playing against a given strategy. This is to be expected in a healthy metagame – some strategies are always effective against certain material. A *fair* game is one for which all material can win by similar, significant amounts. Any material with a win delta of 0 never wins against any effective strategy. We claim that in this instance the game is unbalanced as that material will never be used in high-level play. These values are measures of risk and reward and could be exploited by game developers for whom it would be desirable to have some material with low risk and low reward and others with high risk and high reward.

Outplay Potential. CSG provides an indication of the level of strategic depth in a game, showing how important the use of good strategies is to success. We call this a measure of outplay potential and calculate it as the mean of the difference between each maximum adversarial probability and the mean of the adversarial probabilities for all material once strategies have settled. More precisely $outplayPotential$ is defined as the mean of:

$$winProb(M,k) - \left(\textstyle\sum_{m \in M} winProb(m,k)\right) \big/ M$$

for $K \in \{k^*, \ldots, K\}$. A higher value of outplay potential suggests a greater spread of potential effectiveness between material. Outplay potential shows how important material choice is to the probability of winning, with higher values implying greater importance. This is a significant measure because it allows developers to gauge how important game knowledge is compared to strategic skill, i.e. knowing what material to use rather than what strategy. We argue that a game is more *interesting* if the maximum probability of success is highly dependent upon material choice, provided the material choice is *fair*.

III. Case Study

A. Description of RPGLite

RPGLite is a case-study we have developed as a role-playing game (RPG). It involves turn-based, stochastic combat such as battling in Pokémon or combat in Dungeons & Dragons. RPGLite is a two-player game in which each player chooses two different characters out of three available. The winner is the first player to reduce the health of both of their opponent's characters to 0 or less. The characters are: the *Knight*, who attacks a single opponent; the *Archer*, who attacks both opponents simultaneously; and the *Wizard*, who attacks a single opponent and attempts to *stun* them, preventing them from performing an action on their following turn. A coin is flipped to decide who goes first and play continues in a turn-based fashion. On their turn, a player chooses one action to perform from any of their alive characters that are not stunned and a target for that action from any alive opposing character. These actions result in a *hit* or a *miss*, with a probability determined by the character's attributes. The *Archer* is unique in that their action can *hit-twice*, *hit-one*, *hit-other* or *miss-both* rather than simply *hit* or *miss*. Each of the three characters have three attributes: *health*, *accuracy* and *damage*, which govern how much damage they can sustain before dying, how likely their actions are to *hit* and how much damage their actions inflict. Our purpose is to identify values for these attributes for each character that we know will make the game fair and interesting to play.

Variable	Range	Description
$attack$	$0, \ldots, 9$	Last action selected (0 – no action)
$turn$	$0, \ldots, 2$	Player turn indicator
$p1c1$	$-2, \ldots, 8$	player 1, character 1 health
$p1c2$	$-2, \ldots, 8$	player 1, character 2 health
$p2c1$	$-2, \ldots, 8$	player 2, character 1 health
$p2c2$	$-2, \ldots, 8$	player 2, character 2 health
$p1\_stun$	$0, \ldots, 2$	player 1 character stunned (0 – neither)
$p2\_stun$	$0, \ldots, 2$	player 2 character stunned (0 – neither)

TABLE I: RPGLite variable for an example configuration.

RPGLite is intended to be a simple game in order to allow us to clearly illustrate our approach without complex game mechanics. The game has an associated state space S. Every state $s \in S$ in RPGLite is a tuple of the form:

$$(attack, turn, p1c1, p1c2, p1\_stun, p2c1, p2c2, p2\_stun)$$

where each value is a realisation of a corresponding variable. All variables, with ranges, are described in Table I. The lower bound on health is calculated as 1 minus the maximum damage attribute of all characters.

The characters are intended to excel at different times during a play of the game. The *Archer* is meant to be able to do the most damage early in the game when both opposing characters are alive. For this reason we constrain the product of the Archer's *damage* and *accuracy* to greater than half of the product of the Knight's *damage* and *accuracy*. The *Wizard* is intended to be more powerful towards the end of a play of the game, when one of the opponent's characters is dead. When only a single opposing character is alive a *Wizard* could stun them repeatedly with a high probability, forcing an opponent to skip several turns.

The material choice for players in our case study is a pair of characters. We abbreviate the character pairs to initials, i.e. Knight-Archer is equivalent to KA. A full setup for a game is denoted similarly, e.g. KAvKW.

Config.	Character	Health	Accuracy	Damage
A	Knight	8	0.70	3
	Archer	7	0.80	2
	Wizard	7	0.75	2
B	Knight	8	0.70	3
	Archer	7	0.80	2
	Wizard	7	0.85	2
C	Knight	8	0.70	3
	Archer	6	0.80	2
	Wizard	7	0.85	2
D	Knight	9	0.70	3
	Archer	7	0.80	2
	Wizard	7	0.85	2
E	Knight	9	0.90	2
	Archer	6	0.60	2
	Wizard	8	0.60	2

TABLE II: Configurations for RPGLite.

B. CSG Results

We define several configurations for RPGLite which we analyse using the approach of Sect. II-B. The configurations are described in Table II. These configurations have been developed over a period of time to illustrate how CSG can be used during game development. Having started with a configuration with minor differences between the material, new configurations were created in response to the results gathered from analysis of previous configurations. A candidate configuration can be created and analysed for effectiveness. Our approach can also identify the material and general strategies that are weak, informing decisions about future configurations. We check every configuration for dominant strategies and search for dominated material six times, using a random seed strategy each time. We make sure that all material was selected for the seed strategy at least once in the six executions.

The results of our dominant strategy identification technique are shown in Table III. We obtained the results using a desktop machine running Ubuntu 18.10 with an Intel® Core™ $i7$–7700 CPU with 2×8GB of RAM. Identifying dominant strategies took between 5 and 20 minutes and required 550MB of memory, per configuration. The largest of our models has $816,480$ states and can be built by our machine in 4.38 seconds. The results of identifying dominated material are shown in Figs. 1–6. Each execution took between 10 and 90 minutes. Generating the models of RPGLite requires less than 30MB of memory and takes approximately 10 seconds. It takes fewer than 2 seconds to calculate the adversarial strategy from a model. The majority of this time is spent converting the files generated by PRISM into adversarial strategies. The code used to perform CSG on RPGLite is available at [22].

As an example of how we devised the configurations, we identified that there is a dominant strategy for KA in configuration A. To counteract this we decided to make the Wizard stronger by increasing Wizard accuracy in configuration B by 0.1, to 0.85. We wanted to change configurations minimally, to allow us to examine the effect of small changes on strategies employed at high-level competitive play.

Config.	Material choice	Opposing material		
		KA	KW	AW
A	KA	0.500	0.470	0.483
	KW	0.623	0.500	0.723
	AW	0.579	0.409	0.500
B	KA	0.500	0.594	0.546
	KW	0.559	0.500	0.617
	AW	0.528	0.449	0.500
C	KA	0.500	0.716	0.665
	KW	0.434	0.500	0.532
	AW	0.500	0.516	0.500
D	KA	0.500	0.625	0.350
	KW	0.472	0.500	0.526
	AW	0.716	0.526	0.500
E	KA	0.500	0.539	0.368
	KW	0.479	0.500	0.424
	AW	0.641	0.625	0.500

TABLE III: Comparison of adversarial probabilities against optimal strategies for the same material in all 5 configurations. Rows represent the optimal strategy for a pair when playing against itself, columns represent the adversarial probability for a pair against a strategy. Therefore a cell represents the probability of the column material winning against the optimal strategy for the row material. The two dominant strategies are highlighted (other material cannot beat them with probability greater than 0.5).

In the KA row for configuration A in Table III there is no value greater than 0.5, this shows that there is a dominant strategy for a Knight-Archer pair. A player adopting the optimal strategy for KA in KAvKA cannot be beaten with a probability greater than 0.5 by any strategy for any opposing material. In a real world example, players would eventually discover this strategy. A competitively motivated player aware of such a strategy has no motivation to use any other. For this reason we state that configuration A is uninteresting to play.

We can show with CSG that gameplay will tend towards a dominant strategy if one exists, as illustrated in Fig. 1. The adversarial probabilities reach 0.5 and then stay at 0.5 as the best way to play against the previous strategy is shown to be to play the same strategy. When a dominant strategy is present in a given configuration of a game, executions of CSG tend to show the strategies identified converging to it. Fig. 2 shows how every execution of CSG performed on configuration A converges to the same probability of

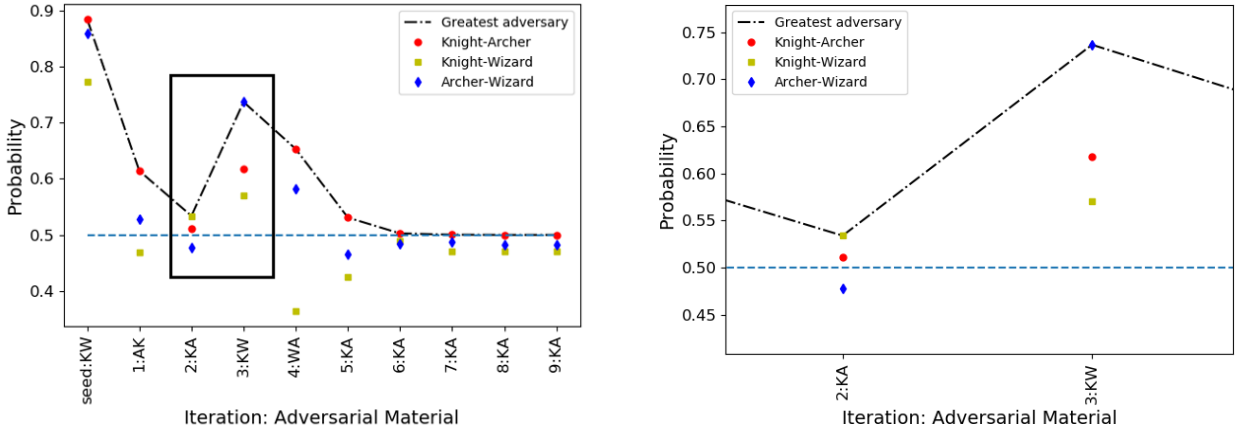

Fig. 1: Configuration A: CSG. (left) A single example where the strategy at iteration 8 is identical to the strategy at iteration 9. (right) A closer examination of the boxed area from (left). The points represent the maximum probability of winning against the previously identified best strategy using the material denoted. The blue diamond at the top of iteration 3 is the maximum probability a player using AW can win by against the KW strategy in iteration 2. Iteration 4 will show the maximum probabilities achievable against the AW strategy in iteration 3.

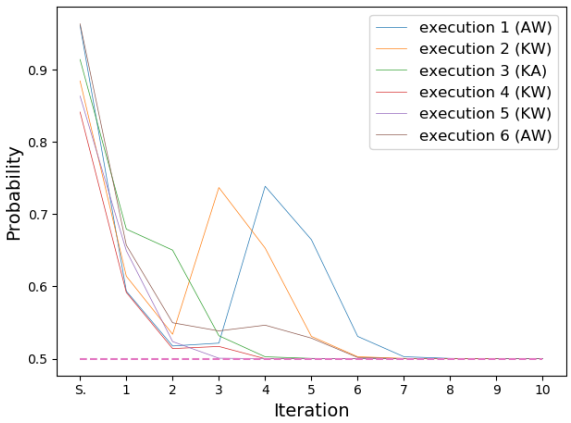

Fig. 2: Configuration A: six executions of CSG showing gameplay converging to a dominant strategy in each instance. The graph plots the highest probability of winning with any material against the previously identified adversary or seed. The seed material is displayed in brackets for each execution. The seed strategy is different for each execution.

0.5. By examining the strategies we can confirm that the same (dominant) strategy is identified in each instance. Our implementation of CSG stops when a strategy is identified which is identical to some strategy generated before. If a strategy is identical to the strategy generated immediately before it, then that strategy is dominant. A game developer would endeavour to develop a game for which a longer cycle of effective strategies exists.

Configuration B has a cycle of four effective strategies which are identified in every execution, as shown in Fig. 3 (left). Fig. 3 (right) shows that eventually the same cycle is identified in multiple executions of CSG, albeit out of step. In the cycle of strategies there is at least one strategy for each material. This shows that high-level competitive play of RPGLite using configuration B would at various points employ all material, which is something game developers would strive for. Once the strategies settle for configuration B (from iteration 3 and beyond in Fig. 3) there is a strategy for all material against every effective strategy identified which can win more often than it loses. This is shown by all values being above the blue line at probability 0.5.

Configurations C and D converge on a cycle of effective strategies and eventually identify the same cycles of strategies every time, as shown in Figs. 4 and 5. Configuration C has a cycle of length 8 whilst D has a cycle of length 6. Both cycles include strategies for all material, although unlike configuration B, some effective strategies cannot be beaten with a probability greater than 0.5 by any strategy for specific material. For example in Fig. 5 (left) the value for AW at iteration 17 is 0.331. This means that the most effective strategy using AW against the strategy identified for KA at iteration 16 wins less than a third of the time.

The results of searching for dominated material with configuration E are important for two reasons. First, there is a dominant strategy for KW in configuration E, as shown in Table III, but it is not identified in any of the six executions of CSG (unlike for configuration A). Second, two different cycles of effective strategies are identified, one of length 3 shown in Fig. 6 (left) and the other of length 16 shown in Fig. 6 (right). Despite this, the results show clearly that AW is dominated by the other material. Once the strategies have settled in both cycles, there are no strategies for AW that can win with a probability greater than 0.5 against any effective strategy. In fact, AW is the worst choice of material at every iteration once the strategies have settled, suggesting it would never be used during high-level play.

Statistical analysis of the results of CSG allow for clearer

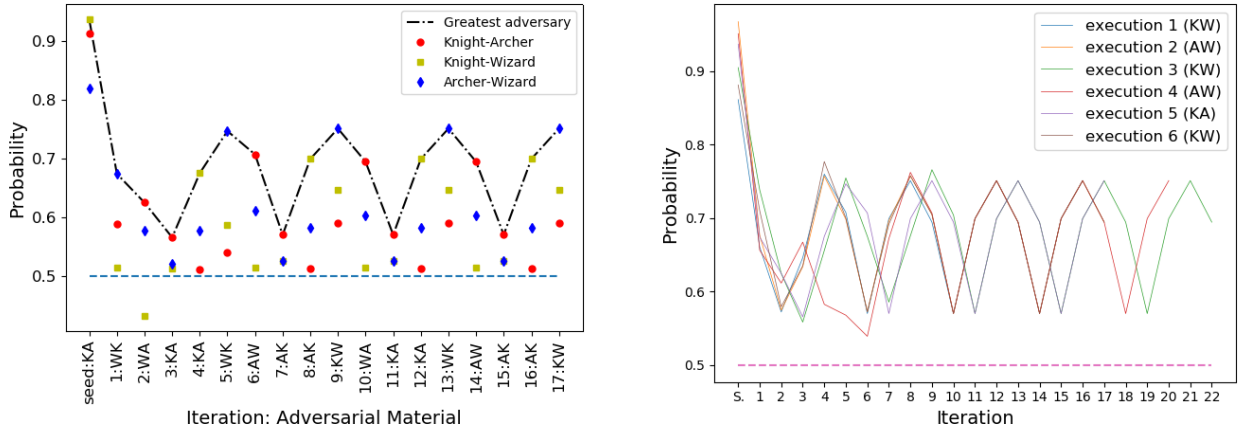

Fig. 3: Configuration B: CSG (left) and six executions of CSG (right).

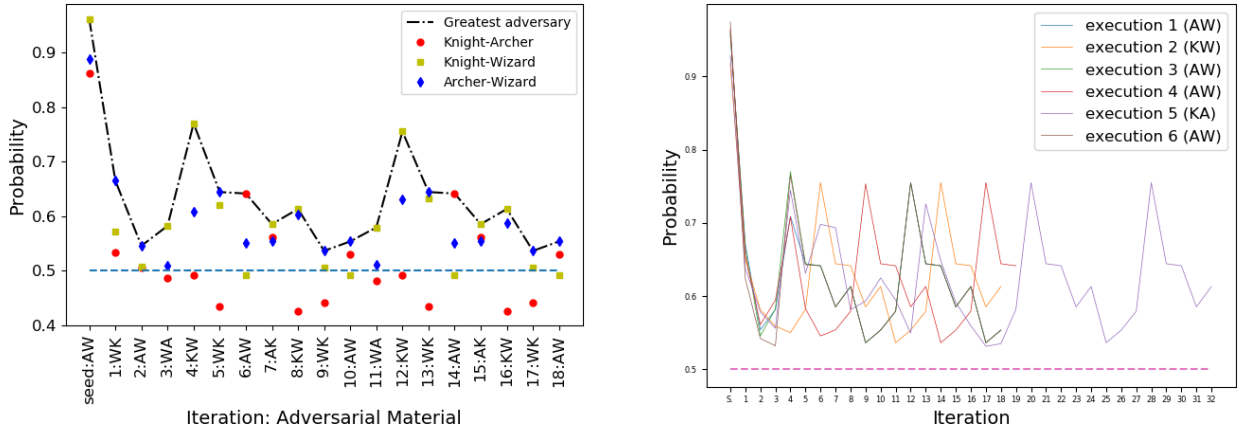

Fig. 4: Configuration C: CSG (left) and six executions of CSG (right).

Metric	Configurations				
	A	B	C	D	E
ρ(KA)	0.524	0.589	0.505	0.565	0.528
δ^{win}(KA)	0.024	0.089	0.029	0.071	0.034
δ^{loss}(KA)	0	0	0.024	0.006	0.006
ρ(KW)	0.472	0.589	0.582	0.592	0.543
δ^{win}(KW)	0.005	0.09	0.084	0.092	0.043
δ^{loss}(KW)	0.033	0.001	0.002	0	0
ρ(AW)	0.505	0.622	0.573	0.523	0.433
δ^{win}(AW)	0.019	0.122	0.073	0.074	0
δ^{loss}(AW)	0.014	0	0	0.051	0.066
outplayPotential	0.033	0.082	0.062	0.096	0.052
μ_ρ	0.5	0.6	0.553	0.56	0.502

TABLE IV: Statistical analysis for the five configurations considered based on the three material metrics (robustness, win delta and loss delta) and the two game configuration metrics (outplay potential and mean robustness).

comparisons of how strategies develop across different configurations. Table IV presents the measures described in Sect. II-C for the 5 configurations.

C. Analysis of RPGLite

Having studied the results of CSG performed on RPGLite we set k^*, the iteration of the first *effective strategy*, to 3. This value is used in calculating the measures of balance outlined in Sect. II-C. We claimed that for a game to be fair all material must be able to win by similar, significant amounts, therefore following CSG, configurations A and E can be regarded as unfair. The results for $\delta^{win}(KW)$ in A and $\delta^{win}(AW)$ in E (0.005 and 0.0 respectively) as well as the consistently low values for the other win deltas show this. Furthermore, the material and mean robustness values show that they are uninteresting to play too. For both configurations mean robustness is only slightly above 0.5 meaning the choice of winning strategies for players is limited. KW is dominated in configuration A and AW is dominated in configuration E, as shown by their low robustness values. Both configurations A and E can be discounted as unbalanced.

Analysis of configurations C and D show why it is important to consider the delta values as well as material robustness. Consider KA in configuration C and AW in configuration D, both have significantly lower values for robustness than the other material. However, we can use the

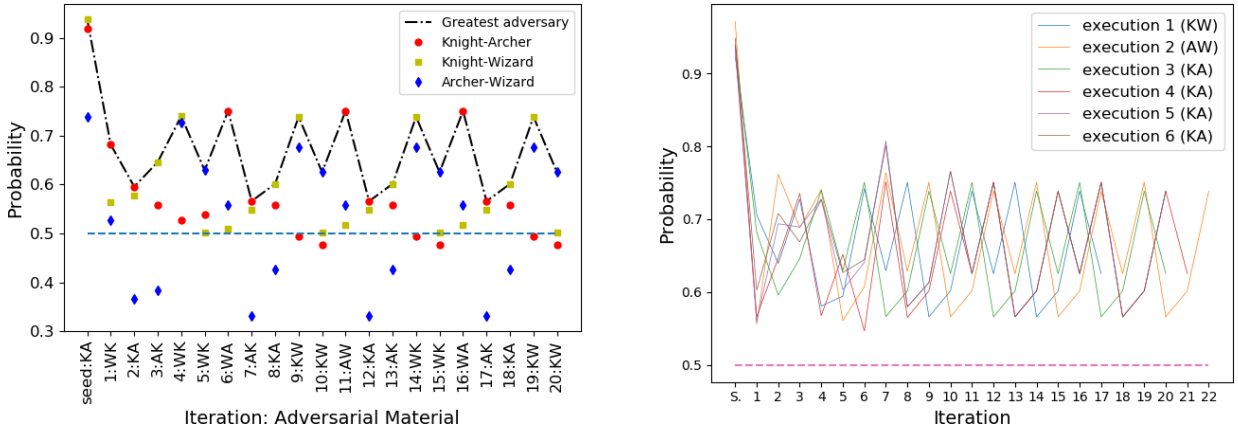

Fig. 5: Configuration D: CSG (left) and six executions of CSG (right).

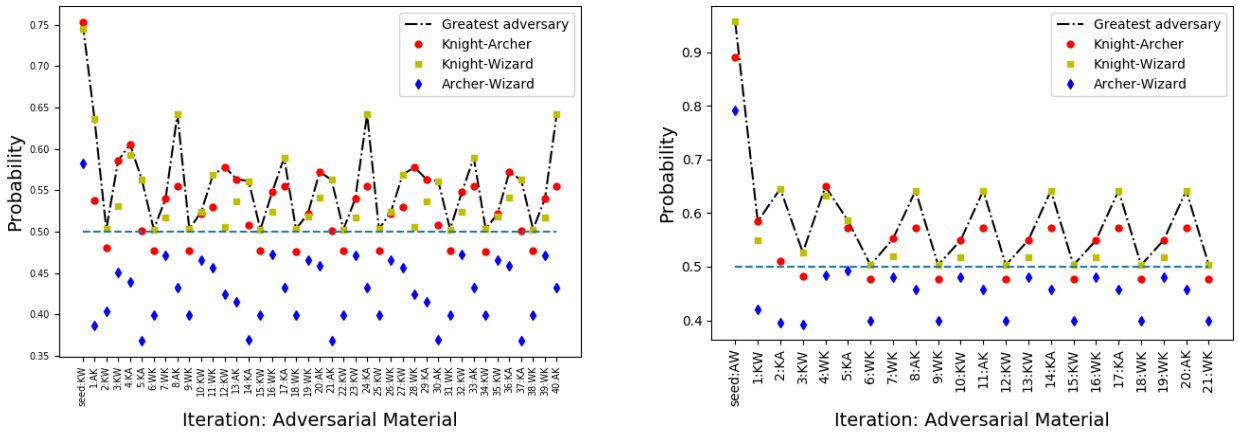

Fig. 6: Configuration E: CSG for two different seed strategies.

delta values to show that AW is viable in configuration D whilst KA is not viable in configuration C. Comparing the loss deltas with other material illustrates the risk associated with playing the material, both are far lower than the alternatives. The win deltas are more significant, 0.029 compared to 0.084 and 0.073 in C, suggesting that the risk outweighs the potential reward for playing KA. In D, the win delta for AW is 0.074 compared to 0.071 and 0.092, a far more justifiable risk to the player, given the potential reward. Although it may appear that AW is dominated in configuration D, the delta values show that it is highly viable, but only at certain times.

Having shown that configurations A and E are imbalanced and that KA is too weak in configuration C, we are left only with configurations B and D. B has the greatest μ robustness, therefore it could be argued that it is the most *interesting*, but D has greater outplay potential, suggesting it has a more varied metagame. Game developers who want to decide on a configuration based on these results could justifiably choose either configuration B or D as the optimal one depending on the type of game they wanted to make.

Configuration B would be a more player friendly game then D because it is always possible to win with any material against the effective strategies identified for B and the loss deltas are 0.0 or 0.001 meaning there is almost always a winning counter for all material against any effective strategy. Fig. 3 shows a clear cyclical hierarchy for configuration B where KA beats AW, AW beats KW and KW beats KA. Configuration D has the same hierarchy, Fig. 5 shows that this is less clearly defined than in B, as many optimal strategies are only slightly better than strategies using different material. The value for $\delta^{win}(AW)$ in B (0.122) suggests that AW is under powered compared to the other material, whereas the values for configuration D are less varied.

IV. Discussion

The results of CSG on our case study were not what we expected in a number of ways. The fact that every execution terminates in a cycle of strategies in a relatively short number of iterations was surprising given the number of strategies available. We assumed that the results would reflect those for identifying dominant strategies, or that we would not necessarily get more information than simply identifying the probability of winning calculated for all material playing optimally against all others. This was not the case – a great deal of extra context is given by generating strategies in turn. For example, simply calculating optimal values would give

no appreciation of the risk associated with using specific material or of the extent to which material is dominated. We were surprised by the dramatic effects caused by small changes in configuration. Some configurations were designed incrementally: C was created from B by reducing the health value of the archer from 7 to 6, while D was created from B by increasing the health value of the knight from 8 to 9. Yet the differences in the results between configurations is profound, demonstrating the value of our approach.

A key point to note about the cycle of effective strategies identified for configurations B, C and D is that they include at least one strategy using each material. This property alone is enough to show that a game is well developed as natural competitive play employs all of the choices offered to the players, which can only be positive for game developers. This is an example of orthogonally differentiated game material constituting an intransitive relationship at high-level play, the confirmation of which is one of the aims of this paper.

V. Conclusion

We have introduced new techniques for analysing game balance by simulating educated competitive play through CSG. We have outlined the technology used and the approach, and demonstrated its effectiveness through the use of a case study. We have shown how our technique offers game developers a deep insight into the interaction between material and lets them ensure that a game is *fair* and *interesting* to the players.

Future work includes scaling the approach to cope with the complexity of a more realistic game. This may require a purpose-built tool for performing CSG as the most costly process is generating files to describe strategies and inputting them back to the model checker. Improvements in this area could mean significant linear time improvement for CSG. We are also investigating the use of CSG on different forms of games, including concurrent games. CSG could be expanded to include automated analysis of results and reasoning about reconfiguration of parameters to perform the entire balancing process without human interaction.

Currently, in order to use our approach, a user would need expertise in model checking. In future we hope to hide this aspect by wrapping the entire modelling and analysis chain within self-contained software. CSG could potentially be the foundation of a very powerful tool for game analytics. This is only an initial investigation into what could be possible with CSG. More work is needed if it is to be used to by professional game designers, however the early results are encouraging.

Acknowledgment

This work is partially supported by the EPSRC Doctoral Training Partnership award EP/M508056/1 and the EPSRC Programme Grant *Science of Sensor Systems Software* EP/N007565.

References

[1] 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 *Proc. IEEE Conf. Computational Intelligence and Games (CIG'16)*, 2016, pp. 1–8.

[2] M. S. Debus, "Metagames: On the ontology of games outside of games," in *Proc. Int. Conf. Foundations of Digital Games (FDG'17)*. ACM, 2017, pp. 18:1–18:9.

[3] H. Smith, "Orthogonal unit differentiation," *Lecture notes from GDC*, 2003.

[4] E. Adams, *Fundamentals of Game Design*. Thousand Oaks, CA, USA: New Riders Publishing, 2014.

[5] M. Morosan and R. Poli, "Automated game balancing in Ms PacMan and StarCraft using evolutionary algorithms," in *Applications of Evolutionary Computation*. Springer, 2017, pp. 377–392.

[6] A. Jaffe, A. Miller, E. Andersen, Y. Liu, A. Karlin, and Z. Popovic, "Evaluating competitive game balance with restricted play," in *Proc. AAAI Conf. Artificial Intelligence and Interactive Digital Entertainment (AIIDE'12)*. AAAI Press, 2012, pp. 26–31.

[7] A. B. Cardona, A. W. Hansen, J. Togelius, and M. G. Friberger, "Open trumps, a data game," in *Proc. Int. Conf. Foundations of Digital Games (FDG'14)*. Society for the Advancement of the Science of Digital Games, 2014.

[8] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," in *Proc. Conf. Genetic and Evolutionary Computation Conference (GECCO'16)*. ACM, 2016, pp. 269–276.

[9] F. G. Glavin and M. G. Madden, "Skilled experience catalogue: A skill-balancing mechanism for non-player characters using reinforcement learning," *CoRR*, vol. abs/1806.07637, 2018. [Online]. Available: http://arxiv.org/abs/1806.07637

[10] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Automatic computer game balancing: a reinforcement learning approach." 01 2005, pp. 1111–1112.

[11] I. Althöfer, "Computer-aided game inventing," *Friedrich Schiller Universität, Jena, Germany, Tech. Rep*, 2003.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[13] M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. Int. Conf. Computer Aided Verification (CAV'11)*, vol. 6806. Springer, 2011, pp. 585–591.

[14] R. Giaquinta, R. Hoffmann, M. Ireland, A. Miller, and G. Norman, "Strategy synthesis for autonomous agents using prism," in *Proc. NASA Formal Methods Symp. (NFM'18)*, ser. LNCS, vol. 10811. Springer, 2018, pp. 220–236.

[15] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre, "Quantitative verification of implantable cardiac pacemakers over hybrid heart models," *Inf. Comput.*, vol. 236, pp. 87–101, 2014.

[16] S. Radomski and T. Neubacher, "Formal verification of selected game-logic specifications," in *Proc. EICS Workshop Engineering Interactive Computer Systems with SCXML*, 2015, pp. 30–34.

[17] R. Rezin, I. Afanasyev, M. Mazzara, and V. Rivera, "Model checking in multiplayer games development," in *Proc. Int. Conf. Advanced Information Networking and Applications*. IEEE, 2018, pp. 826–833.

[18] P. Milazzo, G. Pardini, D. Sestini, and P. Bove, "Case studies of application of probabilistic and statistical model checking in game design," in *Proc. Int. Workshop on Games and Software Engineering (GAS'15)*. IEEE, 2015, pp. 29–35.

[19] J. G. Kemeny, J. L. Snell, and A. W. Knapp, *Denumerable Markov Chains*. Springer, 1976.

[20] A. Condon, "On algorithms for simple stochastic games," *Advances in computational complexity theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 13, pp. 51–73, 1993.

[21] M. Kwiatkowska, D. Parker, and C. Wiltsche, "PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives," *STTT*, vol. 20, no. 2, pp. 195–210, 2018.

[22] https://github.com/WJLKavanagh/chained_strategy_generation/.