Modelling and evaluation of complex scenarios with the Strategy Game Description Language

Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis

Abstract— **The Strategy Game Description Game Language (SGDL) is intended to become a complete description of all aspects of strategy games, including rules, parameters, scenarios, maps, and unit types. Our aim is to be able to model a wide variety of strategy games, simple ones as well as complex commercially available titles. In our previous work [1] we introduced the basic concepts of modelling game rules in a tree structure and evaluating them through simulated playthrough. In this paper we present some additions to the language and discuss and compare three methods to evaluate the quality of a set of game rules in two different scenarios. We find that the proposed evaluation measures are complementary, and depend on the artificial agent used.**

I. Introduction

Game mechanics normally play a vital part in a game's success and appeal to players. In a time where almost every commercially produced game observes certain standards of (audio)visual quality, it is often the differences in game mechanics (in particular the originality) that differentiate the popular (and commercially successful) titles from the unpopular ones. It therefore stands to reason that a working algorithmic method for automating or assisting game rule design would be of immense use not only to game developers, but also to game researchers interested in exploring the roles and limits of game mechanics [2]. Methods for automatically generating various types of game content are developed and studied in the field of procedural content generation (PCG) [3], [4].

However, unlike the creation of other assets — levels, landscapes, weapons, textures etc. — the creation of game mechanics has only recently gained some attention from research within the procedural content generation (PCG) community. That this has not gained more attention in the past is somewhat surprising for us, as we believe that computer aided design of game rules or dynamic, even player tailored, may be a valuable and vital tool to help a game's success. It is an ongoing debate in the creative community debates if software can be really be creative or just perceived as such [5], but our position is that rule-generation systems will at least extend and augment a game designer's creativity. SGDL may therefore also used as a rapid prototyping system.

We propose to approach this problem from a search-based perspective. Within search-based PCG, the main challenges lie in representing the search space and in evaluating possible solutions [4]. Various representations for rules for different types of games been proposed in the past, as well as several measurements of quality. We will discuss some of them in this paper.

In our previous work we proposed the Strategy Game Description Language (SGDL), a solution to represent the properties and mechanisms of (computer) strategy games. We are aiming for a level of abstraction appropriate for this particular domain inasmuch as it allows us to represent a large variety of strategy games but few games outside this domain. We believe that our general approach within a specific genre of games may allow us to express such games in a succinct and hopefully readable manner, avoiding the verbosity that results from a too domain-general description language. To ensure that, we define strategy games as a genre that include a spatial map where one or more players own and take turns to move different units, and which might additionally include e.g. obstacles, resources and buildings. While the theming and visual representation may differ, the underlying games are similar, have often the same base mechanics. In this respect, the strategy games genre can be compared to classical board games such as Checkers, Go and Chess. In theory, SGDL will be able to express commercial available games like *Sid Meier's Civilization*. The main challenge would probably lie in modelling the meta-mechanics (such as the space race or research in general). For the lack of space, we did not include a formal definition of the language in this paper. We would like refer the reader to the project's homepage[1] for additional details instead.

In the following, we will first highlight other approaches to describing game rules, as well as some fitness measures for game mechanics that we believe may be useful. We will then briefly re-introduce the main idea and features of SGDL as well as the additions we have made in the current version. We will conclude this paper with the presentation of data and observations we've made when we tried the highlighted fitness functions on our extended game scenario using the SGDL framework. Particular attention will be given to how much the various fitness/evaluation functions we investigate correlate with each other, and across different agent types.

II. Related research

Before we discuss other research related to modelling and evaluating game mechanics, we would like to point out that we're always working on the premise that the ludologic system [6] can be decoupled from its audiovisual presentation. To what extent theming is necessary for an enjoyable game is an ongoing debate in the computer game

The authors are with the Center For Computer Games Research, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark (email: {tmah, juto, yannakakis}@itu.dk).

[1] http://game.itu.dk/sgdl/

design community and beyond the scope of this paper [7]. But seeing them as abstract games makes it reasonable to compare them to traditional combinatorial games (such as board games) as they are also *finite* and *discrete*. Computer games only differ from Browne's categorization (please see below) in such that they are not necessarily *deterministic* and sometimes involve the aspect of *hidden information*.

Since SGDL is meant to describe the rules of a game, it is natural to put it into the context of other Game Description Languages (GDL). The probably most well-known GDL was developed by Love et al. at the University of Stanford [8]. It is intended to be able to represent any game playable with and by a computer that is discrete and with complete information, and is built on first-order logic (to be more specific, it is a dialect of Datalog). The vast generality lead to vast verbosity and lacking readability; for example, three pages of text is needed to express a game as simple as *Tic Tac Toe*.

A few other, less domain-general approaches have been introduced in the recent past. Smith and Matheas [9] developed the *Variations forever* system which models game mechanisms as logical rules, expressed in AnsProlog. The use of AnsProlog makes it possible to use Answer Set Programming to automatically answer specific questions about the game design. Variations forever was primarily meant as a design support tool, but the system was developed into the *Ludocore* game engine together with Nelson [10].

In 2008 Togelius and Schmidhuber proposed a simple description structure for discrete PacMan-like games for use in their experiment in automatic game design. The experiment introduced the evaluation of the quality of generated game mechanics through learnability testing: a reinforcement learning algorithm is used as a model of human learning, and game mechanics with smooth learning curves (those that are neither trivial nor non-learnable) are preferred [2].

More recently, in 2010, Salge and Mahlmann [11] proposed the measurement of *relevant information* as a quality measurement, saying that a game is more refined if the minimum amount of features observed by players to achieve a certain performance is higher. Although it can be used as a method to evaluate arbitrary game mechanics, Salge and Mahlmann focus more on the quality evaluation then the modelling of the game rule space.

In 2008 Browne proposed his *Ludi* framework [12]. The genre in focus was two-player combinatorial games ("board games"). He also discussed several possible metrics to measure the aesthetics and quality of a game as a base for automatic evolution. One of Browne's fitness measurements that we will concern ourselves with here is based on the move- and lead history during the game. It records the players' performances (scores) throughout the game and counts how many times the leading player changes. According to Browne a game is more thrilling the more lead changes it has.

The tension (or uncertainty, the term is used interchangeably here) calculation on a per-turn basis was first proposed by Iida et al. in 2006 [13]. Like Browne, Iida et al. focused on two-player board games and proposed that the uncertainty of the outcome (player one wins, player two wins or stalemate) may be a good approximation of the abstract quality of a game mechanic: a game becomes uninteresting once its winner is obviously determined. Therefore a fast decrement of the uncertainty should only occur in a late phase of the game. The uncertainty of a game's outcome can be calculated using Shannon's famous definition of entropy as commonly used in information theory [14]: $-\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$ whereas $p(x_i)$ is the probability of the outcome x_i. For two player games only two outcomes are possible (three if the game rules permit a draw). The proposed method to approximate the entropy per step is to record the outcome of a certain number of self-play games. However, depending on the computational complexity of the game, this may lead to problems generating enough data to calculate the entropy (Iida et al. proposed 4,000 games per turn).

The initial case study, using the game "Hex" [13], created the impression that the uncertainty should be monotonically decreasing throughout the game since every step reveals information. This can be falsified by taking into account that the winner of the game is not determined right at the start of the game. If it were, the game could only point it out more in every game turn. In fact, players can make moves that increase their chances of winning. Hence the outcome uncertainty can increase throughout the game, as we will present later in this paper.

III. The Strategy game description language

Before we reveal what additional features we added to SGDL, we will briefly re-introduce the main elements of the description language and what games it may apply to.

We define "strategy game" as a battle of war seen from the general's perspective: the player looks down on the field of battle, deciding where their units and buildings should be deployed. Each player has his unique goals, often including the elimination of all other players' assets. Since the player observe the game from a bird's eye view, he is removed from the immediate action and often only interacts with the game world through his units and buildings.

Strategy games can be decomposed in three different layers:

1) The *mechanics* layer. This layer determines the fundamental rules of the game, such as what an attack action is, what it does mean to win the game and in what type of game environment units are placed on (e.g. on a 2D grid).
2) The *ontology* layer. This layer specifies the types of key elements that may exist in the game (e.g. rivers, mountains, tanks and factories) as well as their properties (e.g. mountains have movement cost 5 for ground units).
3) The *instance* layer. The setup of an individual match, campaign or battle are specified within this layer: the layout of the map, initial placement of units, and any particular conditions that might apply (e.g. the fog of

war[2] gets lifted after 50 turns and the battle is lost if enemy survives after 100 turns).

We propose to describe at least the mechanics and ontology layers using a tree-based representation, similar to the most common representations used in genetic programming (GP) [15]. GP is an evolutionary computation technique for evolving program code. In most applications of GP, expression trees are used to model and evolve programs that fulfil a pre-defined task. The core idea is to model atomic instructions into tree nodes whereas child nodes serve as parameters for their parents. One common application of GP trees are mathematical equations where numerical values are passed as literals into the leaf nodes. Whenever we refer to *nodes* in this paper, we mean nodes in an SGDL tree.

We started out attempts to model strategy games with the mechanics layer, focussing on detailed modelling of the *units*. In most strategy games, units are the most important game elements the player interacts with, and can be compared to the pieces of a board game such as Chess. Furthermore, the challenge of the game is increased since units usually belong to different classes which provides them with dissimilar abilities and properties. A game often uses its theme to make it more intuitive what the differences are, e.g. tanks and airplanes are intuitively seen as objects with different properties. Units (or objects on the map in general) have properties (called *attributes* in the following) and abilities (called *actions* in the following). While having some basic attributes are mandatory (like spatial positioning), having actions is not: debris, lying scattered on the map, may serve the game as obstacles for other units while having no power to act themselves.

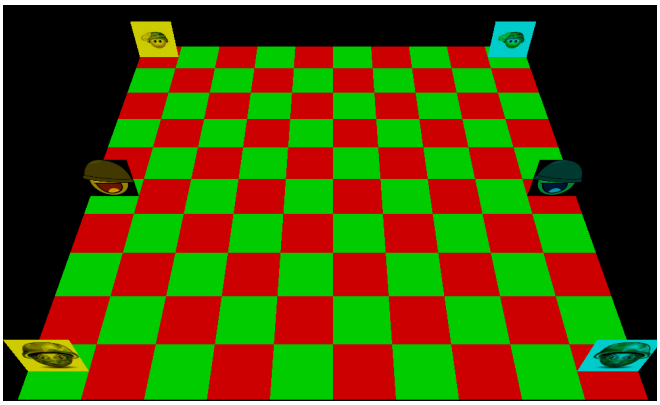
To give you an example, we briefly discuss the SGDL fragment of a unit class. In Figure 1 we can see the unit type "A" that has two attributes in its right branch of the SGDL tree: *x* and *y*. Without any further reference, these variables have no explicit semantics. Only the action "goNorth" in the left branch of the tree assigns meaning to them. Once an instance of that class (an actual unit on the battlefield, identified as *_OBJECT(0)* in the tree) invokes that action, and its condition that there must not be another unit at the map tile north of the unit is fulfilled, its *y* attribute is decreased by one. Naturally, an application that would like to visualize the current game state has to interpret *x* and *y* as coordinates on a map to calculate the position of an object's representation in the visual space.

A. Additions in this version

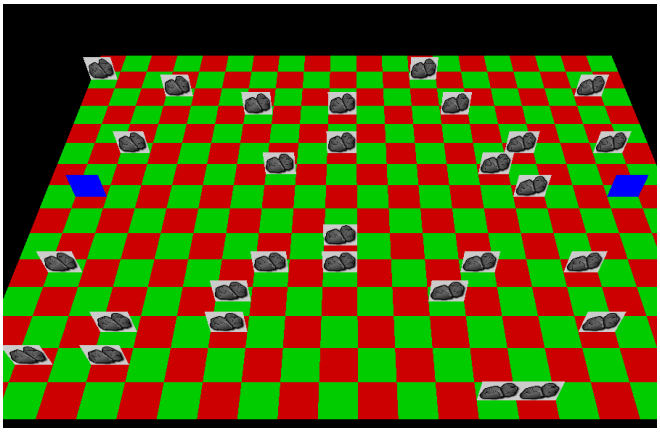
To draw nearer to our goal to model complete strategy games in SGDL without the need for any hard-coded game rules, we made several additions to the description language:

- non-deterministic nodes that represent a random number generated within certain parameters each time the value is queried. These can be used within conditions to model probabilities or consequences to create non-deterministic effects.
- the ability to create new objects on the map. This is an essential mechanic that most games require. Even if the game does not permit the creation of new units, creation nodes can be used to model temporary objects e.g. projectiles that exists over several game turns.
- we divided the set of consequences of an action into several indexed layers. This allows us to script sequential consequences without having to cascade a huge tree of actions
- both the global game state and the players got their own set of attributes and actions. That enables us to integrate more functionality of the ontology layer into SGDL.

[2]fog of war: only a portion of the map is visible, depending on the players' units' positions



(a) Complex Rock Paper Scissor



(b) Rock Wars

Fig. 2. Two exemplary screenshots from our different scenarios.

IV. Game scenarios

We test our proposed fitness in two different scenarios in this paper; one is relatively simple and has been tested in a previous paper of ours, the other is more complex and relies on the newly introduced SGDL features.

A. Complex Rock Paper Scissors (CRPS)

For our previous work [1] we designed a very simple strategy game for our studies. The game takes place on a 10×10 quadratic map. The main goal of the previous

Fig. 1. An SGDL fragment that defines a units ability to move north on the map

experiment was to find complementary unit sets, where one unit beats another but is beaten by a third, and where it is advantageous to use diverse sets of units rather than homogeneous armies. As the classic children's game "rock, paper, scissors" is perhaps the most complementary game ever devised (rock beats scissor beats paper beats rock) we will refer to this scenario throughout the paper as "Complex Rock Paper Scissor" (CRPS). For a better understanding of the following game description we provided a screenshot using our development version of our game engine in figure 2(a).

Each player has three units, and those start evenly spaced out on opposite sides of the map. Each turn, one player can move or attack with one of his units. A unit can move one step north, south, east, west, or attack one of the enemy units. Units cannot move outside the map or attack a unit which is not within their range. Each unit type has seven attributes: health (range [0, 100]), ammunition ([0, 100]), three attack values ([0, 100]) and both maximum and minimum attack range ([0, 6]). The attack values determine the damage that can be done by one shot on each of the enemy unit types.

For CRPS we sampled a number of ca. 7000 uniformly generated games using a heuristic we created for our extended scenario (please refer to section V-B and ca. 900 games using our Monte Carlo Agent. The differing number of samples was simply caused by the different computation costs.

B. Rock Wars (RW)

On major flaw in our previous experiment was the limited complexity of the game mechanics, apparently disallowing deep gameplay and high-level skill differentiation. To increase the complexity of the game and make it more appealing to human players, we increased the map size to 20x30 and randomly placed impassable *rock* objects on 10% of the map tiles. Rocks could be of course any other object that would fit the theming, but we will refer to our new scenario in this paper as "Rock Wars" (RW) for the sake of clarity. Strictly speaking, we would need to check if the non-deterministic creation of the scenario does not create unplayable games (even if the configuration is valid) e.g. through creating impassable barriers, but in practice non-playable maps have never been observed. For a better understanding of the game mechanics please refer to the screenshot in figure 2(b).

Instead of a pre-assigned army, players start with single *factory* object (blue in the screenshot) that is able to spawn new units. Each unit type has an additional unique *cost* attribute, the value of which is subtracted from the acting player's global amount of resources (called money here but could be anything else e.g. ore or crystal like seen in popular commercial games). The money variable is part of the newly introduced player attributes. The goal of the game is extended with the rule, that a player will loose if he has no more units (excluding the factory, which is indestructible) and no more money left to buy a new unit. If this rule is triggered, the other player will win. The condition for the maximum number of turns was left unaltered at 100 turns before a game is considered as a draw. One might add that it would have been simple to give each player a certain amount of money back, to simulate an economy, but our test showed that if a game SGDL model (if generated or mutated with unfavourable attributes) may allow never ending games where players can always buy new units. These cases would still be detectable through our fitness function but would be very costly to evaluate since they would always hit the maximum turn limit.

Another important difference between the CRPS and RW scenarios is the move order. In CRPS, the player can only move one unit each turn, whereas in RW all units can be moved each turn. This makes RW more similar to mainstream computer strategy games, whereas CRPS has the move order of classic board games like chess.

The move restriction in CRPS was due to our initial approach to let agents use the MiniMax game tree search algorithm with Alpha-Beta pruning. The increased number of actions per move available in RW increased the branching factor of the game tree exponentially to x^{y^z}, whereas x is the number of possible actions per unit, y is the number of units they own and z is the search depth. This complexity made it very hard to run many experiments in a realistic time frame, using the MiniMax algorithm, and we therefore had to devise new agents capable of handling this complexity.

V. Game-playing Agents

All of our fitness functions rely on automatic playthrough, and we therefore need agents capable of playing the devised strategy game scenarios with some skill and preferably without too high computational complexity. As we will evaluate a wide variety of game configurations, the agents cannot be specifically tailored to any particular configuration.

A. MCTS Agent

Our first agent is based on Monte-Carlo Tree Search (MCTS), as this is a high-performing game-playing method for many board games, and currently the world-leading algorithm for computer Go [16]. The basic Monte Carlo decision algorithm works by, for each possible action each turn, making a number of random playthroughs until the end of the game and choosing the action that leads to the best outcome on average. MCTS improves on this by building up a tree of approximated state values as it carries out simulations, in order to reduce the need for evaluating less promising move sequences.

Still, the MCTS agent is very complex, and as the branching factor increases (more actions are available to the player) ever more simulations are necessary to achieve reasonable performance. The playing style of this agent can be described

as quite cautious, as it will usually try to avoid fights which it is not likely to win.

Our initial approach was to repeat the evolutionary (to find complementary unit configurations) run with our new RW game, but it proved to be impractical to sample a multiplicity of games in reasonable time due to the time it took our agent to play the game.

To see how the increased complexity affects the fitness values, and to be able to put CRPS and RW into relation, we therefore limited our experiment to a number of 100 configurations based on the three configurations previously found, only varying the new parameters (start money and cost value of each of the three classes) with ranges 0-10 for the unit costs and 0-100 for the start money, and 600 uniformly samples games using our heuristic (please refer to section V-B for details).

B. Heuristic Agent

We also devised a purely heuristic agent, that plays very aggressively but with some rudimentary target selection. The very low computational complexity of this agent allows us to run experiments with complex scenarios in manageable time. It also allows us to test the effect of a very different playing style on the fitness functions, as the heuristic agent has a markedly different "temperament" to the MCTS agent. The heuristic can be described in pseudocode thus:

for all units **do**
  if unit can attack **then**
    attack where most damage caused (the unit against which the current unit is most effective)
  else if next step towards nearest enemy is not blocked **then**
    move towards nearest enemy
  else
    do a random move
  end if
end for

VI. Fitness functions

A. Balance

The aim of this fitness function is to find a suitable configuration (here: sets of attribute values for attack, health etc. In RW this also includes the unit cost variables) of the three classes that fulfilled our definition of "balanced" unit type set. A balanced, or complementary, set is one where the nontransitive power relations between units make it advantageous to compose or armies of diverse units rather than homogeneous ones. The balance of a unit set was measured as followed: six battles were played for each unit type set. Balanced unit sets with units of all three types (denoted ABC) played against unbalanced sets with units of only one type (AAA, BBB and CCC). In RW this denotes the unit types that a player could build through the factory object.

Three games where played where the balanced unit set started the game, and three games where the unbalanced set started. The fitness was defined as the minimum fitness achieved by the balanced set in any of the six games. To minimize noise, the fitness calculation was averaged over 200 trials (t). The configurations found by our evolutionary algorithm from our previous work [1] can be seen in tables I, II and III.

B. Tension

The second fitness we used was the self-play method of evaluating tension, as proposed by Cincotti and Iida [13]. As we have already briefly introduced it in section II, it is based on the computational intelligence definition of quantifiable *information* [14]. The more information is revealed about the run of play, the clearer the outcome of the game becomes. The later the decision point (the last lead change) of who will win the game occurs, the better. It is therefore desirable that the uncertainty of the game outcome stays high until a very late phase of the game.

To calculate the outcome uncertainty throughout the game, we used Iida's proposed method with 1000 roll-outs per turn to approximate the outcome probability. We then used the graph (normalized over the game length, ranging from 0.0 to 1.0 in both dimensions) to create a fitting of the actual uncertainty graph with the least-squares method, increasing the degree of the curve until the standard error was below 0.01. Following the stipulation that the final drop of uncertainty (final game phase) should occur as late in the game as possible, we numerically computed the distance d of the nearest point on the curve close to the point $1.0, 1.0$ (the maximum uncertainty, game length). Since we try to minimize this distance, the resulting fitness function is $1-d$.

C. Lead changes

Another thing that should correlate with tension is *thrill*. Following Browne [12], a thrilling game requires that the leading player changes several times throughout the game. The worst case is that the starting player always wins, if he just plays a perfect game (*Tic Tac Toe* is a well known example). Hence we consider a game more thrilling, the higher the average number of lead changes is.

To determine the number of lead changes for a certain playthrough of a two-player game, each player's score is

	Health	Ammo	Attack 1	Attack 2	Attack 3	Min range	Range
A	53.0	33.0	60.0	20.0	92.0	10.0	0.0
B	82.0	78.0	85.0	60.0	62.0	0.0	23.0
C	39.0	45.0	37.0	100.0	12.0	0.0	0.0

TABLE I

A unit type set with balance fitness 0.0 in the CRPS scenario.

	Health	Ammo	Attack 1	Attack 2	Attack 3	Min range	Range
A	46.0	69.0	61.0	71.0	71.0	2.0	5.0
B	6.0	43.0	22.0	90.0	22.0	3.0	5.0
C	36.0	82.0	40.0	47.0	6.0	2.0	4.0

TABLE II

A unit type set with balance fitness 0.24 in the CRPS scenario.

	Health	Ammo	Attack 1	Attack 2	Attack 3	Min range	Range
A	6.0	82.0	39.0	2.0	67.0	0.0	3.0
B	4.0	31.0	92.0	79.0	3.0	1.0	5.0
C	64.0	79.0	94.0	1.0	90.0	0.0	2.0

TABLE III

A UNIT TYPE SET WITH BALANCE FITNESS 0.57 IN THE CRPS SCENARIO.

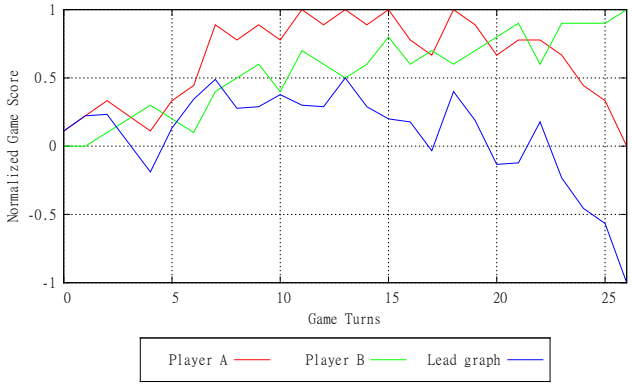


Fig. 3. The normalized scores (red and green) for two players of an exemplary game. The lead graph (blue) indicates the current leading player.

(a)

	Balancing	Lead changes	Turns
Uncertainty	0.04	−0.08	−0.12
Balancing		0.01	−0.1
Lead changes			−0.17

(b)

	Balancing	Lead changes	Turns
Uncertainty	0.17	0.09	−0.16
Balancing		0.19	−0.36
Lead changes			−0.53

TABLE IV

THE CORRELATION BETWEEN DIFFERENT PROPERTIES OF SAMPLED GAMES OF CRPS. 7000 GAMES WERE SAMPLED USING THE HEURISTIC (A) AND 900 USING THE MCTS AGENT (B). SIGNIFICANT VALUES WITH GRAY BACKGROUND.

	Balancing	Lead changes	Turns
Uncertainty	0.01	0.22	0.03
Balancing		0.02	−0.03
Lead changes			−0.19

TABLE V

THE CORRELATION BETWEEN DIFFERENT PROPERTIES OF CA. 675 SAMPLED GAMES OF *Rock Wars* USING THE HEURISTIC. SIGNIFICANT VALUES WITH GRAY BACKGROUND.

recorded in each game turn. After the game has ended, the score is normalized between 0 and 1, whereas 1 represents the maximal score recorded. A third graph (the lead graph) is created representing the difference between the two players' scores. Given that the two score graphs range between 0 and 1, the lead graph can only range between -1 and 1. Is the first player leading the game, the lead graph will have a positive data point in that turn, a negative data point if the second player leads. The example in figure 3 illustrates this more clearly: red and green are each player's performance graph normalized to the maximum score. Blue represents the difference between the score graphs. Every time the sign of the lead graph changes, the leading player changes.

For all our games the players' scores were recorded for every game turn. As a score heuristic we used the health of all units on the battlefield:

$$S_p = \sum_{u_p}^{U_p} health_{u_p} - \sum_{u_o}^{U_o} health_{u_o}$$

Where the score S_p of the player p is defined as the difference between the sum of health of all (U_p) his units and the sum of health of all (U_o) his enemy's units. The actual fitness of the game is the number of lead changes divided by the number of game turns (the average number of lead changes).

VII. RESULTS

In this section we present how we applied the previously introduced fitness measures to our two game scenarios (CRPS and RW) and how they correlate to each other. We conclude this section with a detailed analysis of the fitness recordings on three RW configurations that base on the configurations previously found in the evolutionary run and presented in tables I, II and III.

A number of experiments were made, where hundreds of games were played using both scenarios and agents. For each combination of scenario and agent, all three different fitness measures were calculated, and the correlation between the different fitness measures calculated. We tried to aggregate as much data as possible for each experiment in reasonable time, whereas different computational costs lead to a different number of simulation results for each agent/game combination: 7000 for *Complex Rock Paper Scissors* using the heuristic and 900 using the Monte-Carlo agent. For *Rock Wars* we sample 675 games using our heuristic. Unfortunately we were not able to simulate a significant number of *Rock Wars* games using our Monte-Carlo agent.

Looking at the data presented in tables VII and V we observe several interesting correlations. Some of them can be discarded as non-significant. Correlations are given as the Pearson correlation coefficient (r), while their significance was tested against the null hypothesis ($> 95\%$ confidence):

$$t = \sqrt{\frac{n-2}{1-r^2}}$$

whereas r is the correlation coefficient and n the number of games played.

Although these are rather small, we would like to focus on three details here: A) the correlation between the game lengths and the number of lead changes shows a significant correlation in both games. This is rather expected as we assume that the rate of lead changes is a constant property

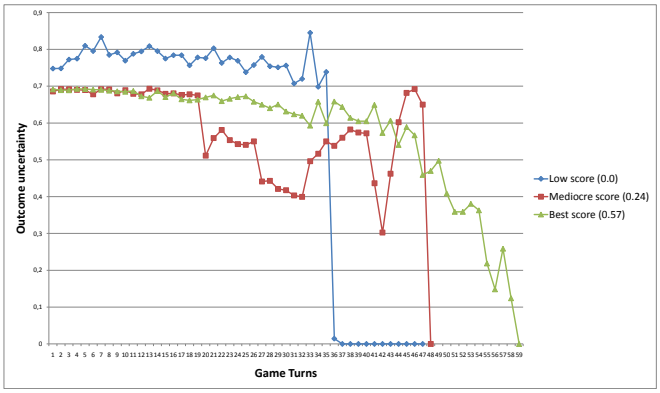


Fig. 4. The outcome uncertainty in RW. The configurations were based on our CRPS configurations found in our evolutionary experiment.

of the game mechanics. It is only natural, that the total percentage of lead changes decreases with an increasing game length. Therefore a negative correlation can be observed. B) There is a weak correlation between the uncertainty measurement and the number of lead changes in "RockWars", while in "CRPS" there is not. While a correlation between tension throughout the game is to be expected to depend on the number of lead changes, we believe that we don't see any correlation in CRPS as it is basically to simplistic to be able to create tension and there are not many lead changes due to the short game length. We discard the negative correlations of -0.08 (Heuristic) and 0.09 (MCTS) as non-significant. C) The simple CRPS scenario shows a significant negative correlation between the uncertainty and the game's length (-0.12/-0.16). We assume that a game is either over very fast, or it is in a non-functional configuration where the game end is only prolonged due to the fact that no party can win (e.g. only having units with a range of zero left).

While the values for the Lead changes and uncertainty behave very similar between the two agents for CRPS, we observe several differences in the other correlations, especially in those that base on the game length. We believe that this is connected to the increased average game length: 20 turns (heuristic) versus 60 turns (MCTS). The MCTS agent is certainly more versatile than the heuristic, which is hand-crafted to work with the current scenarios, but has a markedly different and more cautious playing style. Note that there is no straightforward relation to playing style.

The sampling of RW configurations based on our previous findings proved to be interesting as well: figure 4 shows an example of the development of the outcome uncertainty for our three different configurations as described in section VI. We see that each configuration generates its own unique graph. All three graphs show a very stable first part until the first shots are fired. Graph A) shows that the game is decided immediately once a shot is fired, hence the uncertainty drops to the base level of 0. That there are several turns more needed to end the game could have one of several explanations: either the game requires that the remaining units have to be simply cleared from the battle field to trigger the win condition (e.g. in the first Command & Conquer [17] game a player had to clear all opponent's passive structures like walls and sandbags to win a multi-player match. Even if the opponent was unable to fight back at all since he had already lost his production facilities and units.) or that there exists an obvious action (*killer-* or *finishing-move*) that an agent has to take but due to its non-deterministic nature does not pick. B) shows a slight decrease of the uncertainty in the beginning until the graphs starts oscillating heavily. While we consider A) merely a pathological case (since the configuration is basically non-function since two unit types have a range of zero) and B) simply a "bad" configuration, does C) show a graph we would suspect from a well working configuration. What is surprising here is, that the fittest configuration from our previous experiment shows the slowest decrease of uncertainty in this experiment. Following our measurement of the smallest distance to the point $1.0, 1.0$, this fitness is rather low (0.5) compared to the mediocre (0.38) and bad configurations (0.46).

VIII. Conclusion and Future Work

In this paper we described a number of additions to the Strategy Game Description Language that permit the modelling of more complex game mechanics and scenarios. We also applied three different fitness measurements to two scenarios, one simplistic and another inspired by complex turn-based strategy games. The following fitness measures were used:

- Our definition of balancing, calculating the win-rate of heterogeneous unit sets versus homogeneous ones.
- Iida's measurement of tension using the outcome uncertainty per turn
- Browne's proposed calculation of lead changes per game.

We observed different properties using a large number of uniformly randomly generated unit type configurations of our two game scenarios using different agent controllers. We also analysed the fitness of three hand-picked configurations in detail. We found that certain fitness measurements behave different in the two scenarios while others correlate between scenarios. Our assumption is, that these three fitness measurements are part of an overall approximation that can determine the quality of a game mechanic and therefore its appeal to human players.

We have also observed that the type of agent used has an impact on the fitness measures, as the fitness measures differ considerable for the same scenario depending on whether a heuristic or MCTS agent was used. This leads us to the question what sort of agent should be used to give a qualified measurement of the properties of a game mechanics that can be transferred to human player preferences. Intuitively, it seems desirable to use an agent that plays the game in a human-like manner and with human-level performance. However, in which aspects the agent should be human-like is an open research topic, as is the more fundamental topic of how to play an unknown strategy game scenario and unit

type configuration reasonably well. We are currently putting considerable effort into investigating this.

There are a number of other conceivable fitness measures for strategy games that could and should be investigate. Especially Browne [12] offers several fitness measures for board games that could relatively easily be adapted to work with strategy games. It has yet to be investigate to what extent our fitness measurements correlate with the preferences of human players. It would also be interesting to investigate what type of players a game will appeal to. Combining our approach with player modelling is therefore an immediate goal of this research project [3], [18]. Once a valid and reliable predictors of player preference have been established, these models can be used as fitness functions for evolving new strategy games.

ACKNOWLEDGEMENTS

This work was supported in part the Danish Research Agency (FTP) grant "AGameComIn". Thanks to Christoph Salge for insightful discussions.

REFERENCES

[1] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Towards procedural strategy game generation: Evolving complementary unit types," in *Applications of Evolutionary Computation EvoApplications 2011*, April 2011.

[2] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2008.

[3] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. in press, 2011.

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of EvoApplications*, vol. 6024. Springer LNCS, 2010.

[5] S. Colton, "Creativity versus the perception of creativity in computational systems," in *In Proceedings of the AAAI Spring Symp. on Creative Intelligent Systems*, 2008.

[6] J. Juul, "The game, the player, the world: looking for a heart of gameness," in *DIGRA Conf.*, 2003.

[7] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. MIT Press, 2004.

[8] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General Game Playing: Game Description Language Specification," 2008.

[9] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[10] A. M. Smith, M. J. Nelson, and M. Mateas, "Ludocore: A logical game engine for modeling videogames," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[11] C. Salge and T. Mahlmann, "Relevant information as a formalised approach to evaluate game mechanics," *Proc. IEEE Conference on Computational Intelligence and Games (CIG) 2010*, August 2010.

[12] C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland University of Technology, 2008.

[13] A. Cincotti and H. Iida, "Outcome uncertainty and interestedness in game-playing: A case study using synchronized hex," *New Mathematics and Natural Computation (NMNC)*, vol. 2, pp. 173–181, 07 2006.

[14] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, January 2001. [Online]. Available: http://doi.acm.org/10.1145/584091.584093

[15] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008, (With contributions by J. R. Koza). [Online]. Available: http://www.gp-field-guide.org.uk

[16] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *Proceedings of the Belgian-Dutch Artificial Intelligence Conference*, 2008.

[17] Wikipedia, "Command & conquer — wikipedia, the free encyclopedia," 2011, [Online; accessed 27-March-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Command_\%26_Conquer\&\\oldid=420557245

[18] G. Yannakakis and M. Maragoudakis, "Player modeling impact on player's entertainment in computer games," in *User Modeling 2005*, ser. Lecture Notes in Computer Science, L. Ardissono, P. Brna, and A. Mitrovic, Eds. Springer Berlin / Heidelberg, 2005, vol. 3538, pp. 151–151.