Evolving Card Sets Towards Balancing Dominion

Tobias Mahlmann, Julian Togelius and Georgios N. Yannakakis

***Abstract*—In this paper we use the popular card game *Dominion* as a complex test-bed for the generation of interesting and balanced game rules. Dominion is a trading-card-like game where each card type represents a different game mechanic. Each playthrough only features ten different cards, the selection of which can form a new game each time. We compare and analyse three different agents that are capable of playing Dominion on different skill levels and use three different fitness functions to generate balanced card sets. Results reveal that there are particular cards of the game that lead to balanced games independently of player skill and behaviour. The approach taken could be used to balance other games with decomposable game mechanics.**

I. Introduction

The field of procedural content generation (PCG) is concerned with algorithms that automatically create various types of game content. While isolated examples of PCG in games date three decades back, and the *SpeedTree* software is commonly used for creating vegetation in commercial games, PCG in published games is almost never used for "necessary" content such as levels and mechanics rather than just for peripheral, "optional" content such as textures and collectable items. Further, most PCG algorithms in published games are not *controllable* as they generate random content within bounds.

To address this issue, the term *search-based procedural content generation* (SBPCG) [1] was proposed for PCG algorithms that build on global stochastic search algorithms (such as evolutionary computation) and fitness functions designed to measure the quality of game content. Examples of this approach include the evolution of platform game levels [2] and racing game tracks [3] and the distributed evolution of weapons in a space shooter game [4].

Recently, the idea of including the rules of a game in the definition of "game content" and generating them through SBPCG algorithms has gained some interest in the PCG community. Like for other SBPCG problems, devising game content representations that allows for effective search of the content space and meaningful fitness metrics appear to be the main challenges. Functions that accurately measure the quality of game rules are likely to be simulation-based (according to the taxonomy presented in [1]), meaning that the functions build on the game being played by an algorithm, since their complexity makes an analytical approach very hard. On that basis, Browne measured the quality of board games using a number of custom-defined measurements [5], most of them simulation-based. Togelius and Schmidhuber proposed a learnability-based fitness function, where the entertainment values of 2D predator-prey games are estimated by how they can be learnt by an algorithm [6]. Salge and Mahlmann evaluated simple strategy game battles using the information-theoretic concept of *relevant information* to determine the amount of information necessary to play well [7].

Balance is a key concern in game design, and lack of balance is usually considered detrimental to game quality (conversely, games such as *StarCraft* are often praised for their exquisite balance). Many authors address the problem of game balancing (see [8]–[13] among others), and predictably there a number of differing conceptions of what game balance is. It could mean that the game has a smooth challenge level throughout a game session, that different players have the same initial chance of winning despite different starting positions or resources, or that different strategies can all be effective given equally skilled player. The existence of a dominant strategy that wins against all others is usually considered evidence of poor balancing.

In this paper, we are mainly concerned with balance between different strategies, and use the card game Dominion as a test-bed for the design of balanced game rules. Inspired by earlier work in the field of general game playing we test three fitness functions that map to game balance metrics; these fitness functions are based on three Dominion agents implementing different types of AI. The card sets obtained generate an interesting discussion and pose some initial questions about game balancing in Dominion.

II. Dominion

Dominion is a card game for 2 to 4 players created by Donald X. Vaccarino and published by Rio Grande Games in 2008. The game revolves around building a powerful dominion, which is reflected in all cards' design. It shares many game mechanics with popular trading card games (e.g. *Magic the Gathering* and *Pokémon*) like using resources to buy new cards, separate card decks for each player, and turn based gameplay with different classes of actions available. On the other hand, it features no collectable cards or card trading outside the game; every copy of the game contains the same cards. At the moment of writing (March 2012) six expansions of the game have been published, each extending the game with new cards and rules. Like in other trading card games, each player has his own deck from which he draws cards from every turn. That deck contains just a few cards at the start of the game, but is extended throughout the game. The goal of the game is to accumulate as many points as possible represented by *victory* cards.

Each game session uses a *supply* of seven *basic cards* and ten *kingdom cards*. The ten kingdom cards are selected before the start of the game, randomly or by players' preference, from a pool of 25 different cards (more with expansion

The authors are with the IT University of Copenhagen, 2300 Copenhagen, Denmark. (email: {tmah, juto, yannakakis}@itu.dk)

Fig. 1. A set of five Dominion cards: Copper, Province, Spy, and Festival.

Fig. 2. Typical Dominion setup seen from a player's perspective. It shows the supply and the player's discard pile in the foreground.

sets). This makes more than 2 quadrillion combinations of card sets possible with the base game alone (our current implementation only models the base game).

Dominion cards are divided in four main categories (an example of each card can be seen in Fig. 1):

- Victory cards: contribute to the player's score.
- Curse cards: victory cards with negative point value.
- Treasure cards: used to buy other cards from the supply.
- Action cards: these cards are used in the action phase to a variety of game effects. Action cards that harm other players are called *attack cards*.

The seven card types present in every game are the basic treasure cards *copper*, *silver*, and *gold* (each of those cards provides a different amount of *coin* value) and the victory cards *estate*, *duchy*, *province*, and *curse* (each providing a different amount of *victory points*. A typical setup of the game can be seen in Fig. 2.

Each turn, each player goes through the following phases:

a) Action phase: During this phase a player has one action which he can use to play an action card from his hand, if present. The player may gain additional actions, given as properties of some cards. Example card properties are: "Draw two more cards from your deck", "Every player passes a card to the player to his left", or "Every card in the supply costs one coin less this turn".

b) Buy phase: In this phase the player uses her treasure cards, which add to the coin value gained in the action phase. The coins can be spent to buy new cards from supply. Each card in the supply displays a price ranging from 0 to 8. Normally players want to include cards from every cost tier of the kingdom cards, i.e. both cheap and expensive cards. The player can only buy one card per turn, but there are action cards which grant additional buys.

c) Clean-up phase: when done with his turn, the player discards all the cards he played this turn and those left in his hand on a separate *discard* pile. He then draws five new cards from his deck. If there are not enough cards left in his deck, the discard pile is shuffled and used as the new deck.

The game ends when either the stack of Provinces is empty or three stacks in the supply are depleted. The player with the highest amount of victory points wins the game. The main challenge for players is to develop a strategy to win the game using the cards on the table. Different strategies are effective in different games due to the synergy effects (or the lack thereof) between different cards.

Each play-through generates different dynamics depending on the cards available which, in turn, yield dissimilar playing experiences. However, not all card combinations are interesting or fun to play with. For example, different card combinations reward different playing styles, and some card sets are allow for deeper or more diverse gameplay than others. One card set can be said to more balanced than another both in terms of differentiating better between strong and weak players, and in terms of allowing for more different strategies to be effective when played by a strong player.

The software used in our experiments is based on the *vDom* engine which is part of the popular Dominion implementation *Androminion* [14] for the Android operating system. Both software programs are free and open source but not affiliated with the original publisher, Rio Grande Games. Presently, we only consider two-player games, both in order to simplify our experiments and the analysis of them.

III. Searching for Dominion Card Sets

For the experiments presented in this paper we employ an integer-valued genetic algorithm (GA) [15] that evolves card sets based on two different fitness functions that attempt to approximate different notions of interestingness and balance. Our GA implementation is based on the genetic algorithm toolkit *JGAP* [16]. Every experiment reported in this paper runs for 30 generations using a population of 100 individuals.

The chromosome is a ten-element integer vector where each element represents a kingdom card. The value of each gene corresponds to an index of all available cards. Any chromosome with duplicate cards is invalid; such chromosomes are assigned a fitness value of zero but kept in the population to promote diversity. Below we present the two fitness functions used to evaluate card sets.

A. Skill differentiation-based fitness

From a pilot survey among the authors and their fellow Dominion players we gained the impression that players who self-reported to be a skillful enjoyed a game more if they could achieve a clearer win (leading by a larger amount of victory points) over their opponents. We therefore derived

the hypothesis that an interesting card set would allow more skilled players to win by a larger amount of points. Thus, a card set will be interesting if the victory point difference at the end of the games including that card set is high. For our skill differentiation fitness function we simulated a number of N games per card set (N is 1000 in this paper) and calculated the average victory point difference. The fitness function, f_s is formalised as:

$$f_s = \frac{\sum_i^N \frac{h_i - l_i}{h_i}}{N}$$

where h_i is the score of the winning player in game i and l_i the score of the losing player in that game. Note that in one experiment we instead minimised the score difference; in that experiment the formula $1 - f_s$ was used instead.

B. Game metric-based fitness

The second fitness function we used is based on the work of Cincotti et al. [17] on the *Synchronized Hex* game. Using MCTS roll-outs in each of the game's turn, they measured the uncertainty of the outcome of the game. With the game progressing, it normally gets easier to predict the winner of a game. Cincotti et al. argued that in a good game this uncertainty stays high until a very late point in the game. An uninteresting game, where the winner is determined early, might bore the players if there is no possibility to compete for the second and third place.

The computational cost of the MCTS rollouts proved to be a problem in that study as MTCS performance depends on the branching factor of the game, which is rather high in Synchronized Hex. Browne [5] was inspired by the idea of Cincotti et al. [17] and further developed the concepts of "Lead changes" and "Decisiveness". He applied those measurements for the design of two player combinatorial games and used PCG techniques to create "interesting" board games. We first describe those two principles and the quantitative measures that encapsuate them in Dominion, and we then present the fitness function that we constructed based on these measures.

1) Lead changes: refers to the concept of the leading player. While there are some games where this concept does not apply (especially folk games), board- and video games normally provide a metric that determines how far a player is from winning or losing the game. Note that this information may be hidden from the players during play, even when it is well-defined. For every turn the leading player is determined. If it differs from the previous turn a "lead change" has occurred. Browne follows the hypothesis that more lead changes may provide a more exciting game play.

It is not trivial to determine neither which player is leading nor which player will win a Dominion game. It is not as simple as simply counting the victory cards in the possession of each player at any time (the *material balance* of e.g. Chess and Checkers). The reason is that in most games a player will try to accumulate as many treasure and action cards as possible before buying their victory cards. Since victory cards are also shuffled into the player's deck, they reduce the chance of drawing a more beneficial card in the early game phase. Therefore, in early game phases, most players' deck will contain few victory cards but, instead, numerous treasure cards which are worthless at the end of the game.

This creates the necessity for a precise estimator of which player is leading the game. We opt to train an Artificial Neural Network (ANN) (employing logistic transfer functions) using backpropagation to predict the leader of the Dominion game in every turn resulting in one ANN model per turn. We choose ANN for their non-linear classification abilities and their universal approximation capacities. The input of the ANN is normalised to [0,1] and and it contains the following 20 values per player in the game:

- Number of action cards gained in this turn.
- Number of treasure cards gained in this turn.
- Number of victory cards gained in this turn.
- Accumulated number of action cards gained over the last five turns.
- Accumulated number of treasure cards gained over the last five turns.
- Accumulated number of victory cards gained over the last five turns.
- Number of action played in this turn.
- Number of buys used in this turn.
- Total number of actions available in this turn.
- Total number of buys available in this turn.
- Accumulated number of actions over the last five turns.
- Accumulated number of buys over the last five turns.
- Total number of actions available accumulated from the last five turns.
- Total number of buys available accumulated from the last five turns.
- Number of curses other players have.
- Coin value of the treasure cards held, divided by the total deck size (money density).
- Total number of action cards in a player's deck.
- Total number of treasure cards in a player's deck.
- Total number of victory cards in a player's deck.
- The player's victory points.

For a two-player game we used an ANN with 40 input nodes, a layer of 10 hidden neurons, and 3 output nodes. Note that we trained a separate network for each turn, therefore the current turn is not fed into the network. Each output node represents one of the following outcomes: player one wins the game, player two wins the game, or the game ends in a draw. The output node with the highest value is selected as the most probable game outcome.

For each card set, 1000 games are played and used to train the ANN. Then another 1000 games were played using the trained network and in each turn the predicted leader of the game was tracked. The number of lead changes is normalised to the total number of turns, T, in a game.

2) Decisiveness: is related to the outcome uncertainty of the game and measures the point in the game where the leading player reaches a winning advantage (Decisive Threshold) over the other player, i.e. when the player lead

changes. Ideally this point comes very late in the game and is also followed by a very short end phase. So the decisiveness value is the ratio of turns before and after the decision point. For our experiments we normalise the decision point to the number of total turns in that game:

$$f_d = \frac{t_l}{T}$$

where t_l is the turn in which the last lead change occurred, and T is the total number of turns in that game.

3) Fitness function calculation: Lead change combined with decisiveness measures have already been successfully applied to different configurations of turn-based strategy games [18]. Like in that study, the final fitness function is constructed by forming the average of both measurements:

$$f = \frac{f_d + \frac{L}{T}}{2}$$

where L is the number of lead changes.

IV. AI-Controlled Dominion Players

We used three different AI-controlled Dominion players for our experiments: two agents are shipped with the *VDom* game engine (*Drew* and *Earl*) and a player controlled by a combination of NeuroEvolution of Augmented Topologies (NEAT) [19] and Monte-Carlo Tree Search [20], [21]; this Dominion player was developed by Fynbo and Nellemann [22]. Initially a fourth *random* agent, which bought and played cards on a random basis, was developed for the experiments. However, this player was abandoned due to its very poor performance (low win rate), making experiments involving that agent degenerate. Such a low performance for a random player, and much better performance for existing artificial players, show that there is a considerable skill element in Dominion which can at least partly be captured by heuristics. However, although there are many decisions on a tactical level that are quite obvious (e.g. which cards to discard, or which action card to play) and are easily solved with a deterministic heuristic, it is not straightforward how to develop an overall strategy that provides a fair challenge against experienced human players.

A. The Earl Player

The Earl player goes through a priority queue (also known as "greedy buy") based on the amount of available money in its hand to buy a card. This way high valued cards like provinces or gold are always bought. In the mid-game this decision is overridden by always buying a *duchy* card. Every cost tier has a set of defined cards to buy. If those cards are not available, Earl continues with the next lower tier. In the action phase Earl favours the *Throne Room* card if he has another card from a pre-defined list of cards in his hand. This decision is fed into the decision of the next action card to play. The next action cards preferred are those that give additional actions or (if non of these is available) those that cause negative effects to other players, e.g. the *Thief* or the *Militia* card. If none of these cards are present either, the remaining cards are passed into a sanity check (i.e. cards that make no sense to play in the current context are sorted out) and a random card is selected.

B. The Drew Player

The Drew player is making decisions based on the progression of the game. Drew includes ad-hoc designed thresholds for the predefined card sets (those that come as recommendations with the game's manual) and an "improvise" setting for a random game. It always favours the high victory cards and high treasure cards until the mid game, and has a number of cards set as "valued cards". If those are not available, it buys a random card. In the action phase, Drew first plays all the cards that would give the player additional actions in any order, then any cards that multiply other cards' effects, and finally other reaming action cards in a random order.

C. Fynbo's and Nellemann's Player

This player, which is trained via neuroevolution using the NEAT algorithm [19], was originally developed by Fynbo and Nellemann [22]. The player was ported by us to fit into the vDom framework. The player comprises of two artificial neural networks (ANNs): one for selecting which next card to buy, and one to select which action card to play.

a) Action phase: The player makes use of the general Monte Carlo Tree Search [20], [21] algorithm. The original algorithm is not feasible for this problem (since some actions, e.g. drawing new cards from the player's deck, would branch the tree by a huge number of possible outcomes), and simulating the whole game until its end is rather costly. Instead, Fynbo and Nellemann developed a variant that was inspired by a solution proposed by Ward and Cowling for Magic: The Gathering [23]. For each action card in the player's hand a number of random simulations are performed in which the tree is expanded until the player runs out of actions or action cards. The resulting state is then evaluated by another ANN. The ANN evaluating the game state takes the following features as inputs (all normalised to [0,1]):

- Number of extra buys
- Number of extra coins (from treasure cards and action cards' effects)
- Number of Militias played
- Number of Witches played
- Number of Bureaucrat played
- Total coin value stolen by a thief
- Number of Spy effects
- Number of Remodel plays
- Number of Chapel plays
- Number of Chancellor plays
- Number of Council room plays
- Number of Mine plays
- Number of Money Lender plays
- Total number of coin in gains
- The estimate of the game's progress (using a separate evaluation function)
- Constant bias

Each resulting game state is assigned a value and the action card sequence that yields the highest valued state is played.

TABLE I

FITNESS GROWTH BETWEEN FIRST AND LAST GENERATIONS. MAXIMUM FITNESS IN GENERATION 1 AND 30, AVERAGED OVER 30 RUNS (f_s AND $1-f_s$) OR 5 RUNS (f), STANDARD DEVIATIONS OF THOSE AND P VALUES FOR A TWO-TAILED STUDENT'S T-TEST. THERE IS SIGNIFICANT FITNESS GROWTH FOR ALL EXPERIMENTS.

	$\langle First \rangle$	$\langle Last \rangle$	$\sigma First$	$\sigma Last$	$p <$
Skill (f_s)					
Earl vs. F-N	0.44	0.64	0.07	0.04	0.0001
F-N vs. Drew	0.38	0.57	0.06	0.008	0.0001
Drew vs. Earl	0.28	0.65	0.05	0.04	0.0001
Skill ($1-f_s$)					
Earl vs. F-N	0.98	0.99	0.01	0.0001	0.0001
F-N vs. Drew	0.99	0.998	0.008	0.0002	0.0001
Drew vs. Earl	0.994	0.999	0.005	0.0002	0.0001
Lead changes (f)					
Earl	0.43	0.24	0.191	0.131	0.0337
F-N	0.52	0.62	0.101	0.083	0.0706
Drew	0.06	0.05	0.02	0.01	0.6013

b) Buy phase: The ANN that decides whether to buy a card takes as input the general game state and five inputs for every kingdom card ("every" refers to all the cards that come with the base game, not just those which are in the current card set). Those inputs are:

- average value of the highest value card gained in the last three rounds.
- the average coin value per card, based on treasure cards.
- the same measurement as the previous, but for the opponents.
- the output of a progress evolution function that tries to determine how far the game is to its end

For each kingdom card the following inputs are considered:

- whether the card is available in the current game
- the number of remaining cards in the pile on the table
- the number of cards in the player's own deck
- the number of cards in opponents' decks
- a control signal. The control signal of the current card is set to 1, otherwise the control signal is set to 0. This technique for ANN input selection is inspired by Stanley [24].

Each input value is normalised between [0,1] using standard min-max normalisation. The single output of the network is an indicator of how promising the currently evaluated card is. The card with the highest output is selected to be bought.

Since the number of inputs is fixed, the agent is limited to play only with the cards in the base game, but it could easily be generalised and trained to play with one or more of the game extensions as well. However, the training effort would make this approach improbable on current hardware due to the increased number of inputs.

V. RESULTS

As a first experiment we wanted to make sure that all three AI agents showed different skill levels, assuming that this would correspond to a different playing style for each agent. For that purpose we matched up two agents and let them play a number of 50000 games, each with a different card combination we randomly sampled. The results can be seen in table II. where we observe performance (win rate) differences among the three AI agents. The win rate differences show that the three agents employ different playing tactics. Draws are counted as a win for both players.

TABLE II

THE NUMBER OF WON GAMES BETWEEN THE DREW, EARL AND FYNBO-NELLEMANN (FN) AGENT, USING A NUMBER OF 50000 RANDOMLY SAMPLED CARD SETS (DRAWS COUNT AS A VICTORY FOR BOTH PLAYERS).

Drew	vs.	Earl	23827	29474
Drew	vs.	FN	19484	31958
Earl	vs.	FN	34190	17399

For the main study presented in this paper we run nine separate sets of experiments. In each trial, one of the three fitness functions (f_s, $1-f_s$ and f) was used and the scores of each individual in each generation was logged. 30 repetitions were carried out of the experiments involving the first two functions, but due to computational resource constraints only 5 repetitions were carried out of the experiments involving f. Table I presents the numerical results of several runs of the experiments. As a note on the computational effort of the experiment, a run of the genetic algorithm with a population size of 30 over 30 generations and the f_s fitness function took between 5 and 12 hours on a 2 GHz dual-core computer. We used the second core to train multiple ANN in parallel where applicable.

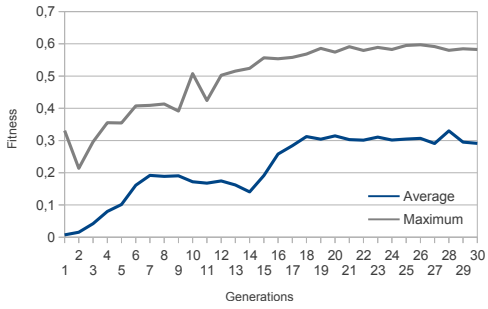
We recorded several runs for each fitness with each AI agent. The skill differentiation function (f_s), which requires two agents, ran with all possible combinations of AI-agents. In this initial set of experiments we did not consider the order of play. The experiment with the game metrics-based fitness ran with two instances of the same agent playing against each other. This was done to prevent the predictor of the leading player learning which agent was first and which was second.

A. Maximising the skill difference

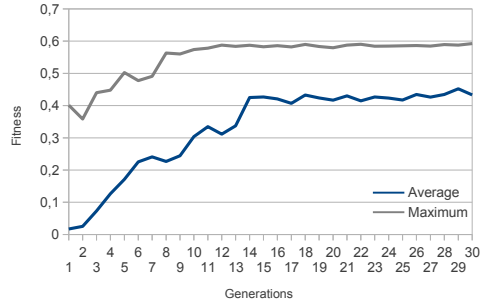
The line-plots of the average and maximum fitness values of the f_s maximisation experiments (see Fig. 3(a), Fig. 3(b), and Fig. 3(c)) show that the GA seems to converge on a local optimum value at the 15th to 18th generation, approximately. This behaviour was repeated by re-running the algorithm for 3 times. Figure 3(a), Fig. 3(b), and Fig. 3(c) show the results of the first run. The only card that was selected all three times, independently of the AI player pair, is the cellar; although Militia, Witch, and Woodcutter show up twice (see table III(a)). The key findings of this set of experiments are further discussed in Section VI.

B. Minimising the skill difference

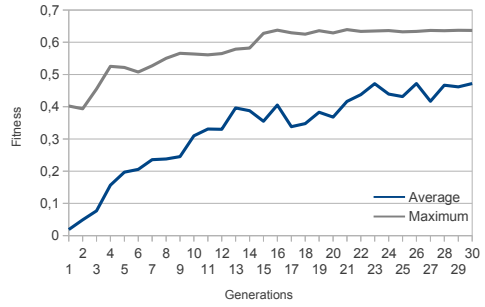
In this set of experiments we, instead, attempt to mimimise $1-f_s$. The resulting card sets that can be seen in table III(b) while the evolution of the average and maximum fitness values are plotted in Fig. 4(a), Fig. 4(b), and Fig. 4(c). Investigation of the highest performing card sets reveal that all three sets include the bureaucrat card (see table III(b)).



(a) Drew vs. Earl

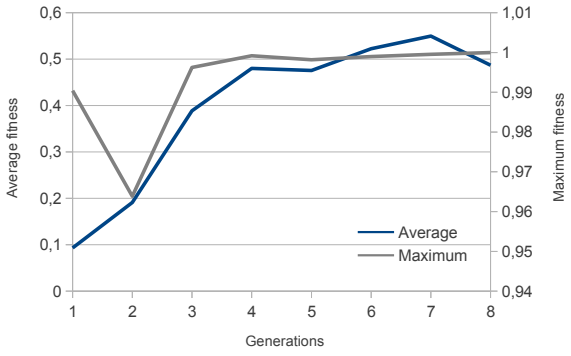


(b) Drew vs. Fynbo-Nellemann

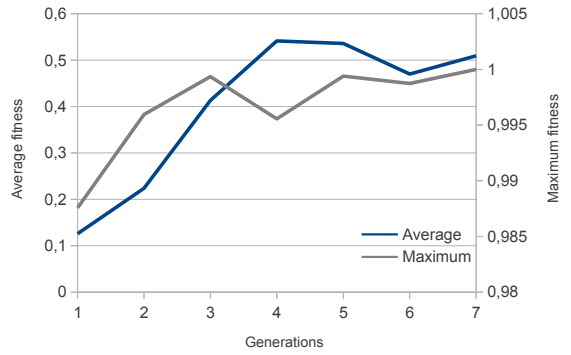


(c) Earl vs. Fynbo-Nellemann

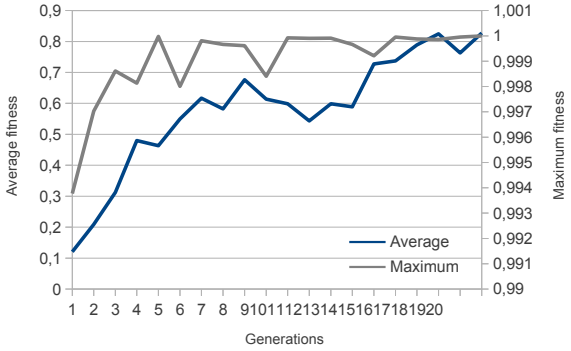
Fig. 3. Maximising skill difference: Evolution of the f_s fitness across different player types.



(a) Drew vs. Fynbo-Nellemann



(b) Earl vs. Fynbo-Nellemann



(c) Drew vs. Earl

Fig. 4. Minimising skill difference: Evolution of the 1- f_s fitness across different player types.

The bureaucrat card forces the other player to put a victory card from his hand back onto his deck thus drawing it again next turn. If played repeatedly, this may actually hinder the opponent player hence countering his skilful play. The second card which exists in all three card sets — across player combinations — is the spy card. Finally, there are five cards which appear at least twice in the sets: the cellar, the chancellor, the gardens, the throne room, and the witch card. Apart from the witch card which is an attack card, and the throne room card which is a multiplier for other cards, it is hard to see what impact those cards have on skill difference minimisation. The results are further discussed in Section VI.

C. Maximising for lead changes and decisiveness

Our attempts to maximise lead changes and decisiveness yielded somewhat inconclusive results, with the high noise level of this fitness measure coupled with the substantial computational complexity hindering us from finding statistically significant fitness growth. As can be observed from Fig. 3(c), Fig. 5(a), and Fig. 5(b), all three runs show similar convergence characteristics; nevertheless, the fitness values differ substantially between the three AI agents tested. The resulting card sets (see table III(c)) are also much more diverse between agents than in the other experiments. The following section discusses these findings further.

VI. Discussion

The results of the first experiment, which compared the performance of the three AI agents, showed that playing skill is not transitive: Drew and Earl showed nearly equal performance against each other; however, Drew lost against Fynbo-Nellemann's player while Earl dominated that player. As part of future research we would like to further analyse which types of games are lost by which AI agent in the future and examine the relationship between playing styles, card sets and playing performance in Dominion.

Optimising card sets for minimising skill differentiation

TABLE III
RESULTING CARD SETS WITH THEIR CORRESPONDING (HIGHEST) FITNESS VALUES OBTAINED IN EACH EXPERIMENT

(a) Maximising skill difference (f_s)

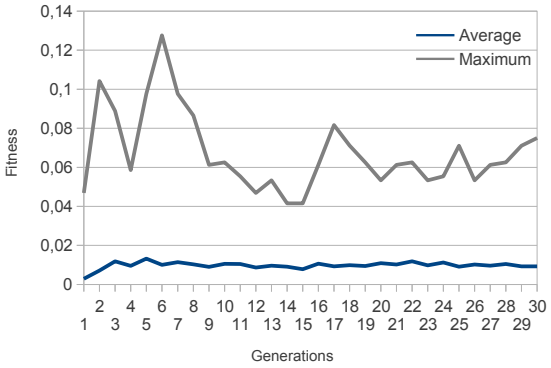
Drew vs. Earl	Adventurer	Cellar	Chapel	Council	Festival	Laboratory	Mine	Remodel	Village	Workshop	0.596
Drew vs. F-N	Chancellor	Cellar	Militia	Moat	Smithy	Thief	Throne	Village	Witch	Woodcutter	0.592
Earl vs. F-N	Bureaucrat	Cellar	Chancellor	Library	Militia	Smithy	Spy	Thief	Witch	Woodcutter	0.639

(b) Miminising skill difference ($1 - f_s$)

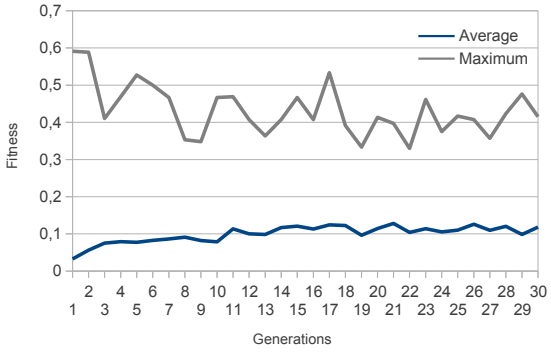
Drew vs. Earl	Bureaucrat	Chancellor	Festival	Gardens	Library	Remodel	Spy	Throne	Village	Witch	1.0
Drew vs. F-N	Adventurer	Bureaucrat	Cellar	Chancellor	Chapel	Council	Gardens	Mine	Smithy	Spy	1.0
Earl vs. F-N	Bureaucrat	Cellar	Feast	Laboratory	Library	Market	Spy	Throne	Witch	Woodcutter	1.0

(c) Maximising lead changes and decisiveness (f)

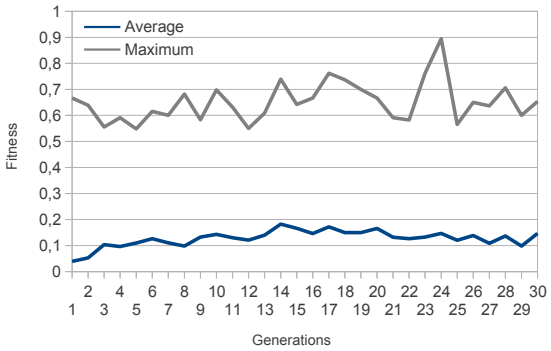
Drew	Adventurer	Bureaucrat	Council	Laboratory	Market	Money Lender	Smithy	Spy	Thief	Witch	0.458
Earl	Adventurer	Cellar	Feast	Festival	Gardens	Market	Militia	Moat	Smithy	Village	0.59
F-N	Council	Festival	Gardens	Laboratory	Market	Militia	Spy	Remodel	Witch	Workshop	0.894



(a) Drew vs. Drew



(b) Earl vs. Earl



(c) Fynbo-Nellemann vs. Fynbo-Nellemann

Fig. 5. Maximising lead changes and decisiveness: Evolution of the f fitness across different player types.

yielded consistently good results, quickly finding card sets that allowed one agent to reliably win over the other. It is likely that evolution found card sets that exploited the inability of the agents to play with arbitrary card sets; in other words, that evolution found exploits or bugs in the agents. Since all generated card sets contained several attack cards, it could be also possible that the AI agents hindered each others play and, therefore scored evenly.

The results from the skill maximisation experiment were less clear: the only card that is included in all three winning sets was the *cellar* card. The cellar card lets players toss unimportant cards (in a given turn) from their hand and draw new ones from the deck. When creating a Dominion strategy one has to consider that all cards that are bought (including victory cards) go into the deck of the player. Thus, the higher the number of potentially important cards in the deck is, the lower the probability to draw a beneficial card from the deck. Novice players often ignore this mechanic and buy as many cards as they can get each turn. Given this observation we suspect that a highly skilled player may use the cellar to counteract this problem and, therefore, achieves a clearer win over an unskilled player.

We have seen that most card sets generated contain one or two attack cards. This was expected as without attack cards the player interaction in Dominion is rather minimal: the only possibility to affect the opponents' progress is to buy certain cards from the supply before the opponents buy them. Yet attack cards are not mandatory to include in the supply and many players enjoy a game with minimal player-to-player interaction. Others enjoy playing Dominion by actively sabotaging the opponents' play through stealing cards from them (Thief card) or giving them curses (Witch card). It is interesting that our experiments — which are driven by fitness functions that relate only implicitly to player experience — show that those cards do not only serve an entertainment purpose, but they constitute a critical element of the game's balance.

This paper presented an initial set of experiments testing balance in Dominion. As part of our future research agenda, we aim to test the game with more AI-controlled players and

different fitness functions. We also believe that this method could be used more generally to help design and balance other games. Each card in Dominion represents a single game mechanic, and including the card in a set translates to enabling the corresponding mechanic. By analogy, one could modify other games to enable the toggling on/off of individual mechanics, and search the space of sets of mechanics for game variants that induce certain kinds of balance or unbalance with regard to agents or sets of agents. Depending on the level of abstraction, even simple games can be said to have numerous mechanics – e.g., Super Mario Bros has running, jumping, run-jumping, wall jumping, shooting, stomping, crouching, jumping while crouching and many others. A complex strategy game such as *StarCraft* of fighting game such as *Street Fighter II* has such a wealth of mechanics that it is scarcely humanly possible to keep track of how they interact with each other. Searching the space of sets of mechanics, on the other hand, is not a hard problem given appropriate fitness functions and agents to base them on.

VII. Conclusion

In this paper we used three different fitness functions to evolve different sets of cards for the Dominion game with the aim to make it more balanced. We tested our genetic search algorithm against three agents that are capable of playing Dominion on different skill levels and we compared their performance. Results obtained show that there are particular cards in Dominion that make the game more balanced independently of playing styles. We have also argued that the method used here can be used more widely for automatic game design and game balancing.

Acknowledgments

The research was supported in part by the Danish Research Agency (FTP) project *AGameComIn* (number 274-09-0083).

References

[1] 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.

[2] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling Player Experience for Content Creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.

[3] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation in racing games," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2007.

[4] E. Hastings, R. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.

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

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

[7] C. Salge and T. Mahlmann, "Relevant information as a formalised approach to evaluate game mechanics," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[8] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game ai with dynamic scripting," *Machine Learning*, vol. 63, pp. 217–248, 2006, 10.1007/s10994-006-6205-6. [Online]. Available: http://dx.doi.org/10.1007/s10994-006-6205-6

[9] G. N. Y. J. K. Olesen and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, December 2008, pp. 87–94.

[10] C. Paul, *The Discourse of Video Games: Word Play*, ser. Routledge Studies in New Media and Cyberculture. Taylor and Francis, 2012. [Online]. Available: http://books.google.dk/books?id=1-QITwEACAAJ

[11] G. van Lankveld, P. Spronck, H. van den Herik, and M. Rauterberg, "Incongruity-based adaptive game balancing," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. van den Herik and P. Spronck, Eds. Springer Berlin / Heidelberg, 2010, vol. 6048, pp. 208–220, 10.1007/978-3-642-12993-3_19. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12993-3_19

[12] G. Andrade, G. Ramalho, A. S, R. Gomes, and V. Corruble, "Challenge-sensitive game balancing: an evaluation of user satisfaction," in *In Proceedings of the 4rd Brazilian Workshop on Computer Games and Digital Entertainment (WJogos05*. AAAI Press, 2005, pp. 66–76.

[13] K. Salen and E. Zimmerman, *Rules of play: game design fundamentals*. MIT Press, 2004. [Online]. Available: http://books.google.dk/books?id=UM-xyczrZuQC

[14] "Androminion," accessed January 10th 2012. [Online]. Available: http://code.google.com/p/androminion/

[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-WesleyEditors, Ed. Addison-Wesley, 1989. [Online]. Available: http://www.amazon.com/Genetic-Algorithms-Optimization-Machine-Learning/dp/0201157675

[16] K. Meffert, N. Rotstan, C. Knowles, and U. Sangiorgi, "Jgap-java genetic algorithms and genetic programming package," *URL: http://jgap. sf. net*, 2008.

[17] 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.

[18] T. Mahlmann, J. Togelius, and G. Yannakakis, "Modelling and evaluation of complex scenarios with the strategy game description language," in *Proceedings of the Conference for Computational Intelligence (CIG) 2011*, Seoul, KR, 2011.

[19] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, June 2002. [Online]. Available: http://dx.doi.org/10.1162/106365602320169811

[20] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *AIIDE*, 2008.

[21] I. Szita, G. Chaslot, and P. Spronck, "Monte-carlo tree search in settlers of catan," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. van den Herik and P. Spronck, Eds. Springer Berlin / Heidelberg, 2010, vol. 6048, pp. 21–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12993-3_3

[22] R. B. Fynbo and C. S. Nellemann, "Developing an agent for dominion using modern ai-approaches," Master's thesis, IT University of Copenhagen, 2010.

[23] C. D. Ward and P. I. Cowling, "Monte carlo search applied to card selection in magic: the gathering," in *Proceedings of the 5th international conference on Computational Intelligence and Games*, ser. CIG'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 9–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1719293.1719306

[24] K. O. Stanley, "The neuroevolution of augmenting topologies (neat) users page," 2010. [Online]. Available: http://www.cs.ucf.edu/ kstanley/neat.html

View publication stats