A Card Game Description Language

Jose M. Font[1], Tobias Mahlmann[2], Daniel Manrique[1], and Julian Togelius[2]

[1] Departamento de Inteligencia Artificial, Universidad Politcnica de Madrid. Campus de Montegancedo, 28660, Boadilla del Monte, Spain
{jfont,dmanrique}@fi.upm.es
[2] Center for Computer Games Research, IT University of Copenhagen, Rued Langaards Vej 7, 2300 Copenhagen, Denmark
{tmah,juto}@itu.dk

Abstract. We present initial research regarding a system capable of generating novel card games. We furthermore propose a method for computationally analysing existing games of the same genre. Ultimately, we present a formalisation of card game rules, and a context-free grammar $G_{cardgame}$ capable of expressing the rules of a large variety of card games. Example derivations are given for the poker variant *Texas hold 'em*, Blackjack and UNO. Stochastic simulations are used both to verify the implementation of these well-known games, and to evaluate the results of new game rules derived from the grammar. In future work, this grammar will be used to evolve completely novel card games using a grammar-guided genetic program.

1 Introduction

The ruleset is essential to any game, defining its mechanics and in many ways being the "core" of the game. Rules can't be removed or edited without changing a game's computational properties. The digital and unambiguous nature of most rules (rules for computer games) makes them easy to model as program code. The modelling of rules can serve several different purposes, but the two most prominent are computational analysis (of gameplay) and generation of new games. Computational analysis uses existing games to simulate many playouts under various circumstances to analyse the balance, depth, variety and other aspects of a game. One approach to game (rules) generation may be done by searching a space of games, expressed in some *game description language* (GDL) to find games with desirable properties. These two purposes go well together, as modelling several related games is a good way of constructing a GDL, and computational analysis is typically used to evaluate candidate games when generating novel game rules.

Card games seem to be an interesting application for automated design and computational analysis for several reasons. An important factor is clearly their ubiquity and popularity; different card games originated from many parts of the world, and have been played since hundreds of years. Another important factor is their computational simplicity: most card games could be simulated with very limited computational effort compared to games which are designed to be played with a computer and are normally very calculation heavy (e.g. simulation games). Card games share these two aspects with another type of games: classic board games such as Chess, Go and Backgammon. But unlike board games, card games share a common prop: the classic French deck of cards (52 cards of four colours and two jokers). Most card games (certainly most Western card games) can be played using only one or two such decks, and perhaps a few tokens representing money or score. This enables us to model a large variety of card games by simply altering the form of their rules.

In recent years, several authors have attempted to formally describe and automatically generate game rules. The two main game genres where this has been attempted are board games and simple 2D arcade games. In board games, the early work of Hom and Marks [7] was followed by Browne's *Ludi* system, which managed to evolve a board game of sufficient novelty and quality to be sold commercially [2,1]. A system generating simple arcade games was proposed by Togelius and Schmidhuber [11], which was followed up by work by Smith and Mateas [10] and Cook and Colton [3]. In a related genre, Mahlmann et al. have defined a GDL for turn-based strategy games [9]. Mahlmann et al. also published similar work to our approach, evaluating different game mechanics of the card game *Dominion* [8]. The representation of game rules and level of abstraction in these examples varies wildly, from expression trees reminiscent of those used in genetic programming, to tables of actions and consequences, to first-order logic expressed in AnsProlog. In all of these examples, the space of games was searched using evolutionary algorithms, except Smith and Mates who use answer set programming and constraint solving. An overview of this line of research can be found in a recent tutorial by Nelson[1].

There are several important considerations when devising a GDL for a particular game genre. Prominent among these is the expressivity of the language: all games in scope have to be expressible in the language. Normally opposing the expressivity stands the compactness or verbosity of the language, i.e. its human-readability, and how efficiently the search space can be traversed by using for example an evolutionary algorithm. Especially the latter is only marginally explored, and more research needs to be done.

2 Definition of a Search Space for Card Games

It is not possible to evolve any kind of game without setting the constraints that define the basics of the search space [11]. Insofar as card games are the subject of the evolutionary process described here, it is mandatory to define a set of axioms that will be shared by any card game in the evolutionary population.

http://kmjn.org/

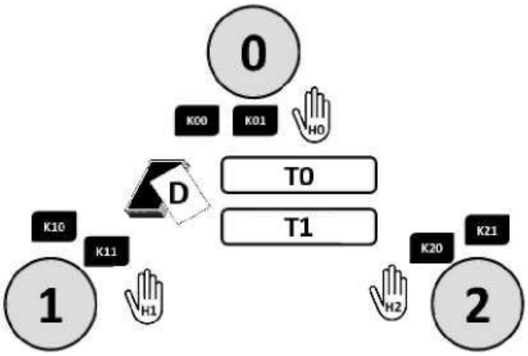

Fig. 1. Main components of a card game with three players and two table locations (P = 3 and T = 2)

Those axioms are the following:

— The card game is played by exactly *P* players.

— A *card location* is defined as a place where any number of cards can be placed during the game. Every game has the following card locations *L:*

- A number of hands *H,* one for each player in *P*. Hereafter, *HX* refers to the "hand of the current player" and *HA* to "hands of all other players". *H0, H1, H*2 refer to the hand of player one, two, or three respectively.
- One standard French deck of cards *D,* composed of four suits with 13 cards each, numbered from 1 to 13 (jokers are excluded). Cards in the deck are always placed face down.
- A number of table locations *T,* which are areas in a virtual table where cards can be placed, always face up.

— Insofar as many card games involve bets, *tokens* are defined as virtual representations of coins or chips. Analogously to card locations, a token location *K* is defined as a place where any number of tokens can be placed during the game. Every player *J* has two token locations, *KJ0* and *KJ1. KJ0* functions as a player's stash), and *KJ1* may be used to place bets. To illustrate the nomenclature, player 0 has two token locations denoted by *K00* and *K01.* The term *KX1* refers to "the current bet of the current player", and *KA1* to "the current cumulative bets of all other players". Figure 1 shows the main components of a card game with *P* = 3 and *T* = 2, therefore including six token locations *(K00, K01, K10, K11, K20, K21),* and three hands *(H0, H1, H2).*

— A card game consists of several stages, each one composed by a set of rules. The stages are played in sequential order.

— Stages are played out in a turn-based way with a round-robin turn order. Within each stage and during his turn, a player may play a variable number of rules. A player's turn is over when he is *done, next* or *out:*

- *done* refers to a player being done with the current stage (but will return to play in the nest stage).

  - *next* indicates that the player has finished his turn but will be able to play again during the current stage when all other players' turns are over.
  - *out* defines a player as being out of the game. He will not play in any remaining stages of the game.

— A stage ends when all players are either *done* or *out.* While one or more player are *next,* their turns alternate in a round-robin order.
— A game ends when all stages have been played or when all players are *out.*

Given these specifications, a card game is defined as a set of stages, a ranking of card combinations, and a set of winning conditions. Every stage comprises a set of conditional rules (production rules) in the form "if antecedent then consequent". Antecedents are conditions that - when fulfilled - trigger actions defined in consequents. For example: "if the player has no cards, then he draws a card". Optionally, a rule may have no antecedents at all, meaning that it can be played without any particular prerequisites. Following this structure, three different kinds of rules can be found:

— *Optional rule.* Standard rule which can be played a multiple number of times. If the current player satisfies the condition defined in the antecedent, he can play the action defined in the consequent.
— *Play-only-once rule.* These rules (marked with a "once" modifier) can only be played once per stage.

Table 1. Types of antecedents and their related conditions

Antecedent	Condition	Example
tokens, KA, RESTRICTION, KB	Compares the amount of tokens in token location KA with the amount of tokens in KB. Condition is fulfilled if the comparison satisfies the RESTRICTION $(<, >, =, \leq, \geq)$	tokens, $K01$, $\geq$, $K11$
play, LA, RESTRICTION, LB	Compares the play in card locations LA with the play in LB. Condition is fulfilled if the comparison satisfies the RESTRICTION $(<, >, =, \leq, \geq)$	play, $H0 + T0$, $<$, $H1 + T0$
sum, LA, RESTRICTION, LB	Sums up the numbers of the cards in locations LA and the ones in LB. Condition is fulfilled if the comparison satisfies the RESTRICTION $(<, >, =, \leq, \geq)$	sum, $H0$,=,$H1$
have, COMBINATION	Check if the player has a given COMBINATION of cards, suits or numbers in his hand.	have, 2 of diamonds
draw	Draw one card from the deck and place it in the hand of the current player	draw
show, RESTRICTION, LA	Show a set of cards from player's hand that satisfy a given RESTRICTION $(<, >, =, \leq, \geq, \mathit{same\ suit}, \mathit{same\ number})$ when compared with a card in LA.	show, card with same suit, $T0$
λ (LAMBDA)	No condition to satisfy.	unconditional

– *Mandatory rules.* Mandatory rules must be played (and only once) by every player at the beginning of his first turn of the stage. They are marked with a “mandatory” modifier.

Additionally, so called *computer commands* are always played once by the game (i.e. a virtual dealer) at the beginning of the stage before the players’ turns. These rules are marked with a “com” modifier. Command rules include no conditions, i.e. no antecedent, and only one consequent which describes the action to be played. Table 2(b) illustrates the implemented types of computer rules.

Once the game’s setup is finished, the first player’s turn begins. After all “mandatory” rules have been invoked (which may be none), he may play as many “once” and/or optional rules as possible. When there are no satisfiable antecedents left, the player is marked as *done*. Optionally, a player may choose to not play any (more) rules, deliberately changing his state to *next* (if he wants to play again during the current stage) or *done* (otherwise). Table 1 shows the different kinds of antecedents possibly involved in a rule. Notice that the terminal symbol “lambda” (λ) is used to represent a “null” value for any given variable, and *LA* and *LB* refer to any two valid card locations.

When played, rules trigger consequents which modify the game state. Table 2(a) shows the types of consequents which can be found in a rule. Notice that the “bet” consequent does not specify any location. Betting always refers to moving tokens from a player’s total amount of tokens to his current bet. “Gain” always implies moving tokens from the specified location to a player’s total amount of tokens.

Every card game has its own card and play values. Those features are specified in the ranking: a table that contains pairs in the form *[play, score],* in order to assign a fixed *score* to a given *play* (card combination). This table is used every time a condition “play” is evaluated in order to know the actual value of the card combinations in comparison. For example, when playing poker, all valid poker hands have to be indexed in the ranking table, therefore it is possible to know that two pairs have a lower value than three of a kind. When a given hand has no specified value, the standard ranking of cards is assumed. Thus: $1 > 13 > 12 > 11 > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2$. Upon finishing the game, winning conditions determine which player wins. This is done by assigning points to the remaining amount of tokens the player has and extra points if the player status is not *out*. An exception here is, that certain consequents may declare a certain player as the winner a priori.

2.1 The Card Game Language

A context-free grammar (CFG) G is defined as a string-rewriting system comprising a 4-tuple $G = (SN, ST, S,P)/SN (\sim\}ST = O/,$ where SN is the alphabet of non-terminal symbols, ST is the alphabet of terminal symbols, S represents the start symbol or axiom of the grammar, and P is the set of production rules in Backus-Naur form. Every grammar generates a language composed where all sentences are grammatically valid, meaning they conform to the constraints defined by that grammar. The context-free grammar $G_{car}d_{ga}me$ has been designed

Table 2. Overview of rule Consequences

(a) Types of consequents in a rule

Consequent	Action	Example
pifr, LA * AMOUNT, FACE	Draw a given AMOUNT of cards from location LA. FACE indicates if cards are drawn face up or down.	pifr, $D*2$, down
puin, LA * AMOUNT, RESTRICTION, FACE	Put a given AMOUNT of cards from player's hand in card location LA, only if those cards satisfy a given RESTRICTION ($<, >, =, \leq, \geq$) .	puin, $T0*1$, $>$, up
bet, RESTRICTION, KX	Bet an amount of tokens that satisfy a given RESTRICTION ($<, >, =, \leq, \geq$, *lambda*) when compared with the amount of tokens in location KX.	bet, $>$,$K01$
gain, KX	Gain the amount of tokens in token location KX.	gain, $K21$
playit	Place the set of cards specified in the antecedent in the location specified in the antecedent	-
next	The player status is set to *next*.	-
done	The player status is set to *done*.	-
out	The player status is set to *out*.	-
win	The player immediately wins and the game ends.	-
end	The game immediately ends.	-

(b) Special Consequences in computer rules

Rule	Action	Example
com_deal, PLAYERS, AMOUNT	deal a given AMOUNT of cards from the deck to a set of PLAYERS	deal, player 1, 4
com_deal, $L0$, AMOUNT	deal a given AMOUNT of cards from the deck to location $L0$	deal, $T0$, 4
com_give, PLAYERS, AMOUNT	give a fixed AMOUNT of tokens to a set of PLAYERS	give, players 0 and 1, 100

to generate the language composed by all the valid card games that comply with the axioms described above.

A grammar-guided genetic program (GGGP) is an evolutionary system that could potentially find solutions to any problem whose syntactic restrictions can be formally defined by a CFG [5]. Individuals are derivation trees of the CFG that, when the algorithm starts, are generated by a grammar-based initialization method [6]. Neither this method nor crossover and mutation operators can generate invalid individuals because they are not contained in the language described by the CFG [4]. Thus the individual population of a GGGP using $G_{cardgame}$ is a set of derivation trees, each of them defining a card game that follows the above

Table 3. Codification of Texas hold'em poker stages

STAGES

NAME	RULES	NAME	RULES
STAGE 0	com _ deal, ALL PLAYERS, 2	STAGE 6	com _ deal, T1, 1
(pre-flop)	com _ give, ALL PLAYERS, 99	(turn)	com _ deal, T0, 1
STAGE 1	mandatory_unconditional_bet, λ, λ	STAGE 7	unconditional_bet, $\geq$, $KA1$
(first bet)	unconditional_bet, =, $KA1$	(3rd bet)	once_unconditional _ *next*
	once_unconditional _ *next*		once_unconditional_*done*
	once_unconditional_*done*	STAGE 8	mandatory_tokens, $KX1$, <, $KA1$
STAGE 2	mandatory_tokens, $KX1$, <, $KA1$	(check bets)	_ *out*
(check bets)	_ *out*	STAGE 9	com _ deal, T1, 1
STAGE 3	com _ deal, T1, 1	(river)	com _ deal, T0, 1
(flop)	com _ deal, T0, 3	STAGE 10	unconditional_bet, $\geq$, $KA1$
STAGE 4	unconditional_bet, $\geq$, $KA1$	(final bet)	once_unconditional _ *next*
(2nd bet)	once_unconditional _ *next*		once_unconditional_*done*
	once_unconditional_*done*	STAGE 11	mandatory_tokens, $KX1$, <, $KA1$
STAGE 5	mandatory_tokens, $KX1$, <, K _	(check bets)	_ *out*
(check bets)	*out*	STAGE 12	mandatory_play, $HX + T0$, >,
		(showdown)	$HA + T0$ _ gain, $KA1$

constraints. Our card games are codified in the genotype of individuals with the structure *STAGES* : *RANKING* : *WINNING CONDITIONS*, where:

- *STAGES* represents several sets of rules, each of them corresponding to a stage of the game.
- *RANKING* is a list of pairs in the form [*play*, *score*] which will be later translated into a ranking table.
- *WINNING CONDITIONS* specifies two natural numbers that are the amount of points awarded to a player for each remaining token and for not being out of the game respectively.

$G_{cardgame}$ has been intentionally designed to generate a high level language containing a great variety of card games. Since individuals of a GGGP have no fixed size, the evolutionary system becomes a flexible tool, being able to design games from simple one-stage games up to long strongly ruled games. Nevertheless, despite the high level approach, this language contains at least three very well known card games: Texas hold 'em poker, Blackjack and UNO. For brevity, the following paragraphs assume that the reader is familiar with said games and their terms.

Table 3 presents Texas hold 'em poker's stages codified as a sentence of the language generated by $G_{cardgame}$. The game is composed by 12 stages which cover the standard parts of the game (pre-flop, flop, turn, river and showdown) as well as additional stages for player bets and bet checking. Bets are checked in order to set the status of players which have folded as *out*. The ranking includes poker hands, and the winning conditions define that the winner is the player with the most tokens at the end of the game. Please note that single cards are not

Table 4. Codification of the two example games Blackjack and UNO

(a) Blackjack

STAGES	
NAME	RULES
STAGE 0	com _ deal, ALL PLAYERS, 2
	com _ give, ALL PLAYERS, 99
STAGE 1	mandatory_unconditional_bet, λ, λ
STAGE 2	unconditional_pifr, D*1, up
	unconditional_*done*
STAGE 3	mandatory_sum, $HX > 21$ _ *out*
	mandatory_sum, $HX > HA$ _ gain, KA

(b) UNO

STAGES	
NAME	RULES
STAGE 0	com _ deal, ALL PLAYERS, 7
	com _ deal, T0, 1
STAGE 1	show, same suit, $T0$ _ playit
	show, same number, $T0$ _ playit
	draw _ *next*
	mandatory_have, λ _ *win*

listed in the ranking table as poker uses the default card ranking. The ranking used for the poker hands is: Straight flush (900), Poker (800), Full house (700), Flush (600), Straight (500), Three of a kind (400), Two pairs (300), and One pair (200).

Table 4(a) shows the codification of blackjack. In this implementation of blackjack, players play against each other instead of the dealer. The winner of the game is the player who earned more tokens during the game, meaning the player who bet most and got the highest score below or equal to 21. The ranking sets all figures' values (13, 12, 11) to 10, and includes the two possible values for an ace: 1 and 10. The only winning condition is one point per token earned.

Table 4(b) shows the codification the basic rules of UNO. For simplicity, we excluded cards with any special effect, e.g. changing the turn order or skipping players. Notice that the ranking table is empty because the goal for a player is to empty his hand (rather than getting the best play). There is no need to set winning conditions or rankings because the winner is determined by the last rule of stage 1.

3 Experimental Results

In order to test the quality of the card games generated by $G_{car}dgame$, random plays have been run over a set of games expressed by its language. All games have been designed for three players and up to two table locations $(P = 3, T = 2)$. The term "random" herein refers to artificial players playing random moves.

The set of games is composed by the sample games presented above: Texas hold 'em poker, Blackjack and UNO. Each of these has been run a 1000 times. In addition to this, 1000 random mutations of each of them have also been tested. Ultimately, we also sampled 1000 randomly generated card games.

For each game the following data has been collected: number of plays won by each player, number of plays ended in a draw, number of games which crashed and the average number of turns needed to properly finish the game. A game is considered to be crashed when it hasn't finished within 100 turns or when a non semantic expression occurs, e.g. forcing a player to bet an amount of tokens lower than zero.

Table 5. Results obtained from running Texas hold 'em poker, Blackjack, UNO, random mutations of them and randomly generated games, 1000 times each

Game	Times run	Times won			Times draw	Times crashed	AVG turns to finish
		0	1	2			
Texas	1000	235	324	437	4	0	39
Blackjack	1000	401	282	317	0	0	20
UNO	1000	319	285	294	102	0	59
Random Texas mutations	1000	237	15	15	523	210	13
Random blackjack mutations	1000	49	30	103	516	302	32
Random UNO mutations	1000	79	45	22	293	561	26
Random games	1000	174	32	36	485	273	19

Table 5 shows the results obtained from these tests. The three sample games have been properly played during all 1000 runs, showing that the language defined by $G_{cardgame}$ allows to codify well-formed and semantically valid card games. These games are finished in a very reasonable number of turns, whereas UNO seems to be the longest one. It seems easy to conclude, that this may because UNO's last stage is not finished until one of the players gets rid of all his cards. UNO is also the game most likely to end in draw. A draw takes place when the deck and the table location run out of cards before any player has been able to win.

As expected, random mutations on these games lead to an increase in their crash rate. Nevertheless, almost 80% of poker mutations, 70% of blackjack mutations and 45% of UNO mutations can be properly played until the end of the game. Draw rates also increased, meaning that changes produced by mutations create games where winners are not clearly defined.

Randomly generated games produce similar results, showing a rate of playable games close to 73%.

4 Conclusions and Future Work

A context-free grammar, $G_{cardgame}$, that generates a game description language for card games has been presented. All card games contained in this language share a fixed set of basic axioms, that represent the main components of a game. Every game defines its very own rules that, when played along with these axioms, compose a playable card game. This language contains a set of high-level instructions that allows the codification of a high variety of games. To show this, examples codifications of Texas hold ’em poker, Blackjack and UNO have been proposed. The experiments conducted show the ability of $\boldsymbol{G}_{cardgame}$ to produce randomly generated but playable card games. Our results indicate, that it is possible to improve the generation of card games with searching and optimizing techniques. For this reason, our grammar is intended to be part of a grammar-guided genetic program that evolves populations of card games in order to automatically generate entertaining and playable card games. Leading the evolutionary process to improve the satisfaction achieved by players when playing these games will allow the possibility of creating interesting card games without human assistance. In other words, mimicking human creativity inside a computer process.

References

1. Browne, C., Maire, F.: Evolutionary game design. IEEE Transactions on Computational Intelligence and AI in Games 2(1), 1–16 (2010)
2. Browne, C.: Automatic generation and evaluation of recombination games. Ph.D. thesis, Queensland University of Technology (2008)
3. Cook, M., Colton, S.: Multi-faceted evolution of simple arcade games. In: Proceedings of the IEEE Conference on Computational Intelligence and Games, CIG (2011)
4. Font, J.M., Manrique, D., R´ıos, J.: Evolutionary construction and adaptation of intelligent systems. Expert Systems with Applications 37, 7711–7720 (2010)
5. Font, J.M.: Evolving Third-Person Shooter Enemies to Optimize Player Satisfaction in Real-Time. In: Di Chio, C., Agapitos, A., Cagnoni, S., Cotta, C., de Vega, F.F., Di Caro, G.A., Drechsler, R., Ek´art, A., Esparcia-Alc´azar, A.I., Farooq, M., Langdon, W.B., Merelo-Guerv´os, J.J., Preuss, M., Richter, H., Silva, S., Sim˜oes, A., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Togelius, J., Urquhart, N., Uyar, A.S¸., Yannakakis, G.N. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 204–213. Springer, Heidelberg (2012)
6. Garcia-Arnau, M., Manrique, D., Rios, J., Rodriguez-Paton, A.: Initialization method for grammar-guided genetic programming. Knowledge-Based Systems 20(2), 127–133 (2007)
7. Hom, V., Marks, J.: Automatic design of balanced board games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), pp. 25–30 (2007)
8. Mahlmann, T., Togelius, J., Yannakakis, G.: Evolving card sets towards balancing dominion. In: IEEE World Congress on Computational Intelligence, WCCI (2012)
9. Mahlmann, T., Togelius, J., Yannakakis, G.: Modelling and evaluation of complex scenarios with the strategy game description language. In: Proceedings of the Conference on Computational Intelligence and Games (CIG) 2011, Seoul, KR (2011)
10. Smith, A.M., Mateas, M.: Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In: Proceedings of the IEEE Conference on Computational Intelligence and Games, Copenhagen, Denmark, August 18–21, pp. 273–280 (2010)
11. Togelius, J., Schmidhuber, J.: An experiment in automatic game design. In: IEEE Symposium on Computational Intelligence and Games, CIG 2008, pp. 111–118. IEEE (2008)