The suitability of Groove as a General Game Playing Language

Daniël Floor
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
d.floor@student.utwente.nl

ABSTRACT

A General Game Playing system is capable of playing a wide variety of games with the use of AIs. Part of these games is perfect information games. To play these games in such an environment, a Game Description Language is needed. This is the language that is used to model the games in a digital environment. There are lots of options to base the Language on, one possibility is graph transformations. Groove uses these graph transformations which can be used to model board games. This paper will analyze the suitability of the Game Description Language of Groove for General Game Playing purposes.

Keywords

General Game Playing, Graph transformations, Groove, Game Description Language

1. INTRODUCTION

There exist several game-specific AI's that can beat even the best human players. Examples of these are Stockfish[1] for chess and AlphaZero [5] for the board games Go, chess, and Shogi. These programs are adapted in such a way that they perform incredibly well on specific games, while not being able to perform on a wide variety of other games.

Since the early 2000s, General Game Playing (GGP) has been gaining popularity amongst computer scientists. GGP is an AI that can successfully play a variety of games in a specific language without a game-specific implementation. These specific languages are called a Game Description Language (GDL). In a GDL, it is possible to model or program a wide variety of games. To play these games accordingly, the models need to adhere to certain coding conventions. If done properly an incredible amount of games can be modeled and later on be used for GGP.

In 2005 Stanford [2] organized a GGP tournament, where students and computer scientists would compete in writing the best AI adhering to the GDL that Stanford has created for this competition. For the competition to work correctly, a game manager needed to be created as well. A game manager is the center of the game, which asks each player for a move input and serves as a communication platform.

The GDL of Groove[2] offers a graphical representation of games. Groove makes use of a graph representation of the current state of a game and combines this with a set of rules also called a grammar. These rules define transformations of the graph. This is a different approach from what most other GDLs use [2]. Most of the GDLs use a logical representation similar to what Prolog[3] has to offer. Like mentioned before, Groove uses an approach of graph transformations to model several instances. A benefit of the graph transformation approach is a simple way of exploring the states and possibilities of the game.

1.1 Problem Statement

Having a GDL for modeling games with the goal of GGP is not enough. Such a GDL needs to fulfill certain criteria to be suitable. Using graph transformations as a basis for a GDL seems promising, but is yet to be confirmed. To conclude whether Groove could serve as a GGP environment certain criteria need to be met. Browne [1] proposed five properties to determine whether a GDL is suitable or not for GGP. These properties are:

- *Simplicity*
  The game descriptions should be easy to write and also be easily modified.
- *Clarity*
  The game descriptions should be readable and comprehensible for the users.
- *Generality*
  The GDL should support a wide variety of games and thus functionalities.
- *Extensibility*
  The GDL should be easy to extend to support new concepts.
- *Evolvability*
  Game descriptions should combine to produce mostly valid (i.e., playable) children with characteristics of their parents.

1.2 Research Question

The question that is going to be answered with this research is: *Is Groove and its Game description language suitable for a general game playing environment?*

[1]Stockfish information: `https://stockfishchess.org/about`

[2]information on Groove: `https://groove.ewi.utwente.nl/about`
[3]information on Prolog: `https://www.swi-prolog.org`

This question will be answered based on the properties of Browne [1]. To answer this question and analyze these properties. The methodology covers which games have been implemented and how the properties have been analyzed. For the Clarity property, an experiment has been conducted. This experiment has been explained in detail how it will be conducted. After this, the results will be presented regarding the game implementations and the experiment. Lastly, each property will be individually analyzed and this is used to come to a conclusion and answer the research question.

2. BACKGROUND

To understand the rest of the paper properly, Groove and graph transformations are discussed in more detail.

2.1 Groove

Groove is a tool that uses graphs for modeling object-oriented programs. Furthermore, it also supports model transformations using rule-based graph transformations. These graph transformations give Groove the possibility to verify model transformations and model checking. While this is the main purpose of Groove it is also possible to model a board game.

2.2 Graph transformations

Graph transformations are rules taking a subset of the current graph and describing how this should be transformed. Each graph transformation rule thus describes a change in the graph. After applying these rules to the graph a new graph is created. In this way, the current graph is the current state of the game. Each of the rules that can be applied represents the possible moves a player can make. Modeling the games in a structured way will result in a board game that can be played. An example of one of these transformations can be found in Fig 1. In this example an edge with label c between a node with label a and a node with label b will be created if and only if there does not exist such a edge yet. The red edge indicates that it may not exist in order to apply the rule and the green edge indicates that the edge will be created with given label.

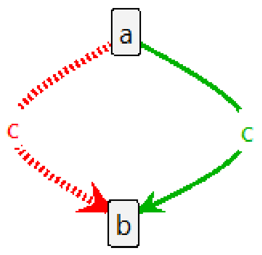

Figure 1. an example graph transformation

3. METHODOLOGY

The research to answer the research question consists out of three parts. The first part is the implementation of the games. This part covers which games will be implemented. The second part is the analysis of the five criteria. To analyze one of the criteria an experiment is conducted, the results of this will then be used to analyze the Clarity of Groove.

3.1 Game implementations

Choosing a good subset of games to use to determine the suitability of Groove for GPP requires a good overview and categorization of the games. Four categories enclose almost all games that are made. In each of these categories, there are still multiple gradations that vary in complexity.

- The first category is perfect information games. Players in these games have perfect information, meaning that players are fully informed of all events that occurred in the game. An example game is Chess or Tic-tac-toe.

- The second category is imperfect information games. These games are exactly the opposite of perfect information games. This means that the players have not a complete overview of the information in the game. An example game would be battleships.

- The third category is perfect information stochastic games. These games contain some form of chance in the game. This may be rolling dice, receiving cards, or spinning a wheel. The perfect information stochastic games are games with perfect information as described in the first category combined with the stochastic elements. An example of this type of game would be ludo.

- The last category is imperfect information stochastic games. These games are stochastic just like the previous category, but instead of having perfect information, these games contain imperfect information. An example of this would be poker Texas Hold'em. Since the cards are distributed randomly each player only knows the cards that were received.

For this research, the games that are implemented will remain restricted to perfect information games and perfect information stochastic games. While this does not give full coverage for all types of games, this still covers a substantial part of games that make General Game Playing possible. the games that are gonna be implemented are as follows.

3.1.1 Perfect information games

The games that will be implemented for perfect information games are Tic-tac-toe[4], Checkers[5], and Chess[6]. Tic-tac-toe will be the most simplistic game that is going to be used, but it already serves as a good indication of how to tackle certain problems. Checkers will be much more advanced than Tic-tac-toe. It requires proper solutions for slaying opponent pieces while using only one piece. Chess will be a much more complex game than Checkers, since this game requires constant checks for legal moves, concerning being in a check position. Chess also offers some unique moves that require multiple pieces and certain circumstances like the Castling move or moving the pawn for the first time.

3.1.2 Stochastic games

For the stochastic games, this will only be Ludo[7]. For Groove, there already exists a working implementation of Ludo. It also contains a standard implementation for the die, therefore Ludo will be used to explore the possibilities of randomized actions.

[4]`https://en.wikipedia.org/wiki/Tic-tac-toe`
[5]`https://en.wikipedia.org/wiki/International_draughts`
[6]`https://en.wikipedia.org/wiki/Rules_of_chess`
[7]`https://en.wikipedia.org/wiki/Ludo_(board_game)`

3.2 Criteria points

To answer the research question, the satisfiability of the 5 criteria needs to be investigated. Using the games that have been previously defined each of these properties is evaluated.

3.2.1 Simplicity

The first property that is investigated after implementing the previously mentioned games is Simplicity. A GDL satisfying the Simplicity criteria has an easy-to-write game description and it should be possible to modify or update current existing implementations without.

After implementing all the games, all difficulties regarding the GDL will be evaluated. To do this properly a distinction will be made between the different game aspects. Firstly, the game logic will be evaluated. This means that for each unique game aspect, a simple and practical solution needs to exist and does not need a large workaround. Secondly, the game design will be evaluated. Points that will be looked at are a simple and effective implementation of the board.

3.2.2 Clarity

The Clarity property will focus on the readability and the comprehensibility of the games. This requires input from outside and therefore an experiment will be set up, see section 3.3 for more information on the experiment. The results of the experiment will sketch a picture of the Clarity of Groove, by combining the results and the reasoning of the participants the satisfiability will be determined.

3.2.3 Generality

For the Generality, the games that were implemented are used to analyze. These games are already chosen in such a way that they already a wide range of functionalities. All these functionalities will be evaluated separately and analyzed what their impact is with other games in mind.

3.2.4 Extensibility

While the other properties mostly focus on the game descriptions, the Extensibility focuses on the GDL and its ability to support new concepts. To test this, the development of Groove will be analyzed to see how Groove is being extended to support new concepts. This can later be used for modeling the games as well. If the process of Grooves development shows that small features can be implemented it will satisfy the Extensibility property.

3.2.5 Evolvability

Evolvability is all about producing valid states and making sure that each move that has been made is valid. Groove supports a state exploration functionality by using this functionality and combining the results of the games the Evolvability can be evaluated. Especially the Chess implementation can give a clear indication of Grooves capacities in this regard since it requires a check after each move that has been made.

3.3 Experiment

One of the properties that will be evaluated is *Clarity.* This checks the readability and the comprehensibility of the game descriptions for its users. To test this properly, an experiment is set up. This experiment will consist out of 3 parts, namely the instructions with an example, the experiment itself, and lastly a feedback moment to address any points the participants might have.

3.3.1 Instructions

The first part of the experiment is the instructions. The instructions will consist out of a general explanation of what Groove is and what graph transformations are. This will not be done in too much detail, because the users do not need extensive knowledge of Groove. This explanation will be done guided by a demonstration program. This program will serve as a guiding tool to prepare the participants such that they have enough information to take part in the experiment.

3.3.2 Game recognition

After the instruction phase, the test phase will start. During this phase, each participant will get to see the same 3 games. These games are Tic-tac-toe, Checkers, and Ludo. For each of these games the same procedure applies. Firstly, the participants get 2 minutes the time to look at the game and try to determine if they know what game it is. If they believe they know what game it is, they are required to explain what game it is and give examples that would make this true, e.g. a certain move or setup of the board. If after the first 2 minutes they haven't figured it out they will get a little help in how to understand it. These will not be game-specific hints, but general remarks that might apply to the game. If after another 2 minutes they haven't been able to give an answer or provide a valid explanation. For the analysis, the results will be divided into the three aforementioned categories:

1. recognized without help
2. recognized with help
3. not recognized

4. RESULTS

This section covers all the results of all game implementations, giving an overview of all functionalities. Furthermore, the results of the experiment are discussed and explained.

4.1 Game implementations

For this research several games needed to be implemented to answer the research question. For each of the games, the new functionalities that are noteworthy are discussed. The games are discussed in the order that they were implemented and discovered.

4.1.1 Tic-tac-toe

The first game that was implemented is Tic-tac-toe. While the game itself is not very extensive, it offered a few possibilities to try out and see how they turn out.

Players

For pretty much all board games the players make moves after one another. Before implementing the game logic the players were modeled. Each player represents a node and has an edge with a label next pointing to the player that has to make a move afterward. Figure 2shows how the players have been modeled in Groove. After each player made a move, the move also removes the turn label at the Player and transfers it to the player which is next.

Priorities

Furthermore, the implementation of this game gave a first glance at how to deal with priorities. In the case of Tic-tac-toe, checking for a winning condition has a higher priority than executing a move. Groove supports a priority system where for each priority (high to low) the number of moves with the highest priority is determined. If there are no

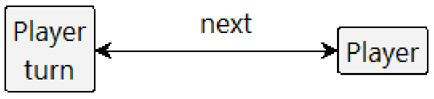

Figure 2. Modeled players

available moves for this priority, moves on a lower priority will be evaluated.

Ending the game
Lastly, are the win conditions as mentioned in Section 4.1.1 like mentioned before. After a game is won, no moves should be played anymore and it should be clear who won the game. This is done by removing the turn label and adding the win and add a label to the winner and loser. Additionally but not required, a score can be awarded to the winner and loser respectively. This can be achieved by connecting the player node with a node of type int. A game only reaches a final node if and only if there are no longer moves applicable indicating and the game is over. This means one of the players won or the game ended in a draw. Which is guaranteed with this implementation.

4.1.2 Checkers

The next game that has been implemented is Checkers, this implementation of this game includes all functionalities that have been described in Tic-tac-toe. Checkers is more complex than Tic-tac-toe and therefore also need additional functionalities that need to be implemented to complete the game.

Multiple moves per turn
The first additional functionality is making multiple moves on a turn. This is the case when you take a piece and after this have to take as many pieces as possible. After a piece is taken, this piece gets the *slay* label, indicating this is the only piece for this turn that can take other pieces. After no pieces can be taken anymore an explicit rule has been created to transfer the turn to the other player and removing the *turn* and *slay* label. This works perfectly as it combines the priorities to make sure the right moves are executed.

Variable move length
After a king is introduced in the game, such a piece can move a variable amount of fields to take pieces or simply move itself. Using the methods like before, this would mean that there must be 9 different rules for this. This isn't necessary as Groove supports regular expressions that make it possible to put this in only one rule. See Figure 3how a king can move downwards.

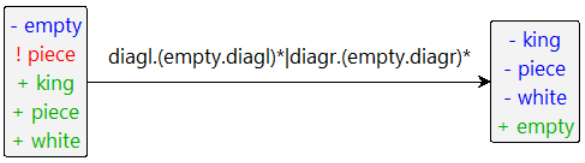

Figure 3. Varying move length

Counting
In Checkers counting is introduced for the first time. To know who won the game, the pieces that are on the board need to be counted. When this number reaches 0 this means one of the players lost the game and a final state is reached. There are two options to keep track of the counting. The first option is assigning a label an integer value like *let: pieces= 20* for initial value and *let: pieces = pieces - 1* for the update. The second option requires manual calculations with the same results as the first option and is not worth further exploring for simple functionalities like these. It will be discussed in a little bit more detail in the Ludo implementation as it proved to be useful there.

4.1.3 Chess

The largest and most complex game that is implemented is Chess. The reason Chess is more complex than the others is that each move can only be valid if after the move your own king is not in a check position. Additionally, this is the first game that introduces much more piece variants, Checkers only has two. To implement Chess with these two new mechanics, 2 Groove functionalities have been used. Besides these 2 functionalities, a combination of previously mentioned functionalities also has been used to model the Chess game.

Recipes
To make sure that each move that has been made does not result in an invalid state, your king is checked, recipes were used. These recipes consist out of one or multiple graph transformation rules. All these rules within the recipe need to be executed, if parts of this fail the entire recipe cannot be executed. While it is possible to execute these rules in the recipe one by one, the state the game is in is not valid and only becomes valid if the entire recipe has been executed. By using the recipes it is guaranteed that a move that has been made is also valid.

Variables
Chess also introduced a new functionality that is necessary to keep the game compact. Since there are a total of 6 different chess pieces, it would be inefficient if there are different rules for all 5 pieces that can be taken (the king cannot be taken). Groove supports the use of variables, which make it possible to define only 1 rule for each piece to take an opponent piece. See Figure 4 for an implementation on how to work with variables. The list after the variable x indicates all values it may not be, this is necessary as it otherwise is also able to take the value of the white label which is undesirable behavior. It is also possible to instead of indicating which values it cannot have, indicate the values it could be.

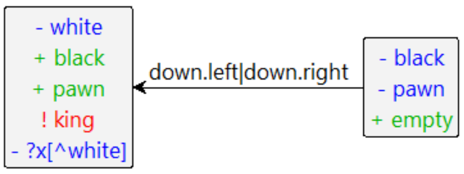

Figure 4. Black pawn takes white piece

4.1.4 Ludo

The last game that was implemented is Ludo. Ludo is a Stochastic game, meaning it got an element of chance in it. In Groove, there already existed an implementation of Groove with working game mechanics. Therefore for the modeling, only the chance element within Groove has been explored. After this two possible ways to implement the dice roll have been explored.

Player
The first option to model the dice roll is by seeing the

dice roll as a third player. This would mean that this additional player randomly chooses one out of six numbers, meaning for the implementation of the dice no chance elements have to be implemented. While this is an easy solution that would make the dice work as it is supposed to, the implementation would mean this third player need to be modeled as well to keep playing the game as intended. The rule in this instance always has 6 possibilities namely one to six.

Generate random number
The second implementation of the dice roll does not require a third player. It namely uses the random int generator that is part of Groove. While this number cannot be ranged from one to six, the modulo of the generated number can be taken. This will result in a number from 0 to 5, so the last operation would be to add 1 to this number. The final number ranges from 1 to 6, the values of a normal 6-sided die. The benefit of this approach is that it doesn't require a third player and instead of having 6 possible rules that can be applied there is only one with a ranging result.

4.2 Experiment

The experiment as described in section 3.3 has been conducted on five test persons. The results of each game will be discussed separately. Additionally to the classifiers that were previously mentioned, three other categories indicate on what basis they recognized the game. A *0* indicates they did not need it to recognize the game and a *1* indicates that they used it to recognize the game. These three categories are board structure, naming(labels and rules), and game logic.
The board structure is the starting representation of the game and how the nodes and edges are positioned.
The naming is all names of the labels and the rule names.
The game logic is the logic behind each of the rules that represent a move in the game.
To conduct the research the test persons gained full control over the computer hosting Groove, this was done to give them as much control over the recognition and exploration of the games.

4.2.1 Tic-tac-toe

Tic-tac-toe was the first game the test persons had to recognize. After a small introduction discussing the essentials of Groove and the mechanics, Tic-tac-toe was shown. While each of the persons managed to recognize the game within 2 minutes, some struggled a little with understanding how the game was modeled. While most of them figured it out without having to look at the game logic, there was one person that used the game logic and more precisely the winning conditions to recognize the game.

Tictactoe	Recognized	Board structure	Naming(labels, rules)	Game logic
person 1	nohelp	1	1	0
person 2	nohelp	1	1	0
person 3	nohelp	1	1	0
person 4	nohelp	1	1	1
person 5	nohelp	1	1	0

Table 1. results experiment Tic-tac-toe

4.2.2 Checkers

After Tic-tac-toe the test persons had to recognize the game checkers. 3 out of 5 people after seeing it for the first time reacted directly that it was the game Chess. After mentioning it all of them immediately redacted their previous answer as they saw this was not correct. The other two immediately did recognize the game, by looking at the board and the naming they clearly indicated that the game was Checkers. The remaining three recognized the game after analyzed the moves that indicated taking another piece. After quickly trying that out, they recognized the move and concluded that it must be checkers. One did indicate that the term king threw them off a little as they associated it with Chess, but after recognizing the game they understood this as well.

Checkers	Recognized	Board structure	naming(labels, rules)	Game logic
person 1	nohelp	1	1	0
person 2	nohelp	1	1	1
person 3	nohelp	1	1	1
person 4	nohelp	1	1	1
person 5	nohelp	1	1	0

Table 2. results experiment Checkers

4.2.3 Ludo

For Ludo, the result was the same for each test person. After being shown the game, they all recognized the unique shape and setup from the game and linked the names of the rules and labels directly to the game, and recognize the game within 30 seconds. They did mention that the unique structure of the game benefited in recognizing the game and clearly separating the players with their assigned pieces. If this wasn't modeled as clearly as it was now, they wouldn't have seen it as fast as they did now.

Ludo	Recognized	Board structure	Naming(labels, rules)	Game logic
person 1	nohelp	1	1	0
person 2	nohelp	1	1	0
person 3	nohelp	1	1	0
person 4	nohelp	1	1	0
person 5	nohelp	1	1	0

Table 3. results experiment Ludo

5. ANALYSIS

With the results of the game implementations and the experiment the five different properties that Browne described can be analyzed. The research question will be answered based on the analysis of the five properties.

5.1 Simplicity

Simplicity is all about simple and easy to write game descriptions. After modeling the different games, it was difficult at first to use the right structures. But this was mostly the cause of inexperience in the program. As soon as multiple features in Groove became clear, the design became much more simple. The functionalities that have been described are a simple way of modeling them and without too much repetition and overlaps, like the regular expressions or priorities. Furthermore, the rules system in Groove makes it easy to create rules or adjust currently existing ones since each rule is separately declared and does not require any links with the other rules. If the functionalities of Groove are clear to the person modeling the games, it becomes a simpler process that results in efficient games. Since the rules and starting graph are easily modifiable implementable satisfies Groove the Simplicity property.

5.2 Clarity

Using the results of the experiment the Clarity property can be analyzed. From the results of the game, each participant recognized the games without help. Which is already an indication of the Clarity of Groove. But other things need to be highlighted as well. The way the participants recognized the games was in most cases because of the clear structures of the board and the clear naming of the labels and rules. the Clarity increases by ordering the graphs and naming everything logically. With the game logic being a bit more advanced, the participants still recognized the games as mentioned before this is not the most important aspect of the game descriptions. So by looking at the results, the recognition, and the factors that determined the recognition, Groove also satisfies the Clarity property.

5.3 Generality

Generality is more about the bigger picture of the GDL. Since the GDL should support a wide variety of games and functionalities. For the games that have been implemented all different functionalities have been listed. All of these games have been successfully implemented, all using a different range of functionalities. While there already have been plenty of functionalities, not all possibilities in Groove were needed. Wherewith each in complexity increasing game no problems in the design have been found. It is highly likely that with other, perfect information, games no problems would be encountered. Therefore Groove satisfies the Generality property.

5.4 Extensibility

Groove already contains a large variety of features that make it possible to model different kinds of games, as can be seen from the results of the game implementations. While several features of Groove have been explored not all features have been used to create the games. Seeing that most features proved to be useful for certain types of implementations, it likely that those unexplored features can help to implement other game mechanics. Furthermore, these features in Groove have been growing over time for one can argue it relatively easy to extend to support new concepts, may it be for existing features or new ones. Because of the aforementioned points Groove satisfies the Extensibility property.

5.5 Evolvability

During the modeling of the games, it is important that no faulty moves can be made. With the several games that have been modeled, there have been two possibilities that make sure this is not the case. Firstly the priorities make sure that if a certain move can be made, this must be done and not a move that has a lower priority. Secondly, the recipes proved to be really useful for checking the validity of a move, by executing everything that is inside the recipe. If at some point a rule cannot be applied the recipe cannot be applied as a whole. This was extremely useful for implementing the check for the check rule in Chess. Both of these implementations make sure that the game cannot end in a non-playable state. The only way the game ends in a final state is when a player wins or it ended in a draw. Since this always applies when the game is modeled correctly using these methods, Groove satisfies the Evolvability property.

6. CONCLUSIONS

With the game implementations and the analysis of the properties, there are several things to take away from. The GDL of Groove proved to be capable of modeling the games that were planned to be modeled. Although, this does not mean that the GDL of Groove is suitable for all types of games. Since the games that have been implemented and analyzed were perfect information games and a take on chance. Therefore the suitability of Grooves GDL for imperfect information games cannot be determined based on this research. But the GDL of Groove does satisfy the five properties that have been presented and analyzed. So we can answer the research question that has been stated at the start of the paper. *Is Groove and its Game description language suitable for a general game playing environment?* The answer to the question is yes, but this is only for the perfect information games and stochastic games which have perfect information.

7. DISCUSSIONS

This research gives a good basis for using Groove as a GGP environment. There are still a lot of parts that could have been different. Because of the limited time of this research, the pool of games that were implemented stayed small. To give a more convincing answer, a larger set of games could have been used to analyze. The same can also be said about the experiment that has been conducted. Currently, the experiment only had 5 participants which limit the diversity in the participants which could lead to a biased result. By extending the experiment with more participants, but also increase the games that need to be recognized the result becomes more convincing than it currently was.

Lastly, this research only focused on the perfect information games. This keeps the imperfect information games left to explore for the GDL of Groove. This research could serve as a good basis for future research that looks into imperfect information games. This research can also serve as a basis to create a game manager that would bring the GGP into practice in Groove.

8. RELATED WORK

In 2005, Stanford University created its own GDL (S-GDL) having Prolog-logic-based representation. They challenged computer scientists all over the world in a competition for the best search algorithm [2]. Stanford would host this competition annually until 2016. While the GDL of Stanford is the most well known GDL to date, they have not been the first to propose a Prolog-logic based representation for a GDL Koller & Pfeffer [4] were the first in 1997.

While the S-GDL remains the most accepted GDL to date, multiple other GDLs have been proposed that do not take the logic-based representation, one of them uses undirected graphs to model the games [3], but instead of using the graph transformations, like Groove, to model the different actions, matrices are used.

Furthermore, Tagiew [6] describes a GDL which bases its representation on Petri nets. This is a place and transition network, this has quite some similarities to how graph transformations work. These Petri nets describe a set of transitions that are applicable if all places in here contain a token that will then fire and result in a new state. Which in a lot of ways overlaps with the functionalities of graph transformations.

9. REFERENCES

[1] C. Browne. A class grammar for general games. In A. Plaat, W. Kosters, and J. van den Herik, editors, *Computers and Games*, pages 167–182, Cham, 2016. Springer International Publishing.

[2] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62, Jun. 2005.

[3] M. Kearns, M. L. Littman, and S. Singh. Graphical models for game theory, 2015.

[4] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, 1997. Economic Principles of Multi-Agent Systems.

[5] S. J. S. K. Silver, D. Mastering the game of go without human knowledge. *Nature*, (550):354–359, 2017.

[6] R. Tagiew. Multi-agent petri-games. In *2008 International Conference on Computational Intelligence for Modelling Control Automation*, pages 130–135, 2008.