General Game Playing with imperfect information in GROOVE

Jorrit van Assen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands

ABSTRACT

General Game Playing is concerned with the exploration of strategies that perform well over a multitude of games. To make sure that these games can be understood by the strategies, a general description language is required to capture the games. A range of languages have been proposed, most notably the Game Description Language of the Stanford Logic Group. In recent years these languages were used for the research into games with imperfect information. Groove grammars can be used to capture systems in graphs and graph transformations. This would allow for application of graph tools on the games and its strategies. This paper proposes a structure for describing perfect and imperfect information games using Groove grammars. This structure will be used to model Connect Four and Simultaneous Krieg Tic-tac-toe. The models will be verified by matches between three types of general game players: a legal move, random move and a Lookahead player.

Keywords

General Game Playing, Imperfect Information Games, GROOVE

1. INTRODUCTION

A lot of computer players are designed for a single game. This means that their respective strategies are optimised for one specific game and are not suitable for others. For a lot of games, optimal strategies have been researched and a lot of knowledge has been gathered from analysing those games. For some games, computers have vastly outperformed human players. Chess is an example of this. Stockfish 13, one of the strongest computer engines in the world, is a far stronger blitz player than even the best blitz professional in the world [2, 7]. However, Stockfish 13 is only good at playing chess variants. General Game Playing (GGP) takes a more general approach to generating strategies.

General Game Playing is a research topic in which computer controlled agents are developed which perform well in a range of games, which they have possibly not played before [5]. An AI which could learn from doing one task and apply this knowledge to another would be beneficial. Instead of training a new AI from scratch for every new scenario, the problem is described in a specific structure and the AI would be designed to be able to solve problems in this structure. This kind of intelligence is more similar to how humans approach new problems. We try to structure the problem and solve it using similar situations in the past. The ability of computers to learn like humans is called General Artificial Intelligence. By researching General Game Playing it is hoped to come closer to creating General Artificial Intelligence [13].

35th Twente Student Conference on IT Jul. 2nd, 2021, Enschede, The Netherlands.

One of the main requirements for General Game Playing is thus that the language can be used to model a wide variety of games. One of the most used languages is the Game Description Language (GDL) developed by the Stanford Logic Group of the Stanford University. One of the reasons why GDL is widely used by GGP algorithms is because of its use in the International General Game Playing Competition. This was an annual competition organised by the Stanford Logic Group from 2005 until 2016. GDL uses logic to describe the game using an initial state, winning conditions, as well as the legal moves of the players [9]. Reasoners can be used to discover legal moves for a given game description and state. These reasoners are utilised by the General Game Players in order to find the best moves for a player. The players employ different tactics for evaluating the moves. In 2007, CadiaPlayer won the competition using a Monte Carlo Tree Search with Upper Confidence bounds applied to Trees (UCT) simulation technique for move selection [3], a technique that uses random playouts of the match to estimate the quality of the moves. This technique has later been successfully combined with deep learning for AlphaGo [14]. The last competition in 2016 was won by WoodStock, who converted GDL into Stochastic Constraint Satisfaction Problems which could be solved to find optimal moves [8].

The last decade has seen an increase in research done into General Game Playing, specifically in the research into games with imperfect information. For games with perfect information, the current state of the game is fully visible to all players, but for imperfect information there is an asymmetry in the knowledge of the players. An example of a game with imperfect information is *Texas hold 'em*[1]. In this game the players do not know their opponents cards and which cards are in the flop, river or turn. The players have to adapt their strategies and take into account the possibility of an opponent with a strong hand. General Game Languages need to be adjusted to allow for those type of games. Thielscher describes in 2011 an extension for GDL called GDL-II which allows for imperfect information games to be modelled [15]. A more recent language is Ludii, proposed in 2020 as part of the Digital

[1]`https://en.wikipedia.org/wiki/Texas_hold_'em`

Ludeme Project [11]. This language can also be used both for perfect and imperfect infomration games.

GROOVE is a toolset created for the use of verifying object-oriented systems using graph transformations. Because the tool was designed with extensibility in mind [12] the toolset has evolved and is now used for many more purposes. It can now be described as a "general purpose graph transformation tool" [6] enabling its users to create graphs and rules to explore the state space. This feature makes it suitable for playing board games. Game rules can be translated into graph transformations and the initial state as a graph. Afterwards, GROOVE can be used to simulate the game and inspect the legal moves and states. This is a rather intuitive way of programming the rules of the game and could serve as an alternative to Game Description Languages like GDL-II and Ludii. Although these last two specifications are optimised for games, the generality of GROOVE could also prove more flexible.

The goal of this paper is to investigate how Groove can be used for General Game Playing with imperfect information. This leads to the following questions:

1. How can GROOVE be connected to a General Game Playing algorithm with perfect information?

2. How can games with imperfect information be modelled in GROOVE?

3. How can a General Game Playing algorithm be adapted to play games with imperfect information?

In this paper, we propose a set of conditions under which Groove grammars representing perfect and imperfect information games can be played using our match-manager. We demonstrate this by modelling two example games in Groove, Connect Four and Simultaneous Krieg Tic-tac-toe, and playing these games using three general game players. The paper first introduces Groove and the requirements for General Game Playing in section 2. Section 4 proposes the conditions for the Groove grammars for managing perfect information games, three simple general game players and the describes the implementation for Connect Four. Adjustments to the conditions for the grammars and subsequent changes to the players are made in section 5. This section also describes the implementation of Simultaneous Krieg Tic-tac-toe. In section 6 the games and players are validated, related work is described in section 7 and the findings of this paper are discussed in section 8. Finally, the paper is concluded in section 9.

2. BACKGROUND

2.1 Groove

Groove is built around the concept of Graph Transition Systems. All states of a GTS are captured in graphs [12] and the states are connected by transitions. Transitions apply a certain rule to a source to reach a target state. The grammar consists of initial states and rules and Groove can be used to explore the state space of the grammar. An example of a partly explored state space is shown in figure 1.

In a GTS the initial state is captured using a graph. This graph consists of nodes and edges. Each node has an identity and, possibly, a type. Nodes can have labels or flags, which is stored as a self edge, or an edge to another node. All edges require a label. An example of a state is in figure 2.

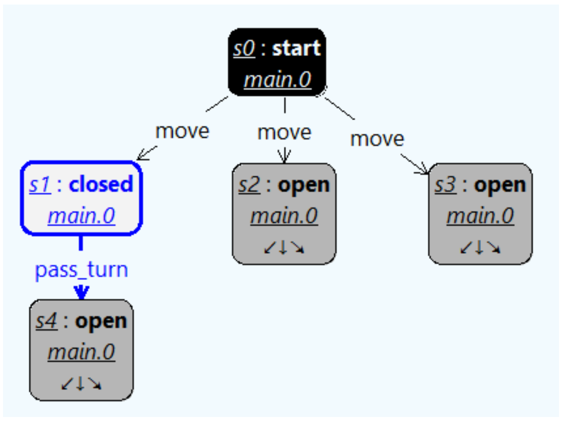


Figure 1. A partly explored state space of three by two Connect Four.

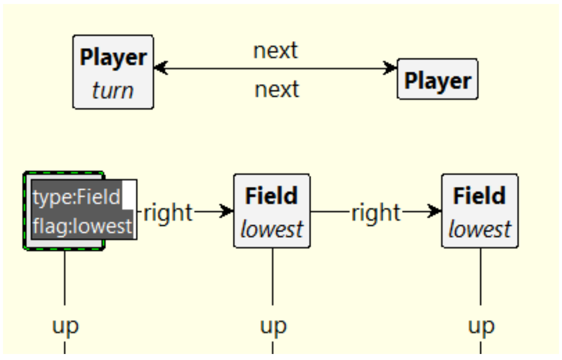


Figure 2. A three by two Connect Four start graph in Groove.

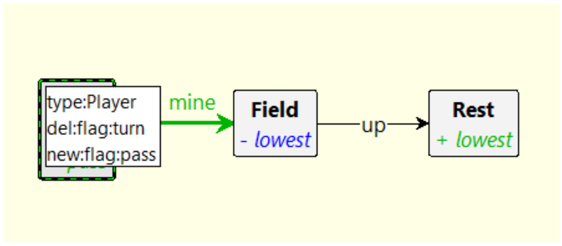
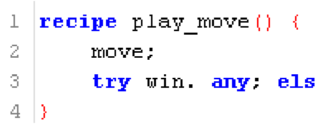


Figure 3. A graph for the "move" rule of Connect Four in Groove.



```
recipe play_move() {
    move;
    try win. any; else try pass_turn;
}
```

Figure 4. Fragment of the Control program of Simultaneous Krieg Tic-tac-toe.

Rules are conditionally applied to states. For every match it allows a transition from a source state to another state. When the source state is the same as the target state, the rule is called a 'graph condition'. Rules which change the graph of a state are called transformers. An example of the transformer "move" is given in figure 3. Groove makes use of colours to model how nodes, edges and labels are affected by a rule. Nodes, edges and labels coloured black are required to be present in the graph in order for the rule to apply. The opposite true for the colour red. The colour blue is similar to black, however, when the rule is applied, everything coloured blue will be deleted. Green is used to create nodes, edges and labels when the rule is applied. If a rule can be applied in multiple ways and result in multiple unique states, multiple transitions will be created. In the start graph in figure 2, the rule "move" can be applied three times. In figure 1 the three transitions created by applying the rule to the start graph can be seen.

Groove allows for finer control of rules using Control. It can define the order in which rules should be applied, if rules can be repeated and combine rules into composite rules. The composite rules can occur either as recipes or functions. Where recipes require all steps to be applied successfully, functions can apply steps until the function is completed or one of the steps fails. A fragment of an example control program is shown in figure 4. Groove supports a wide range of Control syntax, but a small section will be highlighted. To call a rule, the name of the rule can be used. To try a rule the "try" syntax can be used. If the rule cannot be applied in any way, the optional else block is executed. The "any" command allows for the application of any rule inside of a directory. To repeat a command or block of commands as long as possible, "alap" can be used.

GrooveChannel is a an extension for Groove allowing external programs to communicate with Groove through an API [1]. When a client connects, it is first required to send a gps file containing the Groove grammar. This file must be zipped and encoded in base64. After this has been successfully done, the "state", "info", "rules" and "exit" commands become available. The "rules" command returns a JSON representation of all rules in the grammar. The "state" command takes a state number as an argument and returns the state graph with the corresponding state number. The state graph is encoded in JSON format and is described by nodes and edges. Similar to the "state" command, the "info" command also uses the state number as an argument but returns the possible transitions instead. These transitions contain a source state number, name and target state number. For both commands the state number argument is restricted to state 0 or states already discovered by the info command. For example, sending "state 2" is only allowed if a previous info command contained a transition to state 2. It is important to note that states numbers are dynamically assigned on a connection basis for the states are numbered in order of discovery. This means that the previous state numbers are lost when a new connection is made.

3. APPROACH

To show that Groove can be used for General Game Playing with and without perfect information we will demonstrate that Groove grammars can be used for managing general games and playing general games. To manage games with perfect information described in Groove grammars, we first investigate constraints to capturing these games in Groove grammars. These grammars ensure that the match-manager understands the state of the game and can determine what moves are allowed for the players. To demonstrate that perfect information games adhering to these constraints are manageable, we will model and play Connect Four. The constraints for the grammars will be extended to allow for the managing of games with imperfect information. Managing imperfect information games will be shown by modelling and playing Simultaneous Krieg Tic-tac-toe.

For both perfect and imperfect information games, three players will be created. A legal player and a random player will serve as naive players. An additional player will be implemented that can use Groove to perform better than the naive players. These players will be designed as a general game players, and it strategy will not be based upon knowledge of the game.

The games will be validated using their respective players. All players will play one another as both the starting player and the second player. Every match-up will play 1000 games. If the last player is able to significantly play better than the random player it demonstrates that Groove can be used for playing general games.

4. GENERAL GAME PLAYING WITH PERFECT INFORMATION

4.1 Managing games

To be able to understand the states of a game, its modelling is subject to a few requirements. Each node with type "Player" is a player slot. In order to determine which player is to move in each state, the slot of the current player has a flag "turn". Only one turn flag is allowed and at each non final state, one player needs to have the turn flag. To determine the winner of the game the win flag is used. This flag is only allowed in a final state and is not required to be present. It is also possible for multiple players to have the win flag.

Transitions are used to represent moves. All transitions in a state are legal moves the current player can make. Final states are not allowed to have transitions to other states. These requirements ensure that the state space of the grammar represents the state space of the game. Every state is a legal state, and every transition is a legal move in the game.

The match-manager is responsible for managing a single match. It is initialised using a gps base64 string and a list of players. A connection is made with Groove using GrooveChannel and the initial state of the game is explored. The player slots are identified and matched to the list of players. All players are then initialised and receive their player identifier and the shared connection object to Groove. Afterwards the match-manager starts the match and requests moves from the player as described by the GPS.

4.2 Modelling the players

To model general game players for perfect information games, we will first need to discuss how the player knows about its position in the game. Because all information about the state of the match is available to all players, the players can be allowed to reason with the connection of the match-manager. The match-manager requests a move from the player in a certain state number. The player can retrieve information about the state or the legal moves using the state number for as long as the connection is shared.

The Legal Player uses the connection to retrieve all possible moves in the current state. The first legal move is picked. It should be noted that the order in which Groove presents the move is not always the same in all states; What column is chosen for the first move Groove returns dependence on the state. However, the ordering of the moves is not random and does not differ between multiple matches for the same game. For a game with perfect information we expect the legal player to always get the same result against itself, for every time the same path will be taken along the game tree.

The Random Player also uses a connection to retrieve the possible moves for the current state. When a move is requested by the match-manager a random move is picked from all possible moves with a uniform distribution.

The Lookahead player is a very simple improvement upon the Random Player. Whenever a move is requested, it evaluates the game tree with a depth of two moves. If there exist a move in the current state that would allow the Lookahead player to instantly win, that move is picked immediately. If no such move exists, it will look if any move could allow another player to instantly win. It will disregard all these moves and pick a random move from the remaining if possible. If a forced win by the opponent cannot be avoided, it will pick a random move from the legal moves.

4.3 Modelling Connect Four

As perfect information game Connect Four[2] was chosen. This is a game for two players where each player makes a move after the other. All players have the same information and can see the moves of the opponent. The board consists of seven columns and six rows. The player who has the current turn chooses one of seven columns and drops a piece with his or her colour. The pieces stack and a column cannot be chosen once it contains six pieces. When all columns are full the game is a draw. A player wins if 4 pieces of his or her colour are connected either horizontally, vertically, or diagonally consecutively.

There are two player slots and one of the has the turn. The graph for a board with 4 columns and 3 rows is shown in figure 5. The actual board with six rows and seven columns is modelled using a similar fashion.

The flow of Connect Four can be described using a few steps.

1. The player whose turn it is drops a fiche in one of the available columns.
2. The board is checked for a winning position. The player with the winning position is marked as the winner.
3. If no winning position has been found, the next player will receive the turn.

All these steps can be modelled using one or more rules.

The move step (step 1) is modelled in the rule called move, see figure 6. The rule requires a player node with a turn flag and a field node with a lowest flag connected to another node using a up edge. The player loses his turn by taking a field and instead is flagged using the ”pass” flag. The field loses his lowest flag and passes it to the node above.

The win step (step 2) is modelled using four separate rules. One rule for a horizontal connection, one rule for a vertical connection and two rules for the diagonal connections. The horizontal win rule requires four fields directly connected using right edges and a player with the pass flag, see figure 7. If this has been found the player loses its pass flag and receives a win flag. The other rules are constructed in a similar fashion.

The final step (step 3) is responsible for passing the turn of the player, see figure 8. The player loses its pass flag and the player connected with a next edge gains the turn. The turn can only be passed if the next player has a legal move. This is the case if there exists a field that is flagged as lowest.

To make sure that the move rule can be applied to the last

[2]https://en.wikipedia.org/wiki/Connect_Four

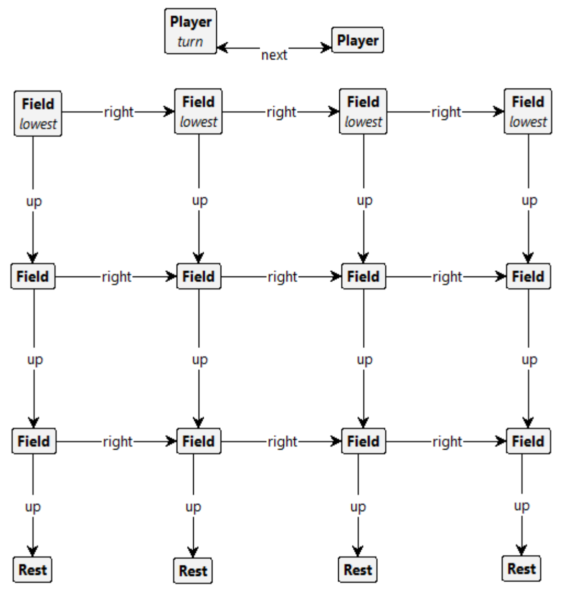


Figure 5. Starting position of Connect Four for 4x3 board.

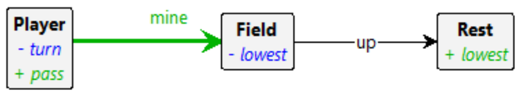


Figure 6. The move rule of Connect Four.

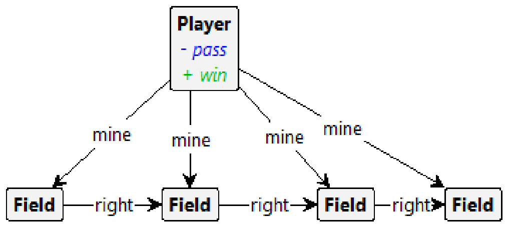


Figure 7. The horizontal win rule of Connect Four.

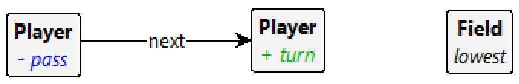


Figure 8. The pass turn rule of Connect Four.

```
recipe play_move() {
    move;
    try win. any; else try pass_turn;
}
```

Figure 9. The control of Connect Four containing the play_move recipe.

row, a row of empty nodes is connected, as can be seen in figure 5. Once all fields of a column are taken, the move rule cannot be applied to that column.

Because the game-manager sees every transition as a move, the three steps have to be combined. We want a unique transition for every possible way the move rule can be applied. We therfore make use of Grooves control functionality. This control defines a recipe called "play_move", see figure 9. This recipe starts with calling the move rule, which means that a play_move transition is created for every transition that the move rule would have. It is possible that a move can result in a position where two winning conditions are met. To ensure these rules do not cause additional transitions we group the winning rules using the "any" syntax. Every state should be checked for the winning conditions, but they are not required to succeed. The "try" keyword let us try to apply a condition. If the game has not been won, the pass_turn rule is tried.

5. GENERAL GAME PLAYING WITH IMPERFECT INFORMATION

5.1 Adjustments to games

To allow for games with imperfect information, a few adjustments have to be made to the GPS. First, the asymmetry of information has to be modelled. The match-manager always has all information, but the information the players have can be limited. For perfect information games the connection to Groove could be shared, however, for imperfect information games this would give the players too much information. Instead of initialising the players with the shared connection to Groove, the players receive the GPS string. This requires them to use an own reasoner to allow them to reason with the current state.

As previously discussed in section 2.1, Groove will number states based on discovery. This means that two players who reason about the game can have different numbers for the same state. The transitions cannot be used anymore to communicate moves between players. To solve this, GrooveChannel was slightly modified. In addition to giving the source, rule name and target it also gives the anchors of a rule. Anchors are used in Groove to identify what nodes and edges are affected by a rule. Because recipes do not have recipes, it is possible to use an argument instead.

When a player is to move, the match-manager notifies the player. The player receives a filtered version of the current state. The filter is a special transition that removes all information from the state which should be hidden for the player. Exactly one filter transition is required in every state where a player has the turn. In both the orig-

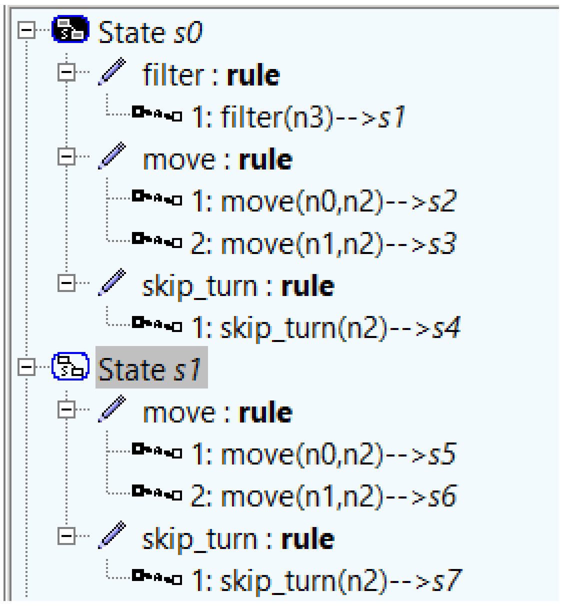

```
State s0
    filter : rule
        1: filter(n3)-->s1
    move : rule
        1: move(n0,n2)-->s2
        2: move(n1,n2)-->s3
    skip_turn : rule
        1: skip_turn(n2)-->s4
State s1
    move : rule
        1: move(n0,n2)-->s5
        2: move(n1,n2)-->s6
    skip_turn : rule
        1: skip_turn(n2)-->s7
```

Figure 10. An original state and a filtered state in an imperfect information game.

inal state and the filtered state, the non-filter transitions should be equivalent. An example is given in figure 10. Every transition that is present in *State s0* is also present in *State s1*, apart from the filter.

To introduce chance in the game, the possibility of moves by nature is introduced. This is not required by all imperfect information games. Krieg Tic-tac-toe, for instance, does not have moves by nature. However, many imperfect information games do have some form of random chance, this includes simultaneous Krieg Tic-tac-toe. Instead of a player node, a node with type random receives the turn. This will be recognized by the match-manager and it will choose a single transition out of all possible transitions using a uniform probability.

5.2 Modelling Simultaneous Krieg Tic-tac-toe

Game chosen as test for playing imperfect games is Simultaneous Krieg Tic-tac-toe. This game is a variant of Krieg Tic-tac-toe where the requirement of simultaneous moves is added. Krieg Tic-tac-toe is a widely used example of a game with imperfect information. It differs from traditional Tic-tac-toe[3] by restricting the observable pieces to the respective owner. If a player chooses a slot occupied by the opponent, the piece is discovered and the player is allowed to try another slot.

The Simultaneous variant of Krieg Tic-tac-toe requires the players to requires the players to choose a slot at the same time. If both players choose the same slot, a coin flip decides who gets control over the slot. The other player will learn that the other player controls the slot. If different slots are chosen, a player will either control the slot if not already controlled by its opponent or receive the information that the slot is already taken. Slots that have been chosen in the past cannot be chosen in subsequent rounds by the player. A player wins if it controls 3 adjacent slots horizontally, vertically or diagonally. When both players win in the same round or they are both out of moves the game is a draw.

[3]`https://en.wikipedia.org/wiki/Tic-tac-toe`

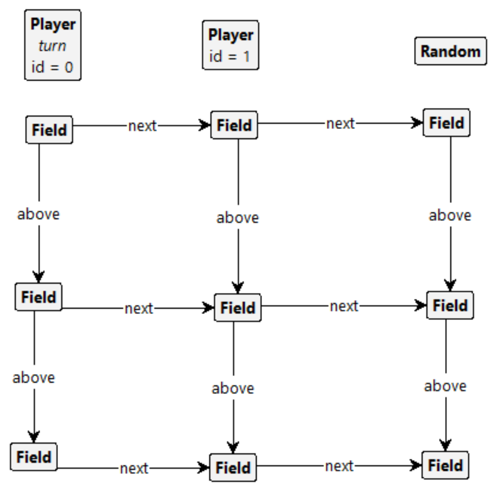


Figure 11. The starting state of Simultaneous Krieg Tic-tac-toe.

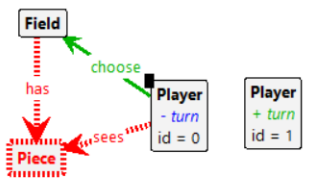


Figure 12. The rule that allows player 1 to choose a field.

Just as with Connect Four, the game has a rectangular board with connected fields, see figure 11. Two player nodes, as well as a random node is created. The players are identified using an id. The first player receives the turn.

Although the game is played simultaneously, the GPS is modelled consecutively. This means that one of the players chooses its move before the other. For the players this does not matter, since the opponents choice can be filtered from the state they receive. The rules used for modelling the player moves are separated. Choose1 allows player 1 to choose a field and pass the turn to player 2, see figure 12. Choose2 has the same functionality but passes the turn to random instead.

The move by nature is modelled using control and consists of three steps.

1. Remove the turn flag from the random player.
2. Either allow or deny each player to take the field they chose.
3. Check if the game is won or drawn or give the turn to the first player again.

These three steps are combined in four separate recipes: deny_both, take_first, take_second and take_both. The choice was made to distinguish these recipes, to allow the moves to be recognized by name alone. The first and last step are similar to the win rule and pass turn rule of Connect Four.

To allow a player to take a field, the rules take and take_arg are used. The graph of rule take_arg is shown in figure 13. The rule takes as argument an integer. This integer must be the same as the id of the player. If the field is unoccupied, a piece will be linked to the field which is owned and seen by the player. The normal take rule does not take the id of the player and can therefore match any of the player nodes.

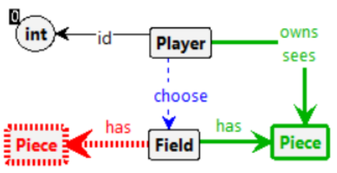


Figure 13. The take rule of Simultaneous Krieg Tic-tac-toe.

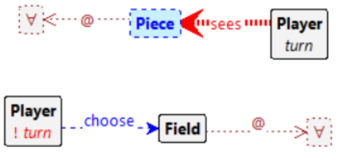


Figure 14. The filter rule of Simultaneous Krieg Tic-tac-toe.

The deny rule used is like the take rule. The difference is that a piece already has to exist for the field. Additionally, after application of the rule, the player will not own, but only see the piece.

To allow for filtering of the states a single filter rule has been made, see figure 14. For the player that has the turn, all piece nodes that are not seen are removed and all choose edges from the other player. The choice for separating the field node and the piece node was made to allow for easy filtering.

5.3 Adjustments to players

The change from perfect to imperfect information games has no effect on the workings of the Legal player and Random player. Their decision-making only relies on the available moves in the actual state. Because filtering the actual state does not affect the available moves, the moves that are available in the filtered state represent all the moves in the actual state.

This, however, is not sufficient for our Lookahead player. Although it knows what moves are possible in its current state, it is not sure if that move will result in a win. Going one step further, it knows even less about the legal moves that could allow the opponent to win in his next turn.

It is possible to reason about the actual state of the game. The filtered state only removes nodes and edges, meaning that all nodes and edges present in the filtered state, are

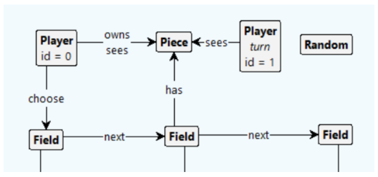


Figure 15. The state of a possible match as seen by the match-manager.

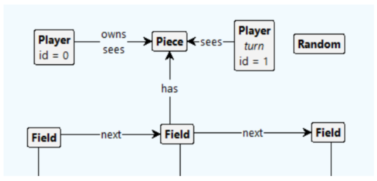


Figure 16. The same state in figure 15, filtered for the second player.

Table 1. Simulation of 1000 matches of Connect Four either resulting in a win for player 1, a draw or win for player 2.

vs.	Legal 2	Random 2	Lookahead 2
Legal 1	0/1000/0	413/0/587	175/1/824
Random 1	666/2/332	563/3/434	288/0/712
Lookahead 1	865/0/135	842/0/158	594/0/406

also present in the actual state. This can be demonstrated with the following example in Simultaneous Krieg Tic-tac-toe. The two players have chosen the same field in the first turn. Afterwards, the first player chooses a second field. It is now up to the second player to make a move. A slice of the current state is shown in figure 15. The second player, however, will perceive the state as shown in figure 16. After a single round and a choice by the first player, the game could be in $(8*9+18)*8 = 720$ different states. But by reasoning, this number can be reduced to only 8. The player knows what moves he has played in the past. Given that seeing but not owning a piece after its first move requires that both players chose the same field, the number of possible states is reduced to every move the first player could have made.

This kind of reasoning about possible states can also be implemented for a look ahead player that uses Groove. When it receives the gps file, it can find all states in which he would receive its first turn. When he receives his turn by the match-manager, it will compare all of these states with the filtered state. Because the filtered state is a subgraph of the original state, it can eliminate any state for which this does not hold. Finally, it will choose a move based on all remaining states and calculate all descending states of the remaining states in which it is to move.

This is not a scalable approach to a general game player, however because Simultaneous Krieg Tic-tac-toe is not a very complex game, we expect reasonable performance.

6. VALIDATION

6.1 Connect four

For each combination of computer players 1000 matches of Connect Four were played. All of these games ended in one of three possible outcomes. The outcome of these matches are in table 1. The number after the player type denotes whether the player was first to play, or second to play. The total win percentage of all players for all positions are shown in table 2.

Table 2. Win percentages of all players for the 1000 matches simulated.

	as player 1	as player 2	combined
Legal	19.60%	15.57%	17.58%
Random	50.57%	39.30%	44.93%
Lookahead	76.70%	64.73%	70.72%

Table 3. Outcomes for the starting player in Connect Four.

	n	win	loss	draw
Groove	1e3	56.3%	43.4%	0.3%
Cato	1e8	55.58%	44.17%	0.26%

Table 4. Simulation of 1000 matches of Simultaneous Krieg Tic-tac-toe either resulting in a win for player 1, a draw or win for player 2.

vs.	Legal 2	Random 2
Legal 1	448/72/480	425/107/468
Random 1	472/121/407	478/112/410

According to a sample of 1000 games, the chance of winning Connect Four is 56.3% when starting or 43.4% when playing second. Additionally, 0.3% of the games were a draw. Although no publication were found mentioning win or draw percentages of Connect Four, the draw percentage was discussed on the Mathematics StackExchange site [10]. A program provided by the user Cato could be used to find this draw percentage, as well as the winning percentages, for random play-outs[4]. This program was ran to simulate one-hundred million matches. The results are shown in table 3, alongside the results of the random player in Groove. The winning percentage of both Groove and Cato are compared. Because the number of samples of Cato's program is extremely large, those numbers are taken as the actual chances. The Z-score was calculated using with $\hat{p} = .5558$, $\hat{p}_1 = .563$ and $n_1 = 1000$:

$$Z = \frac{\hat{p}_1 - \hat{p}}{\sqrt{\hat{p}(1-\hat{p})/n_1}} = .458 \tag{1}$$

This gives a p-value of 0.6447.

To compare random play with informed play, we need to determine for which winning chance is considered significantly higher or lower than the chance of winning randomly. The 99% confidence interval of the chance to win as a starting random player can be calculated as follows:

$$p + -2.33 \cdot \sqrt{\frac{p(1-p)}{n}} =$$
$$0.5558 + -2.33 \cdot \sqrt{\frac{0.5558(1-0.5558)}{1000}} =$$
$$[0.5153, 0.5963]$$

For playing second, the 99% confidence interval of the win-percentage is:

$$p + -2.33 \cdot \sqrt{\frac{p(1-p)}{n}} =$$
$$0.4417 + -2.33 \cdot \sqrt{\frac{0.4417(1-0.4417)}{1000}} =$$
$$[0.4013, 0.4821]$$

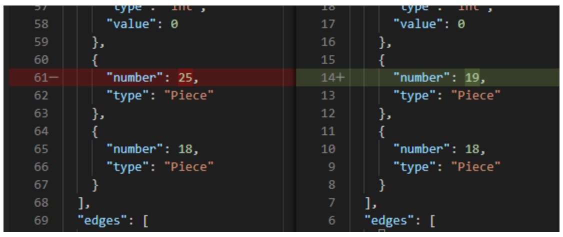
The win percentage of the Lookahead player is 84.2% and 71.2% for playing first and second respectively.

6.2 Simultaneous Krieg Tic-tac-toe

The results for the matches of Simultaneous Krieg Tic-tac-toe are shown in table 4. The Lookahead player was omitted from this table, because it was unable to reason with the game.

Whenever the random player plays itself, the starting random player won more games. Out of 888 games won by

[4]`https://dotnetfiddle.net/5Ja1JS`



```
      "type": "int",
      "value": 0
    },
    {
      "number": 25,
      "type": "Piece"
    },
    {
      "number": 18,
      "type": "Piece"
    }
  ],
  "edges": [
```

```
      "type": "int",
      "value": 0
    },
    {
      "number": 19,
      "type": "Piece"
    },
    {
      "number": 18,
      "type": "Piece"
    }
  ],
  "edges": [
```

Figure 17. Two equivalent states received by different connections to Groove.

Table 5. Results of 150 matches for the Lookahead player against the random player.

win	draw	loss
39	31	80

either player, the player that was allowed first to move won 472/888 = 53.15% of the games. Simultaneous Krieg Tic-tact-toe is a game which should be played simultaneous, but is modelled consecutively. A significant advantage for one of the player could mean a faulty implementation. We define our Null hypothesis as $H_0; p = .5$ and our alternative hypothesis as $H_1; p \neq 0.5$.

$$\mu = p_0 \cdot n = 444$$
$$\sigma = \sqrt{n \cdot p_0 \cdot (1 - p_0)} = 14.90$$
$$P(Z > \frac{471.5 - \mu}{\sigma}) = 3.21\%$$

Because 3.21% > 2.5%, we are not able to reject the null hypothesis with a 95% certainty.

The Lookahead player was not able to reason with the implemented version of Simultaneous Krieg Tic-tac-toe. The issue was avoided by creating an adjusted version. The Lookahead player played 150 matches against the Random player of which the results are given in table 5.

Although reasonable performance was expected given the simplicity of the game, the Lookahead player was significantly slower than both the Legal and Random player. Playing 150 matches between two random players took 24 seconds, whereas playing the 150 games shown in 5 took 1764 seconds. An example of the number of states the Lookahead player considered is given in 6. For every move, "before" depicts the number of possible states the player could currently be in and "after" depicts the number of possible states the player could be in for the next turn. The difference between the number of states between move 1 after and move 2 before is caused by the application of the reasoning.

7. RELATED WORK

Table 6. Example of possible states considered before and after moving.

move	before	after
1	9	80
2	72	560
3	112	684
4	516	2120
5	790	2688
6	392	924

General Game Playing in Groove is also being researched by Daniel Floor [4]. However, his focus is on the modelling of complex games and researching the understandably of the games for humans.

GDL is one of the most common Game Description Language. Although, published in 2006 by Love et al. as a language for perfect information games only[9]. It was extended multiple times. First in 2011, to allow for imperfect games [15], and later in 2017, to allow for epistemic reasoning [16]. Although its use is widespread, other alternatives have also been developed.

Ludii is another Game Description Language and reasoner. It is developed as part of the Digital Ludeme Project which has the goal to "model the world's traditional strategy games in a single, playable digital database.". Ludii is designed as the language which will be used to model the games. The tool is designed with both efficiency and readability in mind. Like our system, the language allows for both expressing perfect and imperfect information games. However, instead of graphs it uses text to model the games. The syntax is more concise than GDL [11]. They conclude that Ludii outperforms GDL and is competitive with RGB in terms of reasoning efficiency.

8. DISCUSSION

The results of Connect Four are as expected. The match-manager was able to manage the game without errors. All 1000 matches where the legal player played itself, were a draw. Because the moves of the Legal player should be predetermined, the outcome of a single match should always be the same. Additionally, the win percentages of the random player did not significantly differentiate from the win percentages given by the external program of Cato. All these findings suggest correct modelling of the game and implementation of the match-manager.

The general game player for the perfect information game also performed well. The win percentage of the Lookahead player was well above the 99% confidence interval of random win chance when starting. For playing second, the win percentage also was above the confidence interval. This means that the Lookahead player performed significantly better than the random player, from both positions. Informed decision-making using groove, without prerequisite knowledge of the game, has been shown.

For the imperfect information game Simultaneous Krieg Tic-tac-toe, the results are indefinite. To start, the legal player performed as expected. Because this game incorporates moves by nature whenever a Field is chosen by both players, the game is not predetermined when the same moves are chosen. This is different from perfect information games, like Connect Four. The results in table 4 reflect this behaviour.

The outcome of the matches between two random players are expected to be equal. The real game has no starting player, however the model in Groove has. The model is designed so that the turn of the either player should not be affected by one another in a single round. Any advantages for either player would suggest that this design has failed. Although the null hypothesis of equality of chances was not rejected, it has not been shown it was correct. The correctness of the model should be further investigated.

Creating asymmetry in information using filter transition used by the match-manager is not optimal. The filter offers the creator of a game grammar fine control over the

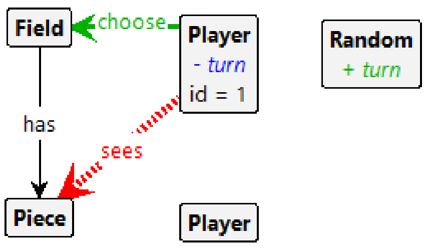


Figure 18. The adjusted graph for the choose rule.

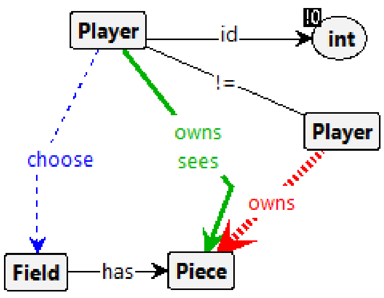


Figure 19. The adjusted graph for the take rule.

graph that the players are allowed to see, however it distorts the state space. Whereas the game tree of perfect information games is represented by the state space of the grammar, for imperfect information games the filter introduces states which are not legal for the game. An alternative could be edges from the player nodes to the observable nodes in a graph. The match-manager would contain an algorithm for eliminating the nodes not observed by the current player. Another option would be to extent Groove with additional syntax. Nodes and edges not containing a certain keyword and player id, could be filtered from the graph when presented to the player.

The reason why the Lookahead player could not reason with the states was investigated and found late in the research process. The issue was caused by the method used for comparing graphs. The numbering of the nodes was assumed to stay consistent, however, this was not the case. In figure 17 it is demonstrated that the nodes in a graphs of an equivalent state are not guaranteed to be numbered the same. This is only true for nodes that are created by the application of rules. This means that models containing rules that create nodes cannot be played using the Lookahead player. A possibility would be to use graph isomorphism theory to test the filtered state with possible states. However, this is likely to provide a significant penalty in performance. Constraining the models even further, by requiring all rules to not create new nodes, would weaken the ability to model general games in Groove. The Lookahead player therefore is not a general game player.

Simultaneous Krieg Tic-tac-toe was adjusted by changing all rules that created or checked for the existence of the Field node and starting the game with all piece nodes already attached to their respective field. Figure 18 shows the new choose rule and figure 19 shows the new take rule.

Although the Lookahead player was able to play the game, it performed significantly worse than the random player. This suggests that the strategy was either not well designed or not well implemented. The reasoning part of the Lookahead player was successfully able to explore the adjusted game. A better strategy or implementation, together with a better approach to selecting states to reason with, should allow for a more accurate and faster general game player.

The connection of proved General Game Playing algorithms to the match-manager could further demonstrate the possibility of General Game Playing with Groove grammars. For perfect information games, the algorithms of players like CadiaPlayer could be adapted into Groove players. Additional games could also be made.

In addition, Groove could also be used for more game theory exploration. The state graph created by the grammars described in this paper, are very similar to extensive form. These could be used to exploring equilibria for different configuration of rules or starting states.

All code and models are available on the UTwente Studenten Net GitLab[5]

9. CONCLUSIONS

This paper demonstrates the possibility of using Groove for General Game Playing. Connect Four and Simultaneous Krieg Tic-tac-toe were used as an example for a perfect and imperfect information game respectively. The perfect information game was shown to be successfully implemented and was playable for both naive players and a simple General Game Playing strategy. Simultaneous Krieg Tic-tact-toe was modeled, but it could not be shown that it was implemented correctly. The proposed general game player was unsuccessful in playing imperfect information games better than a random player.

An answer to RQ 1 was found. To be able to use a Groove grammar as model for a game with perfect information: each state needs to flag a player with having the turn; each transition from a state needs to correspond with a legal move in that state; and the winner of a game is designated with a flag in a final state. These grammars can be managed by a match-manager, who determine which player is to move or or has won using a connection with Groove. The match-manager shares the connection with the general game player which applies its strategy to this model of the game.

To answer RQ 2, this paper proposed additional requirements of the Groove grammars. An additional transition using a move called "filter" has to be present each time a player receives the turn. This transitions a source state into a target state that limits the information for the player. The filtered state has to be a sub-state of the original state and all transitions, apart from the filter transition, need to have an equivalent in the filtered state. The grammars are also allowed to create moves by nature. When the player with type random has the turn, a random move transition will be taken by the match-manager. Although an option for modelling simultaneous moves has been discussed, it has not been shown correct.

RQ 3 has only been partially answered. Imperfect information requires the match-manager to not share its connection with Groove. This change requires the transitions to be uniquely identifiable and makes comparison of states more difficult. Transitions can be made unique by the inclusion of anchors. To allow for comparison of states, the rules of grammars can be restricted to not create new nodes. However, this could hurt the generality of the sys-

[5]`https://git.snt.utwente.nl/s1916661/strategies`

tem. A general game player was able to reason with the filtered information about the state of an imperfect information game, but it performed worse than a random move generator.

10. ACKNOWLEDGEMENTS

This paper would not have been possible without the feedback, motivation and guidance of my supervisor, Arend Rensink. His experience, as researcher and as teachers, helped me to perform when it was needed. I would also like to thank Daniel Floor for the encouraging words and the help in understanding Groove. Last, but not least, I would like to thank Dennis Aanstoot for providing me with his GrooveChannel tool as well as a manual for how to use it.

11. REFERENCES

[1] D. Aanstoot. Graph Rewriters as Components. Master's thesis, University of Twente, 2021. (unpublished).

[2] CCRL Blitz, 2021. `https://computerchess.org.uk/ccrl/404/rating_list_all.html`, visited on 2021-06-25.

[3] H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. *Proceedings of the National Conference on Artificial Intelligence*, 1:259–264, 2008.

[4] D. Floor. Using Groove as a General Game Playing Environment. Master's thesis, University of Twente, 2021. (unpublished).

[5] M. Genesereth and M. Thielscher. General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 24:1–231, 2014.

[6] A. H. Ghamarian, M. de Mol, A. Rensink, and E. Zambon. Saying Hello World with GROOVE - A Solution to the TTC 2011 Instructive Case. *Electronic Proceedings in Theoretical Computer Science*, 74:215–222, nov 2011.

[7] FIDE Top 100 Blitz Players, 2021. `https://ratings.fide.com/top.phtml?list=men_blitz`, visited on 2021-06-25.

[8] F. Koriche, S. Lagrue, É. Piette, and S. Tabary. General game playing with stochastic CSP. *Constraints*, 21(1):95–114, 2016.

[9] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General Game Playing: Game Description Language Specification, 2006.

[10] What's the probability a random game of Connect 4 ends in a draw?, 2020. `https://math.stackexchange.com/questions/3569997/whats-the-probability-a-random-game-of-connect-4-ends-in-a-draw`, visited on 2021-06-25.

[11] É. Piette, D. J. Soemers, M. Stephenson, C. F. Sironi, M. H. Winands, and C. Browne. Ludii - the ludemic general game system. In *Frontiers in Artificial Intelligence and Applications*, volume 325, pages 411–418, 2020.

[12] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3062, pages 479–485, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[13] M. Schofield and M. Thielscher. General game playing with imperfect information. *Journal of Artificial Intelligence Research*, 66:901–935, 2019.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[15] M. Thielscher. GDL-II. *KI - Kunstliche Intelligenz*, 25(1):63–66, 2011.

[16] M. Thielscher. GDL-III: A description language for epistemic general game playing. *IJCAI International Joint Conference on Artificial Intelligence*, 0:1276–1282, 2017.