Researching Hanabi with CardScript

Analysing the rules of collaborative card games

Andrea van den Hooff
`Andrea.vandenhooff@student.uva.nl`

14 December 2019, 44 pages

Research Committee: Riemer van Rozen, Anders Bouwer and Ana Oprescu
Research group: Software Analysis and Transformation (CWI)

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Automated Game Design studies how to support game designers with the iterative process of designing, play testing and adjusting prototypes. But while there are many publications on game design, there are no digital tools for card game design of automated game design accepted as canon by game designers. This thesis focuses on questions game designers might have. We aim to support them by formalising aspects of the prototyping process. We research the mechanics of collaborative card game prototypes and provide tools that give feedback to designers. The design of collaborative games like the award-winning *Hanabi* forms a new frontier for research in automated game design, because the rules alone do not explain the many ways of how players can collaborate. This project tries to formalise this collaboration and relates it to the mechanics and meta-rules within the game.

We present a card game design language and analysis tool called CardScript that aims to help designers express their prototype and informs them on the consequences of the designed mechanics. The tool demonstrates how small adjustments to these game mechanics influence the gameplay and reports back on previously posed designer hypotheses. Ultimately, this can help designers get a better understanding of game mechanics such as collaboration and improve the iterative design process as a whole.

Contents

Chapter 1

Introduction

Seeing people playing cards in a cafe on a Friday night is no longer an unusual sight [1].[1] From educative serious games in the workplace, to e-sports in sold-out Olympic stadiums [2] [3]; the challenges made by game designers enrich our daily lives for fun, recreation and training. But *game design* is a challenge in itself. Designers have to focus on a broad spectrum of disciplines from inventing a story, to designing gameplay and formulate rules to structure the result in an experience for players [4]. Novel ideas for games are usually plentiful, but defining the *rules* in such a manner that they form a great game is the real challenge. Why can one rule be a great addition to the mechanics, while another makes it impossible for someone to win the game if they are not the starting player?

Designers need to *playtest* their design to see their mechanics in effect. Worse, playtesting usually shows that the expected and actual gameplay differ. An essential phase of the game design process is therefore playtesting the prototype. Playtesters play the game and provide feedback on the experience so that the designer can move forward with a fresh perspective [5].

Automated game design studies how to improve this design process, such that this iterative process of playtesting and subsequently adjusting the prototype becomes less time-consuming and error prone. Several phases of the prototyping process can be replaced by a tool. An example of this support is a tool that can analyse the rules and influences on the resulting gameplay, or even come up with its own variations of the game. Multiple studies have tried to develop such tools, such as Smith, Nelson and Mateas, Van Rozen and Bell [6] [7] [8].[2] However, hitherto they have not been accepted as canon by the game designers; there is no 'unified theory of game design' [9].

This thesis aims to support the designers by formalising aspects of the prototyping process. This begins with the design of a language to define games with. It is helpful for game encodings to start with a specific genre. This helps to scope the research and the expressiveness of the language [10]. For this project, we focus on the automated game design of collaborative card games (hereinafter referred to as *CCG*s).[3] In these games the main objects are cards and all players are in the same team, sharing the payoffs and outcomes.[4] CCGs have not yet been addressed in the tools developed by automated game design studies. They are a new frontier for research, because analysing the rules alone does not explain how players collaborate.

We study the critically acclaimed CCG *Hanabi* to analyse how its rules facilitate collaboration. In the game, each player can see all cards in play but their own. By communicating under a strict set of constraints with team mates, the players need to conquer this obstacle as a team. While Hanabi is not the most well-known CCG, it has won many awards for its unique game mechanics and is currently being studied extensively.[5] Furthermore, designing collaborative games requires collaborative playtesting, and card games provide a platform to realise this goal. They are well-known and widely popular since they have their origins in many parts of the world [12]. Furthermore, the mechanics of the games are (mostly) limited to cards. For players, this means that learning the basis of a new game will not be too challenging. For computers this determines the computational simplicity and the possibilities for a simulation [12].

[1]`https://www.iamsterdam.com/nl/zien-en-doen/bars-en-cafes/beste-bars-en-cafes/de-leukste-spelletjescafes`, last visited 21 Oct 2019.

[2]See Chapters 2.2 and 3.

[3]See Section 2.3.

[4]The team is considered an organization in which the kind of information each person has can differ, but the interests and beliefs are the same [11].

[5]See Chapter 3.

While a card game is limited by the variety of game objects, creative designers can still come up with many different games within this design space. For example, there are only 52 playing cards in a standard deck, which over time have become a framework for hundreds of card games.[6]

1.1 Problem Analysis & Statement

Despite the widespread interest, little is known on how to design collaborative games effectively, as well as how to evaluate or analyse them [13]. No formal language exists to describe Hanabi in its full capacity. Today, there is no tool focused on helping game designers with their challenges in regards to collaborative games, neither in creative, nor in technical aspects. Therefore we propose a tool to support designers with the latter.

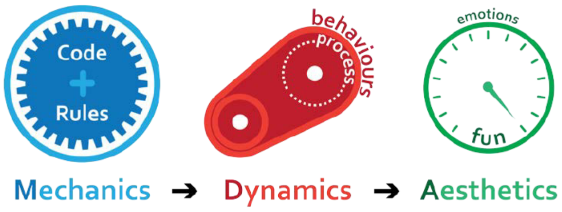


Figure 1.1: Hunicke et al's MDA Framework [14] as discussed in [15].

Each game consists of multiple elements as explained by a framework called the mechanics, dynamics and aesthetics [14]. The relation is visualised in Figure 1.1. The game mechanics have been discussed in the previous section. They are the distinct set of rules that dictate the outcome of interactions within the system with an input, a process, and an output [16]. The second level consists of the game *dynamics*. These are the users' responses to those mechanics and result in the gameplay of players. For example, the mechanics of card games include shuffling, trick-taking and betting from which dynamics like bluffing can emerge [14]. Lastly, there are the *aesthetics* of a game. They are the emotional responses evoked in the player during the game.

This thesis supports designers with the first two concepts of the framework. The tool analyses the mechanics and dynamics and subsequently reports its results to the game designer. Furthermore, it verifies designer hypotheses' of what can occur in the prototype against the resulting gameplay. With this we wonder if game metrics can analyse the MDA framework in a quantifiable manner. This relation between the designer's assumptions, the gameplay and the players' experiences is modelled in Figure 1.2 on the next page. With Hanabi as a use case, we scope the metrics to focus on the forms of collaboration in CCGs.

The tool is created in several steps. We first create a design language *CardScript* to describe the rules of CCGs. We do this with the help of the programming language Rascal, because it is specialised in designing domain specific languages.[7] Subsequently we analyse the mechanics of the prototype in detail. The tool reports on domain specific properties defined in the rules and checks the gameplay against predefined hypotheses of collaboration by the game designer. Therefore we simulate a set of games played by humans and run their actions as game input to look for collaborative aspects. Examples of collaboration in Hanabi are when a player communicates about a certain card and the owner subsequently acts on the given information in his turn, or when players are playing cards on the same pile in the game, further elaborated on in Section 4.4.1. We propose a tool that takes a description of a game in CardScript and returns feedback on these hypotheses to the designer.

We test the tool with Hanabi as case study and therefore implement a simulation of the game. Hanabi is already extensively studied, but as a testbed for AI. The game is seen as the next challenge for algorithms after Chess and Go. Developers try to beat Hanabi with teams of humans and agents combined. They do not focus on the experience of playing the game (the aesthetics), but on finding the best strategy to win the game (the dynamics), sometimes even disregarding some of the rules (the mechanics). We analyse Hanabi from the novel perspective of automated game design and try to relate the game's mechanics to collaboration. The result of our thesis is therefore not a Hanabi playing agent which can communicate successfully with humans. Instead we develop a card game design language and

[6]`https://boardgamegeek.com/boardgame/21804/traditional-card-games`, last visited 21 Oct 2019.
[7]See Section 4.3.

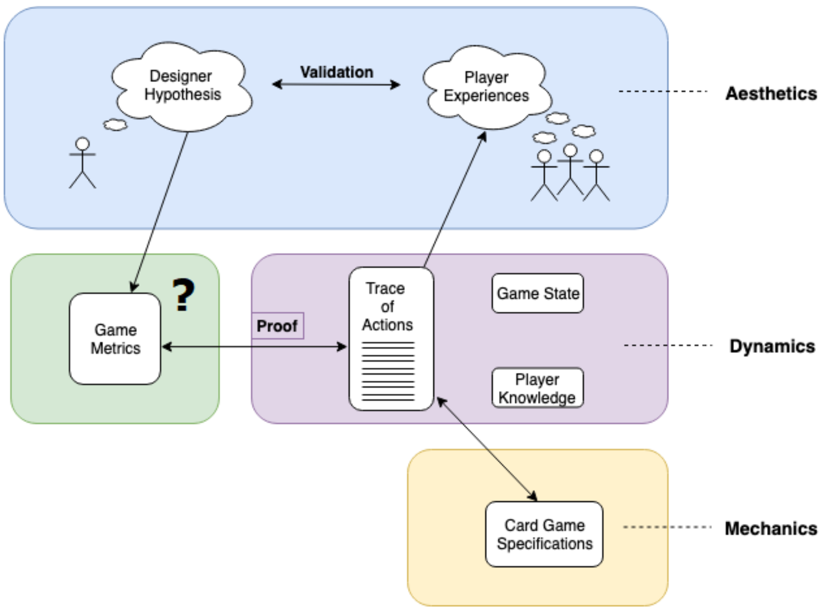


Figure 1.2: Interaction within the MDA Framework. This thesis questions if game metrics can analyse the MDA Framework in a quantifiable manner.

accompanying tool to analyse the mechanics and dynamics of the game, such that CCGs like Hanabi can be designed more efficiently and challenge players with more collaborative puzzles.
The CardScript language and aforementioned analysis tools aim to give insight in collaboration in card games. They are meant as a helpful start for game designers to reduce some iterations of prototyping and playtesting.

1.1.1 Research questions & contributions

We pose the following questions:

RQ1: How can a domain-specific language for card game design be developed?
RQ2: Which elements of card game rules can be analysed automatically with a game defined in CardScript, such that these can be reported to the game designer and give insight in the mechanics of the game?
RQ3: To what extent can we formalise the designer hypotheses of scenarios in a prototype such that we can verify their existence in the gameplay?

We answer these questions to learn how a DSL for card game design can help game designers with their design process, analysing and improving games and player experiences. With the answers to these questions, we contribute the following:

1: A new language CardScript which specialises in defining (collaborative) card games.
2: An analysis tool that informs game designers on their defined mechanics of a prototype and its relations in the aforementioned language.
3: An analysis tool that uses simulated games to verify game designers' hypotheses on the effects of their rules on the game state and player actions.

1.2 Outline

Chapter 2 describes the background of this thesis, such as game design and the rules of Hanabi. Chapter 3 contains related work of studies in different disciplines, including Automated Game Design and Hanabi. This section extends the background and describes the limitations of other DSLs. We present CardScript in Chapter 4, highlighting and elaborating on some of its unique aspects. The case study of Hanabi is in the subsequent Chapter 5. We discuss the results in Chapter 6. Finally, we present concluding remarks and future work in Chapter 7.

Chapter 2

Background

This chapter will present the necessary background information for our thesis project. First, we elaborate on the key challenges of game design. Then we explain card game terminology and essential concepts that will be used throughout this thesis, including an explanation of Hanabi's rules.

2.1 Game Design

Since 1979, a jury of German game critics come together to award the annual *Spiel des Jahres (SdJ)* to an analogue tabletop game. It is a prestigious award to reward excellence in game design.[1] Notable winners of these awards include the famous *Catan, Dixit, Carcasonne* and *Ticket to Ride.* Some of the game designers, such as Catan's Klaus Teuber, have won the SdJ award multiple times. This suggests that designers can have a good understanding of what makes a great game. Before bringing these successful games on the market however, they still required hours of playtesting by friends, acquaintances and game lovers to fine-tune the rules.

Game design is a complex domain. Game designers often start with simple tools of paper, perhaps small wooden pieces and vague descriptions of the gameplay. However, a game must have rules in order to provide structure. They are *'a specific sort of immaterial support [..] the determination of what moves and actions are permissible and what they will lead to'* [17].

There are many concepts, shaped by rules, that make a game a good game. An example is that the rules should make sure that the final result of the game stays undetermined for as long as possible. Another aspect of the success of a game are the player's actions. During a turn, decisions have to be made on which action to execute. Players could struggle from 'analysis paralysis' when too many viable options exist, or feel like there is no room for their own input when they only have mandatory tasks [7]. Meanwhile, there are a lot of pitfalls designers should worry about, some more obvious than others: each participant of the game should have a fair chance of winning, new players should not have such a difficult time getting introduced to the game, while more experienced players still require challenge [5]

2.1.1 Game Design Process

Game designing is more than simply making hundreds of thought out decisions. Each decision needs to be tested extensively. A well-known approach to the design process is the *playcentric* approach. This means that through every phase of development, the designer continually keeps the player experience in mind and tests the gameplay with target players [5]. This is an iterative process and it can look, when simplified, as follows [9]:

1. Come up with a basic design.
2. Figure out the greatest risks in your design.
3. Build prototypes that mitigate those risks.
4. Test the prototypes with players.
5. Come up with a more detailed design based on what you have learned.
6. Return to step 2.

Game designers do not have perfect imagination and many decisions are impossible to make until they

[1]`https://www.spiel-des-jahres.de/faq/`, last visited 27 Oct 2019.

have seen their prototype in action [9]. Therefore, the goal is to test the prototype as early, usefully and frequently as possible. It can be as simple as moving some paper and Lego blocks, to formally printed figures [5].[2] As shown in the previous section, game designers have questions, hypotheses and goals devised for their game. Modifying the mechanics of the prototype and testing the changes, helps them to fine-tune the games' overall dynamics [18].

These aforementioned six steps are a practical framework for building games. However, there is no globally accepted theory of game design. According to game designer Schell, game design is solely based on principles of human psychology, but influenced by many disciplines [9]. The MDA Framework discussed in Chapter 1 fits these steps as well. It supports the iterative approach to the design and tuning. The MDA Framework allows the game designer to reason explicitly about particular design goals and anticipate how changes will impact each aspect of the framework [14] [19].

This thesis project strives to prevent lengthy paper prototyping iterations by means of a tool that speeds-up game design iterations. This requires a way to describe games. To be able to analyse the mechanics and dynamics of a game, a domain specific language could come in handy.

2.2 Domain specific languages

A *domain-specific language (DSL)* is a computer language that e.g. can describe a certain domain and its specifications. Formally, it *'provides a notation tailored towards an application domain and is based on the relevant concepts and features of that domain'* [20]. A DSL is the opposite of a *General-Purpose Language (GPL)* which is, like the name suggests, a multi-functional language. A GPL in the physical world is English. A DSL defines a limited domain instead. Examples in the physical world are the football jargon ('offside') or corporate speak offices. Well-known DSLs are the programming languages *HTML* and *SQL*.

DSLs have several potential benefits. Since all the unrelated clutter is filtered from the vocabulary, the time to market and the maintenance costs are reduced. Furthermore, the compactness of a DSL improves the portability, reliability, optimisability, and testability of the object it is used in [20]. However, having a scoped language has its disadvantages as well: the specified scope causes limitations and the cost of learning a new language versus its limited applicability needs to be evaluated. Furthermore, the cost of designing, implementing, and maintaining a DSL as well as the tools required to develop it should be taken into account [21]. In this thesis we define a DSL to be able to explore the domain of collaborative card games, which we will discuss further in the following sections.

2.3 Card Games

There is no centralised dictionary for game genres [9], except for the obvious difference between e.g. video games and board games. The genre of card games encloses a wide range of games. However, they share a set of objects that define the basics of its domain. These are displayed in Figure 2.1. In this section we specify a *card game vocabulary* to discuss these concepts.

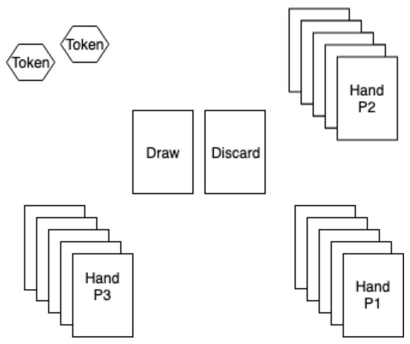


Figure 2.1: Abstract representation of a card game.

A card game consists of a set of *cards*. This can be the well-known 52 playing cards of hearts, clubs, spades and diamonds, or game-specific cards. Each card has a physical location. Either they belong to

[2]`https://nos.nl/op3/artikel/2253582-alexander-heeft-misschien-wel-het-nieuwe-catan-bedacht.html`, last visited 14 November 2019.

a pile on the table (such as the discard and draw pile) or a player's hand, both visualised in the abstract representation shown on the previous page. Furthermore, these cards have a visibility depending on the location. The draw pile cards are face down, others are only shown to the player which have that card in their hand [7]. This causes *incomplete information* of the game state for players. Incomplete information refers to situations where players cannot have all required knowledge for the posed challenge, which is formed by the mechanics of the game. In contrast to chess, where everyone can see all elements in play, the dynamics of card games build upon this incomplete visibility of the cards [22] [10]. For example, the dynamics of Texas Hold 'em and Blackjack 'break' if the card visibility changes. This concept of incomplete information adds a puzzle-like aspect to card games, while also creating a sense of 'luck' based on the order of cards after shuffling.

Besides the required cards, card games might also have an internal game economy, which is symbolised by *tokens* [23]. In Figure 2.1 they are displayed as a shared object of all players in the game, but depending on the game mechanics they can belong to a specific player or the general market as well.

The mechanics make a game unique [18]. They are the framework on which the dynamics and aesthetics are built. We categorise the mechanics in several groups. The rules that specify the mathematical, core regulations which can be found in any game, are defined as *constituative rules* [24]. These include for example the scoring principle, the amount of players and turn order. Furthermore, the turns of players should be described. What actions can they perform? When can they perform them? These are specified in the *operational rules*. In a game they are described in the *stages*, which are played in a sequential order. Lastly, there are games where cards have rules or actions themselves, such as Dominion and Magic: The Gathering. However, these add an extra complexity which is not within the scope of Hanabi and is therefore not included in this thesis.

2.4 Collaboration

In general, games can be either collaborative, competitive or a mixture of both. In competitive games, players win by beating their opponents, reaching a certain goal before they do. In a mixed style, collaborating with another player for a while might give an advantage to secure the individual victory. A recent development in game design concerns these cooperative games where the players have to collaborate throughout the entire challenge [11]. While there are no annual statistics of published games, it seems that collaborative games have become more popular, based on the nominees and winners of the SdJ award [13].[3] The challenge for CCGs is not to destroy friends, but to combine strengths and solve the puzzles together. Whereas the classics of Monopoly, Risk and Catan are heavily influenced by the competition between players, collaborative games are a success because of the teamwork.

The collaboration is defined and constrained in the rules, but the effects of the concept on player interaction are noticeable in dynamic interaction sequences. The aspect of social interaction and how this influences gameplay adds a new dimension to designing collaborative games. For game designers, the focus shifts to collaborative design questions, e.g.: How can players work together effectively? How will a player react to another player's actions? Where the roll of a die will always keep the element of chance in games, teamwork can be practiced and improved.

The mechanics enable certain actions in gameplay. However, what the players cause to happen cannot always be predicted [19] and the response to the same scenario might even change after several games. We define these mechanics that players decide together for their strategies as *meta-rules*. They are personal rules and are on different 'levels' than the explicit rules of the game [24]. When playing the game in the same setting multiple times, these personal manners and strategies can appear. We hypothesise that these potential developments are consciously designed choices. For Hanabi specifically, one of these rules is even mentioned as a tip in the manual: *'A player who is given information can rearrange his hand if he wishes to do so in order to put the cards concerned in an order which is easier for him to remember'*.[4] Other meta-rules can be found in player-made manuals found in online communities, for example on sites such as GitHub.[5] They act as an addition to the original rules of a game, making the design of collaborative games an interesting research topic.

[3]`https://www.spiel-des-jahres.de/spiele/`, last visited 27 Oct 2019.
[4]`https://www.spillehulen.dk/media/102616/hanabi-card-game-rules.pdf`, last visited 14 Oct 2019.
[5]`https://github.com/Zamiell/hanabi-conventions/`, last visited 14 Oct 2019.

2.5 Hanabi

When analysing the rules of Hanabi, knowledge of the game is a prerequisite. Therefore we shall explain the rules of the well-received CCG in this section.

The game objects consist of 50 cards and a handful of tokens. The basic deck has cards in five colours (red, yellow, white, green and blue), numbered 1 to 5. Of each colour there are three 1's, two of the 2's, 3's and 4's, and one 5. Players need to play one of each in the correct order from low to high. Reaching the maximum score of 25 is considered winning. The mechanics of Hanabi are straightforward: based on the number of players (between two to five), four or five cards are dealt to each player. Then, in clockwise turns, each player can either give a hint to another player, play a card on one of the five colour piles, or discard one of their cards onto the discarding pile, removing it from the game. When a card is played or discarded, a new card will be picked up, such that a player has always the same amount of cards in hand, until the drawing pile has been emptied.

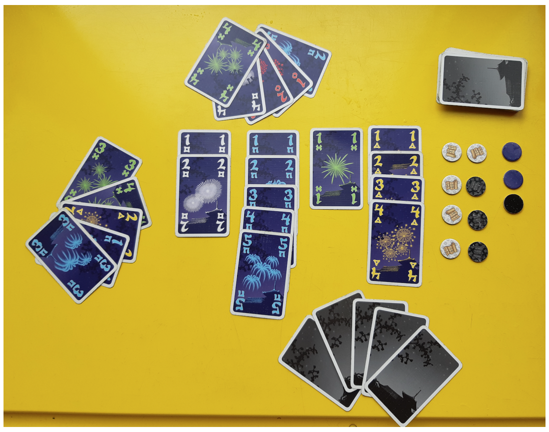


Figure 2.2: Example of a gamestate in Hanabi, with on the left P2's hand, P3 at the top and P1's (your) hand at the bottom.

These basics make it seem like Hanabi is a simple game, but the unique aspect to this game is the visibility of the cards. Your own hand is faced towards your team members, such that you can only see the back of these cards. Each player sees the cards of all teammates, except for their own, which is visualised in the game state shown in Figure 2.2. Players have to rely on their team to inform them which of their cards will be the correct cards to play, and help them in return. Thus the action of giving a hint is essential to Hanabi and this unique visibility rule, which creates the aspect of incomplete information, is a big element of its success.

Two sets of tokens are part of the game as well. There are three tokens representing lives, which are lost when errors are made. When a card is played which is not the next card in the sequence of that colour, a life is lost. The game ends immediately when all three lives are lost. Furthermore, there are eight tokens representing available clues to give. When a player gives a hint to a teammate, one of these tokens is used. A hint token can be returned by discarding a card or playing a card with the number 5 correctly, completing that specific colour pile. Finally, a hint can be one of two options: the player either points at all cards in the teammate's hand of a certain colour, or those with a specific number.

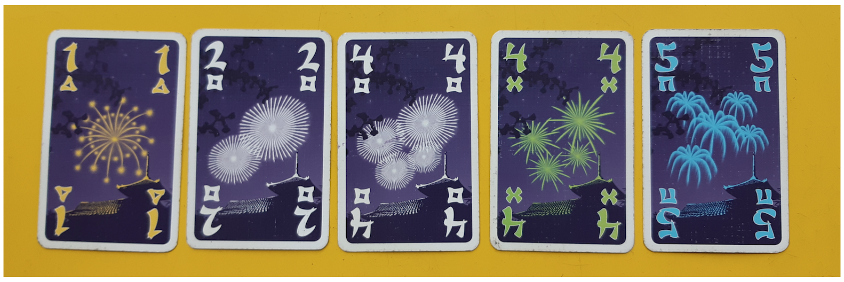


Figure 2.3: Example of a player's hand in Hanabi.

Consider the following scenario: Ben has the hand shown in Figure 2.3: Anna can give several hints, e.g. she can point at the two white cards, stating that they are white. However, Anna can also mention the two fours to Ben, or state that the five is a five. An extra constraint to this mechanic is that Anna has to point at all the cards of that number or colour, and at least one card has to have the specified property. So in this example, she cannot state ‘You have no red cards’.[6]

Depending on the current state of the game, the most ‘useful’ clue to reach the shared goal should be chosen. Anna decides which action to take and Ben then has to interpret the hint and deduce the correct (perhaps initiated by Anna) next play. The game ends when three cards are played incorrectly, when all fives are correctly played, or when there are no cards left in the deck. When the latter happens, all players have exactly one last turn to finish the game. Finally, the shared score is calculated.

[6]In later editions of the game, this kind of hint is allowed, however in this research we use the rules of the original game, thus we do not allow these hints.

Chapter 3

Related work

Many articles and books have been published on games; from the design of the gameplay to the methodologies and theories behind them. We divide the related work into three main perspectives: Automated Game Design, Automatic Game Design and playing Hanabi.

3.1 Automated Game Design

The need for common design vocabularies, game design patterns, specialised game grammars, and computer assisted design tools has been expressed for some time, such as in [6], [25] and [26]. Automated game design is a broad multidisciplinary field focused on automating game design processes to support game designers. Game rule encodings and evaluation functions can encode game design expertise and style, and thus help understand game design [10]. In this approach, DSLs are used to describe game prototypes and build tools for game designers. There are several examples of studies in automated game design we will elaborate on, each researching different phases of the game design process.

In early 2007, Nelson and Mateas explored automated game generation. They described an approach to formalising game mechanics and generating games using those mechanics. Like this thesis, the authors aim to facilitate formal game analysis through the computational expression of game rules, mechanics, and representations [25]. However, Nelson and Mateas factor game design in four interacting domains (or aspects): abstract game mechanics, concrete game representation, thematic content, and control mapping. The latter three scope their game design research towards the story telling of video games, such that the final results are aimed to generate short video games.

In [27], Smith and Mateas respond to this factoring of the game design domain: *'several commonly accepted processes in game designs cross-cut these domains. Conceptualization, prototyping, playtesting and tuning are essential parts of game design; there is no compelling reason to think they should not be addressed in a nuanced automation of game design'.* Similar to Smith and Mateas they scoped their research to digital games and created a logical language, but to explore the possibilities of automated playtesting instead. Both these studies used automated game design to develop tools to automate some of the work of the designer. They implemented logical languages, which are difficult to read for game designers with little programming experience. This makes it difficult for game designers to use their tools. Thus we aim for a notation that is intuitive and easy to understand and maintain while researching a different domain of games.

In 2010, the three aforementioned authors published their shared research on *'Ludocore'*; a *'logical game engine'* which aimed to analyse video game design. Ludocore was capable of generating gameplay traces that illustrate the behavior of a game. The primary function of the engine was to extract gameplay traces in video games [8], to show the game designer the influence of rule modifications, since *'one of the difficulties for game designers is understanding the potential consequences of rule interactions'.* This resembles our objectives closely, but Ludocore was scoped towards video games and therefore several methods of the approach are not directly applicable to analyse card game design.

Osborn et al. developed a tool that delivers reports from critics during the game development (see Figure 3.1 on the next page). This model resembles our approach closely, since *'Gamelan'* was aimed to formalise game mechanics and have game designers interact with the tool during the game design process. The intend of their contribution was more amenable to non-programmer designers than prior approaches [19]. However, the authors scoped Gamelan to the card game *Dominion.* This is a competitive card game where the main mechanics of the games are connected to specific cards. We do not research these kinds of card games. Furthermore, Osborn et al. do not include a parser with Gamelan. Therefore the language is difficult to use by game designers for other games.

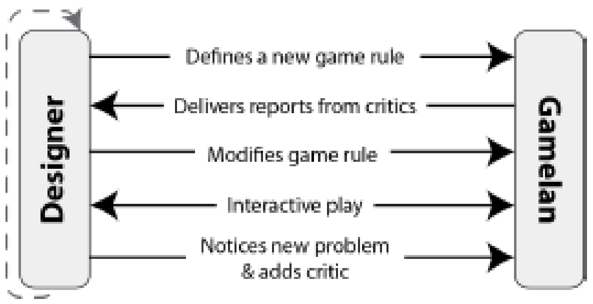


Figure 3.1: Designer interaction with the design support tool Gamelan [19].

Bell and Goadrich constructed a card game design language for multiple card games. In 2016, they published their GDL *Recycle* and accompanying tool *CardStock*. By slightly modifying rules, Bell analysed the game design properties with a set of metrics of three card games, such as the points per player, choices per game and turn order advantage. This approach resembles our thesis project closely. However, there are several differences between the scopes. Firstly, we aim to measure collaborative card games with any kind of cards, instead of competitive games with standard playing cards. Furthermore, Bell states specifically that Hanabi cannot be encoded with Recycle, since it does not have ways to support the reversed visibility. Finally and perhaps most importantly, all of the languages discussed focus on generating objective quality metrics. We argue that criticising design decisions requires the expertise of game designers. This project does not aim to judge these decisions, but to report on the designer hypotheses' of what these decisions might enable instead. Furthermore, we focus on the collaborative aspect of games, which to the best of our knowledge has not been addressed in any other Game Description Languages so far. The rules of collaborative games such as the SdJ winning game *Hanabi* remained undefinable in these game languages.

Each of these studies, and many more which we do not elaborate on such as [28] and [29], aim to support game designers with the iterative design process. However, so far no tool, method, or framework has surfaced as an industry standard in automated game designs. No system exists that can predict the success of a game by analysing its rules [13]. As a result, game design still relies on the designer's expertise for the iterative process discussed in Chapter 2. Furthermore, to the best of our knowledge, collaborative card games have not been addressed as main scope in these studies and thus our questions posed in the introduction remain.

3.2 Automatic Game Design

Games can also be defined in *Game Description Languages (GDLs)*; DSLs specific for the game domain [28]. Automatic Game Design uses GDLs to develop algorithms that can design games. The difference in the objectives between Automated and Automatic Game Design is that the latter develops GDLs and algorithms to possibly replace game designers, instead of support them.

There are already dozens of GDLs in existence, e.g. [30], [31] and [32]. The first of these rule-generating systems was *Metagame*, which generated 'symmetric, chess-like games' in 1992 [10]. Nelson discusses these languages and states that Browne's work might be the most successful example. The latter published *Ludi* in 2010, focusing on two-player combinatorial games with perfect information.[1] The system designed more than 1300 new games through several algorithms. *Yavalath* is the most successful result of Browne's studies and can be bought commercially: it is very similar to the classic 4-in-a-row game, however you lose when you make 3-in-a-row.[2]

The 1300 prototypes generated by Browne's algorithms had to be tested for their originality and playability with *General Game Glaying (GGP)* GGP lets computers play unfamiliar, turn-based games

[1]Browne defined combinatorial games as finite, turn-based, without chance or hidden information and made for two players. These are further explained in [33].

[2]`https://boardgamegeek.com/boardgame/33767/yavalath`, last visited 23 Okt 2019.

[8]. Automatic Game Design uses their generated games as a testbed for these game playing AI, to improve algorithms such that they can play different games well [34]. In contrast to DSLs, one of the objectives of a GDL is that it can be used by systems to understand the rules of games and learn to play them [35]. Most of the 1300 generated games turned out not to be playable by the AI.

While there are dozens of GDLs, there are only a handful languages for card games specifically. Font et al.'s domain consisted of card games with a standard deck of playing cards, such as Poker and Blackjack [12]. The authors focused on the descriptive part of the language, which might help our domain analysis in Chapter 4. They used the language to design simple card games using evolutionary algorithms and confirmed Browne's findings that this was quite difficult, with many games crashing or resulting in a tie. In a later paper they were able to generate games with playable rules, but these were not considered fun to play by humans [36] [10]. These studies showed the importance of the expertise of a game designer during the designing and analysing of prototypes.

Besides domain-specific studies in automatic game design, there are also examples of general studies in automatic game generation, e.g. De Mesentier Silva et al. and Togelius et al [37] [38]. The characteristics of games that can be generated are analysed by the latter. According to them *'rules are arguably the core feature of a game, and for an algorithm to generate a game it needs to generate the rules in some form'*. Furthermore, they note that another core feature that makes games interesting is what they call *heuristics*, which we define as meta-rules in Chapter 2 [39]. According to them, the methods for determining whether a game supports the possibility of heuristics, is a key challenge for automatic game generation. Similarly, De Mesentier et al studied heuristics for games for novice players (and therefore, automated game agents) in [37]. Both studies laid emphasis on the importance of game mechanics and we therefore conclude that these deserve further research.

The fields of Automatic Game Design and Automated Game Design do not exclude each other. This project uses the language approach of the latter, while we do not aim to replace the game designer. If the computer agents work as intended, we may leverage GGP in the automated game design tools to replace the non-aesthetical part of human playtesting completely [8].

3.2.1 Collaboration Engineering

Despite the fact that collaboration has been an important research topic across several disciplines for a long time, only recently has it become prominent in game research. The interest in collaborative and cooperative processes in several fields of study and the increasing popularity of collaborative games give a clear indication that this is a relevant topic for investigation [40]. One of the main disciplines is Collaboration Engineering, which we elaborate on with two example studies.

Azadegan and Harteveld stated that *'despite the widespread interest, little is known on how to design collaborative games successfully, as well as how to evaluate or analyse them'* [13]. They propose to use Collaboration Engineering to study collaboration in games. However, they concluded that there is a conceptual difference between game design and CE. According to them, the collaboration framework in game design is not always clear and sometimes even opposed by the game mechanics, since the nature of games challenges players. Azadegan and Harteveld showed that game design needs other approaches to collaboration frameworks.

Kolfschoten et al.'s discussed the concept of ThinkLets; a codified facilitation intervention in a group process to create a desired pattern of collaboration, which is a key concept to CE [41]. However, the study the collaboration placed ThinkLets in daily life and were difficult to link to collaborative card games. Most studies on collaboration had different scopes on the collaborative concepts. This supports Azadegan and Harteveld's conclusion. For future work, with a broader focus, these could be a possible view to elaborate on. However, the question how collaborative games can be represented remains largely unanswered. Our approach in Chapter 4 try to take the first steps in this direction.

3.3 Hanabi

The unique way of communication in Hanabi has sparked the interests of many researchers since the game was published in 2010. As mentioned in chapter 2, the academical focus on Hanabi has been from the perspective of artificial intelligence. Since the essence of the game is effective communication between agents, researching Hanabi is helpful from a societal point of view. By getting the highest score in Hanabi with computer agents and humans as players, we can learn how to have robots and humans interact in the technological society of the future. Here, we summarise the related studies to Hanabi.

The success factor of the Hanabi studies is the average highscore of agents playing the game. A trend can be seen in the publications with the highest results: their programs either introduced very specific meta-rules or disregarded some of Hanabi's original rules. Thus, while the Hanabi studies research the dynamics of the game instead of the mechanics, they are heavily influenced by the latter.

The first that tried his hand at creating agents for Hanabi was Osawa in 2015 [42]. He developed a couple of strategies his agents would follow and reported the results. Osawa found that the score in Hanabi was the highest (between 15 and 20 points out of the maximum 25) when agents would expect their team mates to act the same way as they would with that particular game state. This 'self-recognition' strategy scored the best after the ideal game, where each player could see his/her own hand.[3] This latter approach is the first example of disregarding Hanabi's original rules and is thus not regarded as a valid strategy.

Not long after Osawa, Cox et al. came up with a way to interpret the rules in such a way that AI agents would achieve the highest score possible in the game in over 75% of the games [43]. While the computer agents abide all Hanabi's rules, they implemented unique meta-rules. By creating a counting system and an encoding scheme, a hint would be able to inform all players on their hand. Since this scheme had to be known by heart, this implementation would not be applicable to human players. Furthermore, by introducing these meta-hints, the essence of the game was lost and therefore they did not 'solve' Hanabi's original puzzle.

In August 2019, Eger implemented computer agents playing Hanabi with other meta-rules. He tackled the communication problem from a different point of view, utilising timing as a covert channel so effectively that the players could eschew the communicative actions provided by the game entirely [44]. They addressed these issues in a paragraph called 'Isn't this cheating?' in their article. This is a discussion that we think should be explored by more studies and it would be insightful to hear the opinion of Hanabi's designer Bauza on the endless possibilities of meta-rules.

In the last four years, more and more authors have tried to solve Hanabi's challenges with AI. Most of the studies start with Osawa's original strategies. Noteworthy is the competition laid out by CIG to come up with the best performing AI Hanabi player.[4] This competition resulted in several publications, where most authors such as [45], [46] and [47] achieved a slightly better score with a slightly different strategy compared to Osawa and earlier studies. Most recently in February 2019 fifteen Google employees tried to beat the known algorithms [48]. They called Hanabi 'a new frontier of AI research'. Bard and his colleagues were able to replicate earlier invented techniques. While their paper was more extensive than most, even they conclude that 'the learning algorithms using deep neural networks are largely insufficient to surpass the current hand-coded bots when evaluated in a self-play setting' [48]. Thus, the true essence of Hanabi remains undefeated.

The dynamics of Hanabi resulting from the hint mechanics have been dismantled by the studies on the game. Their focus was to beat the game, not analyse the dynamics created by the rules. The aesthetics and joy of playing a game are not solely based on winning [15]. We look at Hanabi as a frontier in automating collaborative game design instead and combine the fields of game design, collaboration and Hanabi's mechanics into one coherent analysis on how the game and its rules work. The result is therefore not a Hanabi playing agent which can communicate successfully with humans: instead we develop a card game design language and accompanying tool to analyse the mechanics and dynamics of the game, such that CCGs like Hanabi can be designed more efficiently and challenge academics with more puzzles by designing them quicker.

[3]Note: Even with the strategy of perfect visibility, the average score of a perfect game in Hanabi is not 25, since the game is dependent on the shuffling of the deck and it cannot be won in some cases, such as when all 1's are at the bottom of the drawing deck.

[4]`http://hanabi.aiclash.com/rankings.html`, last visited 31 Aug 2019.

Chapter 4

CardScript

Here we describe CardScript, a DSL for card game design that helps designers create, analyse and play test collaborative card games. First, we present its requirements by considering designer questions. Next, we describe a domain analysis that relates essential concepts. Based on that analysis we develop a syntax of rules and create a framework to formalise collaboration, which we explain using examples. Finally, we show several techniques to formally analyse the rules. With these steps we take a technical approach to the practical questions of game designers.[1]

4.1 Requirements

4.1.1 Designer Questions

The objectives of CardScript are to support the game designer during their practical iterative process of prototyping and play testing card games. Therefore we formulate questions designers might pose during their designing process, similar to the approach in [49]. These posed questions could be:

1. **Expressiveness**: How does the material prototype of a card game translate to a clear and structured rule set?
2. **Collaboration**: How is collaboration represented in a card game prototype specification?
3. **Root-cause analysis**: How can we identify during the iterative designing process which modifications influenced the evolution of the prototype (negatively or positively)?
4. **Translation**: How can a hypothesis of the expected gameplay be formulated such that it can be verified by play testing the current prototype?

Other relevant questions not further discussed here are, e.g.,

1. **Playability**: How can a game be fun to play for both new players and more experienced players?
2. **Challenge**: How many turns or minutes does it take to achieve certain goals in the game?

4.1.2 Technical Challenges

CardScript is not meant solely as an expressive language for game designers, but also as support to their gameplay assumptions with quantifiable measurements. The designer questions the effect of the mechanics on the dynamics and aesthetics, as discussed in Chapter 1. While not every question can be easily answered, a technical approach can solve some of their issues. Our tool addresses these questions and provides partial answers with helpful metrics. Here, we identify technical challenges that need to be addressed to answer the questions posed in the previous subsection.

> **TC1 Expressiveness**: How can the rules of card games be formalised as an expressive textual notation that facilitates iterative game design? One of the requirements for our language is to be expressive such that game designers without programming experience understand the structure, but at the same time have a logical language such that it can be parsed and processed to turn the game definition into a playable prototype.

[1]All code presented in this thesis can be found on `https://github.com/andreavdh/masterthesis`.

TC2 Collaboration: How can the collaborative aspects of CCGs be formalised in a model? We have discussed in Chapter 2.4 that collaboration is dynamic, relates to gameplay and exists in the head of players, whereas the mechanics are static definitions. Is it possible to isolate the collaborative components such that we can analyse the concept within a game?
TC3 Correctness: How can a syntactically correct card game specification be checked with respect to contextual constraints, such as trying to draw a card from a player instead of a draw pile? The parsed game needs to be reviewed for mistakes made by the game designer, before it can be turned into a game, to avoid syntax-related errors.
TC4 Root-cause analysis: Which aspects of the mechanics and dynamics of a prototype can we measure such that it gives insight in the resulting gameplay? While every game is unique, they consist of similar patterns. For example, each game has a starting position, specified turns and a win condition. What can we report on the effects on the gameplay when a game designer makes a small adjustment?
TC5 Simulation: How can the game defined in a card game language be evaluated with a simulation? There are dynamics of a prototype that can only be detected by simulating the gameplay to see what the players cause to happen. What are the options without proper knowledge of AI or General Game Playing?
TC6 Translation: How can designer hypotheses be formalised such that a tool can relate quantified metrics to the qualities of a game? Playing a game is more than following the rules and the questions game designers have can be broad and general, transcending the mechanics. However, the basis of a hypothesis is a set of actions, which we can check for and this might answer the posed questions.

4.2 Domain Analysis

To approach the first challenge (TC1) we need to model the defined scope before we can design an expressive textual notation. By creating a model for card games we can identify, capture and organise relevant information in objects [50]. We do this on the basis of a domain analysis and a class diagram. A basic class diagram shows the relations of all objects within our scope. We use existing approaches and reverse engineer known card games to facilitate the diagram. Figure A.1 in the Appendix A.4 shows the full diagram, but we discuss it in components throughout this section.

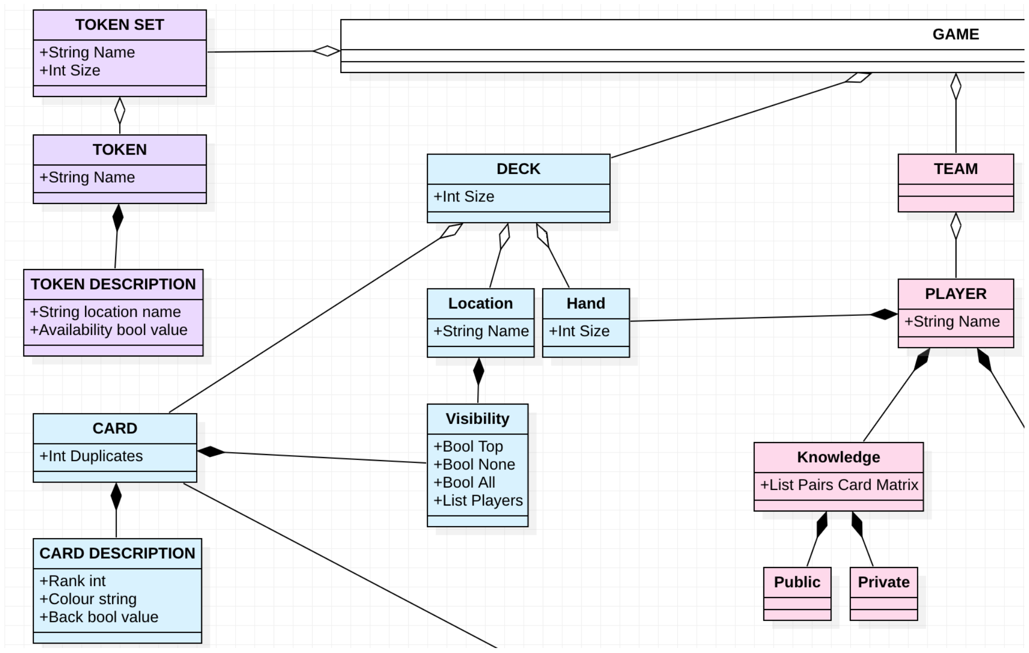


Figure 4.1: A snippet of a trace of actions of a game of Hanabi.

Figure 4.1 shows the first three sections of the diagram. The blue section represents the *deck* object. Each deck consists of a *location, visibility* and a set of *cards*. Using Hanabi as an example, the deck consists of the unique Hanabi cards on several piles. Each player has their own hand that only the other players can see. The shared draw, discard and colour piles can be seen by respectively no one and everyone. Furthermore, each card has properties, which in this case are a *rank* and a *colour*. The purple selection representing the *tokens* is similar to these deck objects. They include the location and visibility properties as well. For Hanabi, the tokens (representing lives and hints) are shared and visible to all players. The last physical aspect of a game is the *team* and its *players*, seen in the pink nodes.

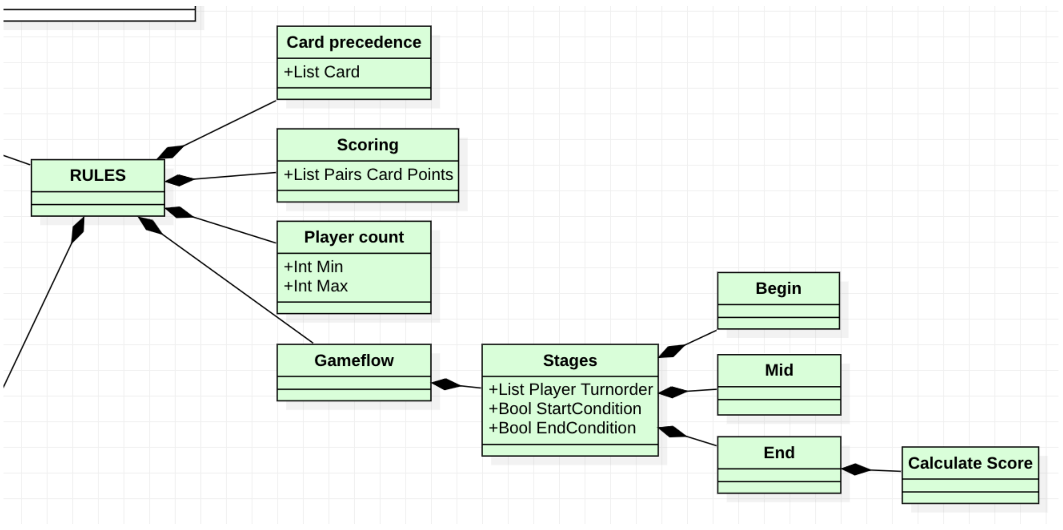


Figure 4.2: The section of the class diagram concerning general rules.

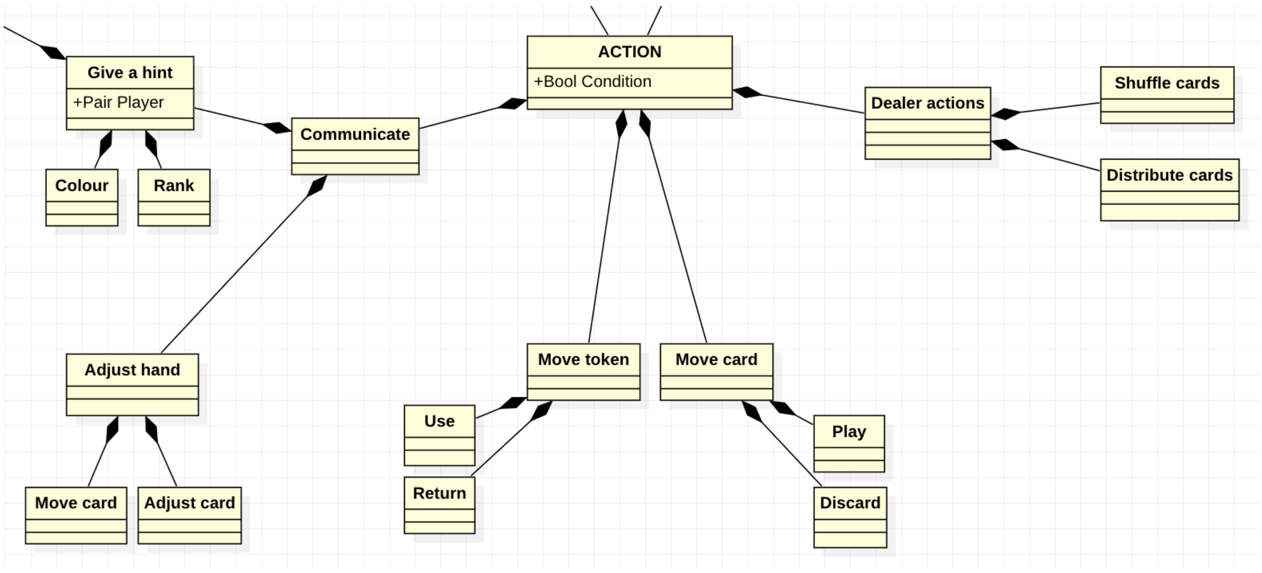


Figure 4.3: The section of the class diagram concerning player and dealer actions.

Figures 4.2 and 4.3 represent the *actions* and *rules* of a game respectively. The first shows all constituative rules (as defined in Section 2.3) related to general properties of the game, such as the card precedence and the number of players. Here we also specify the general flow of the game; the order of the stages. For example, does the game start with an imaginative dealer that shuffles and distributes cards? These *actions* are specified in *turns* in the second Figure. While they belong to the gameflow, they are such a big part of the mechanics as a whole, that we display the details separately. Therefore, the nodes in Figure 4.3 visualise the operational rules, e.g. these dealer actions, the moving of cards and tokens, and something we introduce as 'communication'. This latter is a unique asset of collaborative card games and its form is game-specific. The sub-nodes are examples of communicating in Hanabi.

4.3 Implementation Language

This project has several objectives, from designing a DSL from the above diagram to a language workbench that can analyse game prototypes defined in the new language. We therefore implement CardScript with Rascal. This is a meta-programming language developed by the national research institute for mathematics and computer science in the Netherlands (CWI). It is focused on analysing, transforming and generating source code. When using another basis for a DSL, one will end up with many different tools, possibly even written in different languages. The problem is then to integrate these tools again and this is one of Rascal's qualities by combining it all in the same language. Furthermore, Rascal has several features that support the creation of a DSL. Lastly, Rascal is noteworthy for its compactness. Compared to other languages, programming in Rascal requires few lines of code and improves speed with its list comprehensions and visit statements to traverse trees.[2]

These advantages come with certain downsides. First of all, programmers are required to write code in the Eclipse IDE, which is prone to lagging, vague error messages and other small bugs. However, with the help of the original Rascal developers these disadvantages were less of an obstacle and thus the pros outweighed the cons in this trade-off.

4.4 Language Design

With a concept diagram in place, we can now translate all elements to a textual notation.[3] As mentioned before, one of the objectives of our language is to be expressive and usable for designers, while also structured in such a way that it is easy to transform with our tools. The abstract syntax of CardScript therefore closely resembles the aforementioned diagram in Figure A.1.

To cater to the needs of game game designers, all declarations within the game are separated quite easily from each other. Every syntax element is an isolated node and can be removed and added without having to rewrite many lines. With this approach, we intend for the designers to be able to make quick modifications, and subsequently run a new analysis within minutes to learn how the removal of card X changed the dynamics of the game. This goes for specific properties of objects as well. For example, decks can have properties, such as a location, values and visibility. Usually, a hand can only be seen by the owner of that specific hand. To be able to include Hanabi-like games, the visibility property is extended. If a new game is published with another unique set of visibility rules, it can easily be added with CardScript. Furthermore, our grammar uses syntactic sugar with the intention to keep the game description readable for game designers that have little knowledge of programming languages. With the additions of small words and signs such as in the example below (=, from, to, []), the game definitions aims to be more readable for game designers.

At the top level, CardScript consists of a set of *declarations*, which define the main objects of a game. While designers can write down these declarations of their game in any order, the preferred order would be to first define all physical objects (cards, tokens, players) and subsequently describe the operational rules of the game. Some example cards, a deck and the action of distributing cards are defined as follows:

```
    AceOfHearts = [hearts, 11, 1, A],
    RedSeven = [7, red]

    deck yellowPile = []
        yellowpile top everyone
        [value = 1 higher than current, colour = yellow]

    distribute 5 from DrawingPile to [Anna, Ben, Charlie]
```

Thus, a card is defined by a variable name and then a list of properties of that specific card. Distributing cards requires a number of cards, a source pile and a list of possible destinations.

Stages

Several other language design decisions are worth mentioning, such as the definitions of the gameflow. Each game consists of *stages* in a gameflow declaration and the run-through of a game consists of

[2]`https://www.rascal-mpl.org/`, last visited 30 Oct 2019.

[3]The full code can be found in the appendix or in our GitHub repository at `https://github.com/andreavdh/masterthesis`. Several noteworthy data types will be discussed in this thesis in more detail.

executing these stages until the last is reached. Stages can have a list of conditions to specify when that specific stage should end. Examples of these conditions are when a player's score reached a certain amount of points, or when there are no more cards left in the draw pile. However, if the stage does not specify any conditions, each player will run the *turn* defined in that stage once. A stage can also consist of actions by a dealer who performs dealer-unique turns shuffling the deck and distributing cards. Otherwise, players will execute their turns in rounds until one of the conditions of that stage is no longer met and the next stage is executed. This continues until the end of the last stage, and thus the game, is reached.

Turns

Player turns can consist of required and optional actions, or a choice between several actions. In certain scenarios of the game state, choices can be added or removed depending on the situation. Furthermore, based on the decision made by players, it is possible for them to have several more required actions to take. For example, a turn in CardScript's Hanabi is shown in the following code:

```
    1 of [
        giveHint [Anna, Ben, Charlie] Value
        and then useToken hints,

      moveCard [1 .. 5] from [Anna, Ben, Charlie]
          to [bluePile, yellowPile, whitePile, greenPile, redPile]
      and then takeCard from DrawPile
          to [Anna, Ben, Charlie],

      moveCard [1 .. 5] from [Anna, Ben, Charlie]
          to [discardPile]
      and then takeCard from DrawPile
          to [Anna, Ben, Charlie]
      and then returnToken hints
    ]
```

In this snippet, players have to choose between giving a hint to another player, or move a card of their own. When giving a hint, players require to select one of the other players and subsequently pick one of the cards' values to specify what to communicate. The receiving player is then informed which cards in his/her hand have that property. When players decide to discard a card from their hand instead, the action to pick up a new card from the draw pile and the action of returning a token is added to that turn.

The full CardScript grammar can be found in Appendix A.1. In the table below we list some of the main language features and their specifications:

Primitive	Parameters	Description
Deck	ID, Cards, Visibility, Location, Condition	A deck object specifies properties such as a visibility for the players and a physical location, and conditions on how it can be accessed. See the listing on previous page.
Cards	ID, Properties	A card object has a variable name and one or more properties, such as a rank or a colour. See the listing on previous page.
Turn	Keywords, Action	A turn object specifies which actions can be taken, specified with the parameters. See the listing above.
Action	Keywords, ID, Values	Actions are defined by specific keywords and require other objects to be performed on. See the listing above.
Visibility	Keywords	One of the possible properties of a physical object. See the listing on previous page.
Expression	Operators / ID / Value	Expressions are used in conditions of decks and stages. They can consist of the standard operators (e.g. <and !=) and variable names. See the listing on previous page.

Table 4.1: Key language features.

4.4.1 Collaboration

With the basic notation for rules described, the first question and challenge posed in Section 4.1.1 is answered: CardScript aims to be an expressive language that can be mastered and used by designers quickly, while also maintaining a logical structure. However, the essence of this thesis, collaboration in card games, has not been completely formalised in CardScript yet. While the giveHint action can communicate information, there are more ways to collaborate in Hanabi. We find that collaboration does not fit in the language as a separate object: it is dynamic and interwoven in the mechanics in several ways. But this raises the question how we can model this essential aspect of CCGs, to be able to analyse it in a later stage (TC2). While frameworks for collaboration have been discussed in Chapter 3, we find them difficult to apply to card games. We propose a division of collaboration in CCGs into two categories:

1. **Direct Collaboration**: Players share their private knowledge directly with other players to reach the shared goal. The game state can remain unaffected.
2. **Indirect Collaboration**: Players use their private knowledge to perform an individual action. This influences the game state, which in turn informs other players, indirectly influencing their subgoals and next actions.

This idea resembles the difference between direct and indirect communication [51], but to the best of our knowledge it has not been described in relation to collaborative card games before. We hypothesise that collaboration is a consequence of how the actions and constraints are defined in the mechanics. These give the space to carry out specific actions and subsequently collaborate with other players. For example, the limited visibility of cards ensures that the information of the current game state is unique for each player. Players search how to overcome the information gap and this motivates players to somehow share their private knowledge to help each other. How players can collaborate in the game state of a CCG is schematically shown in Figure 4.4.

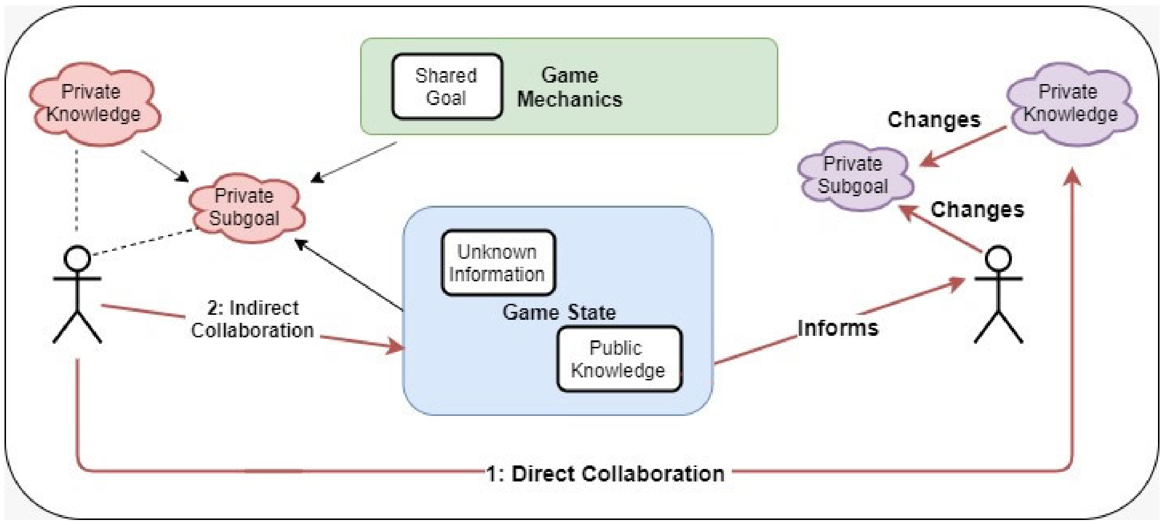


Figure 4.4: Collaboration model in a game state.

In Hanabi, direct collaboration happens if Anna gives a hint to Ben. In this case, Anna directly interacts with Ben and this changes his available knowledge. This can influence Ben's current subgoal as a result. Indirect collaboration occurs when Anna does not interact with Ben directly, but informs him through indirect communication. For example, when Anna plays a card from her hand to either one of the shared colour piles or the discard pile, she performs an individual action that changes the public knowledge and game state. This informs Ben and can cause him to change his subgoal as a result too, which influences his actions in his next turn.

The direct communication is clearly defined in the game mechanics of Hanabi, which we introduced with the action 'giveHint' in CardScript. This action lets Anna share some of her private knowledge with Ben, such that it becomes general knowledge. However, the indirect collaborative actions are not clearly defined in one specific action. This makes it possible to come up with original algorithms (of AI) to collaborate by using time or meta-hints as discussed in Chapter 3. These strategies are related to the meta-rules we introduced earlier in Chapter 2. The strategies and heuristics of players can change when playing with different team members, who might interpret the collaborative actions in ways a player did not intend, which we will show examples of in Chapter 5. We therefore conclude that the collaborative aspects of CCGs are game elements with multiple layers, which make them a challenge for game designers to implement.

4.5 Analysis

With the game mechanics defined by the game designer, we can subsequently parse the file and run it through our tool to receive feedback on the prototype. This is done in several phases, from checking the correct usage of CardScript to analysing the game for specific designer hypotheses shown in Figure 4.5 below. We will discuss each of these phases in this section.

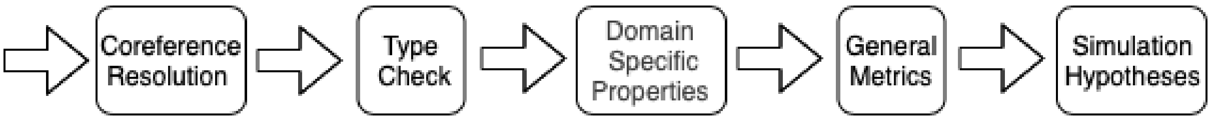


Figure 4.5: Tool Architecture

Name Resolution

Before we can analyse the game mechanics in CardScript and report relevant information to the game designer, we need to check the game definition for its correctness (TC3). These first tests only verify the correct usage of the language by the game designers and does not inform them on their design choices in this stage. However, these tests are needed to ensure we won't run into errors later.

We start with the name resolution; verifying the name usage of all objects defined in the game. While the parsing function checks for some mistakes, designers can have made contextual errors which are not spotted. Therefore we add the data type *scope*, which is similar to a lookup table. We compare the definitions and check if they are the object the syntax expects it to be. The resulting report can look like the following:

```
ERROR :: Cannot define BlueCard as card, since this variable name is already used
    somewhere else. Please use unique identifiers.
-------------------------------------------
Found errors during name resolution (definitions).
Please adjust your object definitions.
```

Type Check

When the game passes these first checks, the use of these definitions and its references is next in the type checker. For example, when the action "Move card A to B" is called, two decks are specified in that action, since a card has to move from A to B. Both A and B should have been defined as a deck and this is therefore verified. Similarly, attributes of cards, such as colours, should have been defined with a typedef as well. The references are added to a separate list connected to the referenced object. If any mistakes have been made by the designer, these are reported back in a clear message describing what the grammar expected and what it instead found in the description of the game.

```
ERROR :: Your references are incorrect.
    Expected a card but got player on line 19, column 14.
ERROR :: Your references are incorrect.
    Expected a player but got deck on line 20, column 82.
-------------------------------------------
Found errors during type checking.
Please adjust your object definition.
```

Object Relations

When the game passed these contextual grammar checks, the tool reports an overview of the relations of all defined game elements, printed to the terminal in well-arranged text, which can be seen in the listing on the next page. This overview aims to help the designer get insight in the connections between game objects. For now these relations are printed in a list, but a visualisation might be more insightful for the game designers. Therefore we propose that in future work, this mapping is changed to one or multiple graphs.

```
    For the following variable: DrawPile, the next rules have been found:
    ------------------------------------------
    shuffle DrawPile
    distribute 5 from DrawPile to [Anna, Ben, Charlie]
    takeCard from DrawPile to [Anna, Ben, Charlie]
```

Domain Specific Properties

Now that the tool has verified and confirmed that the game defined by the game designer is correctly written, we implement two analyses of the defined components (TC4). The first consists of simple static computational checks on objects such as the number of tokens, cards and players. Our analysis does not judge on the resulting statistics, since 'bad' design is difficult to qualify based on these numbers. The thresholds should be decided by the expert game designers instead.

Our tool also reports on a few components of the game state, which could spark questions in designers. For example, an important factor of games is the *decision space* of a player's turn. How many options do players have and what is the average number of choices they have to make in their turn [26]? When a game designer adds a new rule to the prototype and it influences this decision space exponentially, this is important information for the game designer to have. Some of these results, such as the above example, might not seem important when running the analysis once, but get valuable over time when evolving the prototype. With the report on these elements of the game, the designer can quickly respond to unusual changes. An example of the output of these analyses look as follows:

```
---------------------------------------------------------------
Your prototype was defined in 74 lines of code.
The following stats have been calculated for this prototype:
---------------------------------------------------------------
Required # of players :: 2 to 4
Total # of players    :: 3
Total # of teams      :: 1
Total # of cards      :: 52
Total # of decks      :: 3
Total # of token types:: 0
Total # of stages     :: 2

Actions per turn per stage ::
-    midgame has a minimum of 2 and a maximum of 5 required action(s) in each turn.
-    endgame has a minimum of 1 and a maximum of 3 required action(s) in each turn.

Decisions per turn per stage ::
-    midgame has a minimum of 2 and a maximum of 4 decision(s) to make in each turn.
-    endgame has a minimum of 2 and a maximum of 3 decision(s) to make in each turn.
```

Simulation Hypotheses

The generic framework we propose needs to be applied in a game-specific setting to analyse the indirect collaborative properties of a prototype (TC2, TC4, TC6), since collaboration is a non-trivial concept in CCGs.[4]. The last part of our analysis tool aims to give insight into these dynamic aspects of a CCG as an example of how our tool, with the right questions, can formalise these dynamic elements originated from the static mechanics of a game.

When a designer has specific ideas on what the players should be able to achieve in the game, these can be formalised in hypotheses. It is difficult to predict what players will do in a game, but we can make expected scenarios concrete to describe examples of what the mechanics may enable. In Chapter 5 we explore how this works in practice with a case study of Hanabi and we describe how our tool can help the designer with this analysis.

[4]See Section 4.4.1

Chapter 5

Case study: Hanabi

In this chapter we bring CardScript into practice by implementing the use case Hanabi and examine if the posed questions in the previous chapter on collaboration can be developed further. Therefore we first describe Hanabi in the DSL. Next, we code an interactive prototype which can also run simulations of played games. Finally, we compare the traces of actions of played games to a list of hypotheses on collaboration to analyse the collaborative aspects of Hanabi's rules with our tool.

5.1 General Metrics

Implementing the prototype of Hanabi begins with defining the game in CardScript.[1] When defined correctly, the following results will be reported by the analysis tool:

```
----------------------------------------------------------------
Your prototype was defined in 112 lines of code.
The following stats have been calculated for this prototype:
----------------------------------------------------------------
Required # of players :: 2 to 5
Total # of players    :: 3
Total # of teams      :: 1
Total # of cards      :: 50
Total # of decks      :: 10
Total # of token types:: 2
Total # of stages     :: 4

Actions per turn per stage ::
-    midgame has a minimum of 2 and a maximum of 3 required action(s) in each turn.
-    endgame has a minimum of 1 and a maximum of 2 required action(s) in each turn.

Decisions per turn per stage ::
-    midgame has a minimum of 2 and a maximum of 4 decisions to make in each turn.
-    endgame has a minimum of 2 and a maximum of 4 decisions to make in each turn.
```

This prototype is played by three players in four stages. The first stage consists of the dealer shuffling and distributing the cards, and the final stage calculates the score. The player turns are thus in the middle two stages, which have been named 'midgame' and 'endgame' in this example. When presented these numbers without further explanation, a designer might observe that the number of decks is relatively high for the amount of cards: it consists of the drawing and discard pile, the hands of three players, and the five shared colour piles. However, this number is put in perspective when looking at the defined mechanics in detail. For example, when players move a card from their hand to one of the colour piles, they do not have to know on which pile of the five colour piles their card has to be placed on. This results in two decisions and a reduced decision space for that specific turn.

The computed actions and decisions per turn show that every player has to make at least two decisions in their turn in each stage. Players first have to decide which action they want to perform and depending on that choice, more decisions are added.

Hanabi can be defined in 112 lines. However, half of these lines are used by the definitions of the unique cards. This might be shortened in the future with more list generators and keywords, to make sure that defining a game takes as little effort and lines as possible.

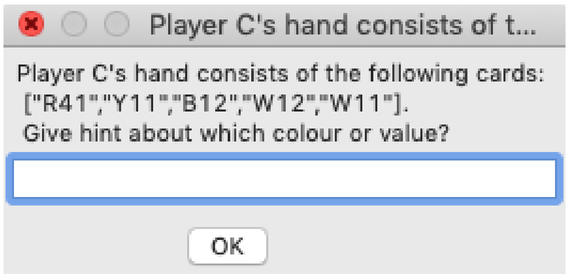
[1]The full implementation of Hanabi in CardScript can be found in Appendix A.2.

5.2 Simulation

After a game designer has defined the rules of a new game in CardScript, the next step is to test them in a prototype. Therefore we need to translate the game definition to functioning gameplay. We implement Hanabi in two ways: to play manually or run recorded gameplay in a simulation.

The translation from a CardScript definition to a simulation is done in several steps (TC5). First we translate the defined objects to data types in Rascal. This is generic for each prototype. Subsequently, all actions are translated to functions with the data types as parameters. Finally, the main function makes function calls to run the different stages until the booleans that represent the stage's conditions are no longer met. The stage constraints can be generic, such as that the draw pile is usually completely face-down, but can also be game-specific. For example, the unique constraints on visibility of cards in hands require a few lines of code in the action functions.

The first of the aforementioned approaches requires terminal input to perform player turns. When players have to make decisions, the terminal will prompt the players for input. This happens in several scenarios in Hanabi: when players decide which action to take in a turn, when they choose which specific card they want to move from their hand, or when they hint another player. An example of such a prompt can be seen in Figure 5.1, where Anna chooses to use a token to give a hint to Charlie.


Player C's hand consists of t...

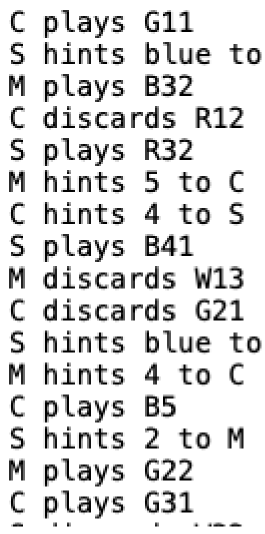
Player C's hand consists of the following cards:
["R41","Y11","B12","W12","W11"].
Give hint about which colour or value?

Figure 5.1: A turn in Hanabi where Anna decides to give a hint to Charlie.

With a few alterations to the implementation of the manual playable game, the prototype simulation can take the trace of actions as input instead of Rascal's prompts. However, the ideal automated tool tests the designer's prototype without human input to play test (TC5), which involves General Game Playing (see Chapter 3). As a first step towards this goal, we record the actions of several played games with a physical prototype and translate these *traces of actions* to digital input for the simulation. An example of such a *script* can be seen in Figure 5.2a on the next page. Each line of the script represents a turn in the game, where in this example C, S and M are the players. In the first three lines the following happens: C plays a green 1, S points out all blue cards in M's hand and M plays a blue 3. The card variable names in this example are an abbreviation of the colour, value and the number of the copy of that combination. The action 'play' means that the player had the intention to move a card from their hand to a colour pile. If this is a correct play, it is carried out by the simulation. However, if the blue pile has a blue 1 on top, instead of a blue 2, the constraints of the deck would automatically change M's action from 'move to colour pile' to 'move to discard pile and lose a life'. These are game-specific additions to the mechanics.

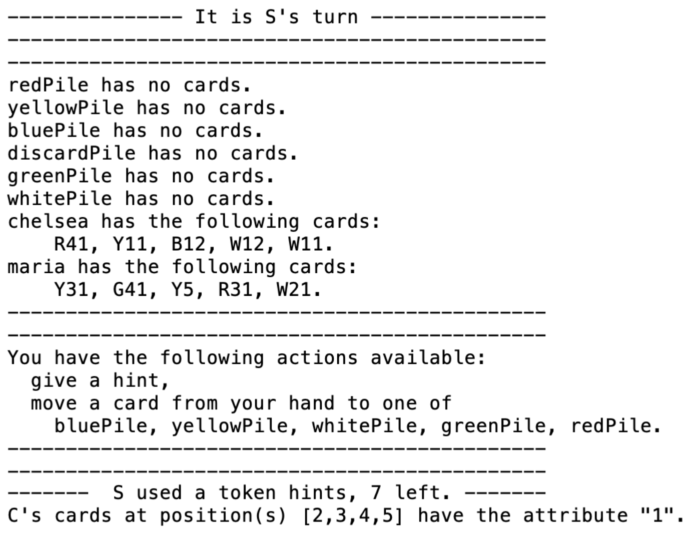
The result of both versions of the prototype is a game visualised in the terminal, of which a turn is shown in Figure 5.2b on the next page.[2] For each turn, the terminal prints the visible game state for the current player. Subsequently, the available actions in that turn are displayed. If the game is simulated, it will then pick the decision made by the player in the input file. In this specific example, S chose to inform C of the four 1's in her hand.

[2]The full trace of actions and resulting gameplay can be found on GitHub as examples of output.



```
C plays G11
S hints blue to M
M plays B32
C discards R12
S plays R32
M hints 5 to C
C hints 4 to S
S plays B41
M discards W13
C discards G21
S hints blue to C
M hints 4 to C
C plays B5
S hints 2 to M
M plays G22
C plays G31
```

(a) A fragment of the trace of actions of a game of Hanabi.



```
--------------- It is S's turn ---------------
----------------------------------------------
----------------------------------------------
redPile has no cards.
yellowPile has no cards.
bluePile has no cards.
discardPile has no cards.
greenPile has no cards.
whitePile has no cards.
chelsea has the following cards:
    R41, Y11, B12, W12, W11.
maria has the following cards:
    Y31, G41, Y5, R31, W21.
----------------------------------------------
----------------------------------------------
You have the following actions available:
  give a hint,
  move a card from your hand to one of
    bluePile, yellowPile, whitePile, greenPile, redPile.
----------------------------------------------
----------------------------------------------
-------  S used a token hints, 7 left. -------
C's cards at position(s) [2,3,4,5] have the attribute "1".
```

(b) A snippet of a simulated game of Hanabi.

5.3 Designer Hypotheses

With the Hanabi prototype implemented, we can now check to what extent the tool can report on the assumptions of designers (TC6). Game designers have certain intentions or *hypotheses* with their defined mechanics. By formalising these hypotheses of the game designers, we can argue in which *scenarios* they might occur such that the tool can filter the gameplay for these expected player actions. We take collaboration as the theme of these hypotheses as example.[3] While direct collaboration is defined with a clear action in the mechanics of Hanabi, indirect collaboration requires more context. Specific scenarios are thus helpful to recognise the teamwork in a played game. Therefore, we construct a list of hypotheses of collaborative plays in Hanabi a game designer might pose, with the theory behind them. These can subsequently be related to specific examples and scenarios. A selection of these possible hypotheses is added in the Table 5.1 on the following page.

Our tool creates patterns for these specified examples and subsequently checks the occurrences of these expected scenarios during the simulation of the trace of actions. Reconsider the first example of the hypotheses in Table 5.1. We implement code to check for the following pattern:

```
    [ Player hints property of card X ]
    [ Turn ]*
    [ Player plays card X ]
```

All found scenarios in the simulation are stored in a log. At the end of the game, the tool returns a report on all posed hypotheses. An example of these conclusions ran on our trace of actions can be seen in the listing below. The found hypotheses are related to the first and third example in Table 5.1. This relation is not very clear from just the report. However, we manually implement the selection of scenarios and the resulting report for each specific hypothesis of the game designer in this use case. Thus, the designers can link them to their original examples. This makes the distinction between general and game-specific code unclear. We desire to generalise and automate these in future work, but we decided against this since we have only one use case.

```
    The following hypotheses were checked against the gameplay:

    - (1) Player successfully used information obtained via a teammate.
    - (3) Player misunderstood information obtained via a teammate.
```

[3] See Chapter 4.4.1.

Hypothesis	**Theory**	**Example**	**Contradiction**
Hanabi requires collaboration to win.	Due to the card visibility, players have incomplete information.	Anna hints Ben about card X. Ben plays X successfully.	If all players ignore Ben, he will never know how his cards contribute to the shared goal.
Collaboration requires shared subgoals.	By working on different aspects, the shared goal will not be reached.	Anna plays a 4, Ben plays the 5 of the same colour, making finishing that colour pile their shared subgoal.	Anna hints Ben about X. Ben ignores this, making Anna's turn useless.
Collaboration can go wrong.	Games should be a challenge and not always be won.	Anna hints X. Ben arguments this action to a different intention and makes a wrong play.	
In Hanabi, players can collaborate by reducing the decision space of another player.	By limiting someone's options, their personal sub goals can not be executed and thus they change their subgoal.	Anna uses the last hint token, this forces Ben to change his actions (perhaps for the worse).	With 1 hint token left, Anna decides to leave it to Ben, such that all options remain available in his turn.

Table 5.1: Collaborative scenarios in Hanabi, according to designer hypotheses.

5.3.1 Reflection

With the trace of actions, we can relate parts of Hanabi's gameplay to the collaboration framework we discuss in Section 4.4.1. However, subtle actions of indirect collaboration stood out during the original playing of these games that are not visible in the reported gameplay. For example, the test games showed how several meta-rules influenced the collaboration as well: players moved cards around in their hand, to abide to certain heuristics (see Section 2.4). One of their meta-rules stated that players added drawn cards to the right side of the hand. If players wanted discard a card, they had to throw away the left-most card. The consequence was another meta-rule: when a card with the number 5 was close to, or in the left-most position of a hand, another player had to hint this to the owner of that card.

The players introduced these meta-rules in Hanabi to simplify the challenges of collaboration. They cannot be found in the scripts, because they are not part of Hanabi's mechanics, thus these will only be logged when the designer adds these actions in the scenarios on collaboration. Only then can they be researched with our tool. Again, these meta-rules in the example above are Hanabi-specific and require game-specific context. The exact opposite is also true: the collaboration discussed in our case study might be unique to Hanabi: collaboration in other card games are broader than the examples discussed here, which makes these scenarios difficult to generalise.

With this in mind, by adjusting the rules between simulations and rerunning the hypothesis checks on the prototype, the resulting report can show how specific adjustments influence the presence of specific scenarios, similarly to the earlier analyses. As a result of these tool reports, the time spent on the iterative process of prototyping for game design can be reduced.

Component	**KLOC**
Grammar incl. IDE	0.2
Tests + Test data	0.5
Grammar + Rule Analysis	0.4
Hypotheses Check	0.1
Extra Game Specifications	0.4

Table 5.2: CardScript Metrics.

Table 5.2 gives an overview of CardScript and workbench with lines of code (LOC) required for these analyses. Most of the code to play Hanabi resembles its definition in CardScript, with some small additions. For example, we introduce a few extra data types to preserve all relevant information for the later analysis, such as mappings to log the cards and their location to save all game states. The number of LOC for these extra specifications is relatively high, because we implement these extra data types and functions for one game specifically. We do not separate the components for general actions and game-specific components of a card game. This is something we would like to address in the future. To automate parts of the design process we need generalised functions for actions in games. While each game is unique, they all consist of specific objects such as cards and card movements. Recognising these patterns will reduce the LOC of game simulations even further.

Chapter 6

Discussion

In this chapter, we discuss our findings of automated collaborative card game design as described in the previous chapters. We have split these in successes and limitations and will discuss them accordingly.

6.1 Successes

Finding 1: The game design language CardScript can describe the mechanics of collaborative card games such as Hanabi.

With CardScript, we have introduced a DSL with the intention to support game designers through automated game design. The language aims to be both expressive and logical, such that game designers with little programming experience will be able to master it, while the defined prototype can also be parsed for later computations.

The first research question was 'What does a domain specific language for collaborative card games look like?'. We have shown that the mechanics of Hanabi, which could not be defined in other DSLs, can be translated to CardScript. The language resembles other card game languages closely and the majority of the mechanics of collaborative card games do not differ from competitive card games. The genres of card games consist mostly of the same concepts, such as cards, piles and the mechanics of how to move cards between them. However, the addition of teams requires a few alterations to specific mechanics such as the visibility and ownership of game objects, which CardScript can describe. Other CCGs such as The Mind or CodeNames might require a few modifications because of the lack of a turnorder or other unique mechanics, but we do not expect these to be a challenge to implement.

Finding 2: CardScript can analyse mechanics and dynamics of collaborative card games and report quantifiable metrics.

By using CardScript to describe a prototype, our presented tool reports several statistics of the defined game back to game designers. This might give insight in the mechanics and as a result help designers to improve the iterative design process. CardScript's tool reports several metrics when given a game definition in a correctly written file.

The second research question, 'What elements of card game rules can be analysed automatically with a game defined in CardScript?' describes the objective to explore the limits of formalising game rules. The answer is two-fold. Our static analysis can report on general mechanics. Some of these results are immediately helpful, such as the number of cards and actions per turn. Others might give more insight during the evolution of the prototype. When one of these statistics, such as the number of decisions a player has to make, unexpectedly grows exponentially, the game designer can compare the game to a previous prototype to find which modifications might be the root of the cause. The experimentation with gameplay traces that our tool supports might help designers understand the possibilities of their design by iteratively 'zooming in' on specific kinds of gameplay.

Secondly, our analysis studies parts of the dynamics of a prototype with a game simulation. For example, if a pile has no more cards during the game, while a rule forces players to take cards from that same pile, the tool shall report this to the game designer. We implement several of these checks, as a first selection for this academic prototype. Design experts will be able to come up with many more objectives that might extend our tool in the future, such that they can gain more insights through the analyses.

Finding 3: Different hypotheses of game designers about direct and indirect collaborative gameplay in Hanabi can be verified by CardScript's accompanying tool.

We find that some collaboration is mentioned in the mechanics of a CCG like Hanabi, where the rules explicitly state that *'players win or lose together'*, concealing that there are more ways to work as a team in the game than at first glance. With the actions and concrete constraints defined in the mechanic, Hanabi steers players to collaborate in certain ways, but these do not include the effects of the interaction between players. This is subtle indirect collaboration and is left to the players to discover. This teamwork puzzle is part of the game. Therefore the collaboration can evolve and take different forms when playing the same game multiple times, as seen in the development of meta-rules.

The final research question is posed as follows: 'To what extent can we formalise the designer hypotheses of scenarios in a prototype such that we can verify their existence in the gameplay?' We find that the analysis of game simulations can verify parts of the hypotheses of designers. When the game designer has clear ideas of scenarios and player behaviour as a result of developed meta-rules, we look for certain patterns in the traces of actions. The description of the scenarios in combination with the trace of actions of played games of Hanabi show how our tool can verify and report on their existence.

The objective of our tool is to support game designers in the design process. It is important for them to find out how their ideas of gameplay are originated in the rules and its interactions. With our tool we aim to give more insight in this design process and thereby improve the iterative process for game designers. Of course, CardScript is an academic prototype, and the case study on the collaborative dynamics of Hanabi is a relatively small informal evaluation. But because the approach is general, reusable and maintainable we believe it is a step towards an industrially applicable game design tool.

6.2 Limitations

Next to these successes, there are several limitations to this presented work. Here we discuss these threats to validity on the usability, scope, limited analyses, simulation and validation of the project.

Usability

From the approach of the game designers, questions about the usability of CardScript and the language workbench may arise. Designers require knowledge of basic programming to be able to use the tool. Furthermore, they need to master CardScript before they are able to define their game in the language. These are known cons of using a DSL and did not weigh against the pros of using a DSL. However, the original idea for this project includes an interactive visualisation as front end to the language, to simplify the use of our tools for game designers. Due to scoping we cut this out of the final result. By adding it as an extension to the tool in future work, we aim to facilitate the use of CardScript for game designers.

When the designers have mastered CardScript, they still need to write the code. Creating a prototype of some cubes and papers is way more time-efficient. While this is correct, the time-consuming phase for game designers currently is the play testing of the prototype to discover certain characteristics and errors. This is part of CardScript's reports, which moves the focus from the gameplay to the formalisation of designers workflow, to empower the game designers. By using CardScript several iterations of the time-consuming prototyping process might be removed.

Finally, the games that can be created are limited to the domain of CardScript's specifications. Other mechanics can therefore not be implemented without adjusting the grammar. With the domain analysis and reverse engineering of multiple card games, we have ensured that the patterns of most general mechanics of known card games can be described in CardScript. Furthermore, as shown with the addition of Hanabi's unique visibility, with some small adjustments it is possible to add new game-specific mechanics. We acknowledge that languages require maintenance and therefore it might help to have tool engineers to support game designers, separating the concerns with the game design team.

Scope

Several critical points on the scope can be made. First of all, collaborative card games have specific mechanics, that might not be relevant to other games and lack mechanics such as a die. However, as discussed in the introducing chapters: the domain of games extends in many directions. It is therefore helpful for game encodings to start with a selection of game mechanics, and when that selection of games is analysed successfully, we can broaden the scope. Our reasoning for choosing CCGs can be found in Chapter 1. Furthermore, by researching how to support the game design progress, we can gain more insights in game design in general.

More specifically, the designer hypotheses only focus on the collaboration in card games. We focus on this game mechanic because of its non-triviality. By showing how the hypotheses on dynamics and meta-rules in the gameplay of card games can be applied, we imply that the trivial mechanics likely require less effort to research.

Finally, it might be argued that the scoping of the collaborative aspects is too narrow, the framework for collaboration rudimentary. We have explained that collaboration and social interaction are very dynamic concepts, defined in definite rules but influenced by developing meta-rules. With the framework used in Chapters 4 and 5, we have attempted to explore limits of the collaborative formalism. A different approach to try could be to use ThinkLets or the programming language ABL, and we encourage this for future work [41] [52].

Limited analysis

The reports on the analysis can be considered incomplete: only a selection of the game mechanics and dynamics is analysed by the tool. While CardScript is a solid first step on the path of automated rule analysis, many more elements may be analysed when looking at the mechanics of a game. Questions in regard to meta-rules and heuristics, which to the best of our knowledge are an important concept in games, are still difficult to answer. We opened up this area of research and created a basis for further studies with this academic prototype. The project touches on several different fields of study and could benefit from each when extending the scope in any of the directions, e.g. a pattern-based or machine learning approach.

Furthermore, the reports do not inform game designers on the quality of the rules. Ideally, the tool specifies constraints that likely indicate the absence of good play. Game designers would want to be informed why and how their game is 'bad' or 'not fun' to play. This requires a quality measurement of the whole game, which needs a judgement of the quantity metrics. We have shown that this last step of quantifying game mechanic metrics is already a challenge. Therefore we need the expertise of the game designers and the designers can decide if the numbers make it a 'good' game.

Simulation

CardScript and the accompanying analyses support the design process with automated game design, but they still need actual gameplay of humans, live or recorded, to be able to run the analysis. While our project could benefit from the AI expertise, this specific limitation of predicting gameplay will remain hard, especially in collaborative games where interaction is essential. The challenges found in General Game Playing show how difficult it is to simulate human behavior and predict resulting emotions.[1] We think that human play testing will not completely disappear in the near future. Regardless, when collaborating with human play testers, the presented work can overcome this hurdle and support game designers with their design challenges.

Validation

Finally, the use of CardScript is only shown by implementing one game: Hanabi. We have tested the grammar with a few dozen tests, to make sure that the syntax is well defined. Furthermore, we tested trivial mechanics of known card games, such as playing tokens, drawing and discarding cards. However, we acknowledge that the simulation and hypothesis checks are limited to the specific collaboration found in Hanabi. The scope of the research was decided on the award-winning game, as a small but representative case for CCGs. We present CardScript as a first example of a collaborative card game analysis language and tool. In future work, CardScript may be used to research other card games, both collaborative and competitive.

[1]See Chapter 3.

Chapter 7

Conclusion

This study presents the new game design language CardScript. It describes collaborative card prototypes of game designers and aims to support them with the designing process. We study how a DSL for card game design can formalise game mechanics and subsequently analyse these with quantifiable metrics. Furthermore, by posing designer hypotheses on collaboration, we analyse if the tool can verify them with the gameplay of the prototype as input.

In Chapters 3 and 4 we showed how collaborative aspects are not addressed in other constructed languages. CardScript includes collaborative elements such as teams, communication and visibility to address this gap. The analysis of the mechanics and dynamics of a prototype are subsequently explored in Chapters 4 and 5. With the use of meta-programming language Rascal, three analyses are done. First, we checked the correct usage of the language by the game designer. Subsequently, we computed several statistics on the game objects and dynamic concepts such as the decision space. Here we also proposed how collaboration is linked to the mechanics and meta-rules of a game. Finally, we implemented a simulation to compare the trace of actions of a played game against formalised hypotheses on collaboration. These scenarios are suggested by the game designer and the tool can filter these patterns of actions. With CardScript and its accompanying workbench, we analysed the rules of the award-winning Hanabi and showed that collaborative components of the game can be extracted from the trace of actions. The game designers might use these reports to validate their ideas of scenarios in a game. CardScript aims to help the designers to understand how small adjustments to the mechanics may influence the game state as a whole.

CardScript and the accompanying tool have the potential to remove a few prototyping loops of the iterative game designing process. We argue that by extending the scope in the future, patterns might be found to specify and extend the selection of metrics further. However, how meta-rules develop and influence the gameplay remains a challenge to predict and therefore human players remain required to play test new games.

7.1 Future work

In this thesis we formalised the rules of collaborative card games in a card game design language, which analyses the game mechanics and dynamics. However, we do not claim to have defined a canonical set of language features for collaborative card games. There are several fields of research that can be considered future work to extend CardScript, which we will discuss here.

Usability

We have kept the visualisation of the simulation and reports to a minimum. However, we know that this is a challenge for designers with little knowledge of programming. A web interface to show simulations in combination with the running mechanics could aid game designers. This might be further extended and allow designers to graphically encode games in CardScript's web interface as well, such that it is no longer necessary for them to master textual notations.

Patterns

The approach taken in this thesis can be applied to many other card games. We would like to extend the use cases to other collaborative and competitive card games. By implementing these, we might stumble upon more objective metrics for the mechanics and dynamics of card games through reverse engineering.

Game designers can support the evolution of this project further as well, by discussing their assumptions and hypotheses during game design. This can strengthen our last analysis and result in insights of specific mechanics, such as the influence of the turn order or other design choices.

General Game Playing

Artificial Intelligence might also bridge several gaps in this automated game design project. If we improve general game playing agents, simulations become more independent and the time spent play testing by humans with physical prototypes is further reduced. With better simulations, game metrics such as the win-ratio, average game length or the number of decisions for players in a turn with the current game state can be measured.

In regards to Hanabi, studies use the game as a pioneer for Artificial Intelligence already, but to the best of our knowledge, they have not focused on how the meta-rules by human gameplay developed. This could be a new approach towards the challenge of the game.

Meta-rules and collaboration

CardScript creates a framework to analyse game mechanics and dynamics. One of the themes we touch upon is the relation between the mechanics, collaboration and meta-rules. These latter concepts require further study to understand game dynamics fully. Using concepts of other fields that research communication and collaboration can be used as a new approach to the dynamics in gameplay.

We deem each of these sections possible future studies. New research can extend our tool's analyses, verify other designers' hypotheses and decrease the required time for their iterative game design processes even further. This project left the aesthetics of games out of scope with reason. However, we believe that with the above-mentioned suggestions, formalising the concepts of what make a game 'fun' are within grasp.

Acknowledgements

First of all I would like to thank my supervisor Riemer van Rozen, for guiding this research. His previous work, personal involvement and endless enthusiasm have been of great help for this thesis.

I would also like to thank everyone involved with the master Software Engineering at the University of Amsterdam and at CWI. Especially my fellow students, including but not limited to Aynel, Anna and Stephan, whom I have intensively filled brain cells with, alternated with lunches and laughter.

Finally, I would like to extend my gratitude towards my family and friends, who have supported my decisions, played endless amounts of Hanabi with me for 'scientific research', read and corrected my interestingly worded sentences and listened to my complaints with endless patience. They have been an immense help in regard to the stress I have endured during these intense months. I could not have done this without any of them.

Bibliography

[1] M. Dummett, “The history of card games”, *European Review*, vol. 1, no. 2, pp. 125–135, 1993.

[2] U. Ritterfeld, M. Cody, and P. Vorderer, *Serious games: Mechanisms and effects.* Routledge, 2009.

[3] T. L. Taylor, *Raising the Stakes: E-sports and the Professionalization of Computer Gaming.* Mit Press, 2012.

[4] K. Salen and E. Zimmerman, *Rules of Play - Game Design Fundamentals.* The MIT Press, 2003, ISBN: 9780262240451.

[5] T. Fullerton, *Game design workshop: a playcentric approach to creating innovative games.* AK Peters/CRC Press, 2018.

[6] R. Van Rozen and J. Dormans, “Adapting game mechanics with micro-machinations”, 2014.

[7] C. Bell and M. Goadrich, “Automated playtesting with recycled cardstock”, 2016.

[8] A. M. Smith, M. J. Nelson, and M. Mateas, “Ludocore: A logical game engine for modeling videogames”, in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, IEEE, 2010, pp. 91–98.

[9] J. Schell, *The Art of Game Design: A book of lenses.* AK Peters/CRC Press, 2019.

[10] M. J. Nelson, J. Togelius, C. Browne, and M. Cook, “Rules and mechanics”, in *Procedural Content Generation in Games*, Springer, 2016, pp. 99–121.

[11] J. P. Zagal, J. Rick, and I. Hsi, “Collaborative games: Lessons learned from board games”, *Simulation & Gaming*, vol. 37, no. 1, pp. 24–40, 2006.

[12] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius, “A card game description language”, in *European Conference on the Applications of Evolutionary Computation*, Springer, 2013, pp. 254–263.

[13] A. Azadegan and C. Harteveld, “Work for or against players: On the use of collaboration engineering for collaborative games”, 2014.

[14] R. Hunicke, M. LeBlanc, and R. Zubek, “Mda: A formal approach to game design and game research”, in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, 2004, p. 1722.

[15] M. A. Schnabel, T. Lo, and S. Aydin, “Gamification and rule based design strategies in architecture education”, in *DesignEd Asia Conference*, 2014.

[16] A. Marczewski, “Gamification: A simple introduction and a bit more, (self-published on amazon digital services, 2013)”, *Kindle edition, Loc*, vol. 1405, 2013.

[17] J. Juul, *Half-real: Video games between real rules and fictional worlds.* MIT press, 2011.

[18] K. Salen, K. S. Tekinbaş, and E. Zimmerman, *Rules of play: Game design fundamentals.* MIT press, 2004.

[19] J. C. Osborn, A. Grow, and M. Mateas, “Modular computational critics for games.”, in *AIIDE*, 2013.

[20] A. Van Deursen and P. Klint, “Domain-specific language design requires feature descriptions”, *Journal of Computing and Information Technology*, vol. 10, no. 1, pp. 1–17, 2002.

[21] A. V. Deursen and P. Klint, “Little languages: Little maintenance?”, *Journal of Software Maintenance: Research and Practice*, vol. 10, no. 2, pp. 75–92, 1998.

[22] S. Edelkamp, T. Federholzner, and P. Kissmann, “Searching with partial belief states in general games with incomplete information”, in *Annual Conference on Artificial Intelligence*, Springer, 2012, pp. 25–36.

[23] E. Adams and J. Dormans, *Game mechanics: advanced game design.* New Riders, 2012.

[24] K. Salen, K. S. Tekinbas, and E. Zimmerman, *The game design reader: A rules of play anthology.* MIT press, 2006.

[25] M. J. Nelson and M. Mateas, "Towards automated game design", in *Congress of the Italian Association for Artificial Intelligence*, Springer, 2007, pp. 626–637.

[26] S. Bjork and J. Holopainen, *Patterns in game design.* Charles River Media Hingham, 2005, vol. 11.

[27] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games", in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, IEEE, 2010, pp. 273–280.

[28] R. van Rozen, "A pattern-based game mechanics design assistant.", in *FDG*, 2015.

[29] M. J. Nelson and M. Mateas, "An interactive game-design assistant", in *Proceedings of the 13th international conference on Intelligent user interfaces*, ACM, 2008, pp. 90–98.

[30] J. W. Romein, H. E. Bal, and D. Grune, "An application domain specific language for describing board games", in *Parallel and Distributed Processing Techniques and Applications*, vol. 1, 1997, pp. 305–314.

[31] F. E. Hernandez and F. R. Ortega, "Eberos gml2d: A graphical domain-specific language for modeling 2d video games", in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, Citeseer, 2010, pp. 1–1.

[32] A. W. Furtado and A. L. Santos, "Using domain-specific modeling towards computer games development industrialization", in *The 6th OOPSLA workshop on domain-specific modeling (DSM06)*, Citeseer, 2006.

[33] C. Browne and F. Maire, "Evolutionary game design", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[34] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition", *AI magazine*, vol. 26, no. 2, pp. 62–62, 2005.

[35] M. Thielscher, "A general game description language for incomplete information games", in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[36] J. M. Font Fernández, D. Manrique Gamo, T. Mahlmann, and J. Togelius, "Towards the automatic generation of card games through grammar-guided genetic programming", 2013.

[37] F. de Mesentier Silva, A. Isaksen, J. Togelius, and A. Nealen, "Generating heuristics for novice players", in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, IEEE, 2016, pp. 1–8.

[38] J. Togelius, M. J. Nelson, and A. Liapis, "Characteristics of generatable games", *intelligence*, vol. 9, p. 20, 2014.

[39] G. S. Elias, R. Garfield, and K. R. Gutschera, *Characteristics of games.* MIT Press, 2012.

[40] C. I. Sedano, M. B. Carvalho, N. Secco, and C. S. Longstreet, "Collaborative and cooperative games: Facts and assumptions", in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, IEEE, 2013, pp. 370–376.

[41] G. L. Kolfschoten, R. O. Briggs, J. H. Appelman, and G.-J. De Vreede, "Thinklets as building blocks for collaboration processes: A further conceptualization", in *International Conference on Collaboration and Technology*, Springer, 2004, pp. 137–152.

[42] H. Osawa, "Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information.", in *AAAI workshop: Computer Poker and Imperfect Information*, 2015, pp. 37–43.

[43] C. Cox, J. De Silva, P. Deorsey, F. H. Kenter, T. Retter, and J. Tobin, "How to make the perfect fireworks display: Two strategies for hanabi", *Mathematics Magazine*, vol. 88, no. 5, pp. 323–336, 2015.

[44] M. Eger and D. Gruss, "Wait a second: Playing hanabi without giving hints", in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, ACM, 2019, p. 14.

[45] M. Eger, C. Martens, and M. A. Córdoba, "An intentional ai for hanabi", in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, IEEE, 2017, pp. 68–75.

[46] B. Bouzy, “Playing hanabi near-optimally”, in *Advances in Computer Games*, Springer, 2017, pp. 51–62.

[47] R. Canaan, H. Shen, R. Torrado, J. Togelius, A. Nealen, and S. Menzel, “Evolving agents for the hanabi 2018 cig competition”, in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2018, pp. 1–8.

[48] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, *et al.*, “The hanabi challenge: A new frontier for ai research”, *arXiv preprint arXiv:1902.00506*, 2019.

[49] R. van Rozen and Q. Heijn, “Measuring quality of grammars for procedural level generation”, in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, ACM, 2018, p. 56.

[50] R. Prieto-Dıaz, “Domain analysis: An introduction”, *ACM SIGSOFT Software Engineering Notes*, vol. 15, no. 2, pp. 47–54, 1990.

[51] C. Joyce, “The impact of direct and indirect communication”, *Journal of the International Ombudsman Association*, 2012.

[52] M. Mateas and A. Stern, “A behavior language for story-based believable agents”, *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 39–47, 2002.

Appendix A

Appendix

A.1 CardScript Grammar

```
module lang::crds::grammar

start syntax CRDS
 = game: "game" ID Decl*;

/**************************************************************************
 * Main game objects of the syntax.
 **************************************************************************/
syntax Decl
 = @Category="Decl"
   typedef: "typedef" ID "=" "[" {Exp ","}+ "]"        // E.g. names, values, colours.
 | deck: "deck" ID "=" "[" {Card ","}* "]" ID Prop+ "[" {Condition ","}* "]"
 | team: "team" ID "=" "[" {ID ","}+ "]"           // No teams defined == FFA.
 | gameflow: "gameflow" "=" "[" Turnorder {Stage ","}+ "]"
 | players: "players" "=" "[" {Hands ","}+ "]"
 | tokens: "tokens" "=" "[" {Token ","}+ "]"        // Set of all possible tokens.
 | rules: "rules" "=" "[" {Rule ","}+ "]";

syntax Card
 = card: Exp "=" "[" {Exp ","}+ "]";

syntax Token
 = token: ID "=" "[" "start" VALUE "," "max" VALUE "]" ID Prop+;

syntax Rule                           // General rules.
 = playerCount: "players" "=" VALUE "to" VALUE
 | points: "scoring" "=" "[" {Scoring ","}+ "]";      // Points per card.

syntax Stage                          // Script of the game.
 = stage: "stage" ID "=" {Condition ","}+ Playerlist "[" {Turn ","}* "]"
 | basic: "stage" ID "=" Playerlist "[" {Turn ","}* "]";

syntax Turnorder
 = turnorder: "turnorder" "=" "[" {ID ","}* "]";

syntax Turn
 = req: "req" Action
 | opt: "opt" Action
 | choice: VALUE "of" "[" {Action ","}+ "]";

syntax Action                         // Specific rules
 = @Category="Action" shuffleDeck: "shuffle" ID
 | distributeCards: "distribute" VALUE "from" ID "to" "[" {ID ","}+ "]"
 | takeCard:     "takeCard" "from" ID "to" "[" {ID ","}+ "]"           // from drawpile
 | moveCard:     "moveCard" Exp "from" "[" {ID ","}+ "]" "to" "[" {ID ","}+ "]"  // from a to b
 | moveToken:      "moveToken" VALUE "from" ID "to" ID
 | useToken:     "useToken" ID
 | returnToken:    "returnToken" ID
```

```
    | obtainKnowledge: "getInfo" ID
    | communicate:   "giveHint" "[" {ID ","}* "]" Exp    // list of locs, value
    | changeTurnorder: "changeTurns" Turnorder
    | calculateScore:  "calculateScore" ID+
    | endGame:      "endGame"
   > left sequence: Action "and then" Action;

  /***************************************************************************
   * Main properties of objects.
   ***************************************************************************/
  syntax Prop
   = visibility: Vis
   | usability: Usa;

  syntax Vis
   = allcards: "all"
    | none:    "none"
    | top:     "top"
    | everyone: "everyone"
    | team:    "team"
    | hanabi:    "hanabi"
    | hand:    "hand";

  syntax Usa
   = draw:    "draw"
    | discard:  "discard"
    | play:    "play"
    | use:     "use"
    | ret:     "return";

  syntax Playerlist
   = allplayers: "all"
    | dealer:  "dealer"
    | turns: "turns"
    | teams:    "teams";

  syntax Scoring
   = s: ID "=" VALUE
    | allcards: "each" "=" VALUE;

  syntax Hands
   = hands: ID "has" ID;

  syntax Condition
   = deckCondition: "if" Exp "then" Action
    | stageCondition: "while" Exp "do"
    | totalTurns: "for" Exp "turns" "do"
    | higher: "value" "=" "higher than current"
    | lower: "value" "=" "lower than current"
    | xhigher: "value" "=" VALUE "higher than current"
    | color: "color" "=" ID;

  syntax Exp
   = var: ID
    | val: VALUE
    | l: LIST
    | obj: ID"."ID
    | empty: "empty"
   > left (
      gt: Exp l "\>" Exp r
    | ge: Exp l "\>=" Exp r
    | lt: Exp l "\<" Exp r
    | le: Exp l "\<=" Exp r
    | neq: Exp l "!=" Exp r
    | eq: Exp l "==" Exp r
    | and: Exp l "&&" Exp r
    | or: Exp l "||" Exp r);

  /***************************************************************************
   * Basis.
   ***************************************************************************/
  syntax ID
   = id: NAME;
```

```

    syntax LIST
     = l: "[" VALUE ".." VALUE "]";

    syntax BOOL
     = @category="String" tru: "true" | fal: "false";

    lexical NAME
     = @category="String" ([a-zA-Z_$] [a-zA-Z0-9_$]* !>> [a-zA-Z0-9_\$]) \ Reserved;

    lexical VALUE
     = @category="Number" ([0-9]+([.][0-9]+?)?);

    /*************************************************************************
     * Layout. Should be ignored by the parser.
     *************************************************************************/
    layout LAYOUTLIST
      = LAYOUT* !>> [\t-\n \r \ ] !>> "//" !>> "/*";

    lexical LAYOUT
    = Comment
    | [\t-\n \r \ ];

    lexical Comment
     = @category="Comment" "/*" (![*] | [*] !>> [/])* "*/"
     | @category="Comment" "//" ![\n]* [\n];
```

A.2 Hanabi in CardScript

```
game Hanabi

typedef Value = [1, 2, 3, 4, 5]
typedef Colour = [blue, green, red, white, yellow]
typedef location = [bluepile, greenpile, redpile, whitepile, yellowpile, discardpile
    , drawpile, handA, handB, handC, tablelives, tablehints]

deck allCards =
[
  B11 = [blue, 1],
  B12 = [blue, 1],
  B13 = [blue, 1],
  B21 = [blue, 2],
  B22 = [blue, 2],
  B31 = [blue, 3],
  B32 = [blue, 3],
  B41 = [blue, 4],
  B42 = [blue, 4],
  B5 =  [blue, 5],
  G11 = [green, 1],
  G12 = [green, 1],
  G13 = [green, 1],
  G21 = [green, 2],
  G22 = [green, 2],
  G31 = [green, 3],
  G32 = [green, 3],
  G41 = [green, 4],
  G42 = [green, 4],
  G5 =  [green, 5],
  R11 = [red, 1],
  R12 = [red, 1],
  R13 = [red, 1],
  R21 = [red, 2],
  R22 = [red, 2],
  R31 = [red, 3],
  R32 = [red, 3],
  R41 = [red, 4],
  R42 = [red, 4],
  R5 =  [red, 5],
  Y11 = [yellow, 1],
  Y12 = [yellow, 1],
  Y13 = [yellow, 1],
  Y21 = [yellow, 2],
  Y22 = [yellow, 2],
  Y31 = [yellow, 3],
  Y32 = [yellow, 3],
  Y41 = [yellow, 4],
  Y42 = [yellow, 4],
  Y5 =  [yellow, 5],
  W11 = [white, 1],
  W12 = [white, 1],
  W13 = [white, 1],
  W21 = [white, 2],
  W22 = [white, 2],
  W31 = [white, 3],
  W32 = [white, 3],
  W41 = [white, 4],
  W42 = [white, 4],
  W5 =  [white, 5]
] drawpile none []

deck yellowPile =   [] yellowpile   top everyone [value = 1 higher than current,
    color = yellow]
deck greenPile =  [] greenpile  top everyone [value = 1 higher than current, color =
    green]
deck redPile =    [] redpile    top everyone [value = 1 higher than current, color =
    red]
deck whitePile =  [] whitepile  top everyone [value = 1 higher than current, color =
    white]
deck bluePile =   [] bluepile   top everyone [value = 1 higher than current, color =
    blue]
    deck discardPile = [] discardpile all everyone []
    deck handPlayerA = [] handA all hanabi []
    deck handPlayerB = [] handB all hanabi []
    deck handPlayerC = [] handC all hanabi []

    team allPlayers = [A, B, C]

    tokens = [
      lives = [start 3, max 3] tablelives all,
      hints = [start 8, max 8] tablehints all
    ]

    players = [A has handPlayerA, B has handPlayerB, C has handPlayerC]


    rules = [
      scoring = [ each = 1 ]
    ]

    gameflow = [
      turnorder = [A, B, C]
      stage dealing = dealer [req shuffle allCards, req distribute 5 from allCards to [
          handPlayerA, handPlayerB, handPlayerC]],

      stage midgame = while lives != 0 do, while allCards != empty do
        turns
        [1 of [ giveHint [handPlayerA, handPlayerB, handPlayerC] Value and then useToken
             hints,

            moveCard [1 .. 5] from [handPlayerA, handPlayerB, handPlayerC] to [bluePile,
                 yellowPile, whitePile, greenPile, redPile]
            and then takeCard from allCards to [handPlayerA, handPlayerB, handPlayerC],

            moveCard [1 .. 5] from [handPlayerA, handPlayerB, handPlayerC] to [
                discardPile]
            and then takeCard from allCards to [handPlayerA, handPlayerB, handPlayerC]
          and then returnToken hints
          ]],

      stage endgame = while lives != 0, for 1 turns
        turns
        [1 of [ giveHint [handPlayerA, handPlayerB, handPlayerC] Value and then useToken
             hints,

            moveCard [1 .. 5] from [handPlayerA, handPlayerB, handPlayerC] to [bluePile,
                 yellowPile, whitePile, greenPile, redPile],

            moveCard [1 .. 5] from [handPlayerA, handPlayerB, handPlayerC] to [
                discardPile]
            and then returnToken hints
          ]],
      stage calculateScore = dealer [req calculateScore yellowPile greenPile redPile
          whitePile bluePile]
    ]
```

A.3 Example Trace of Actions

```
S hints 1 to C
M hints white to C
C plays W11
S hints white to M
M plays W21
C discards W12
S hints 5 to M
M hints 5 to S
C plays B12
S discards B11
M hints 1 to S
C hints blue to S
S plays R11
M discards R31
C plays Y11
S plays B21
M hints 3 to S
C hints 2 to S
S plays Y21
M hints white to C
C plays W31
S discards W41
M discards G41
C discards R41
S hints yellow to M
M plays Y31
C hints red to M
S hints 2 to M
M plays R22
C discards R21
S hints 4 to M
M plays Y41
C discards B13
S hints green to C
M discards B22
C discards R5
S hints 1 to C
M plays Y5
C plays G11
S hints blue to M
M plays B32
C discards R12
S plays R32
M hints 5 to C
C hints 4 to S
S plays B41
M discards W13
C discards G21
S hints blue to C
M hints 4 to C
C plays B5
S hints 2 to M
M plays G22
C plays G31
S discards W32
M plays G42
C plays W42 // last card
S plays G5
M hints white to S
C plays R42
```

A.4 Domain Analysis

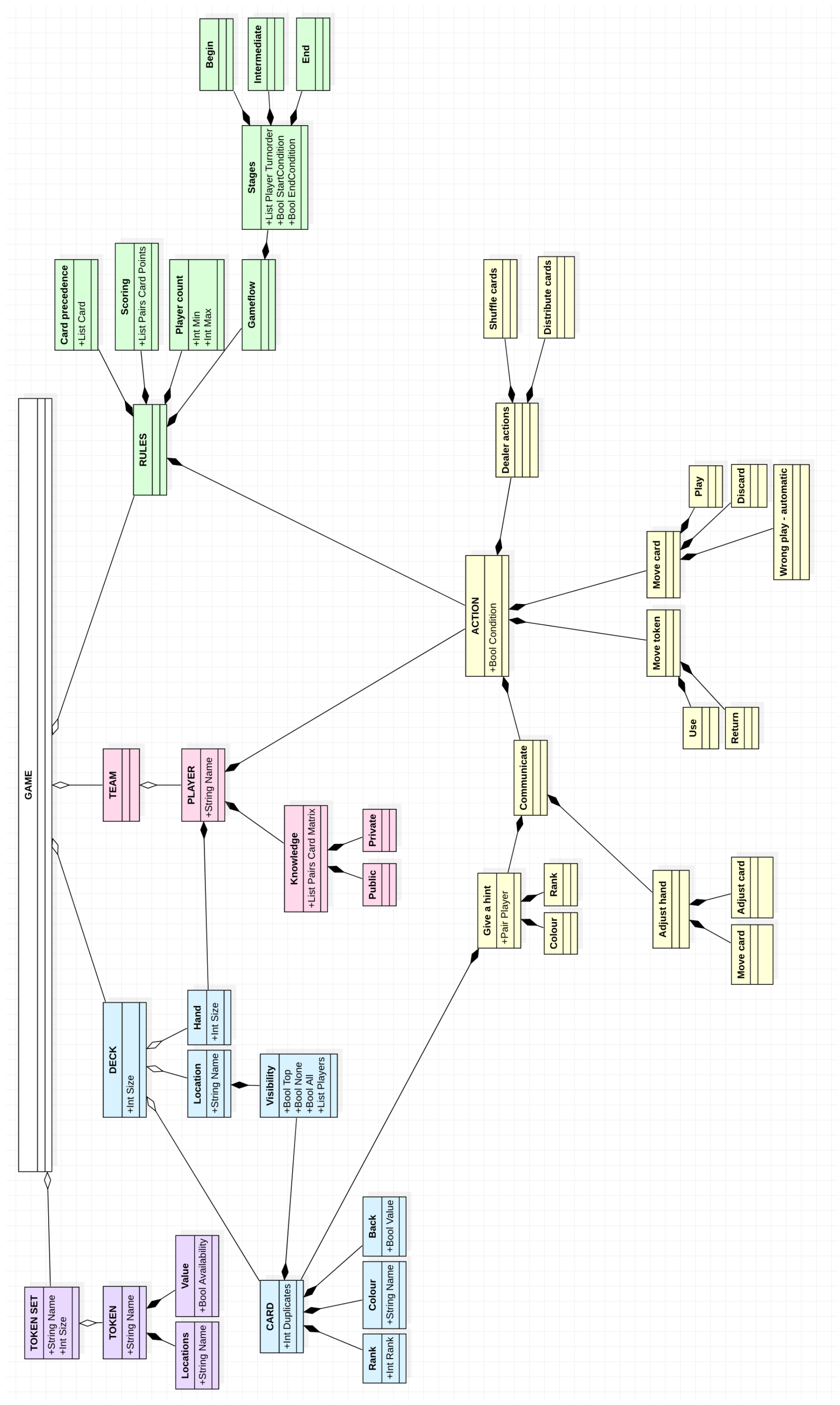


Figure A.1: Class diagram of collaborative card games.