I Play, You Play: A Framework for Card Games in a Multi-touch Table

Nuno Filipe de Matos Monteiro Maia

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Dr. Ana Maria Severino de Almeida e Paiva

Examination Committee

Chairperson:	Prof. Dr. António Manuel Ferreira Rito da Silva
Supervisor:	Prof. Dr. Ana Maria Severino de Almeida e Paiva
Member of the Committee:	Prof. Dr. Teresa Isabel Lopes Romão

November 2015

To my grandfather

Acknowledgments

I would like to thank my advisor, Professor Ana Paiva, for her invaluable experience, expertise and guidance provided throughout the development of my thesis. Her patience, optimism and wise advices were always of the utmost importance and kept me going and striving to accomplish the best possible outcome.

Additionally, I would like to thank my colleagues in the research project PArCEIRO, particularly Tiago Ribeiro and Patrícia Alves-Oliveira, for all their valuable insights and prompt guidance.

Lastly, I am grateful to my family, in particular to my parents, who have always believed in me, and to my closest friends, who have always supported me.

Resumo

Os jogos de cartas datam de há muitos anos. Existem enumeras variações dos jogos e um grande número de diferentes famílias. Normalmente, as pessoas juntam-se em torno de uma mesa para jogar às cartas mas, com o aparecimento dos jogos de vídeo, os aspetos sociais do tradicional jogo de cartas foram alvo de uma transformação. Em alternativa, as pessoas sentam-se em frente a um ecrã e jogam versões mais modernas dos mesmos jogos e interagem com outros sem terem que estar presentes no mesmo espaço físico. Investigadores têm tentado combinar as características tradicionais e mais modernas dos jogos de cartas, de forma criar uma extensão do jogo tradicional recorrendo à tecnologia como RFID, dispositivos portáteis e computadores, e até agentes virtuais. O objetivo deste trabalho recai sobre a criação de uma framework para fazer jogos de cartas para mesas multi-toque. Este conjunto de bibliotecas e componentes que constituem a framework permitem o desenvolvimento do estado e das regras dos jogos, e a integração de agentes autónomos capazes de jogar esses mesmos jogos. Mais importante é o facto de estes jogos serem jogados com cartas físicas ao invés de cartas digitais que aparecem no ecrã. A utilização de marcadores, que são detetados e identificados pela superfície multi-toque, permite uma interação direta e mais natural com os jogos desenvolvidos.

De maneira a avaliar e validar a framework, foi desenvolvido um conhecido jogo de cartas Português: a Sueca. O jogo foi testado por grupos de três pessoas que jogaram com, e contra, um agente autónomo. Os resultados destes testes mostram que, por vezes, a identificação dos marcadores é errada. No entanto, a percentagem de identificações erradas é diminuída quando se limita o número de marcadores a comparar, resultando num menor impacto durante os jogos.

Palavras-chave: jogos de cartas, marcadores, framework de jogos de cartas, mesa multi-toque, robôs sociais

Abstract

Card games have been around for thousand of years. There are countless variations of games and a large number of different families. Usually, people gather around a table to play card games, but the social aspects of the traditional card game have undergone a transformation, with the appearance of video games. As an alternative, people sit in front of a screen and can play a more modern version of the game and interact with others without having to be in the same locations as their peers.

Researchers have been trying to blend the traditional and modern facets of card games by extending the traditional game with technology such as RFID tags, portable devices and computers, and even virtual agents. This work's intent was to create a framework for building card games for a multi-touch table. The set of libraries and components of the framework allow the development of the state and rules of the game, and the integration of autonomous agents to play the games. More importantly, the developed games are played using physical playing cards. Using fiducial markers that are detected and recognized by the multi-touch surface, it is possible to interact directly with the game using physical cards instead of digital representations.

So as to evaluate and validate the framework, it was used to build a well known Portuguese card game: *Sueca*. The game was tested by groups of three people that played with and against an autonomous agent. Results show that the recognition of the cards is, sometimes, incorrect. Nevertheless, limiting the range of codes available to the game, the percentage of incorrect values decreases, reducing their impact during the game.

Keywords: card games, fiducial markers, card games framework, multi-touch table, social robots

Contents

List of Tables

List of Figures

List of Acronyms

CGDL	Card Game Description Language
GDL	Game Description Language
GUI	Graphical User Interface
MOM	Message Oriented Middleware
PDA	Personal Digital Assistant
RFID	Radio-frequency Identification
SDK	Software Development Kit
TUI	Tangible User Interface
UI	User Interface

Chapter 1

Introduction

Traditional games have gathered people around tables for centuries. They are a great source of entertainment and joy to everyone who plays them. Throughout the years, some of these games have been ported to the digital world. However, the game experience changed as keyboards and mouse pointers started to replace the fast hand movements and the enthusiasm of collecting their winnings. Recent advances on multi-touch surfaces and object recognition algorithms have allowed these games to recover some of their essence. Tabletop games have proven to enhance the game experience with visual and sound effects, making the game more dynamic [1].

Frameworks for game creation are not a new concept. Even on touch surfaces, there are already frameworks to develop board games using tangible objects [2]. However, previous research shows that card games only gained some attention recently, and there is still space to explore new ideas. Therefore, the creation of a framework for building games played with cards on multi-touch tables, adding the use of real playing cards and the interaction with autonomous agents, represents a novel approach.

The PArCEIRO project[1] focuses on the social interaction between humans and robots, aiming to study and create robots with the capabilities of a social companion. This interaction is sustained by the development of entertaining activities such as card games on tabletop displays. One possible scenario is having elderly people playing card games with a robot, and using real playing cards. The early stage of the project requires some studies and research to be made before the beginning of the development process.

1.1 Challenges

The challenge of this work is to develop a set of tools that can be used to build tabletop card games. Furthermore, instead of having a digital representation, the framework promotes the usage physical playing cards to play these games. Simultaneously, it also offers an easy integration of autonomous agents into the games.

[1] http://gaips.inesc-id.pt/parceiro

1.2 Contributions

The framework was the result of addressing the challenges purposed for this work. The assembling of these tools made possible the development of different scenarios and activities for the PArCEIRO project, such as the *Sueca* and *Coup* card games. Both games are prepared to be played using physical playing cards and with an embodied agent that interacts with other players and is capable of playing the game.

*

The following chapters of this document aim to provide a better understanding of some concepts (Chapter 2), as well as giving an overview of the state-of-the-art (Chapter 3). Before the development of the work, a study was arranged to better understand the different interactions that occur during a card game (Chapter 4). A description of the developed framework and its components is also given (Chapter 5), followed by a detailed description of some cases where the framework was used (Chapter 6). Finally, the document ends with the evaluation of the framework (Chapter 7) and the conclusions and future work proposals (Chapter 8).

Chapter 2

Background

The work developed relies on previous knowledge about some key concepts. The following sections provide a broader perspective of these concepts.

2.1 Card games

A card game is a social activity which attracts friends, families, and strangers together [3]. The world of card games is extremely diverse, from the most simple game to the most complex.

For people that are new to card games, there are some basic concepts that are important to refer and explain. A standard *deck* has fifty-two cards, each with an associated *suit* and *rank*. These cards are equally divided into four suits: Spades ($\spadesuit$), Hearts ($\heartsuit$), Clubs ($\clubsuit$), and Diamonds ($\diamondsuit$). In each suit, there are thirteen rankings or ranks that range from 2 to 10, plus a Jack, a Queen, a King, and an Ace. Although some games have different ranking systems, the most common one defines 2 as the lowest rank and Ace the highest. Card games are often divided by several stages. First, there is the *hand* in which all players compete and points earned count toward an overall game score. Hands usually are broken into *rounds*, in which each player is required to perform an action on their own *turn*. Moreover, some games are played in *tricks*. A trick is composed of all the cards played in a single round, typically one from each player. Another concept that is common on trick-taking games is the *trump*. A trump is a suit that beats all the other suits during one hand. However, if two trumps are played in the same trick, the winner is the trump with the highest rank value.

2.2 Tangible User Interfaces

The evolution of operating systems made Graphical User Interfaces (GUI) the standard paradigm for human-computer interaction. GUIs are represented on displays and manipulated using peripheral devices, such as keyboards and mice. However, without the use of these devices, interacting with the interfaces becomes too cumbersome. Tangible User Interfaces (TUI), on the other hand, can take advantage of physical objects that people can easily manipulate. TUIs use physical forms that fit seamlessly into a user’s physical environment, providing a different approach to control digital information [4]. This type of interfaces enables collaboration and learning scenarios, taking advantage of the natural way humans interact with objects.

Chapter 3

Related Work

Creating a new framework involves a profound knowledge about every aspect of the problem that it is trying to solve. Given the main goal of this work, it is relevant to analyze different approaches on how to create a representation of a game, particularly the representation of card games. At the same time, it is important to do an extended analysis of the state-of-the-art on applications which take advantage of tangible technology.

The following sections will focus on previous research towards the description and representation of card games (Section 3.1) , and the adoption of multi-touch tables and real-life objects to enhance the game experience (Section 3.2). On top of that, the interaction between multi-touch systems and robots will also be discussed (Section 3.3).

3.1 Card Games Representation

Every game has a list of objectives that players have to accomplish in order to win, or complete it. Moreover, players are often restricted by some rules they need to act in accordance with. Hence, the creation of games revolves around the various paths that can lead players to finish the games.

Developing a framework for card games demands a deep knowledge of both the common and specific concepts of a set of card games, and their rules. For that reason, this section focuses on a description language built specifically to describe a set of card games, including their general concepts. Additionally, it refers a web-based application for creating card games and how these are created based on their rules. Lastly, in Table 3.2, a comparison between the different approaches is made.

Card Game Description Language

The Game Description Language (GDL) [5] and GDL II [6] were developed to represent the rules of any game. The first is used to describe finite, discrete and deterministic games of complete information. In complete information games, players are aware of every possible move of other players and have complete knowledge of the games' current state. The second is intended to extend the original GDL in order to describe incomplete information games as well. Incomplete information games contain elements of chance, which means it is impossible for players to have full knowledge of the game state. In addition, private information, such as a player's hand on a card game, prevents other players from having that knowledge. But, although GDL and GDL II are able to describe most games, there are particular cases which require a slightly more expressive approach. The absence of expressiveness often results in a language with added verbosity, and card games are a suitable example of games that can benefit of a more meaningful description.

The Card Game Description Language (CGDL) [7] is an expressive language that is able to describe a great variety of card games. The main purpose of this description language is to provide a mean towards the automatic generation of new card games. Even though the CGDL does not share the same goals as this work, it is still valuable to know the components that make the generated card games and how they are able to manipulate the state of these games.

A considerable variety of card games have some similarities. A significant amount of those games only need one or two decks of cards, and some might require a collection of tokens to bid. Given these and other analogous characteristics, it should be possible to represent a large number of card games by adjusting their rules. The language is capable of representing a set of games that share more principles than just tokens or decks of cards. Therefore, it is important to detail these principles and get a deeper insight of the building blocks of card games before describing the language.

- A card game has a certain number of stages, defined by a set of rules
- A card game is played by P players;
- *Tokens*, which virtually represent the concept of chips;
- A *card location* is a place where any number of cards are placed during the game. There are several card locations L in every game:
  - A set of hands H, one for each player;
  - A standard deck of cards D that consists of thirteen cards of each suit (*Spades*, *Hearts*, *Clubs* and *Diamonds*);
  - A variable number of table locations T, where cards are placed facing up.
- A *token location* K, similarly to card locations, defines a place where any number of tokens reside.

Adding to the axioms listed before, there are also the concepts of *ranking*, that defines the score of each card combination, and *winning conditions*.

A card game is represented by a tuple with three elements. The tuple consists of a set of stages, a ranking, and a set of winning conditions. Stages are described with sets of rules, which have the form "if *antecedent* then *consequent*". Rules state that, in order to perform a certain action specified by the *consequent*, the *antecedent* must be conformed. However, some rules do not need a precondition (antecedent) to be played.

A list of preconditions (antecedents) and actions (consequents) is provided by the CGDL, so that different rearrangements can be made to produce rules. The antecedent *have, 3 of Spades*, for instance, checks whether the current player has a card with rank three and suit of Spades in his hand. Moreover, a consequent such as *playit* enables the player to play the card checked on the precondition. Hence, the rule would have the form "*have, 3 of Spades _ playit*".

Similarly to rules, players have three distinct states: *done*, *next*, and *out*. A player can be *done* if they do not satisfy the antecedent of any of the stage's rules or if they want to continue playing the current stage. However, if a player want to pass their turn then they activate *next* state. Finally, a player is *out* when they decide to quit the game.

The ranking defines the value of individual cards or combinations of cards. It maintains an association between plays and their corresponding values. Nonetheless, certain games do not assign a score to a determined set of plays, therefore a standard ranking of cards is assumed: $1 > 13 > 12 > 11 > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2$. As an example, Table 3.1 illustrates the representation of the card game UNO[1]. On the first stage, the computer deals seven cards to all players and an additional one to *T0*, which represents the shared table location where cards are played. The second stage represents the actual game cycle. The first two rules state that if a player has card with the same rank or suit of the first card on *T0*, they can play it. Alternatively, they can draw a new card from the deck and end their turn. The last rule states that a player wins the game if they have no cards left (represented by the λ character).

Table 3.1: Representation of UNO

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

The language supplies a collection of instructions that allow the description of a high range of card games, but there are some details that are not taken in consideration. The concept of teams is present in some trick-taking card games, such as *Sueca*, which is crucial for building games that fit into this category. Teams are composed of two players or more players and have their own score, which corresponds to the sum of the scores of their players. Despite the absence of this notion, one could argue that a team can be represented by a regular player. Nevertheless, it does not feel natural to associate other players, real players, to a fictitious player that would represent their aggregate. Another concept that is common to some games and it is not present in the language is the trump. Given the list of existing rules and preconditions, there is no way to specify that kind of information.

The absence of some directives that help the construction of common card games proves that, even though the CGDL is capable of representing existing card games and generating new ones, the language is somewhat limited. Trick-taking games are just an example of games that would need a further analysis

[1] http://www.letsplayuno.com

in order to supported them.

Web-based Application

Although description languages are able to describe an application of a certain domain, they tend to miss some essential characteristics at the cost of being too general. Furthermore, these languages generally are not as simple or straightforward for a person without a programming background to understand and use. Therefore, Law and Deane [8] decided to build a multi-player card game application giving people with no programming experience the opportunity of creating their own card games. The card games are created by inputting the game rules in the form of a flow chart. Their application generates the game based on its rules and is capable of reasoning about the best strategies and tactics to play the game.

This application is web-based and focuses only on a subset of card games - trick-based card games. Supporting all possible games would be possible but harder to accomplish, since generating games from game rules was not the only goal of the project. Unlike CGDL, which describes the different stages of a card games, the web application uses game rules only to check if the players' moves are legal and how they influence the game state. Another difference between the two projects is the language they use to describe, or represent, the game rules. The latter represents the game in the form of a flow chart. Flow charts facilitate the input and gives the user a better perception of how the game is structured. They are built using three simple elements:

- **terminal**, which marks the start and the end of the game;
- **statement**, which indicates some action should take place;
- **choice**, which is used as an *if* statement, choosing the next path depending on the truthfulness of the condition.

Users construct the flow chart, which is translated to a description language that the system can understand. This language is needed in order to have access to the information in a form that is easier to manipulate when reasoning about the game itself.

Unlike the CGDL, the authors of this web application only commit to a specific subset of card games. The way they are able to structure the game in the form of a flow chart is interesting and suggests that it may be possible to apply this same system to other games, in order to control the game state and their rules, being validating moves or performing some action. Furthermore, it should be possible to reuse existing statement or choice elements in different games that share some of the rules or behaviors.

Table 3.2: Comparison of card game description approaches

Project	Game Representation	Supported Categories	Key Concepts
CGDL	Textual	Many, but not trick-taking	Description language; Details the game stages; Mean towards automatic generation of card games.
Web-application	Graphical	Trick-taking	Based on game rules; Flow chart representation; Validates plays.

3.2 Enhancing the Experience

Extending the gaming experience is the purpose of pervasive games. There are a number of different ways to turn a game into a more interesting and enjoyable experience. Multi-touch tables, sensors and wireless communication are the main building blocks for a pervasive game [9].

The present section focuses on the various uses of touch interfaces, as well as tangible objects, to improve several applications. These include a couple of projects that target card games, and others that take advantage of tangible technology for different purposes, such as sound creation and teaching mathematics.

3.2.1 Tabletop Interaction

Board games are known examples of games that gather people together around a table. The fact that these games revolve around a single table makes them the perfect case to apply in a multi-touch table. Designing games for multi-touch tables is different from designing a video game that people will play on their computer. It requires a much more thoughtful process because many of them will allow multiple players to interact at the same time with the display. These interactions may differ from game to game depending on their intent. Some may require people to only interact with their hands, and some may have to recognize objects as well. The design process has to take into account several aspects such as the recognition of graphical objects, the concept of individual and shared spaces, and public and private areas [10]. Therefore, there are already some platforms [2] that aim to make possible the creation of an augmented version of traditional board games. The combination of hardware and software that the platform is built upon, provides the potential to refine the overall experience of the game.

Card games, on the other hand, are still subject of active research. Patel and colleagues [11] developed a multi-player card game on the DiamondTouch system. The system was built alongside with a Software Development Kit (SDK) so that anyone can develop their own card games. It is composed of three layers: input, game module, and display layers. The first is responsible for managing the users' gestures like tapping, double tapping, or dragging. The second layer defines the representation of the game objects and the different game areas. There is a particular class, *i.e.* Game class, whose main functions are dealing with the game's rules, managing the players, and keeping the game state updated. Lastly, every action needs to be replicated on the screen and the display layer assures that that happens correctly. In order to test their system, they built a version of Blackjack. The most noticeable problem of this system resides on the requirement of hand gestures, losing the physical affordances of real playing cards.

Another example of a card game built on top of a multi-touch surface is the BriscolaTable. BriscolaTable [12] is a tabletop system that permits a single user to play the Italian trick-taking card game Briscola[2] against a simple conversational agent. The system uses a small capacitative screen which demonstrates an admirable dragging performance. Therefore, in order to play the game, the users just have to perform drag-and-drop actions, with one finger, on the cards that appear on the display. During

[2]http://en.wikipedia.org/wiki/Briscola

Figure 3.1: DiamondTouch table

the game, the conversational agent, named Alice, comments the player's moves and expresses some reactions which are triggered by certain game events. The motivation for developing the system was to acknowledge if this kind of system can be helpful for an elderly person to maintain their cognitive capacities when they are restricted to their home. Therefore, the BriscolaTable was placed on a senior citizens' center so that visitors could experiment with it (see Figure 3.2). Overall, the conversational agent was very well received, being particularly entertaining, and the system was easy to use. Nevertheless, the elderly had difficulties while performing the drag-and-drop actions. With age, people start losing their full senses, including the sense of touch. Thus, they cannot control very accurately how they press the tabletop surface. For this reason, there should exist an alternative mechanism to help people perform some tasks. On the other hand, there are certain occasions where users would like to execute the tasks themselves. For instance, gather the cards after winning a round.

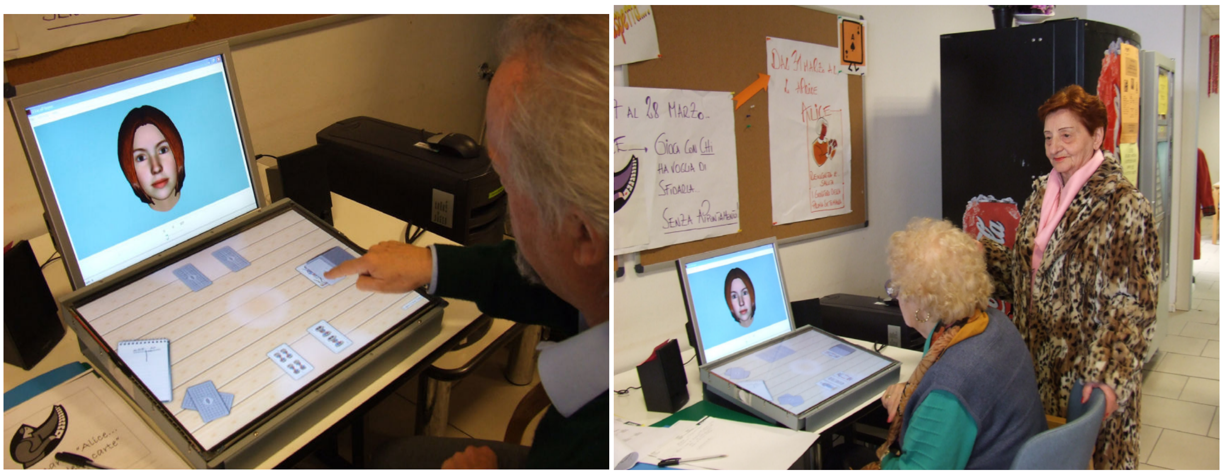


Figure 3.2: Elderly interacting with the BriscolaTable

Even though digital card games are a practical way to enrich the experience of their traditional counterparts, some important aspects of the original games are lost in the process. Adding to the clear problems hand manipulation of digital playing cards originate to the users, the perception of space is often left behind. People are used to throw cards randomly, placing them on top of the others, and these behaviors are really challenging to simulate on a digital game. Interacting with tangible objects, instead of their digital representations, on a multi-touch table, can solve these problems. However, they require extra hardware and software development. Hence, more issues may emerge, which demands the formulation of additional solutions.

Before exploring tangible systems and their approaches to enhance applications, Table 3.3 reviews the characteristics and key concepts of the tabletop systems previously discussed.

Table 3.3: Review of the discussed tabletop systems

Project	Table System	Target Audience	Type of Application	Key Concepts
Multi-player card game	DiamondTouch	Everyone	Card game	SDK to develop card games; Game elements are manipulated with hand gestures
BriscolaTable	Elo TouchSystems	Elderly	Card game	Play Briscola against a conversational agent; Maintain cognitive capacities; Perform drag-and-drop gestures to manipulate the cards.

*

Multi-touch tables allow developers to build games and applications that elevate the level of interaction and cooperation, considering that people share the same screen and can interact personally with each other. Based on the knowledge derived from video games, which run and are played in a regular computer, people started to port the same concepts to multi-touch surfaces. But, while some metaphors work on one type of system, that does not mean they will continue to work for other situations. Usually, video games expect that players know how to use keyboards, mouses, or game controllers. However, on a multi-touch tabletop, that is no longer a requirement (although they can still be used) because users can use their own hands to manipulate the game objects. At first, using *"the best pointing device in the world"* [3] seems a very good idea since it enables players, with no previous experience on maneuvering peripheral devices, to efficiently get some work done. Conversely, gestures and features alike create a barrier for people with diminished motor skills. The slow movements and loss of sensibility are essential factors to have in mind while developing multi-touch applications. Nonetheless, many fail to take this into consideration.

Every card game has one thing in common, and that is the deck of cards. Some of them may also have additional items, called tokens. These are objects that everyone can maneuver easily because people have been dealing with them for their whole life. This fact leads to a possible solution for the problem of using hand gestures: tangibles. Tangible objects provide a more familiar way of controlling the application and also a more enjoyable experience [1]. There are several ways these objects can be detected or tracked. Physical objects, like playing cards, can either be tracked by optical cameras or detected by Radio-frequency Identification (RFID) tags. Some projects, discussed below, use different techniques concerning the tracking of objects on multi-touch tables, as well as RFID technology. A summary of these projects is shown in Table 3.4.

Before starting to describe and discuss the relevant research on tangible interaction, it is important to mention some toolkits. Developers use the following toolkits to build applications that require the tracking of objects:

[3]Steve Jobs, on the announcement of the iPhone (2007)

ARToolKit

A library for developing Augmented Reality applications [13]. It uses various computer vision algorithms to track the position and orientation of physical markers. Moreover, it can also know in which direction a user is looking at so it can render the imagery accordingly.

Chilitags

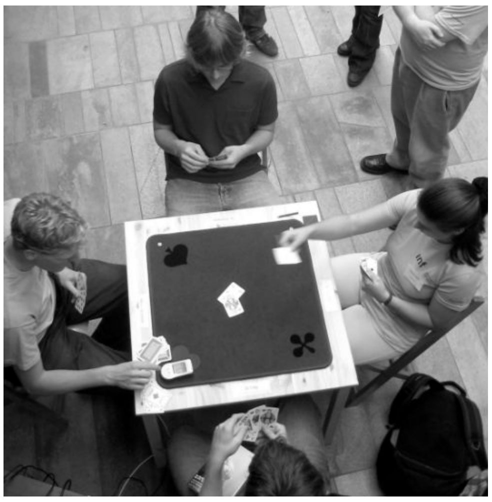
A library to detect and identify 2D fiducial markers [14]. Its two main features are the detection of tags and estimation of their orientation and position in a 3D space. These take advantage of various utilities to deal with imperfect detection and smooth the position of a given tag.

reacTIVision

A framework specially designed for building table-based tangible user interfaces [15]. It uses a computer vision algorithm to track fiducial markers with a specific design to improve their recognition. The framework uses its own communication protocol, TUIO [16], which was specifically designed to serve as an abstraction layer for the description of tangible objects. Together, reacTIVision and TUIO have been used to prototype and implement a vast number of projects that require tracking tangible objects.

3.2.2 Tangible Objects on Card Games

Regarding card games, research is at an early stage. Researchers are still trying to find the best gaming experience when using tangible objects and/or multi-touch surfaces. Both Floerkemeier and Mattern [17], Römer and Domnitcheva [18] use RFID tags to augment the game experience and provide relevant information to the users during the game, such as the exact score at the end of a round. But Floerkemeier and Mattern want to extend the experience so that novice players can easily play with more experienced players without the concern of not knowing the game rules or anything specific to the game. Therefore, one of their goals is to make it equally entertaining to both novices and experts. Card games are mostly very fast, with cards flying from different directions. Because of this, the detection of cards must be very accurate and nearly instantaneous. The author believes that vision systems and other systems, that require physical contact to some surface, make the overlapping of cards a very difficult problem to solve. That said, RFID tags were the most viable choice. The five antennas placed under the game table, connected to a single reader, make the detection of cards straightforward as well as the ability of knowing the owner of a card based on its position (see Figure 3.3). The state of the game is determined by a computer connected to the reader. This computer interprets the data received by the RFID reader, and informs the players about the current score and any wrong moves, for example. But, for players to have access to this information, they need to use a mobile phone that communicates with the main computer.



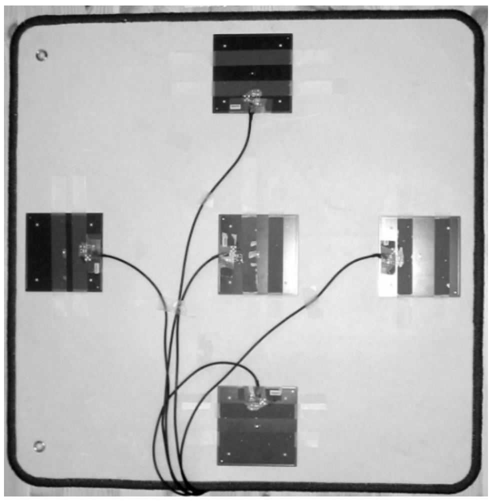


Figure 3.3: Floerkemeier & Mattern version of Smart Playing Cards

Likewise, Römer and Domnitcheva also use RFID technology to extend the regular card game. The two systems are very similar since, besides the RFID reader, this system uses a computer and a set of Personal Digital Assistants (PDA), but not in an identical way. The game state is computed by the computer, which has an external screen attached in order to show the relevant information shared by all the players. The PDAs, however, have a different purpose. Since each player has his own device, this can be used to present private information that the other players should not have access to. Besides all this, it also behaves as an authentication device. Before the start of the game, every player has to identify themselves. After the authentication step, cards are dealt on the table, four at a time. A player must take their card before the next round of cards can be placed on the table. Dealing the cards in different rounds gives the computer the opportunity to detect each individual card, associating it to the corresponding player. Thus, the computer knows all the cards that the players have in their possession.

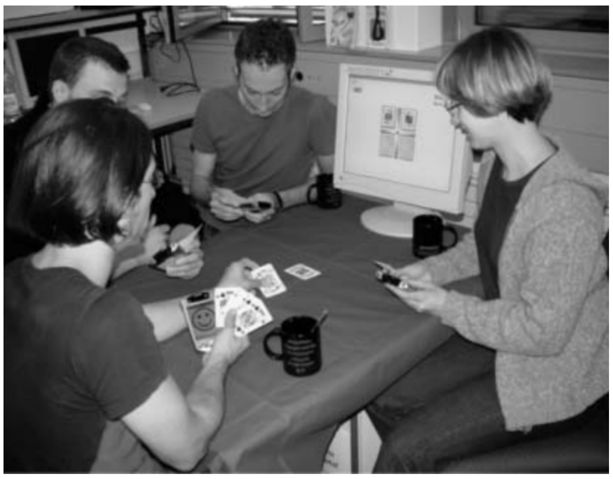


Figure 3.4: Römer & Domnitcheva version of the game, using PDAs

During the game, the player is presented with private information on his PDA. The information revolves around the quality of the current move, showing happy, sad, and indifferent smiley faces accordingly (see Figure 3.4). The feedback is provided by a decision tree consisting of filters and branches. Filters take the player's hand and modify it in some way, whether branches evaluate some condition based on the same set of cards.

The problem of card overlapping was the major motivation for attaching RFID tags to each card. RFID tags have a resonance frequency that the reader uses to detect them. However, when cards overlap, the resonance frequency of the tags suffers some changes that prevents the reader to recognize them correctly. To address this problem, tags have to have a greater resonance frequency so that, when overlapped, they still have a frequency value close to the operating frequency of the reader. Nonetheless, while running tests with users, the problem persisted when the users stacked the cards too quickly.

The usage of mobile devices to present shared or private information to the players can be helpful in some situations but distracting in others. Moreover, the interaction with the hand-held breaks the natural flow of the game. Another characteristic of these systems is that they require players to behave in a very strict way. Since people have their own way of playing, having to take the cards individually, from the table, goes against what they are used to.

3.2.3 Tangible Objects on Other Contexts

An alternative to RFID tags, for object detection, is fiducial markers. Fiducial markers are special tags that are recognized by an imaging system and are usually added to an object, to know its orientation and location. Figure 3.5 shows some examples of fiducial markers recognized by distinct imaging systems.

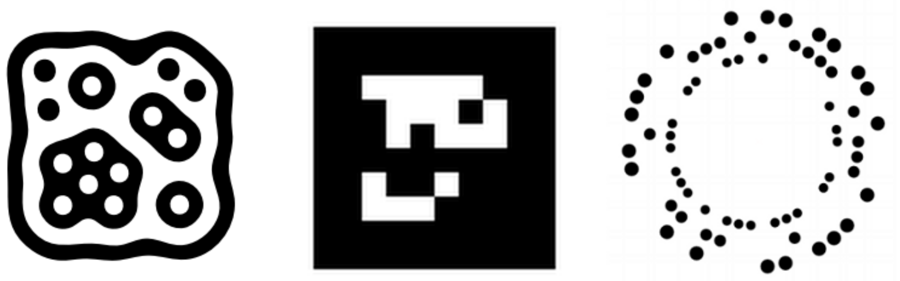


Figure 3.5: Examples of fiducial markers

Attaching a fiducial tag to an object gives the ability to use it to interact directly with the multi-touch surface. The fact that fiducial markers enable the usage of objects, on these surfaces, allows the development of much richer and complex applications. Thus, applications exploit fiducial markers for many different purposes.

Combinatorix [19] takes advantage of fiducial markers to build a learning environment so that students can collaboratively learn probabilities. This project aims to help students having a better understanding of some abstract concepts inherent of mathematical problems. Using and manipulating physical objects, students have the freedom to experiment and observe how their changes affect the final result. Although they manipulate the object on the surface, the corresponding abstract representation is produced in another screen (see Figure 3.6). The system uses the reacTIVision engine for the detection of fiducial markers. These are attached to placeholder objects, which can have letters, in order for the student to be able to form new combinations.

In the same way tangible objects proved to be very useful when building learning environments on multi-touch tables, they are also being used on the music industry. The reacTable system [20] lets musicians share the control of a music instrument simply by manipulating physical entities on a sensitive surface. Each object serves as a modular synthesizer with the function of generating and modifying sound. The state of each object is represented as an aura, providing information about its behavior, parameter values, and configuration. Furthermore, to distinguish objects with distinct behaviors, they have different shapes. Despite having different shapes, some of them can interact with others. For instance, a sound generator, controlled by a step-sequencer, does not produce the same sound as if it was producing the sound by itself. This is just an example of the capabilities of connecting different entities. The system uses the multi-touch surface not only for the detection of objects, but also to provide a visual feedback to the end-users (see Figure 3.7). Changes of sound frequency and volume result in an updated sound wave that is immediately represented on the surface. As Combinatorix, the reacTable uses the reacTIVison engine for tracking the fiducial markers. Overall, the reacTable system proves that a proof-of-concept can provide a rich, complex, and powerful experience.

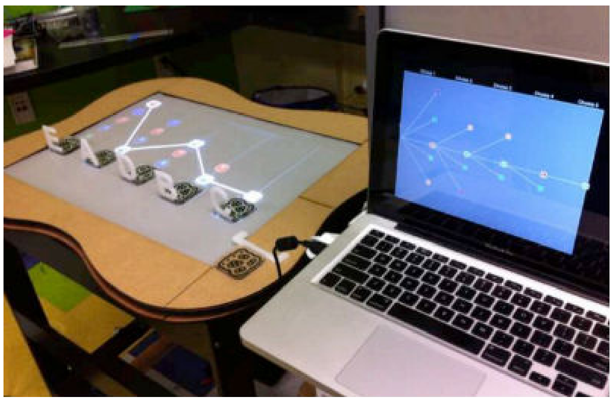


Figure 3.6: Combinatorix setup

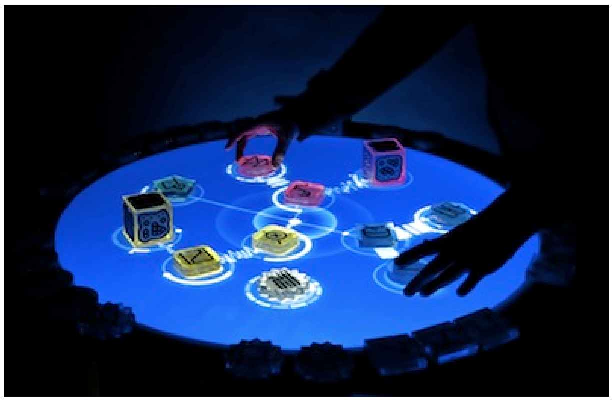


Figure 3.7: reacTable in action

Research shows that tangible objects and multi-touch surfaces can be used to develop learning and entertaining experiences. Furthermore, these systems proved to be well received and capable of delivering great experiences to their users. This success leads other fields of study to follow their steps and innovate. The fun.tast.tisch. project [21] determined that an interactive multi-user multi-touch application could be beneficial to patients on neuro-rehabilitation. Cognitive impairment reduces the person's performance for executing most mundane activities, such as eating and dressing. Hence, patients with this type of impairment benefit from the possibility of using physical objects on interactive tabletop systems. The team behind this project aims to develop different modules that can be applied, depending on the patient's condition. The first module created consists of a Tangram puzzle, which is described as an effective and recommended training for neuro-psychology therapy (see Figure 3.8). The main idea was for the patient to mirror a certain Tangram figure on the multi-touch surface, which provided feedback. The feedback consists on filling the shape of the Tangram's *tan*, on the table, for the patient to know if the *tan* is correctly positioned. However, the feedback provided can only be seen if the *tan* is translucent, which means that attaching a fiducial marker to it is not reasonable (see Figure 3.9). Therefore, the developers had to find an alternative approach to recognize the Tangram tiles. They developed an algorithm that extracts the outlines of the shapes from a monochrome image, which are then compared to the provided Tangram templates. The resulting polygon is described in terms of the aspect ratio of its edges and its angles. If both properties are within certain intervals, the Tangram shape is considered recognized.

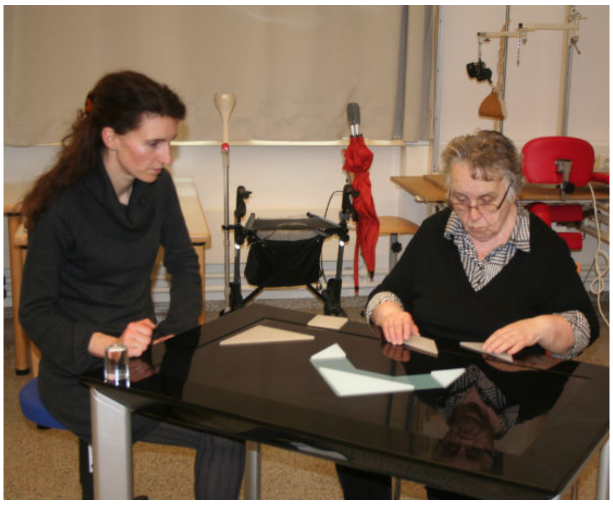


Figure 3.8: Patient using the application

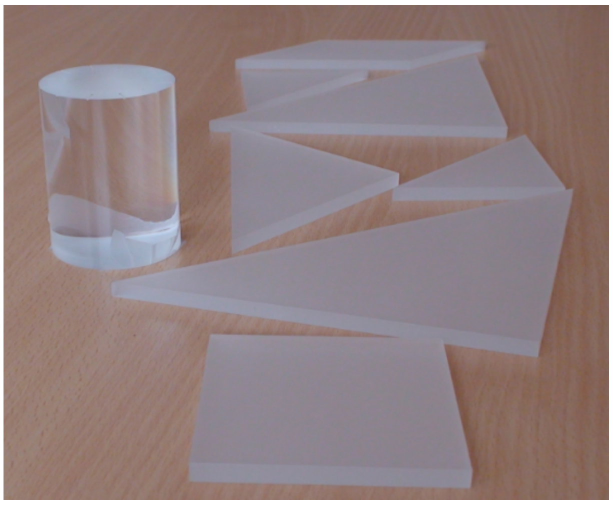


Figure 3.9: Translucent Tangram tiles

Regarding playing cards, tangible technology like RFID and fiducial markers is, without doubts, the best approach. Trying to detect cards without this kind of tags seems almost impossible. Even though the developed algorithm on the fun.tast.isch project is able to recognize Tangram shapes, it would be extremely difficult to do the same for playing cards. The reason is that Tangram tiles have distinct shapes and sizes while cards have all the same dimensions and form.

*

Albeit multi-touch tables and tangible technology seem to fit the necessities for building enhanced versions of traditional games, there are some factors that need to be addressed like occlusion or stacking of cards, and private information. Some of the projects previously discussed add their own solutions to solve these problems, such as using mobile devices to protect private information from other players. Yet, the authors of TaPS Widgets [22] claim that people do not like to use external devices because it makes the task harder to complete. Thus, their solution is to use a lightweight, transparent acrylic spacer that scatters the light the table emits, depending on the viewing angle.

Stacking, or occlusion, is a much serious problem. Card game players often like to throw their cards to the table, meaning there is no way of knowing if they will end up standing on the surface or on top of other cards. RFID tags are much closer to a solution than fiducial markers because, once a marker is hidden, the tracking system cannot see it. Bartindale and Harrison [23] developed a proof-of-concept that aims to resolve the physical order using fiducial markers with transparency. Markers are made of reflective and non-reflective areas and, in order to solve the problem of occlusion, transparency is introduced to them. Having these three distinct areas, stacked markers form an all new marker, a composed marker, that represents those stacking objects. It is exactly as if an unique ID were assigned to the combination of the two objects. However, the objects need to be perfectly aligned for the system to detect them. Although this may be useful for some applications, it is extremely improbable that playing cards align perfectly on top of the others.

Table 3.4: Summary of projects relying on tangible interaction

Project	Tangible Technology	Toolkit	Hardware	Key Concepts
Combinatorix	Fiducial markers	reacTIVision	Multi-touch table Placeholder objects Additional screen	Collaboratively learn probabilities; Abstract mathematical problems.

fun.tast.isch.	Shape detection	-	Multi-touch table
			Translucent Tangram tiles
				Applications for neuro-rehabilitation patients;
				Recognize Tangram shapes;
				Extract outline of Tangram tiles.

reacTable	Fiducial markers	reacTIVision	Multi-touch table Placeholder synthesizers	Control sounds manipulating objects; Compose objects to produce different sounds.
Smart Playing Cards (1)	RFID	-	RFID antennas RFID reader Computer Mobile phones	Novice players do not need to know the rules; Use mobile phones to display relevant information.
Smart Playing Cards (2)	RFID	-	RFID antennas RFID reader Computer PDAs	The computer screen shows shared information; PDAs displays private information and quality of the play; PDAs also function as authentication devices.

3.3 Extending the Experiences to Artificial Players: Agents and Robots

Robots and agents are becoming increasingly more social. Tutors or companions, they are being used to interact with people in a natural manner. These interactions happen based on the their perceptions of the world. Unlike humans, robots and agents do not have senses to help them having a perception of what surrounds them. Therefore, they require additional hardware, components, to overcome these limitations, such as cameras, microphones, motion sensors, or even touch surfaces.

In BriscolaTable, a conversational virtual agent named Alice was employed as an opponent for a computerized version of the Briscola card game. Alice reacts to the moves of the other player similarly to how a human player would react. Additionally, Alice helps the player to focus on the game. This kind of interactions are possible due to the communication between the game and the agent. All the interactions between the agent and the player are spoken, which captures the player's attention and makes the communication more natural and appropriate. The feedback given by the touch screen, or lack of it, provides information to the agent that is then capable of acting accordingly.

Likewise, Pereira [24] integrated a robot into the Risk board game[4]. The goal was to create a robot that is capable of interacting with humans and being perceived as socially present in long-term inter-

[4]http://en.wikipedia.org/wiki/Risk_(game)

actions. Similarly to Alice, the robot plays as an opponent against three human players, which use a touch surface to interface with the game (see Figure 3.10). The value of face-to-face interactions and non-verbal behaviors in board games justifies the use of an embodied agent instead of a virtual one. The robot used was EMYS, the EMotive headY System, which is able to perform facial expressions and gaze at the surrounding environment. It uses a camera, which is embedded in his nose, and the touch table to take perceptions of the interactions history and the environment, to interact with both the game and the human players.

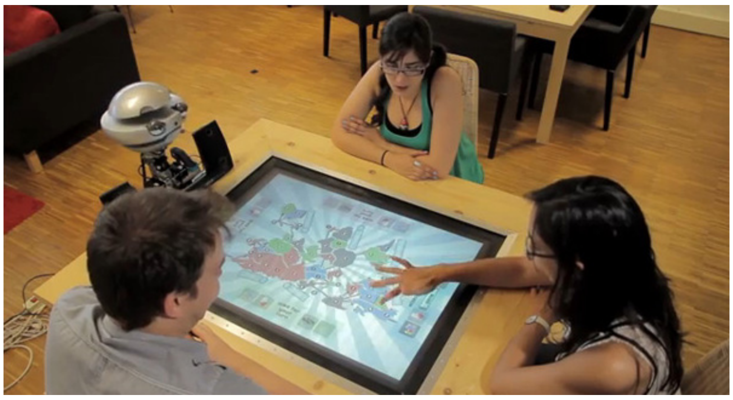


Figure 3.10: EMYS playing the Risk board game

Overall, all components used to build these interactive systems have to communicate with each other, in order to obtain the desired information. However, most hardware have their own interface to communicate with the exterior. Thus, there is the need to generalize the integration of these components so they can easily exchange information between them, and be replaced without interfering with the rest of the system, which should continue to work seamlessly.

Thalamus [25] is a component integration framework that supports the seamless integration of the interactive agent's logic with components for virtual and/or physical embodiments. The motivation behind the development of this framework was that an embodied agent, instead of having all the necessary physical interfaces for perception and actuation, can be composed of different components to interface with the environment. This trait of Thalamus provides great modularity, meaning that, for instance, in a system composed by different modules, such as a robot, a Kinect$^{©}$ camera, and a touch surface, one can easily replace a robot for another while maintaining the system's functionality unbroken.

The framework centralizes all communications between modules, providing a communication layer. The communication happens through a publish-subscribe pattern, so each Thalamus module can therefore publish, subscribe, announce, and receive messages.

Another tool being developed is Skene [26], a semi-autonomous behavior planner. The different components that compose a system, *i.e.* robot, camera, and a touch surface, meet in Skene. It receives high level intention-directed instructions and perception information as input, and it outputs expressive commands (*e.g.* Speak, PlayAnimation, Gaze) to be executed by a particular component that knows how to accomplish each action. Tools such as Thalamus and Skene made possible the integration of these components and are used in several projects, such as LIREC [5] and EMOTE[6].

The LIREC project was a research project that explored how digital and interactive companions can be designed in order to establish a long-term relationship with humans. During this project, Pereira and other researchers developed the socially interactive scenario in which EMYS plays the Risk board game along with three human players.

On the other hand, instead of designing interactive companions, the EMOTE project aims to develop robotic tutors that are able to interact with children on diverse scenarios. An example of such scenarios is the Enercities game[7]. Similarly to the Risk board game, Enercities is played by two human players and a robot. The role of the robot is to guide and collaborate with the other players to build a virtual city that is aware of environmental issues.

In short, both projects demonstrate how applications that use touch surfaces can communicate and exchange information with robots in order for them to be part of experience and actively interact with other players.

[5]http://www.lirec.eu
[6]http://www.emote-project.eu
[7]http://www.enercities.eu/

Chapter 4

User-Centered Study

Most elderly people like playing games. Their primary motivation to play games is to have fun and maintain their social network [1]. Card games, for instance, revolve around tables, which allow the elderly to interact with other people and, therefore, widen their connections.

Solving the inherent problems of card games, such as occlusion, requires a better understanding of the different interactions that occur during a game, and the way people play the game. Taking all this into consideration, a user-centered study, in a care home, was arranged. The purpose of this study was to analyze the players' behaviors, and collect information in order to build a game experience that meets their expectations.

4.1 Methodology

The chosen card game was *Sueca*, mainly because it is very commonly known and played game among the elderly. *Sueca* requires the presence of four players. Additionally, a deck of cards, a table and four chairs, two video cameras, and an audio recorder with four microphones were required as part of the setup.

4.2 Procedure

The setup components were arranged to capture the players' behaviors and interactions during the game. The video cameras were strategically positioned to have a clear view of the players' hands and game area, and the microphones placed on the players' clothes to record any reactions. The players were filmed playing a series of games during a one-hour period.

4.3 Preliminary Results

The analysis of the footage focused on the players' behaviors, including the way they throw the cards to the gaming area and manipulate the cards in general.

In *Sueca*, the four players are assigned to two different teams. Each team is composed of two players, which sit in front of each other. Pairing players in teams has a great impact in the way cards are spread across the gaming area. Throwing cards to the gaming area produces situations of occlusion, with cards ending up on top of each other. However, in order to have a clear perspective on who played a given card, players often rearrange the cards on the center of the table. Although this study was made with only one group of four people, some conclusions can be extracted from this observations. The fact that players rearrange the cards after a play indicates that occlusion may not be a great concern.

Chapter 5

The Framework

The current chapter gives a detailed description of how the proposed framework was developed. Additionally, it focuses on the problems that occurred during the development and the decisions made to overcome those same problems.

The chapter is divided into four sections. First, a brief overview of the developed solution's structure (Section 5.1). Next, the implementation details of the building blocks that constitute the framework (Section 5.2) and how the usage of physical cards was made possible (Section 5.3). And last, but not least, a section that describes how games, built with this framework, can connect with agents that are able to play and interact with them (Section 5.4).

5.1 Overview

Going back to the goals of this work, this framework should be able to represent the fundamental components for the creation of different card games for multi-touch tables. It should also enforce the usage of physical cards to play these games.

The game applications follow the structure represented in Figure 5.1. For these applications, it is important to separate the physical parts, or layers, from the virtual ones.

5.1.1 Physical layer

The physical layer includes all hardware that is capable of providing input to the games. This consists of touch tables, physical objects, and other peripherals that can capture information from the environment and translate it to bits of information that prove to be useful for the developed games, which may include cameras, microphones, or robotic figures. Although many peripherals can be used in a game application, the ones that are strictly required for this project are the physical cards and a touch table (the physical cards will be discussed in more detail in Section 5.3). The touch display used to deploy these games was the MultiTaction Cell 55” Full HD LCD Ultra Thin Bezel Display[1]. According to the available information

[1] http://www.multitaction.com/products/displays/ultra-thin-bezel/

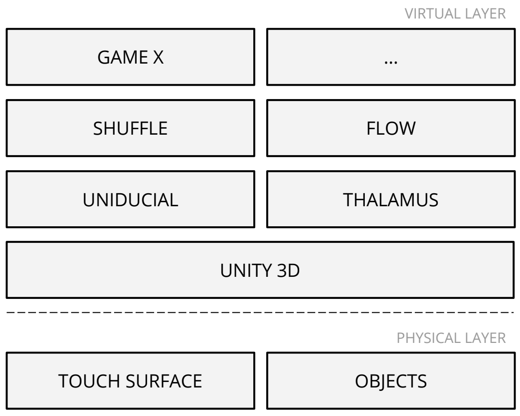


Figure 5.1: Architecture of a game application

on the manufacturer's website, this display is able to detect an unlimited number of fingers and hands, and also specially crafted fiducial markers.

5.1.2 Virtual layer

The virtual layer aggregates all the software and tools to build the games and deals with the events generated by the physical layer and its components. In other words, it represents the game engine that is responsible for running the games. There are already some game engines available that could be extended in order to build this framework. Nevertheless, due to compatibility issues of some required libraries, the number of possibilities decreases. Therefore, the Unity3D[2] game engine was chosen both because of the compatibility problems already mentioned and because it facilitates the design and development of the games. This game engine was extended with libraries and frameworks, detailed in the next sections, in order to provide the needed functionality to build the games. All the components of this layer are programmed using a single programming language, C#, keeping an uniform code base and taking advantage of some of the .NET framework capabilities.

5.2 Building blocks

Most card games have some characteristics in common. On the one hand, they can use the same type of cards, players arranged by teams, or even use tokens. On the other hand, these games can share other properties such as rules, *e.g.* follow the lead suit. The definition of such entities or directives, and the abstraction of some behaviors that can be reused, are the building blocks of this framework. The implementation of these concepts was divided into two separate libraries: Shuffle and Flow. The first defines the properties of common elements that can be found in many card games, while the second focuses only on managing and structuring behaviors.

[2]http://unity3d.com/

5.2.1 Shuffle

As previously mentioned, all card games share some elements with each other. All card games are played using cards and it is not possible to play a game without players. Like players and cards, there may exist other properties that are required in order to represent and play a given card game. Therefore, a representation of these common elements, and also some specific ones, must exist to construct a certain card game.

Starting with the card game itself, there are some abstract concepts that have to be defined. Furthermore, these concepts need to be generic enough to give developers the appropriate tools to build their game states.

The following primitives are the base primitives for every card game:

- Card
- Card Game
- Player
- Team
- Token

Every primitive is implemented as an interface. The reason for using interfaces instead of abstract classes is that interfaces can be easily composed, which means the developer has more freedom to construct more specific primitives composing existing interfaces.

The core concepts of card games are cards, players, and the card game itself. The `ICard` interface represents the most common and simple version of a playing card, which stores only their `Type`, `Symbol` and `Value`. More specific playing card representations must implement this interface. Similarly, the most basic representation of a player consists of some kind of identification, such as a `Name`, and a list of cards denominated `Hand`. These properties are imposed by the `IPlayer` interface. Finally, a card game is described by the `ICardGame` interface. This interface exposes two properties: a `Deck` and `Players`, which are a list of `ICard` and a list of `IPlayer` objects, respectively.

With these concepts defined, more complex and some secondary primitives can be represented. Examples of such primitives are teams, tokens, and plays. The notion of a team takes advantage of the already defined concept of player with the `ITeam` interface. Every team has a `Name` that identifies itself, and a list of `IPlayer` objects, denominated `Players`, that keeps a record of the players that belong to it. Tokens are the most simple primitive in this library and are implemented by the `IToken` interface. Tokens are used to represent some value, often to emulate the real value of money, so they only have a single property called `Value`. The representation of a play, on the other hand, is more abstract than the other primitives. A `Play` is a data structure that stores information about a play made by a player. The difference between other primitives is that there is no way to say what a play can or cannot be. It may be a card played by a player, or a certain amount of tokens a player is betting at some point of the game. Therefore, a play stores information about the player that is doing it and also its `Contents`, which can be any object that implements the `IShuffleObject` interface.

Breaking functionality and properties down to separate interfaces enables composition and therefore the ability to build more specific primitives. Having interfaces with the sole purpose of providing a single property or behavior, such as `IScorable` and `ITeamable`, gives developers more freedom to create their own primitives when needed. An example of a primitive that takes advantage of the composition of interfaces is the `ITeamPlayer` interface, which shares the same properties of a `IPlayer` but also has information about the team they belong to by implementing the `ITeamable` interface. This interface exposes the `Team` property. Likewise, the `IScorable` interface can be implemented by any primitive that needs to provide a `Score` property. Nevertheless, there are some situations where this composition is not needed or does not provide any advantage. That is the case for a more specific type of card game such as trick-taking. In order to represent a generic trick-taking card game, this library also provides an `ITrickCardGame` interface. This interface extends the regular card game by adding more properties such as the concepts of `Trick`, which is a list of plays, `Trump`, and `LeadTrump`. Since the support of physical objects is an important feature of this framework, the library also provides the `IFiducialObject` interface that exposes the properties needed to identify an object with fiducial markers.

In short, the Shuffle library promotes the composition of interfaces to build more complex and more specific primitives the developers find useful to build the state of their card games. This composition is not mandatory but having reusable pieces of code leads to a more structured and maintainable way of defining these primitives.

5.2.2 Flow

In the same way Shuffle tries to break functionality and properties to their own interfaces, and promote the composition of those interfaces in order to build more specific and perhaps complex representations, breaking behavior down to their composing units can also be advantageous, in the sense that every unit can be composed to form a different behavior. Furthermore, having a system that simplifies the connection between these reusable units could ease the development of more complex behaviors. That is the purpose of the Flow library.

Flow provides the tools for developers to visually create the desired behaviors. These behaviors can then be used anywhere in the code. The difference between this library and the projects already discussed is that it is not limited to the card games domain. Moreover, Flow can be used for the validation of a certain play, but it can also be used to control the state of the program and act accordingly. To accomplish these goals, a graph structure is used to represent the behavior. This graph structure can be composed of three different types of nodes: the `Action`, `Predicate`, and `Flow` nodes (see Figure 5.2).

The `Action` nodes execute a single action, changing the current state of the application and continuing to the next node. The `Predicate` nodes have a predicate attached, which returns true or false depending on the current state of the game or application. Unlike the `Action` nodes, that pass the execution to a single node, they are often connected to two nodes which are chosen depending on the truthfulness of the predicate. Finally, `Flow` nodes have another flow (graph) attached. More complex behaviors often require a large quantity of nodes and, as the number of nodes grows, laying them out becomes a difficult task. For that reason, the flow nodes offer a way to encapsulate another graph into

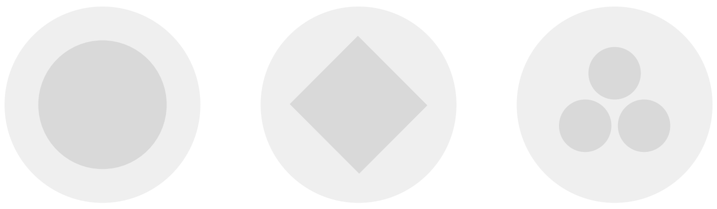


Figure 5.2: Flow nodes (action, predicate, and flow)

a single node which can be connected to any other node (see Figure 5.3).

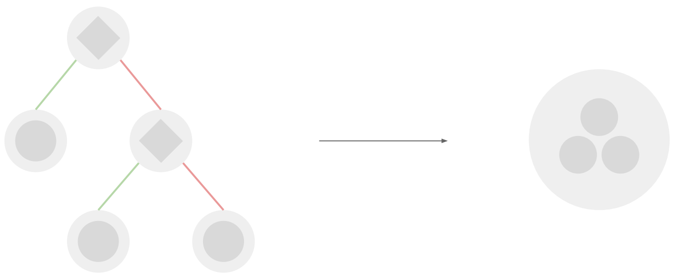


Figure 5.3: A graph represented as a flow node

Every node has a `Validate()` method that is called to execute the desired behavior. In order for developers to add new behaviors, they have to extend the `Action`, `Predicate`, or `Flow` classes and implement this method.

The connection between the nodes could easily be accomplished in code. However, as the complexity of the graph grows along with the number of nodes, having a location where the developer can insert and remove nodes, drag them, and connect them with each other, provides a simpler interface for interacting with the created graphs. Hence the creation of a simple graph editor, illustrated by Figure 5.4. The editor eases the creation of graphs by allowing the manipulation of nodes. It also has the ability to save the graph to disk, as well as load already existing graphs to make changes.

Overall, the Flow library provides a simple way to transform a single but complex behavior into a sequence of steps in the form of nodes that can be visually connected to build a graph that represents the original behavior. These steps are independent and can be generic enough to be reused.

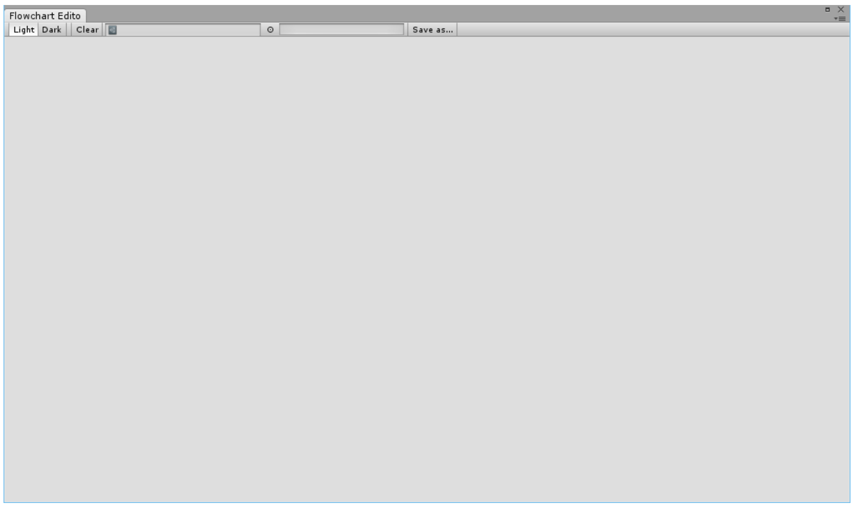


Figure 5.4: The Flow editor for Unity3D

5.3 Playing with physical cards

Besides having a set of tools to help the development of card games, the goal of this work is to explore the usage of physical objects during those games. As seen on a previous chapter, some people [17, 18] have already tried to use tangible technology in the context of card games. However, both projects use the same technology (RFID) and do not try to explore other available alternatives. Therefore, this work explores an alternative to RFID tags: fiducial markers. The touch display available for the development of this work already provides the technology to detect and recognize this type of markers. Despite the existence of many variations of fiducial markers, the touch display only recognizes a specific type of marker, crafted especially for this device (see Figure 5.5).

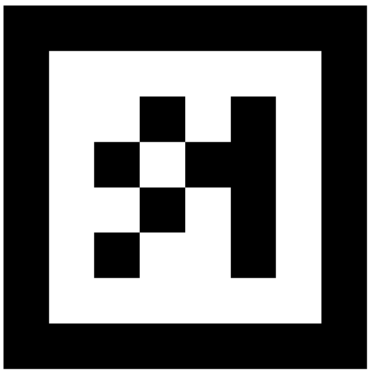


Figure 5.5: Example of a marker recognized by the MultiTaction display

The touch display is able to detect fiducial markers with a minimum and maximum sizes, which influences the amount of fiducial markers that can be produced as well. Using the minimum size (3$\times$3), only 32 unique markers can be generated, which is lower than the amount of cards present in a regular deck of playing cards. Alternatively, a 4$\times$4 marker can hold enough information so that 4096 unique markers can exist, which it is more than sufficient for the needs of this work since, for a deck of 52 cards, only 104 (52$\times$2) markers out of 4096 are needed (assuming only two distinct markers are used for each card).

The detection and recognition of the markers are possible using the TUIO protocol. This protocol is used for translating the received input, such as touch and marker events, to a more understandable and manageable format that can be processed to extract the desired information. The touch display runs a TUIO server that sends messages to every client that is connected. Therefore, games have to run an instance of a TUIO client in order to listen to the events sent from the display. Using the TUIO protocol, the games treats fiducial markers as instances of `TuioObject`. Every time a marker is detected, a `TuioObject` is created and added to a list. When a marker is no longer in contact with the screen, the object is removed. Accessing the contents of that list, developers can control which fiducial markers are currently on the touch surface. However, many card games have different areas in which players are allowed to place their cards or tokens. Given the way the TUIO client handles the markers' detection, there is no way to tell the display to detect the markers only in some specific region of the screen. As a result, a single Unity3D component was developed to overcome this problem. This component can be attached to another User Interface (UI) component and only cares for markers that are contained inside the limits of the latter. Having this component gives developers the freedom to have many areas to detect the fiducial markers and deal with them separately.

The available hardware and software allow the detection and recognition of fiducial markers. Moreover, being able to limit the detection to certain areas is an advantage when building the game's interface. This also allows for a greater control of the behavior required for each area.

5.4 Connecting games and agents

Another purpose of this work is to let an agent, virtual or robotic, connect with the games that are developed using this framework. In addition, these agents should be able to play the games with the human players. Depending on the game, the agents would play against or in cooperation with other human players.

The Thalamus framework provides all the mechanisms to make a connection between the game and the agents. The game and the agents correspond to different modules that are able to exchange messages through the Thalamus framework. Thalamus centralizes all the communications between the modules, providing a communication layer based on a Message Oriented Middleware (MOM). The communications happen through a publish-subscribe pattern. Each module can publish, subscribe, announce, and receive messages, which are named *events*. Considering that the game and agent modules have to exchange the same type of events, they have to follow an interface, or a "contract". By subscribing to an interface, a module subscribes to every event that the "contract" defines.

In order to make the game application a module that can be integrated into a system like the one illustrated by Figure 5.6, the game must be an instance of the `ThalamusClient` class. Since it is not possible to have an instance of a `ThalamusClient` running on Unity3D, some workarounds had to be made. The solution was to build a bridge that exchanges information between the game and Thalamus (see Figure 5.7). When the game wants to send information to Thalamus, it sends that information to the bridge which redirects it to Thalamus. For the game to receive information from other modules, through

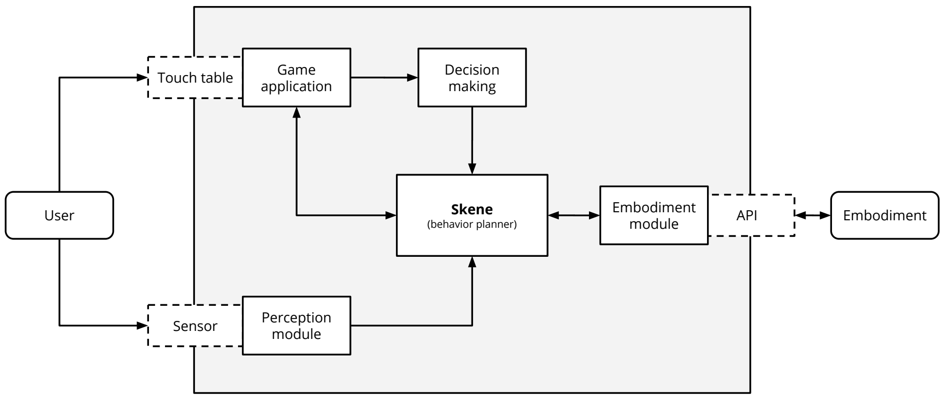


Figure 5.6: Example system that incorporates the game application

Thalamus, the opposite process happens: the bridge subscribes to the events and sends the information back to the game application. This communication between the Unity game and the bridge is done via XMLRPC.

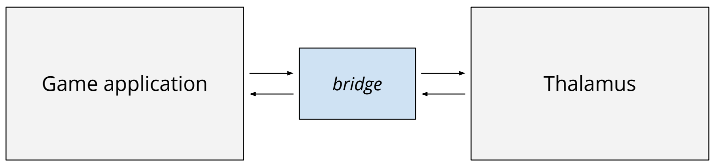


Figure 5.7: Interaction between the game application and the Thalamus framework

Every game is different from the others and therefore the messages defined for the interaction between the game and the agents may also be distinct. The number of events sent from the game to the agents defines the amount of interaction between the two. As the game publishes more information, the agent is more aware of what is happening during the game execution. This allows the agent to react and interact with the game and other players, making it more present. Nevertheless, there are some messages that can be relevant for every card game that may be developed using this framework:

- *SessionStart* and *SessionEnd*, to inform other modules that a game session is about to start or just ended;
- *GameStart* and *GameEnd*, to send relevant information about the game state at the beginning and at the end of the game;
- *NextPlayer*, to tell which player will play next;
- *Play*, which sends information about the plays happening during the game.

Providing the tools to communicate with other modules that are part of a bigger system, this framework also enables the communication between the card games and virtual or robotic agents. The framework does not have any restrictions on the number of human players or agents in a game. The developer has the liberty to decide how many human player or agents make up the game. This means that a game can be played only by human players or only by agents, but also by a mixture of human players and agents.

Chapter 6

Development of Cases

Frameworks are useful because they are capable of providing tools developers can use to better build their applications. These frameworks can offer abstractions that developers can use without having to worry about their implementation details, or just a set of primitives that allow them to have a more structured and maintainable code base.

In order to know if a framework does what is intended for, some applications must be built. The development of a game from scratch provides a testing environment to understand if the several components that form the framework can be used and are sufficient to build that card game. Therefore, the *Sueca* card game was built using the tools provided by the framework. Moreover, some of the framework's components were used in the *Coup* card game.

The following sections discuss the development process of the *Sueca* card game (Section 6.1), and also how the tools were integrated in another game like *Coup* (Section 6.2).

6.1 *Sueca*

Before starting to explain the process of building the *Sueca* card game, it is important to first provide a brief introduction to the game. The *Sueca* card game belongs to the family of trick-taking card games. The game is played by two teams, with two players each. The deck is composed of 40 cards, which are distributed to the players. Each player starts with 10 cards, one for each trick (there are a total of 10 tricks in the game). At the end, the team that scores more than 60 points wins the game. *Sueca* is a simple game, with a handful of rules. The first player of a trick decides what is the lead suit. From that point on, but only for the current trick, all players have to play cards with the same suit as long as they have one. Otherwise, they can play any other card. Assuming that a player does not play a card with the same suit in the current trick, but does so in a following one, the other players may notice and the game ends immediately. This is called a renounce and, when it happens, the player's team loses the game.

The development process of this game focused mainly on how the game state is built, how the validation of plays is made, how the usage of physical playing cards is done, and what information agents need to have a concrete knowledge of the game. Furthermore, the UI also plays an important part by connecting all these aspects to make a playable game.

6.1.1 State and rules

From the description previously given, it is possible to extract some of the concepts that are represented in the Shuffle library and that can help building the state of the *Sueca* game. Examples of these concepts are the notion of a trick-taking game, players and teams, and cards. In Shuffle, there is a `ITrickCardGame` interface that provides the skeleton for trick-taking card games (see Listing 6.1). Since *Sueca* belongs to this family of card games, this interface can also support this game. Therefore, the representation of the game implements this interface. As expected, every trick-taking game has its own properties and rules. For instance, not every trick-taking card games has the concept of renouncing and, for that reason, those properties must be added to the game representation. In order to keep track of the suits each player does not have, an `History` property was added. Another concept that is not present in the trick-taking card games interface is the concept of teams. *Sueca* is played by two teams and is important to keep a reference to them during the game, even if it is only to keep the scores updated. A sample of the code representation of the game is present in Listing 6.2.

```
public interface ITrickCardGame : ICardGame
{
    List<Trick> Tricks { get; }
    List<IPlayer> Players { get; }
    List<ICard> Deck { get; }
    // ...
}
```

Listing 6.1: `ITrickCardGame` interface

```
public class Sueca : Shuffle.ITrickCardGame
{
    Dictionary<Shuffle.Suit, bool> History { get; }
    List<ITeam> Teams { get; }
    // ...
}
```

Listing 6.2: Code representation of *Sueca*

Having the state of the game constructed, the next step was to determine how it could be handled in order to validate the plays and to be updated. To accomplish this, the Flow library was used. Flow helps creating graph-like structures to, in this case, control the game state. This type of structure is appropriate for the validation of the plays, in the sense that each node can access the current state of the game and trigger some behavior according to the current play. Figure 6.1 illustrates the graph made to validate the plays. The nodes that constitute this graph were made for the specific case of *Sueca* but some of them can also be used to build a graph for other card games. The *NextTrick* node is an example of a node that can be reused in other trick-taking games since its sole purpose is to save the current trick and prepare the new one (see Listing 6.3). On the other hand, the *IsPlayerCheating* is exclusive to this game, since it has to know about the existence of an `History` property. The validation of a play starts with the

IsPlayerCheating node. The node verifies if the player is renouncing or not. If so, the team of this player automatically loses the game, otherwise the validation continues to the *IsFirstPlay* node. Being the first play of the current trick implies that the suit of the card must be marked as the lead suit, following the path to the *LeadSuit* node. Alternatively, if it is not the first play then the play must follow the lead suit (*IsFollowingSuit* node). However, a player may not have a card with the lead suit on their hand meaning that the `History` property should be updated accordingly, which is done by the *MissingSuit* node.

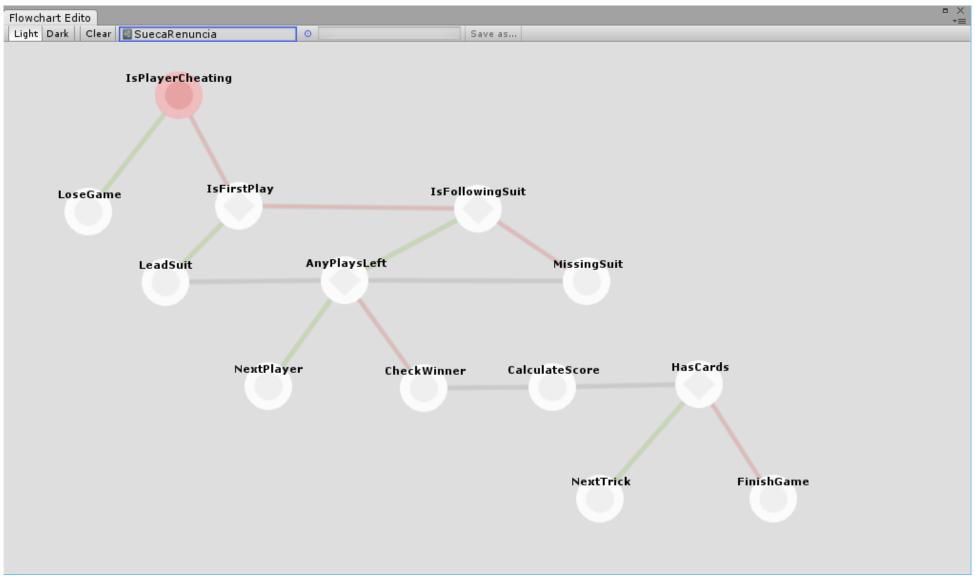


Figure 6.1: The flow graph for the *Sueca* card game

```
public class NextTrick : Flow.Action
{
    ITrickCardGame state;
    public override Validate()
    {
        // ...
        var trick = new Shuffle.Trick(plays, winner);
        state.Tricks.Add(trick);
        // ...
    }
}
```

Listing 6.3: *NextTrick* adds the current trick to the game's list of tricks

After validating the play, the next set of nodes manage the sequence of the game. For instance, the *AnyPlaysLeft* node inspects the game state to know if the current trick is complete. Once the trick is complete, its winner must be found, as well as its score value. On the other hand, if there are still more plays to play, the *NextPlayer* node decides which player will play next. Only after these steps are complete (by invoking the behaviors defined in the *CheckWinner* and *CalculateScore* nodes), the game can continue to the next trick or finish, depending on whether players have more cards to play or not.

6.1.2 Building the UI

The next natural step of the development process was to build the game from the state representation and rules. Building the game consisted in connecting the state and the behaviors with a simple UI that presents the players with relevant information and feedback of their actions.

A session of *Sueca* games always starts with players having to complete some actions with the deck of cards. These actions involve shuffling and cutting the deck of cards, and then dealing the cards to all the players. Nonetheless, these same actions cannot be executed in a random manner, as they need to follow some conventions. At the beginning of a session, any player can volunteer to shuffle. After the shuffling is done, the teammate is the one that breaks the deck in two. Next, the player on their right is the one dealing the cards. When the game finishes, these actions are repeated but starting with the player on the right of the player shuffled the deck at the start of the previous game. Due to the existence of such conventions, the game has to recreate this behavior, informing a player that is their turn to do some of the actions described (see Figure 6.2).

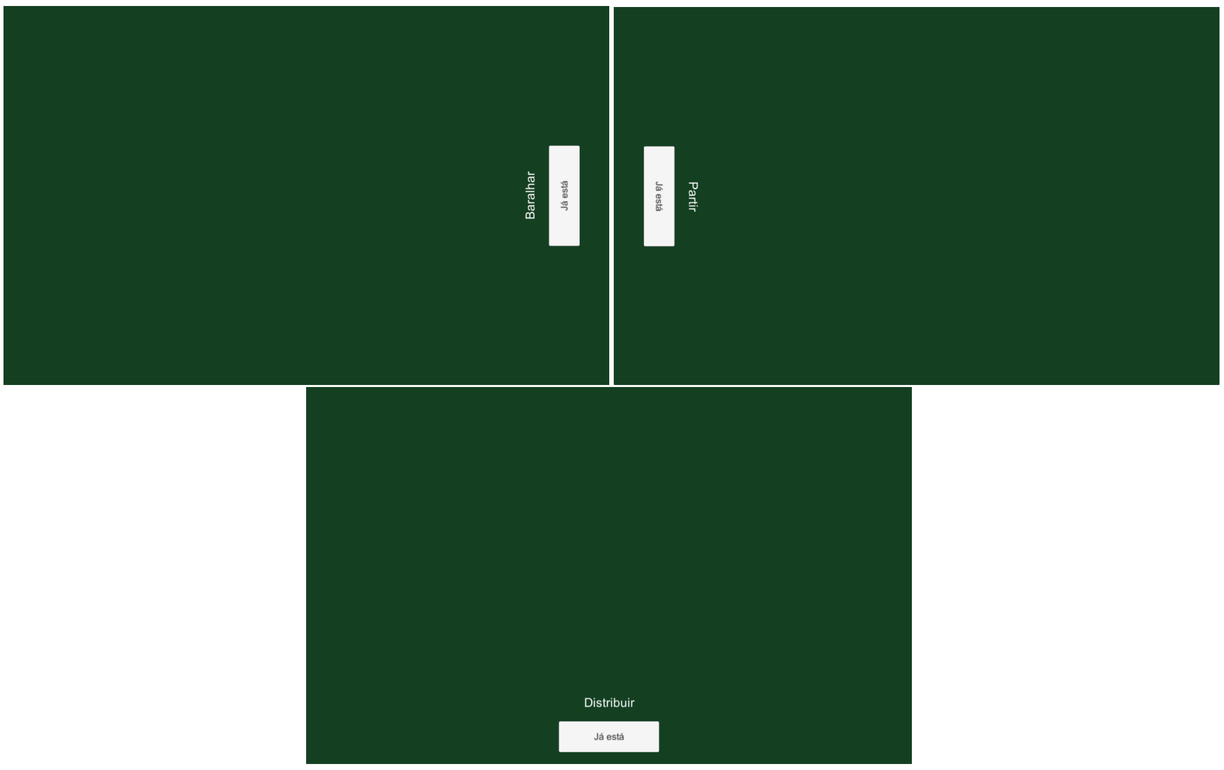


Figure 6.2: Shuffle, cut, and deal views of *Sueca*

At this moment, even when all the players have the cards in their hands, there is still some information they need to share in order to prepare the game state. Since the game is played using physical cards, there is no need to know what cards they have in their hand. However, there is one card that is more relevant than the others - the trump. The trump suit is the only element of interest at the beginning of a game (except when the human players are playing with an agent, as will be discussed in Section 6.1.3). For that reason, players are presented with a screen that allows them to indicate what is the trump suit, before the main view of the game. The main view of the game displays a very simple arrangement of elements. At the center, the area where players place the cards they want to play, and at the right lower corner is the trump suit and the number of the current trick, so that players can easily pick at this information when they need to. Teammates are located at opposite sides of the table. In order to distinguish the two teams, each team has a different color: one is red, the other is blue. Every time it is a player's turn, an indicator with their team's color lights up in front of them (see Figure 6.3). Although there are not many graphical elements composing this view, there is one that deserves more explaining: the center area. This area is an example of a component that is capable of detecting when cards are inside its boundaries and that was explained in Section 5.3. For this game, this component was extended so that it would only accept cards that were not yet played. This means that when a player places a card inside the limits of the component, the component is responsible for detecting the cards but, more specifically, only the ones that were not previously played. When such card is detected, it triggers an event to process that card and start its validation. Additionally, to provide a more visual feedback to the players, a white glow appears beneath the card so that they know the card was well detected and recognized. The center area component is also responsible of triggering other events to control the UI and to make the game-play feel more natural (see Listing 6.4). These additional events are triggered only if there are no cards inside the area but knowing that previously there were and depending on some conditions of the game state. This behavior allows certain actions of the game to be performed only when the players remove the cards from the center area, just like it would normally happen in the classic game. Usually, the events are triggered when all four players have played and one of them removes the cards, but situations like a renounce may anticipate the trigger. While the game is waiting for the players to remove the cards, all the indicators are turned off to indicate that they are required to do something in order to continue.

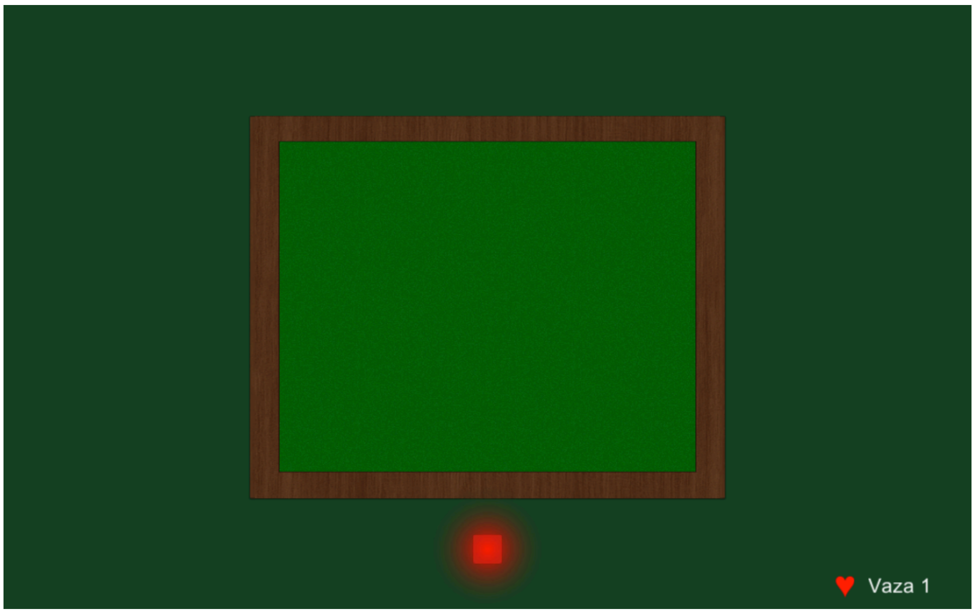


Figure 6.3: Main view of the *Sueca* card game

When the game comes to an end, a view showing the scores replaces the main one (see Figure 6.4). This score view shows not only the score of the last game but also the overall score of the session. A

```
[RequireComponent(typeof(MarkerReaderUI))]
public class SharedReader : MonoBehaviour
{
    // ...
    if (_reader.Markers.Count == 0 && _hadMarkersBefore)
    {
        // Trigger an event depending on the state
        EventManager.Trigger(RENOUNCE | GAME_END | NEXT_TRICK);
    }
    foreach (var marker in _reader.Markers)
    {
        EventManager.Trigger(Events.CARD_PLAYED, marker.getSymbolID()));
        hadMarkersBefore = true;
    }
    // ...
}
```

Listing 6.4: The events triggered by the center area of the game

session can have a different number of games which is chosen at the beginning, based on the number of games won (see Figure 6.5). From this view, players can continue playing the remaining games or go back to the start when the session is over.

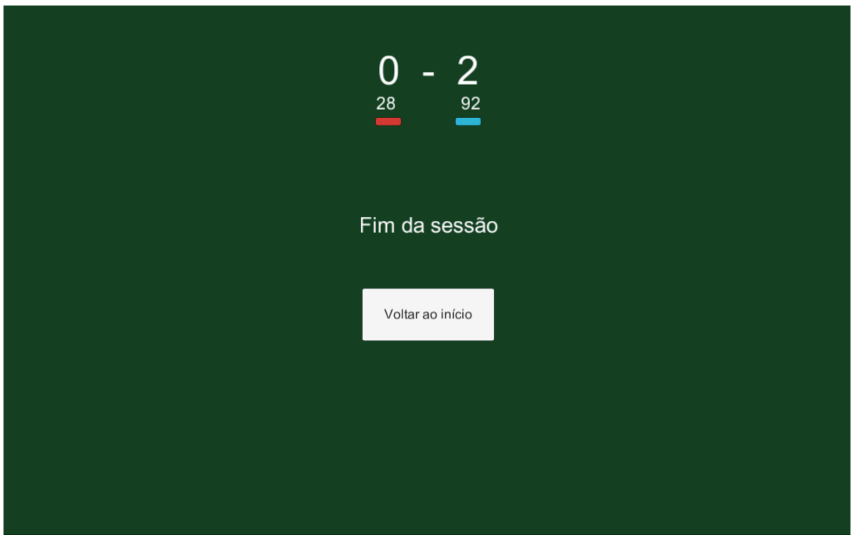


Figure 6.4: The scoreboard appears when a game ends

6.1.3 Integrating an agent

Integrating an agent in a game requires some changes both in behavior and in the UI. As mentioned in Section 5.4, the communication between the game and the agents is made through a bridge that defines the subscribing and publishing actions needed. For the context of *Sueca*, the most relevant actions that are sent to Thalamus are listed in Listing 6.5. These perceptions are sufficient to inform the agents about the current state of the game:

- `Play` is sent every time a `player` plays a `card`;

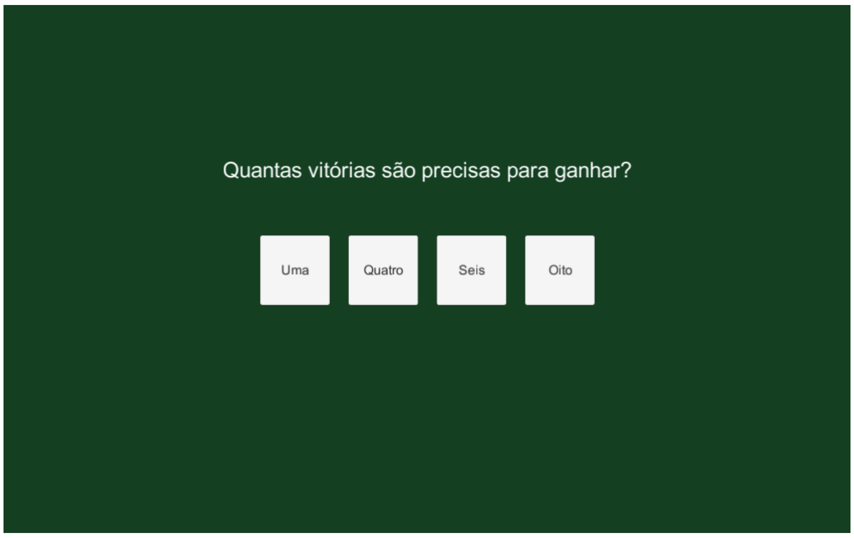


Figure 6.5: A session ends when a team wins the number of selected games

- `GameStart` provides relevant information about the game before the game actually starts, as the current `gameId`, the `trump` for that game, and the agent's `cards`;
- `GameEnd` informs the agent the final scores of each team, at the end of a game;
- `NextPlayer` indicates which player will play next;
- `SessionStart` announces how many games will be played during the session;
- `SessionEnd`, similarly to `GameEnd`, sends the accumulated scores for each team;
- `Shuffle`, `Cut`, and `Deal` tell which player will perform each action;
- `ReceiveRobotCards` informs the agent the game is in that specific phase;
- `TrickEnd` announces the trick's winner, as well as their points;
- Finally, `Renounce` shares which player has renounced.

From time to time, the agent also needs to send information back to the game (see Listing 6.6). The agent only needs to perform a single action which, in this case, is the play it is making.

The interaction between agents and the game demands some changes in the UI of the game. The adjustments made in the game's UI considered only the existence of a single agent. In order to support more agents playing at the same time, some of the solutions presented next would have to be rethought. Previously, after shuffling, cutting, and dealing the cards, all the players knew the cards they would be playing with. An agent, however, is not capable of holding its cards. For that reason, the encountered solution for this problem was to have a view where a player is responsible for placing the agent's cards on the display to be detected and recognized (see Figure 6.6).

The cards are placed in a area similar to the one present in the main view, but with another behavior attached. After all the ten cards are detected, the game can proceed. At this point, the agent does not need to deal with the physical cards, which leads to another problem. Since the agent will not handle the

```
public interface ISPerceptions
{
    void Play(int player, string card);
    void GameStart(int gameId, int playerId, int teamId, Suit? trump, List<ICard> cards);
    void GameEnd(int team1Score, int team2Score);
    void NextPlayer(int playerId);
    void SessionStart(int numGames);
    void SessionEnd(int team0Score, int team1Score);
    void Shuffle(int playerId);
    void Deal(int playerId);
    void Cut(int playerId);
    void ReceiveRobotCards();
    void TrickEnd(int winnerId, int trickPoints);
    void Renounce(int playerId);
}
```

Listing 6.5: Perceptions sent from the game to the agents

```
public interface ISActions
{
    void Play(int player, string card);
}
```

Listing 6.6: Actions sent from the agents to the game

physical cards, how can the game display what plays the agent makes? The answer to this question is by having a digital representation of the agent's cards. When the game starts, the 10 cards are displayed in front of the agent (see Figure 6.7). At the moment the agent sends its decision, one card is removed and appears, facing up, on the center area, illustrated by Figure 6.8. This way other plays know what card the agent played and play accordingly. Every time a trick is done, and the players remove the cards from the center area, the agent's card disappears. An alternative to these two changes in the UI would be not to have the intermediate view to read the cards and the digital representation of the cards, but instead placing the physical cards in front of the agent, which would be detected, and provide some visual feedback to indicate the agent's decision. This behavior would force one of the players to pick the chosen cards and placing it in the center area of the screen. Although this alternative would require less workarounds, imposing the task of placing the agent's cards on the center area, for every trick, would break the game flow.

Finally, at the end of a game, the score view appears. Even though the score is presented to the players, they can easily count the points won by reviewing the won tricks. Nevertheless, the tricks will be a card short when human players are playing with an agent. Therefore, to provide a way for players to count the points on their own, the view shows which cards belonging to the agent were won by each team (see Figure 6.9).

6.2 *Coup*

The *Coup* card game is very different from *Sueca*. It is a game of bluffing, bribery and manipulation for two to six players. The game has several states and there are many ways to transition from state to state

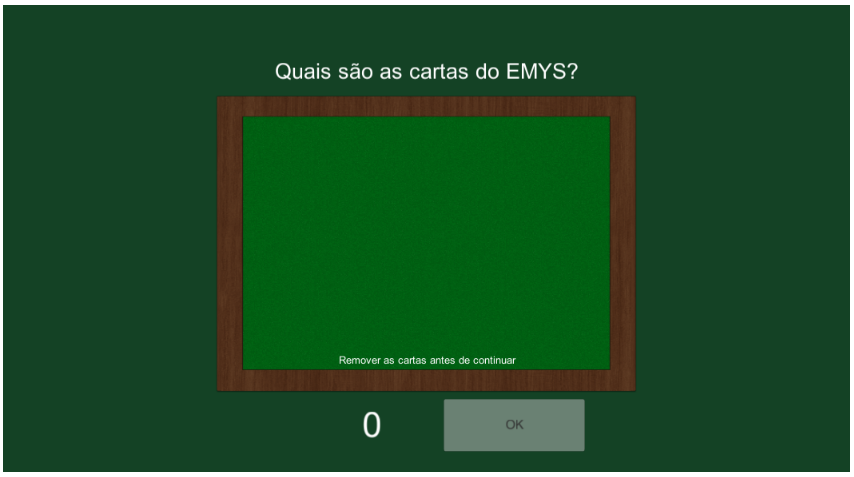

Quais são as cartas do EMYS?
Remover as cartas antes de continuar
0
OK

Figure 6.6: View to detect and recognize the agent's cards

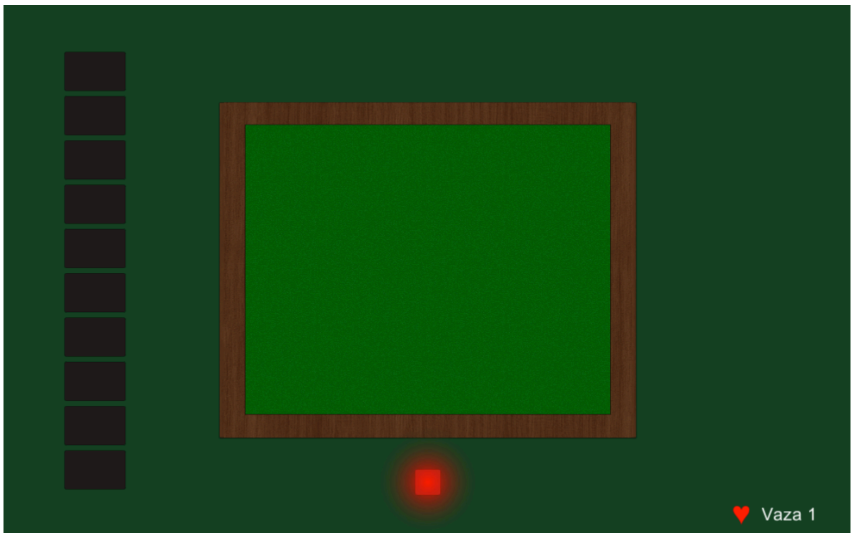


Figure 6.7: Main view of *Sueca*, with cards facing down

during a session. Using Flow, it is possible to manage these transitions and to structure the behaviors in order to have a clearer picture of what happens in a certain moment of the game. For that reason, the Flow graph built for this game is much more complex than the one from *Sueca* (see Figure 6.10).

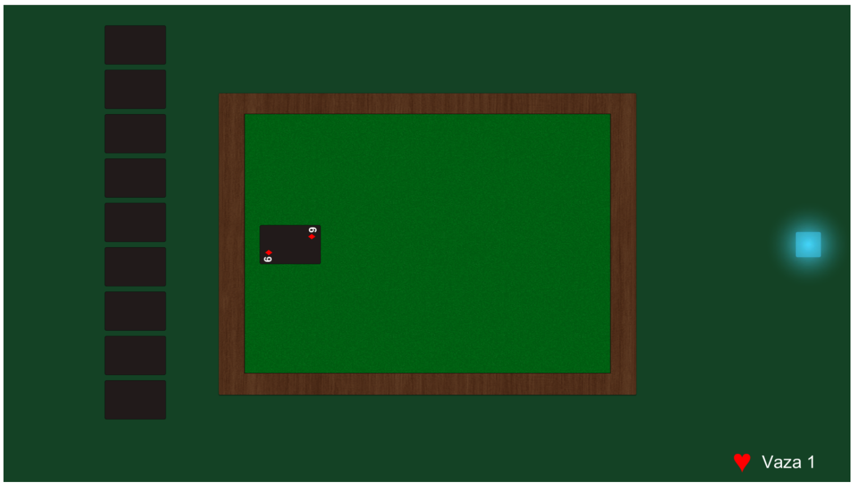


Figure 6.8: A digital card appears in the center when the agent plays

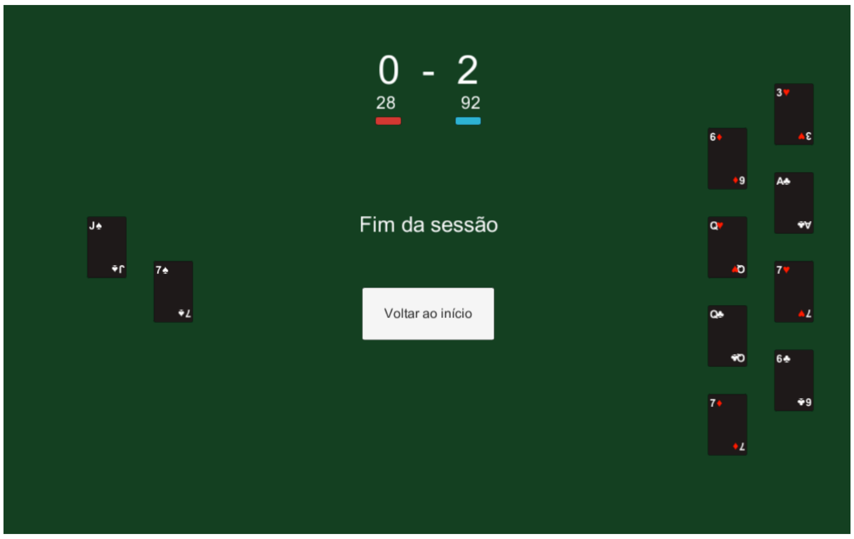


Figure 6.9: The agent cards won by each team are displayed in the scoreboard

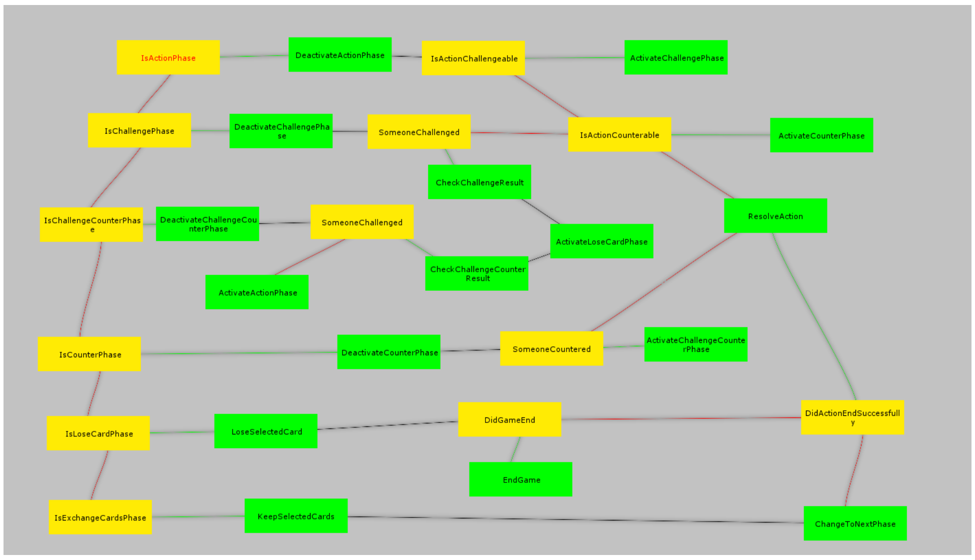


Figure 6.10: Graph to manage the states of the *Coup* card game

Chapter 7

Evaluation

A framework is composed of several tools, each with its own purpose. For that reason, each tool can be evaluated individually or, alternatively, as a whole. This framework is divided into several tools: Shuffle provides the building blocks for card games, Flow offers a graph-like interface to structure behavior, components to allow the detection and recognition of physical cards using fiducial markers, and also a way to integrate autonomous agents with the games. Although they all have different purposes, in the context of this work, it is not essential to evaluate each component individually, except for the detection of physical cards. The other components are evaluated based on their usefulness for building card games.

The following sections detail the performance of physical cards on a multi-touch surface, how the design of a card can influence their detection, and also their overall experience in the *Sueca* card game (Section 7.1). Finally, a brief assessment is made to the usage of the other tools for building card games (Section 7.2).

7.1 Detection and Recognition of Physical Cards

The idea of using physical playing cards to play card games on a multi-touch display is entirely new, which means there is no information available on how these cards should be in order to have a flawless detection and recognition. For that reason, some different designs were tested to try getting the best possible performance. The original intention for the cards was to change the design as little as possible comparing to standard cards. That means that the card would be white, with the fiducial markers replacing the illustrations, and with the rank and suit information (see Figure 7.1). Additionally, the size of the playing cards should be the same so that players feel comfortable playing with them. After trying this design on the touch display, some of its limitations are immediately revealed. The touch display uses a set of cameras to capture the display input and the quality of the captured black-and-white image is poor (see Figure 7.2). Given the constraints on the markers' size this fact is not surprising, however the white of the card is so bright that prevents the marker to be detected, as the marker blends with the background. Consequently, the background color of the cards had to change and, after trying different color options such as red and shades of gray, the best option was to make the cards black (see Figure 7.3). Whereas the white background had a bad impact on the marker's detection, the dark background extends the markers' borders and isolates them making their detection much easier.

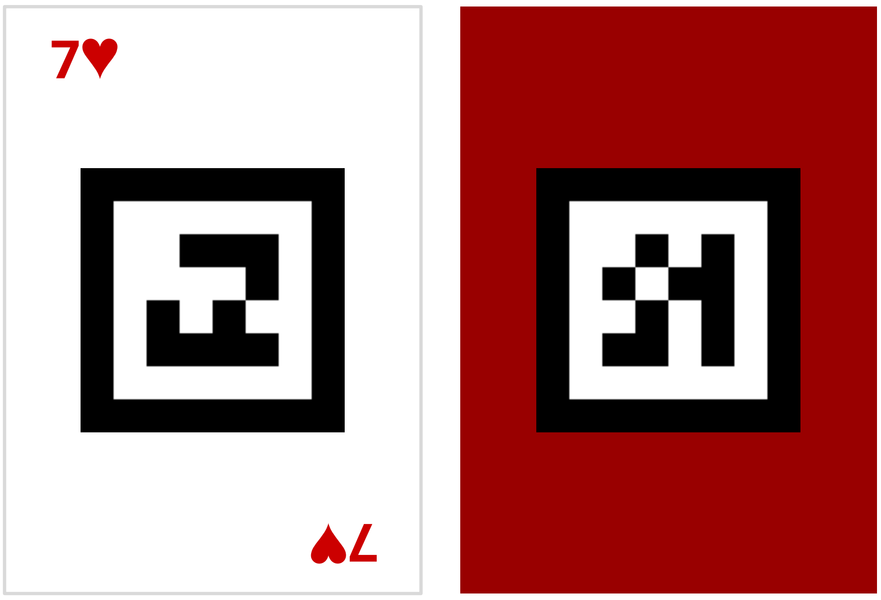


Figure 7.1: Original idea for the card design

Having the design finalized, some tests had to be made to verify if it is viable to use physical cards to play card games on a touch display. Therefore, two different tests were made. The first consisted in placing all the cards available on the touch display and see if the detected marker was, in fact, the correct one. Equally important was to understand if the detected marker could change during a certain interval. Instead of using time, the interval was measured in number of updates of game loop: 10, 50, and 100 updates. For every interval, each card was placed on the display 5 times and the percentages of correct and incorrect recognition were recorded. The results of this experiment can be seen in Figure 7.4. For an interval of 10 updates, the percentage of correct recognition was 78%, while for 50 and 100 updates the results show a small improvement, with a value of 82%. The achieved values are too much alike to conclude that, after a certain interval, the chances of recognizing the correct marker code increases. However, the amount of incorrect values can be justified by the unsatisfactory quality of the captured image, which is not able to distinguish the markers with precision due to the similarities among them. In addition, comparing a marker with all 4096 possible markers available increases the probability of getting an incorrect value.

Unlike the first round of testing, the second set of test results gathered information of 20 *Sueca* card game sessions with students. Each session was composed of 5 games, which translates to a total of 100 games played. In order to declare that the usage of physical cards is viable, the majority of the played games must end successfully, meaning that all cards were correctly recognized. Results show that 96 of the total number of games ended without any problem, corresponding to a percentage of 96% correct

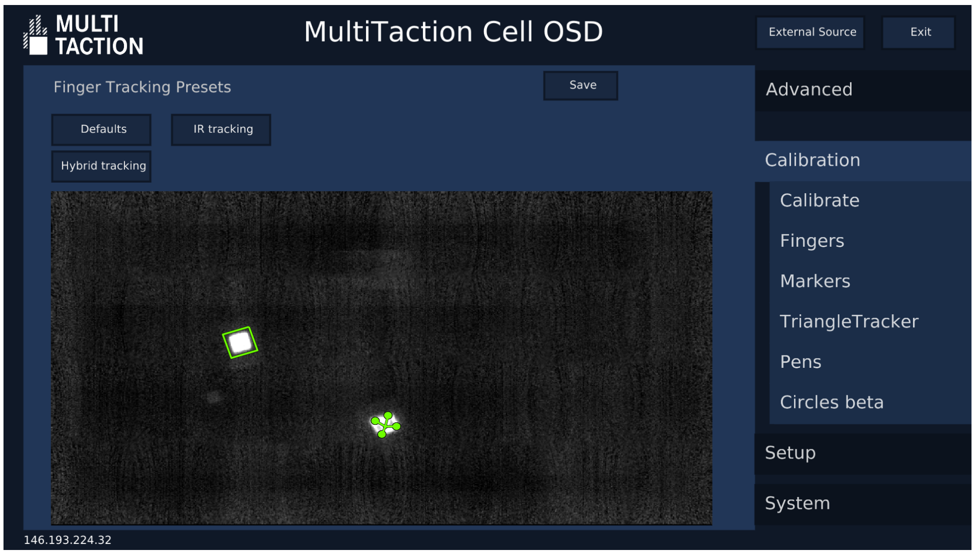


Figure 7.2: The quality of the captured image while detecting two markers

identifications. Comparing this result with the results of the test mentioned before, the improvement is noticeable. First, it is of great relevance to mention that the number of card detections is much higher than the previous, *i.e.* 4000 versus 600. Second, instead of comparing the markers with all possible values, they are only compared with markers attached to cards that are still left to play. By reducing the amount of possible values, the probability of an incorrect recognition decreases significantly.

7.2 Libraries and integration of agents

The main purpose of these libraries is to ease the creation of card games by providing a set of concepts that constitute these games and a graphical way to structure behavior. Such libraries cannot be measured empirically, given that is impossible to extract any numbers or statistics from their usage. Alternatively, they can be evaluated by their suitability for constructing card games. As described in chapter 6, these libraries were used to build the *Sueca* card game, and also to provide some organization for dealing with state changes on the *Coup* card game. The Flow library was used in both cases, proving that it is applicable for, at least, more than one type of card games. Moreover, it shows that Flow can be used to represent simpler behaviors but also more complex ones. On the hand, the Shuffle library was used only for building the *Sueca* game, but proved to be rich enough for building trick-taking card games. Nevertheless, in order to have more confidence about the coverage needed to build a large range of card games, the development of a bigger number of card games is required.

Relatively to the integration of autonomous agents in card games, the developed cases were sufficient to demonstrate that is possible to combine card games with these agents in a simple way. Integrating more than one agent in any game is only a matter of changing the UI to match the needs of the game.

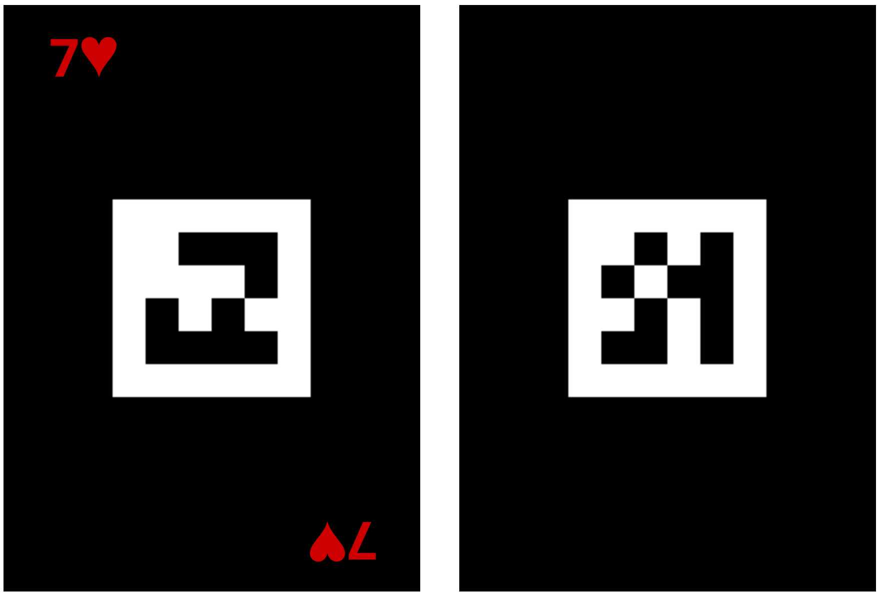


Figure 7.3: Final card design with fiducial markers

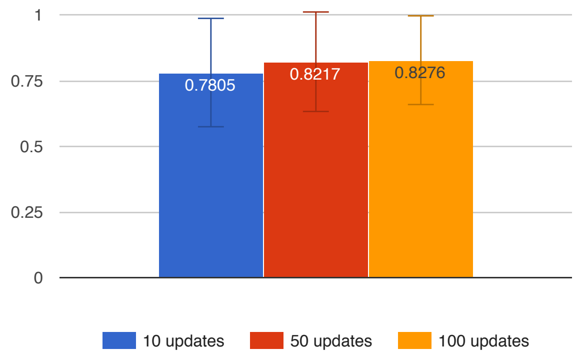


Figure 7.4: Average percentage of the correct recognition of the markers

7.3 Overall experience

Testing the system in the university domain is useful, but sometimes it is more important trying to test it with people that are accustomed to play in a day-to-day basis and that, perhaps, are not as tech-savvy as the students. Therefore, in an attempt to perform tests outside the university, the whole system was taken to an event featuring a *Sueca* tournament. Although having the system as part of the tournament would have been interesting, the scenario was installed in a hall to invite anyone curious to play the game. Some of the researchers present started to play first to show how the game works and the interactions with the agent, and second to captivate more people and invite them to play. Despite the limited time available, some people demonstrated great interest and enthusiasm in participating in the experiment (see Figure 7.5). The participants, after playing several games, were asked to fill a questionnaire to provide their insights about the experience. The questionnaire had two questions that focused more on the interaction with the touch display and the physical cards:

Figure 7.5: Participants show enthusiasm while playing *Sueca*

1. *Did you enjoy playing Sueca on the multi-touch display?*;

2. *Does the experience of playing Sueca on a multi-touch display have any issues?*.

The answers to these questions are highlighted by Figure 7.6. For the first question, the majority of the participants (approximatelly 65%) loved the experience of playing *Sueca* with physical cards and an agent on a multi-touch table, whereas 35% of the participants thought that it felt strange to play in this conditions. As for the second question, most of the participants thought that there were some problems with the interaction (almost 75%). Over 41% of the participants concluded the game does not flow in a natural fashion because of all the rules that are imposed in order to keep a correct game state. Moreover, more than 35% feels that the surface takes an excessive amount of time to detect the cards, which also contributes for a unnatural game play, and almost 12% of the participants has difficulties to identify the cards. Despite all the problems identified, close to 30% of the players thought the experience was good and that it did not have any issues.

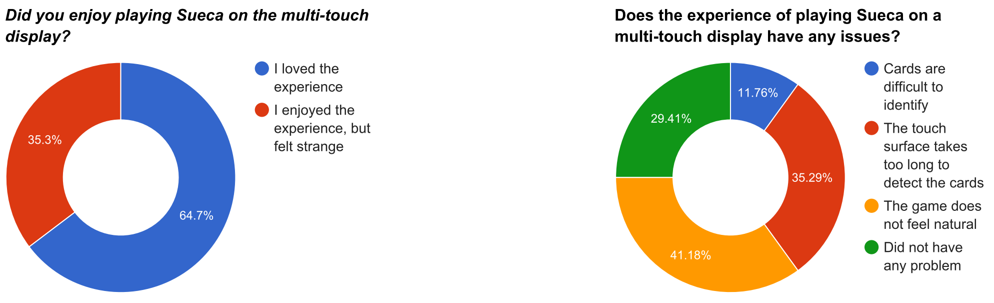


Figure 7.6: Results of the two questions about the experience

Chapter 8

Conclusions

The creation of a framework with the purpose of building card games for multi-touch table displays is an opportunity to develop a set of tools that encourages the development of new games and scenarios, but also to explore new approaches for the interaction with those games.

The development of this work led to the creation of two libraries: Shuffle and Flow. The first is intended to be a starting point for the development of new card games, offering a set of components that represent concepts that are common to several card games, as well as the representation of well known families of card games. The second proposes a graph-like approach to structure a collection of behaviors. Abstracting the behaviors in nodes promotes composition as nodes have a single purpose and therefore can be reused in other contexts. These two libraries were used to build the *Sueca* card game from scratch, where Shuffle helped in the description of the game state and Flow in the construction of a validation graph for plays. Flow was also used to control the game state of the Coup card game. Although these libraries were sufficiently complete to accomplish the desired tasks, more scenarios have to be developed to prove they cover a great number of games.

Exploring new approaches to play card games on multi-touch surfaces was also one of the goals of this work. The idea of using physical cards could change the way people play card games on touch surfaces. The limitations in terms of hardware are evident, causing the recognition of the cards to be incorrect from time to time. Nevertheless, results show that, limiting the range of codes available to the game, the percentage of incorrect values decreases, reducing their impact during the game.

Although the work done was able to accomplish the purposed goals, there is always room for improvements. The next section describes some of the problems the current implementation has and presents some suggestions that may overcome those problems or, at least, diminish them.

8.1 Future Work

As expected, the time available for this project is not sufficient to try or implement every idea that would be useful. In this case, the following sections discuss the different ways the work could be improved, by correcting some of the current implementation, and also by adding additional features to match the ones already present.

8.1.1 Shuffle

As a framework to assist the creation of card games, having only a single example to show proves that the framework is adequate for building that specific game. Therefore, more card games have to be developed in order to prove the robustness of this work. Furthermore, these games can expose the flaws of the framework's several libraries which can then be improved accordingly. A flaw that is evident by looking at the Shuffle library is that it lacks the representation of other families of card games. The library has the representation of a generic and trick-taking, but not other types of card games. Consequently, the representation of other families of card games should be made. Another improvement that can be made to Shuffle is promoting the usage of immutable data structures. Using immutable data structures prevents the modification of an object, ensuring that the object does not change once it is created. This way, to modify a certain object, a new object of the same type must be created with the new information which may lead to less errors. In addition, immutability can also be useful when restoring some previous state of the application. For instance, keeping track of all the states the *Sueca* card game would provide a great advantage either during debugging or a game session: the state of the game could be easily rolled back without having to worry if the previous state was modified after it was stored.

8.1.2 Flow

Following the modifications and functionality that can be added to Shuffle in the future, Flow must be updated accordingly. With the idea of immutability in mind, the way Flow nodes have access to the state of the game, or application, also has to change. Currently, nodes grab the current state of the game to query or modify it. Alternatively, nodes will have to pass the new state to the next node and return the resulting state at the end. On top of that, having a greater number of predefined actions, predicates, and flows would also be important in the future, once more games are developed.

Flow also has a graphical component that could be improved, as well, to make superior editing experience. On the one hand, the nodes and the way connections are currently made could receive a little more attention. Nodes are too simple and, despite having different symbols that represent them, it may not be obvious what a node can do. The node illustrated by Figure 8.1 shows how the nodes can be improved. This new node represents the current *IsFirstPlay* node, exposing the notion of input and output ports. These ports connect the nodes with each other, and provide immediate feedback of the connections made. Having these ports allows the developer to build much more complete nodes, instead of being limited by the ones currently in use. On the other hand, the manipulation of the graph is too limited. As nodes are inserted and connected with each other, the resulting graph starts to grow to the point it becomes really difficult to continue working with it effortlessly. For that reason, the editor could feature dragging and zooming functionalities.

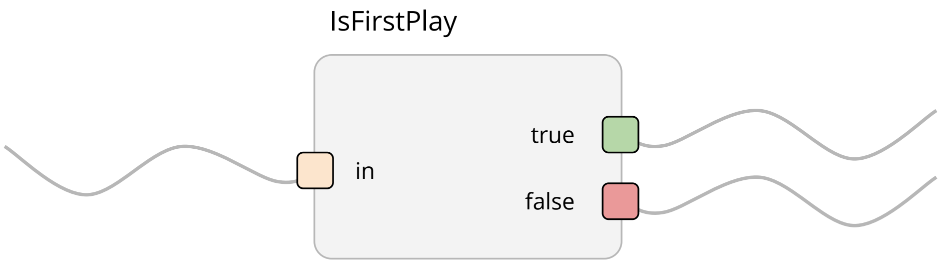


Figure 8.1: Future version of nodes in Flow

8.1.3 Card detection and design

Concerning the card's interaction and design, there are also some improvements that can be tested. First, MultiTaction added support for another type of markers (see Figure 8.2). These markers should be tested against the current ones to know if the performance of the detection and recognition increases. Second, some research can be done in order to find if there exists a direct relation between a marker's code and the incorrect ones. The used markers are built by painting white squares against a black background, which attributes a value to the code. By predicting the possible codes that can be misinterpreted by the touch display, it may be possible to reduce the number of incorrect occurrences. Instead of assigning a single code to a given card, a range of code could be applied.

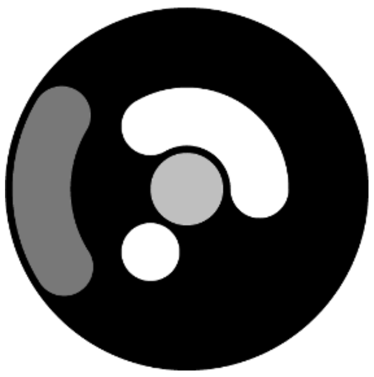


Figure 8.2: MultiTaction now supports circular 2D markers

Bibliography

[1] A. Al Mahmud, O. Mubin, S. Shahid, and J.-B. Martens. Designing and evaluating the tabletop game experience for senior citizens. *Proceedings of the 5th Nordic conference on Human-computer interaction building bridges - NordiCHI '08*, page 403, 2008.

[2] C. Magerkurth, M. Memisoglu, T. Engelke, and N. Streitz. Towards the Next Generation of Tabletop Gaming Experiences. In *Proceedings of Graphics Interface 2004*, pages 73–80. Canadian Human-Computer Communications Society School of Computer Science, University of Waterloo, 2004.

[3] S. McNeely. *Ultimate Book of Card Games: The Comprehensive Guide to More than 350 Games*. Chronicle Books, 2009.

[4] H. Ishii. The tangible user interface and its evolution. *Communications of the ACM*, 51(6):32–36, 2008.

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

[6] M. Thielscher. A general game description language for incomplete information games. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 994–999, 2010.

[7] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius. A card game description language. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7835 LNCS, pages 254–263, 2013.

[8] M. Law and G. Deane. General card game playing. 2013.

[9] S. Benford, C. Magerkurth, and P. Ljungstrand. Bridging the physical and digital in pervasive gaming, 2005. ISSN 00010782.

[10] T. Whalen. Playing Well with Others : Applying Board Game Design to Tabletop Display Interfaces. In *ACM Symposium on User Interface Software and Technology*, pages 4–5. 2003.

[11] S. Patel, J. Bunch, K. Forkner, L. Johnson, T. Johnson, M. Rosack, and G. Abowd. The design and implementation of multi-player card games on multi-user interactive tabletop surfaces. *Entertainment Computing–ICEC 2004*, pages 339–344, 2004.

[12] S. Gabrielli, S. Bellutti, A. Jameson, C. Leonardi, and M. Zancanaro. A single-user tabletop card game system for older persons: General lessons learned from an in-situ study. In *2008 IEEE International Workshop on Horizontal Interactive Human Computer System, TABLETOP 2008*, pages 85–88, 2008.

[13] H. Kato. Artoolkit. *http://www. hitl. washington. edu/artoolkit/*, 1999.

[14] Q. Bonnard, S. Lemaignan, G. Zufferey, A. Mazzei, S. Cuendet, N. Li, A. Özgür, and P. Dillenbourg. Chilitags 2: Robust fiducial markers for augmented reality and robotics., 2013. URL `http://chili.epfl.ch/software`.

[15] M. Kaltenbrunner and R. Bencina. reacTIVision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, pages 69–74. ACM, 2007.

[16] M. Kaltenbrunner. reacTIVision and TUIO : A Tangible Tabletop Toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 9–16. ACM, 2009.

[17] C. Floerkemeier and F. Mattern. Smart Playing Cards – Enhancing the Gaming Experience with RFID, 2006.

[18] K. Römer and S. Domnitcheva. Smart Playing Cards: A Ubiquitous Computing Game, 2002. ISSN 16174909.

[19] B. Schneider, P. Blikstein, and W. Mackay. Combinatorix. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces - ITS '12*, page 129, New York, New York, USA, Nov. 2012. ACM Press. ISBN 9781450312097. doi: 10.1145/2396636.2396656.

[20] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 139–146. ACM, 2007.

[21] M. Augstein, T. Neumayr, R. Ruckser-Scherb, I. Karlhuber, and J. Altmann. The fun.tast.tisch. project. In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces - ITS '13*, pages 81–90, New York, New York, USA, Oct. 2013. ACM Press. ISBN 9781450322713. doi: 10.1145/2512349.2512808.

[22] M. Möllers and J. Borchers. TaPS widgets: Interacting with Tangible Private Spaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '11*, page 75. ACM Press, 2011.

[23] T. Bartindale and C. Harrison. Stacks on the surface: resolving physical order using fiducial markers with structured transparency. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 57–60. ACM, 2009.

[24] A. Pereira. *Socially Present Agents for Tabletop Games*. PhD thesis, 2014.

[25] T. Ribeiro, E. Di Tullio, L. J. Corrigan, A. Jones, F. Papadopoulos, R. Aylett, G. Castellano, and A. Paiva. Developing interactive embodied characters using the thalamus framework: A collaborative approach. In *Intelligent Virtual Agents*, pages 364–373. Springer, 2014.

[26] T. Ribeiro, E. Tullio, P. Alves-Oliveira, and A. Paiva. From thalamus to skene: High-level behaviour planning and managing for mixed-reality characters. 2014.