FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Development platform for elderly-oriented tabletop games

Tiago Manuel Alves Pereira Marques

Master in Informatics and Computing Engineering

Supervisor: Rui Pedro Amaral Rodrigues (PhD)

Second Supervisor: Paula Alexandra Silva (PhD)

July 19th, 2011

Development platform for elderly-oriented tabletop games

Tiago Manuel Alves Pereira Marques

Master in Informatics and Computing Engineering

July 19th, 2011

Abstract

Peripherals are one of the most common obstacles when talking about computers and elderly people. This thesis stands on the notion that Natural User Interfaces and Games may contribute to a solution to this problem and studies how this can become true. It was conducted using a tabletop as the research device.

The main techniques and relevant frameworks available, commonly associated with the use of tabletop technology, are studied in order to predict which combination of those is the most beneficial to the project and to the end-user. Hardware and software tests, as well as low-fidelity prototypes were designed and used to examine which input mechanisms could be implemented on the final application. This also enabled a better understanding of what benefits older adults would have, by using these types of interaction.

With the observations made and results obtained from the tests performed, some gaming aspects were identified as more common among several potentially interesting game types. A framework for game development was conceptualized and created considering those results. In order to test the framework created, a game especially conceived for older adults was designed and implemented on top of the framework. The game implementation is described in adequate detail to this project and its validation process is explained in order to confirm its usefulness to the end-user. The validation process revealed some aspects in which the game could be improved, however, these modifications did not require any change to the framework's concepts or implementation. This process also indicated that the usage of the tabletop can promote social interaction and cognitive and motor stimulation, as intended.

Resumo

Um dos obstáculos mais comuns quando se interliga computadores e pessoas idosas são os periféricos. Esta tese assenta na noção de que Interfaces Naturais para o Utilizador e Jogos podem constituir parte da solução para este problema e investiga como tal pode ser feito. Para isso, utiliza um tabletop como base da investigação.

As técnicas e frameworks disponíveis e consideradas mais relevantes, usualmente associadas ao uso de tabletops, são estudadas para tentar perceber qual a combinação destes componentes que este projecto e o utilizador final mais beneficiarão. Testes em ambos hardware e software, bem como protótipos de baixa fidelidade foram efetuados para examinar que mecanismos de input poderiam ser implementados na aplicação final. Este estudo possibilitou uma perceção dos benefícios inerentes para as pessoas idosas do uso deste tipo de dispositivos

Com a informação reunida das observações e testes feitos, foram identificados alguns aspectos mais comuns de serem usados em tipos de jogos benéficos para as pessoas idosas e para este projecto. Uma framework para o desenvolvimento de jogos foi concebida e criada de maneira a responder aos objectivos deste trabalho. Para testar a framework criada, foi desenhado um jogo direcionado para a população idosa, usando os conceitos desta nova framework. A implementação do jogo é descrita e o processo de validação do mesmo é mostrado, de maneira a confirmar a sua utilidade para o utilizador. O processo de validação revelou que, embora alguns aspectos do jogo poderiam ser alterados, nenhuma modificação ao conceito ou implementação da framework seria necessária. Este processo também indiou que o uso do abletop promove a interacção social, bem como estimulação cognitiva e motora, tal como pretendido neste projecto.

Acknowledgements

First and foremost, I would like to express my gratitude for both my supervisors Rui Rodrigues and Paula Alexandra Silva for their unbiased and ever-so willing support. Their vast knowledge on Multi-touch technologies and Human-Computer Interaction provided me with a great stepping stone to build my work on. Their inciting answers and remarks of both my questions and my work fuelled my own quest for solutions, alternatives and answers and I recognise their importance.

I would like to thank my parents for their support throughout the years in my professional and personal matters. I truly am grateful to them for being able to pursue my dreams and for their belief in me.

To my friends for putting up with me, for keeping me company in good and bad times, for telling me off when I need, for the fun classes we attended, for all the stunts and all-nighters we needed to pull, among many other unforgettable moments.

I would also like to thank Fraunhofer AICOS for the research scholarship granted and the environment in which I had the chance to work in.

Tiago Marques

“Computer Science is no more about computers than astronomy is about telescopes”

Edsger W. Dijkstra

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

API	Application Programming Interface
AS3	ActionScript 3
DI	Diffused Illumination
DSI	Diffused Surface Illumination
FLC	Flash Local Connection
FTIR	Frustrated Total Internal Reflection
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HD	High-Definition
HDMI	High-Definition Multimedia Interface
ID	Identification
LCD	Liquid Crystal Display
LLP	Laser Light Plane
NI	Natural Interaction
NUI	Natural User Interface
OS	Operating System
OSC	Open Sound Control
SDK	Software Development Kit
TCP	Transmission Control Protocol
TUIO	Tangible User Interface Object
UDP	User Datagram Protocol
UI	User Interface
USB	Universal Serial Bus
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Context

The use of computers is commonly associated with peripherals such as the mouse and the keyboard. These devices have been recognized as a computer's usual means of input and have become efficient devices for most of the population. However, they are far from being considered as an intuitive approach to interaction and present major difficulties for the user, especially elderly people, mostly due to the indirect mapping between the input device and the application [KK96]. In the more recent generations, the availability and the growing social acceptance of computers drove people to learn how to use them, while older generations, because they did not face the need to use them, did not become familiar with them and therefore may now have trouble dealing with keyboards and mouses [Cza96, Cza97]. By correctly designing an interface that suits users, this issue can be counteracted. However, interface design problems are not the sole cause of this problem. Other difficulties arise due to arthritis, tremors, or other physical problems that make mouse manipulation and keyboard entry difficult for these users [Han01].

The unfamiliarity with a system may cause a confusing or unsatisfactory usage of an application to any user. The interaction between software and user should be as smooth as possible so that the user is able to complete the task they want with no difficulty other than the one associated with the task itself. Natural Interaction (NI) may be presented as a part of the solution to this problem. NI is achieved through clever designs that constrain this problem in ways that are transparent to the user [Lee10]. Devices that employ the concepts of NI are able to interpret gestures or sounds that may come natural to the user [Lee10]. These characteristics may make NI a viable choice when developing a system that focuses on Human-Computer Interaction (HCI).

An example of a device based on NI is the tabletop system. A tabletop is, roughly, a device that allows interaction with a computer application through gestures, touch, finger movement on the surface or even sounds, depending on how it is built and on the application itself. Its surface works both as a computer monitor and an input method for the system. Thus, a computer connected to a tabletop can be manipulated through these inputs. More specifically, these devices may support single or multi-touch interaction. Multi-touch devices, such as the tabletop, are touch-based devices that allow interaction through multiple fingers. This form of technology is a more direct approach to computer interaction than the regular input methods mentioned that require the use of peripherals, as it attempts to eliminate the barrier between the user and the computer by removing the need for external peripherals as input methods [vHB01].

Multi-touch technologies are recently being developed in the context of NI as they present more options than the regular single-touch interaction [HRB$^+$09]. Recently, a growing number of studies [HRB$^+$09, SBD$^+$08a, RHM$^+$09] are being conducted using multi-touch technology due to its potential as it enables the combination of the advantages of NI devices, with a more tactile user interaction.

Some types of users - elderly people for example - may require specific user interaction studies due to their potentially different cognitive and perception capabilities, when compared to others. Natural User Interfaces (NUI) offer a direct mapping between the input device and the application in an effort to solve the issue raised - the link between older adults and technology. Multi-touch technologies and NUI, usually more intuitive approaches to HCI than the usual keyboard and mouse, may be presented as a way of replacing these peripherals as the computer's main input without having a significant learning curve associated [KK96].

To summarise, NI methods may be a solution to ease the interaction between elderly people and computers, and the diverse possibilities and usages that computers have to offer may also be of benefit to the elderly, in terms of entertainment and motor/cognition stimulation.

1.2 Motivation and goals

One of the popular and recurrent sources for studies in HCI are gaming environments, often associated with entertainment [YH07]. Nowadays, this definition is becoming more and more lacking as games are also being used for purposes other than entertainment, such as learning or training instruments, i.e. serious games [Jac10]. For that reason, games have been playing important roles in today's society. A combination of educational and entertainment games is usually attempted to try and motivate the player. Common combinations include games that can teach children subjects like mathematics while entertaining them at the same time and games that help older adults with their daily health-care and still provide social interaction. As such, the possibilities that games provide are of utmost interest.

This thesis aims to blend a tabletop system in senior people's lives with as little interference as possible in their habits or traditions. As such, the goals that have been established for this thesis are:

- establish which combination of touch detection techniques and frameworks is most favorable to be used in multi-touch game development for elderly people
- provide a calibration method for these devices, in order to optimize touch and object detection, and develop small applications to test the capabilities of the hardware and framework chosen
- create a concept and implement a new game framework to be used in multi-touch games for elderly people
- implement and test a game prototype to validate the new development platform's concepts, flexibility and capabilities

Considering the usefulness of games for older adults and the interesting ongoing studies in HCI for elderly people, the previously described tabletop represents an opportunity to develop a game framework for this emerging device. The framework developed in this project attempts to create an abstract layer for game development on multi-touch surfaces. This framework is indended to be useful for developers attempting to create tabletop games for older adults. The research conducted and results obtained in this thesis, incorporate HCI concepts for application development for older adults to enable faster and more correctly designed user inrefaces.

1.3 Structure

With the purpose of addressing the goals established, a line of work was devised. This document's structure is representative of that work plan and is divided in eight chapters.

This chapter introduces the research areas this document refers to with a review of existing literature and research on related work, which was necessary to identify existing solutions and open issues. The reader is contextualized to better understand the aim of this thesis. Sections regarding the project and its purpose are also included, along with the writer's motivation and the structure of the document.

Following the introduction, chapter 2 presents an analysis of the existing concepts in which the benefits of games for elderly people are explained to the reader. Some context regarding the usefulness of NUI and respective devices is also provided. The last section of this chapter describes the core of this research. It presents different approaches to detecting and tracking touch in multi-touch devices, comparing techniques and frameworks in order to better understand what combination of both touch detection technique and development framework the end-users will benefit more from.

Chapter 3 describes the methodology and approach taken, along with a more detailed explanation of the development process. This chapter is intended to support the choices made during the several stages of the project, including research, prototyping and development.

From this chapter on, this document describes the development process, starting with the hardware and software tests and the mock-ups used to test functionalities with the elderly in chapter 4. These tests were performed in order to grasp the limitations of both hardware and software, while the prototypes were made to identify types of interaction, relevant to the use of tabletops and common to usually played games.

The actual implementation stage of this thesis is described in chapter 5, where an architecture is proposed and explained. Moreover, and according to the user's feedback and reactions observed during the testing and prototyping phase, an application implementing an appropriate game type and gameplay was developed and its concept, along with the implementation's description is presented.

Chapter 6 addresses the validation process of the final application and draws results from the test performed.

In chapter 7, all the development process is analysed in detail. Alternatives to the approach are considered, weighed and compared to the one used. The final application and its results are also object of appraise and implementation decisions are questioned and balanced with possible alternatives.

As a summary of this work, chapter 8 draws conclusions from the final application and its results and possible ways to augment and improve the study performed in the scope of this thesis as means of future work are hypothesized.

Chapter 2

Computer games for elderly people

This chapter provides an overview of the main concepts involved in this work and describes the alternatives that may be adopted for the project at hand. In the first section, the objective and benefits of the use of games for the end-users of this project are described. Later, the concept of Natural User Interface is presented along with some types of interaction and devices associated, such as the tabletop.

With such concepts in mind and the instrument of work - the tabletop - described, its hardware and functioning are explained and compared to other NI devices. The various alternative techniques for building the device and frameworks for application development are analysed considering their usefulness to the end-user.

2.1 Introduction to games for elderly people

Games may have different goals according to purpose and to the target population. Games can be considered for pure entertainment or they can be of a more serious nature (for teaching and training for example). For an application to be considered a game, it must have an entertaining purpose, hidden as it may be, otherwise they would be considered regular tools for training or working. However, in many cases, what is hidden is the serious purpose itself, so users will focus on getting through the game, not realising they are being taught or trained [Fer10, Jac10, Wik11g, Wik11f].

2.1.1 Benefits of games for elderly people

Games are known to have many effects on people and in spite of some negative reactions, games, due to their entertainment purpose, can cause the human brain to be more attentive. This can enable the establishment of connections, thus creating new memories and relationships and consequently making it possible to work cognitive-wise [GAea09].

There are a number of demographic characteristics and age-related sensory, cognitive and motor properties that may influence an older adult's experience when interacting with digital games [INdKP07], which may not apply to younger adults. This is due to the decline of some kind of perception and/or cognition that is more likely to occur in elderly people rather than in younger adults [Gre09, GAB$^+$06, NSA10]. Modifications in the body are not the only changes that occur; the number of social acquaintances also decreases with age [NMS10]. While younger adults grow large social networks with the purpose of finding a mate, older adults concentrate on the most satisfying and humanly rich relationships to maintain an emotional balance [FHH06]. If we consider that most older adults live unaccompanied [FEL09], one quickly understands how the loss of loved ones and the geographical isolation can impact the social interaction of the individual. Moreover, social interaction and community participation are very important to maintain one's mental health and well being [All08]. Therefore, it is determinant that they are stimulated accordingly [DSVA10].

Consequently, there is the need to design interfaces that transmit rich and rewarding experiences, combining interaction styles with content that will directly speak to and engage elderly users [INdKP07, Fab06]. Therefore, games for elderly people must balance both aspects for the benefit of the end-users. More specifically, the application must render the user the "right" information, at the "right" time, in the "right" way, so that it is more likely to succeed in its objectives [Fis01].

2.1.2 Results obtained

The Information Society Technology Programme concluded [Fab06] that games for older adults should focus on maintaining and improving cognitive abilities, enhancing users' social networks and improving their physical condition by means of training fine psychomotor skills. They should promote feelings of independence and the opportunity to learn while playing. This study concluded that games for elderly people should include games to improve cognitive abilities such as attention, memory, executive function and fine psychomotor skills, prioritising subcategories of these, such as selective and divided attention, short-term memory, problem solving capacities and categorisation processes.

In a study by C. Shawn Green and Daphne Bavelier, "The Cognitive Neuroscience of Video Games" [GB04], video game effects on people were conceptualized and tested in order to determine whether video games had a positive impact. This study suggested that video game experience could be a powerful tool in slowing, stopping, or even reversing the age-related declines in perceptual, motor, and cognitive capabilities faced by the elderly population. Video game experience was also found to be beneficial as compared to no video game experience.

In another study by Jeffrey Goldstein entitled “Video games and the elderly” [GCO$^+$97], results indicated that playing video games was related to a significant improvement in reaction time and to a relative increase of wellness. In this study, the group selected to play a video game manifested faster reaction times and a more positive sense of well-being compared to the group that did not play it.

Research indicates that cognitive abilities can be stimulated through games and that these may play a significant role on the well-being of their players. This allows us to conclude that games may be a way to help older adults stay active and lessen the psychomotor functions deterioration associated with ageing.

2.2 Natural User Interfaces

The concept of NUI refers to a User Interface (UI), that can be, in practical terms, imperceptible and natural to use [Lee10]. In regular user interfaces, devices need to be used to interact with a system and a learning period is usually needed to be able to use that device accurately. In NUIs, the device that separates the user and the system should be as unobtrusive as possible or hidden so that the user does not notice it. Naturally, there will be a learning period for the UI, although it is greatly reduced due to the fact that NUIs have a more natural feel to the user [Blo11]

Natural HCI seeks the integration of human language into technological applications, and the mimic of the way we live, work, play and interact with each other in everyday life, thus avoiding interaction complexity and reducing the cognitive load that is typically associated with standard interfaces [BBL08, KKL07].

2.2.1 Types of interaction

A NUI usually relies on different types of interaction such as touch detection, voice recognition, gesture analysis (without touching) and haptic interfaces. These types of interaction intend to deviate from the usual keyboard or mouse as main input for the system. However, in terms of external components, tangible objects can also be used with any of these types of interaction. Some techniques may also be more suited for some types of users or activities. Moreover, an implementation may rely on a combination of more than one of these techniques.

More specifically, considering these types of interaction, we may for example:

- within touch interaction, use single or multi-touch techniques to capture the users’ movements on top of a surface [KKL07, BS02]
- adapt touch technologies to use tangible interfaces that recognise objects to interact with the system [JL06, Fab06]

- use voice recognition that, despite the advantage of only needing a microphone and appropriate software, needs to deal with different languages and accents [Kam95]
- use gesture analysis that, in spite of being a flexible approach, may require space and may be dependent on the surrounding environment.
- or use haptic user interfaces that use hand movement to communicate with the computer [BS02]

Bi-directional feedback can be found in either one of these techniques. Some examples include speech synthesis, visual feedback and haptic feedback.

Each individual type of interaction has its own advantages and disadvantages; however, a combination of multiple types of interaction may also be possible - multi-modal interaction. By using several interaction types at the same time, some errors caused by their individual usage can be eliminated [HW04]. For example, speech, sketching, and gesture almost always provide more reliable when used in combination [AEO$^+$04].

2.2.2 NUI-based devices

There are several types of devices that implement NUIs. Smartphones, tablets and tabletops are examples of devices that use touch detection technology in relation to NUIs. Smartphones and tablets are mobile devices that have been available for some time and are gaining popularity. Examples of tabletop implementations are the reacTable [KJGA06] and the u-Table [JL06].

Game consoles have recently been implementing gesture-based approaches of NUIs. Examples of such are the WIIMote for the Nintendo Wii, the PS Move for Sony Playstation 3 and Microsoft's Kinect for XBox360.

One example of the usage of voice recognition software is present in voice dial-on mobile devices and some GPS locating systems where the user can interact with the system by using their words.

The PHANTOM Haptic Interface [MS94] is an example of a haptic device. It measures a user's finger tip position and exerts a precisely controlled force vector on the finger tip. The device enables users to feel and interact with a wide variety of virtual objects and can be used to control remote manipulators.

2.2.3 Choice for device

Given the multiple types of devices capable of implementing a NUI, tabletops were selected for this project due to their shape and concept, size and potential regarding older adults as the final users. Their size will allow older adults with trouble reading to better understand applications. Tabletops may also have more processing power when compared to the tablet or a mobile phone, increasing their potential and overall capabilities.

Although their size and shape may lead to portability issues, that same property brings one obvious advantage of tabletops, and that is the ability to blend with some environments due to its shape - the table - already being familiar to people.

2.3 Tabletop technology

Paradigms regarding user interaction with computers exist, but they represent only part of the study involving application development. How and why this interaction is done and what mechanisms are there to process and ease the interaction are both relevant factors to that matter; thus, they must be studied and researched as well.

In this section, the hardware alternatives that can be used in this project are depicted. Firstly, the tabletop and its architectural layers are described. Following, the relevant issues that differentiate detection techniques are presented along with some existing touch detection techniques for the tabletop. A study of protocols to transmit touch information and programming languages is carried out, and finally, a study of which frameworks can be used to implement the game prototype is presented.

2.3.1 Tabletop architecture

A tabletop consists of a table whose surface is a screen which users can touch to interact with the system. The tabletop was designed to be a station where several users can work without interfering with each other, but still collaborate if they wish to. It can therefore be used for collaborative work or collaborative games.

There are several methods of extracting information about finger movement to be used on NUIs. One of them, capacitive-based techniques, was primarily developed for single touch interaction and is very suitable for many kinds of touch displays. However, this technique is relatively expensive to produce and, when applied on large multi-touch surfaces, the number of simultaneous touches is tipically limited by firmware or by the design of the controller [SBD$^+$08b].

Keeping in mind the goal of this thesis - collaborative games - more common techniques, specifically designed for multi-touch surfaces, will be the object of study. They all make use of IR light sources and IR cameras so they do not to interfere with the projected image.

In terms of hardware, the tabletop can be subdivided in an input device, a processing unit, and an output device. Considering the fact that NUI hardware does not require peripherals such as the keyboard or the mouse, the input and the output devices are usually connected.

A possible implementation, represented in Figure 2.1, would be: a projector displays the image below the surface - which can be seen by the user - and a camera is pointed at

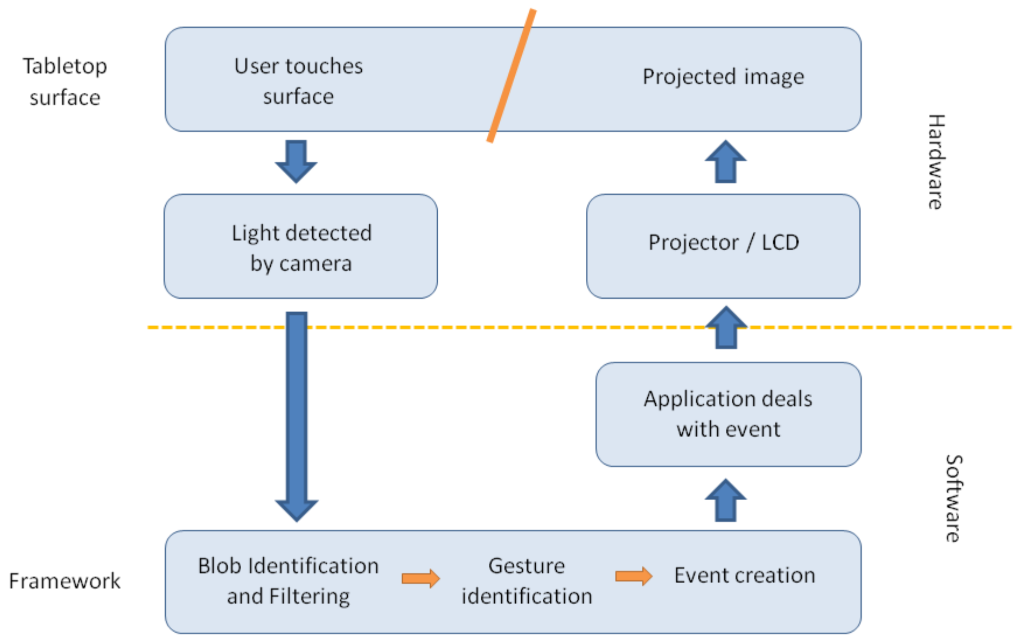


Figure 2.1: Tabletop architecture

the surface from inside the device so it can see the users reaching and touching the surface with their fingers.

There are several techniques for touch detection which are described in the next section. The processing unit is usually a computer running appropriate software. That software will use an appropriate framework that receives and processes the images captured by the camera to detect touch information and convert it into events that can be used by applications.

Section 2.3.3 describes the touch detection techniques, commenting on their advantages and disadvantages while section 2.3.5 describes the processing frameworks that are commonly used.

2.3.2 Relevant comparison factors

Some factors regarding the way a tabletop system is built may weigh more than others in the decision of which techniques may benefit the end-user. Upon analysis and among all the characteristics that define and come from motion detection techniques, the following were seen as more relevant to be discussed in this project.

2.3.2.1 Light direction

The light direction of the technique may require changes depending on the environment that the tabletop is inserted in. Light - in these cases IR light, merged with the visible spectrum - will need to be abundant in the environment so the device can establish a clear detection. This may pose a problem when light in the environment is not constant and/or is not enough to establish a clear detection. A solution to this problem may include adding artificial light sources to the environment.

2.3.2.2 IR light usage

The fact that IR light is used in these techniques has its advantages and disadvantages to the user and to the detection process. The visible spectrum can be used to project the image onto the surface and the use of IR light will not interfere with the image as it is not detected by the human eye. This approach requires some careful analysis to make sure that the IR light that reaches the camera was emitted from the correct IR light source. To solve this problem, filters can be used to block the light.

One advantage to the user is that they will not be able to see the light and therefore the interaction would be transparent. However, the use of IR light implies careful construction of the device itself as the IR rays may damage the human eye if in high concentrations.

2.3.2.3 Type of detection

A touch detection technique can detect motion on the surface by using the reflected IR light to detect the finger (light reflected) or by using its absence (light occluded). The first is based on the presence of light on the spot where the finger is touching, while the later requires abundant and constant IR light to be reflected on or emitted through the surface. In light reflection, the environment must not introduce extra IR light so not to cause a faulty detection and, in light occluded systems, a touch is detected when there is absence of light on a spot.

Either technique may detect the difference in illumination on the region of the finger by comparison to a clear image of the surface or by detecting it solely with the captured frame by analysing discontinuities. By using the method of comparison between two images, the device requires previous calibration so that an image of the surface without touch is stored for comparison. In addition, it also requires an environment that presents no change in the illumination, if light direction is outside in, or it will result in a poor detection or even a wrong one.

2.3.2.4 Light source exposure

Whether the light source can be seen by the user or not matters only to the users themselves. Precautions regarding any potential risks to their health due to IR light exposure need to be considered.

2.3.2.5 Diffused Illumination usage

Using diffused illumination requires an equal distribution of the IR light through the surface. Relying on this technique causes more pressure to be needed, in order for detection to occur, due to the abundance of IR light reflections that result in less noticeable difference in illumination. However, it may eliminate the risk of detecting stray IR rays. In addition, detection on some regions may be harder than others due to the difference in lighting not being substantial.

2.3.2.6 Tangible objects' support

Tangible objects are physical objects that can be recognised and used to interact with a NUI. In a general way, they may link a physical object to digital media contents [OW04], allowing users to access media content by using an object they can grab. The support of tangible objects in touch-based NUIs may be a relevant feature for comparison due to the fact that tangible objects may provide the user with a richer experience. They may also enable or disable features, by letting the software know the user through "proof by possession". Shape and marker detection are common ways to represent tangible objects.

- Shape detection - A technique can detect an object on the surface by its shape - triangles, rectangles or pentagons for example - however, depending on the technique and how it reflects IR light, the object might have to create the same pressure as a finger would in order to be detected [SCG10].
- Unique marker detection - To be detected, some objects may have a unique marker so that the camera picks up the marker and identifies it as that particular object. Some black and white markers may reflect IR light as well, so no special features to the camera need to be added. In order for this to happen, the IR light emitted needs to reach the marker. Again, this may vary depending on how and where the light is emitted [KB99].

These characteristics were analysed for each studied technique and, after a description of each technique in the next section, a comparison based on these factors will be made.

2.3.3 Touch detection techniques

As mentioned earlier in section 2.3.1, only projection-based techniques such as FTIR, Front/Rear DI, LLP and DSI, were the object of our study. The first step in detecting finger movement on a tabletop that uses these characteristics consists of capturing images with a camera situated directly below the surface, inside the tabletop. Where the light source is located may vary between techniques as they can use the presence or absence of

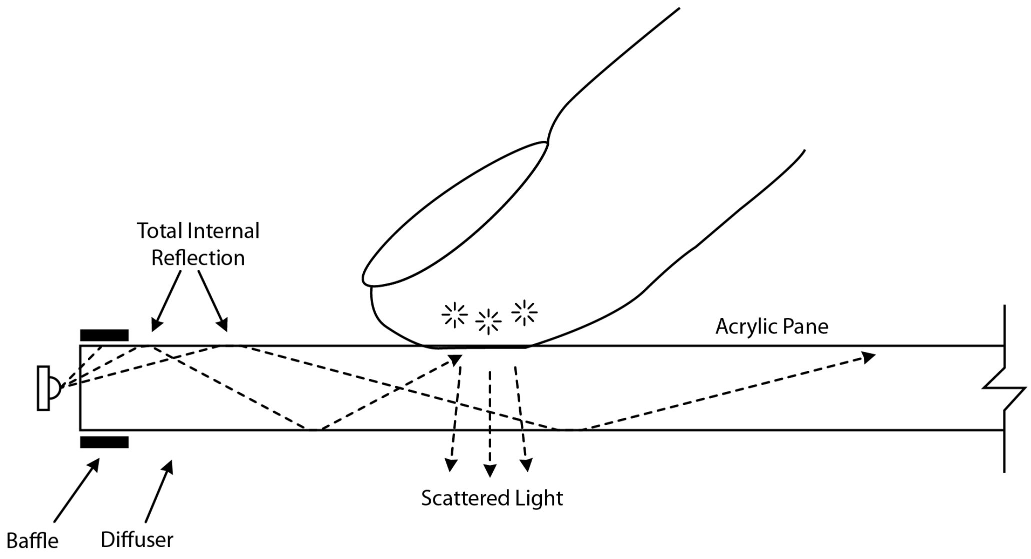


Figure 2.2: Explanation of FTIR touch detection [Han05]

light to detect where and if a finger is touching the surface - this finger area is known as a blob.

In the following subsections, the FTIR and Front/Rear DI techniques are presented in more detail for being the most common for home-made and commercial products, like Microsoft Surface [Cor11]. Other techniques such as LLP and DSI are also explained by comparison with the two previous techniques.

2.3.3.1 Frustrated Total Internal Reflection (FTIR)

This technique was used and made popular in 2005 by Jeff Han and is very well-known and commonly used in NUIs. It relies on the principle of Total Internal Reflection which states that when light enters one material from another material with a higher refractive index, at an angle of incidence greater than a specific angle, no refraction occurs in the material and the light beam is totally reflected [Gro09, wik11d]. As a consequence, the light beam floods the surface's material.

When the user comes into contact with the surface of the tabletop, the light rays in that area are redirected downwards, breaking the "perfect" angle they had before - they are said to be frustrated. That light is not reflected inside the material but transmitted outside, directly below the finger that is touching the surface, evidencing any finger touch wherever it may be. If installed correctly, no finger occlusion occurs due to hardware limitations. The camera installed bellow picks up the IR light and identifies a blob in the image [Gro09]. That blob represents the finger touching the tabletop surface. This description is shown graphically in Figure 2.2.

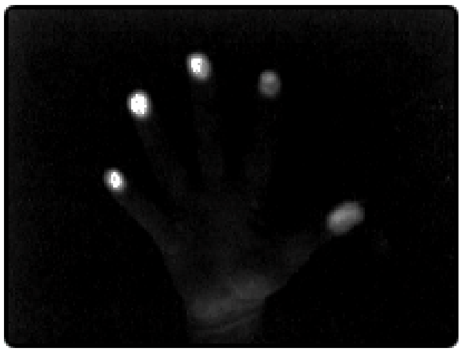


Figure 2.3: Example of FTIR image [Han05]

The composition of the surface is very important in this technique. The material used, according to Han, needs to be acrylic because of its properties, that allow IR light to go through it, and because it is relatively cheap. Special care needs to be taken in relation to the edges of the acrylic: in order for light to be transmitted correctly, it needs to be correctly polished.

The baffle and the diffuser are responsible for making sure no IR light can escape between the light source and the acrylic and eliminating any "noise" that other objects may cause by hovering the surface, respectively [Gro09]. Due to the image that it produces, exemplified in Figure 2.3, the tracking in this technique is directly dependent on the amount of frames per second the camera captures [Han05]. FTIR is familiar to the biometrics community where it is used for fingerprint image acquisition. It acquires true touch information at high spatial and temporal resolutions, and is scalable to very large installations [Han05].

Despite being a widely accepted and easy to implement technique, FTIR presents the disadvantage of dealing poorly with ambient light. Because it relies on IR being detected, it may falsely detect positive interaction when leaks occur - outside IR light enters the device - or false negatives if the contrast is reduced.

2.3.3.2 Diffused Illumination (Rear/Front DI)

The technique of DI comes in two main forms: Front DI and Rear DI. Both techniques rely on the same principle, which is detecting touch on the surface based on a constant beam of IR light illuminating the surface. The detection process may be done by either absence or presence of light on a spot [Gro09]. The difference lies on the location of the IR light source and its consequences on the obtained image.

Front DI is a simpler but more fallible method than Rear DI as it relies on exterior illumination to create a uniform IR distribution along the touch surface. A diffuser is placed at the bottom or at the top of the surface so that when a finger touches the surface [wik11a], a shadow is created due to the blockage of IR light and the camera picks up the difference in

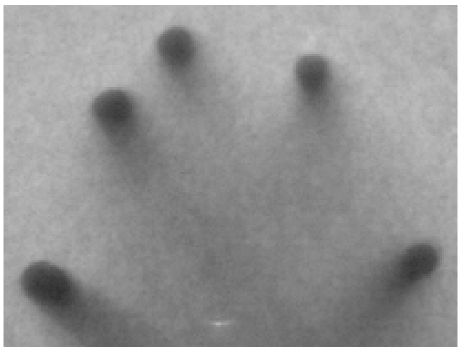


Figure 2.4: Example of Front DI image [wik11a]

illumination, interpreting it as a blob, similar to the FTIR technique [Gro09]. An example of an image with finger interaction, captured by the IR camera is shown on Figure 2.4.

Rear DI however does not rely on outside illumination. It consists of having IR light emitted from inside the hardware rather than the outside [wik11a] as shown in Figure 2.5. In this case, the diffuser now diffuses the exterior light that reaches the surface. Figure 2.6 shows the result the camera picks up.

Depending on the size and configuration of the table or of the environment lighting, it can be quite challenging to achieve a uniform distribution of IR light across the surface for Rear or Front DI setups. While certain areas are well lit and touches are easily detected, other areas appear darker, thus requiring the user to press harder in order for a touch to be detected [Gro09].

Rear DI and Front DI have the disadvantage of being highly sensitive to variations in external lighting, which may cause problems in blob identification. As a consequence, there is the need for a calibration once the tabletop has been moved from an environment to another or if the environment's illumination itself changes. However, provided that this flaw is corrected or minimised by blob analysis on different frames, the advantage of these techniques relies on the fact that stray rays are more easily blurred and mixed with the background.

2.3.3.3 Other techniques

Less commonly used techniques like the LLP illumination and the DSI are described in this section as alternatives to the ones described previously.

The LLP illumination is a technique in which lasers are positioned to intersect the object that touches the surface, as illustrated in Figure 2.7. When the finger touches it, it will hit the tip of the finger which will register as an IR blob [Gro09]. Infrared lasers used to achieve the LLP effect carry inherent risks, possessing risk factors with regard to eye damage [Gro09]. This technique, if used with a low number of IR light sources, may result in occlusion [Gro09].

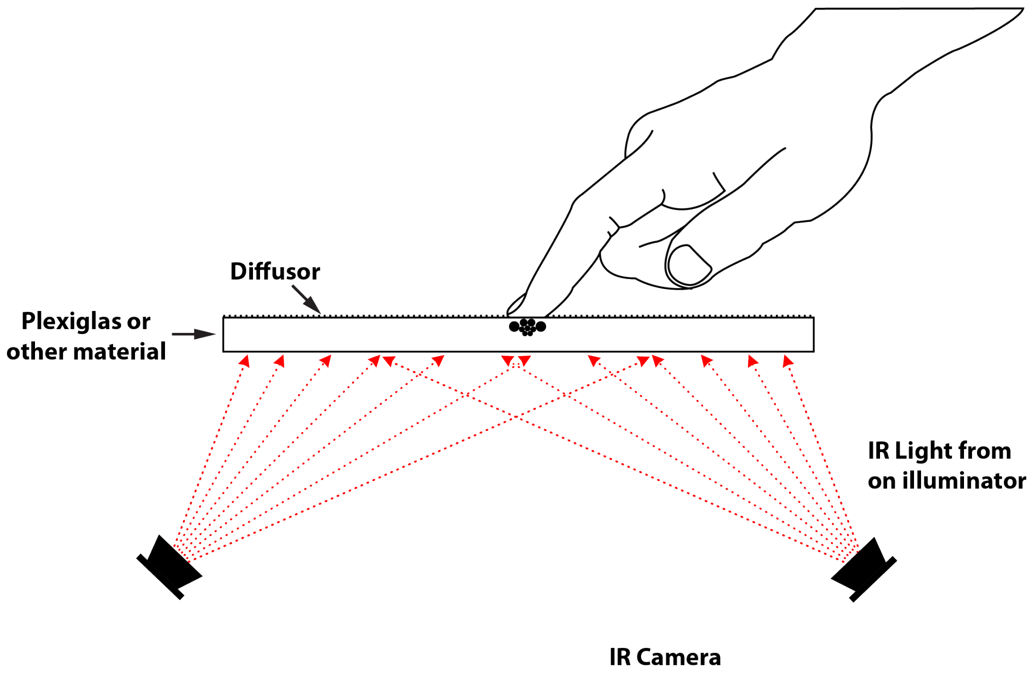


Figure 2.5: Explanation of Rear DI touch detection [wik11a]

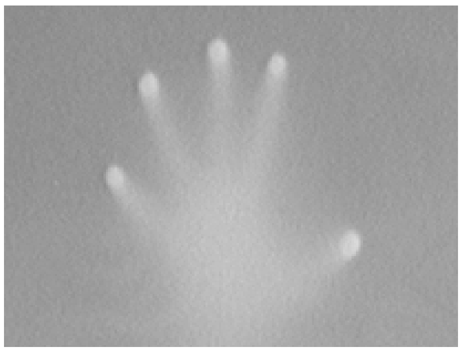


Figure 2.6: Example of Rear DI image [wik11a]

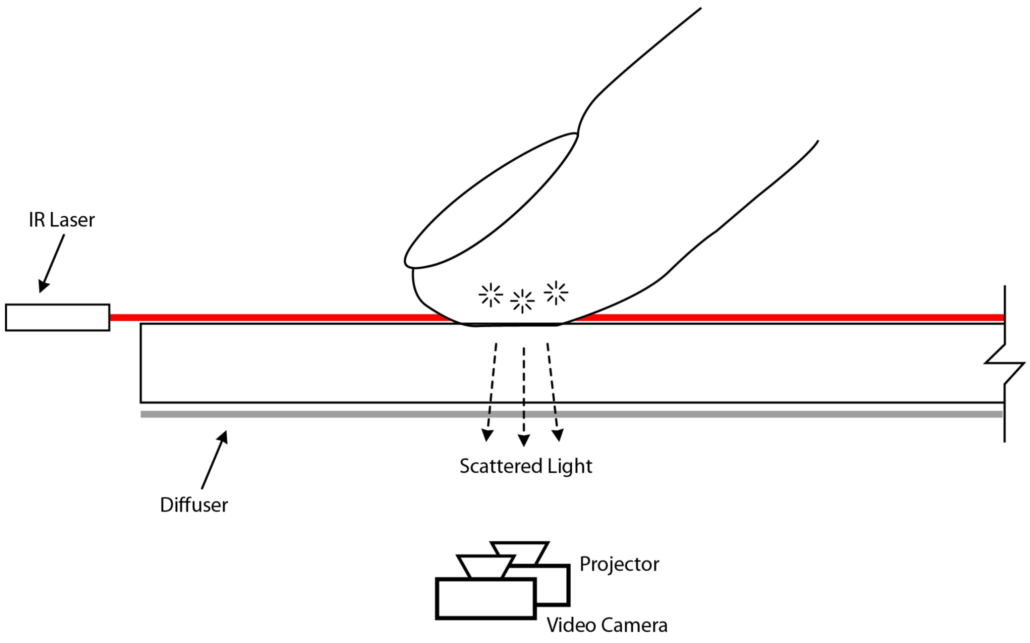


Figure 2.7: Explanation of LLP illumination touch detection [wik11e]

On the other hand, the DSI is a technique that uses concepts from both the FTIR and DI techniques. Instead of using carefully placed illuminators as a IR Source like in DI, DSI, represented in Figure 2.8, uses a special and more expensive acrylic to distribute the IR light evenly across the surface; however, the position of the IR LEDs is the same as in the FTIR technique. Light gets reflected both upwards and downwards [Gro09]. This approach may cause problems due to sending the IR light to the camera as well, resulting in less contrast when detecting a touch [wik11c].

2.3.3.4 Comparison

Table 2.1 presents some characteristics of each technique that are relevant for the study at hand.

Table 2.1: Comparison between touch detection techniques

Characteristics	FTIR	Front DI	Rear DI	LLP	DSI
IR light	Yes	Yes	Yes	Yes	Yes
Light direction	Inside Out	Outside In	Inside Out	Outside In	Inside Out
Light detected	Reflected	Occluded	Reflected	Reflected	Reflected
Light exposed	No	Yes	Yes	Yes	Yes
Diffused Illumination	No	Yes	Yes	No	Yes
Shape detection	No	Yes	Yes	No	Yes
Marker detection	No	No	Yes	No	Yes

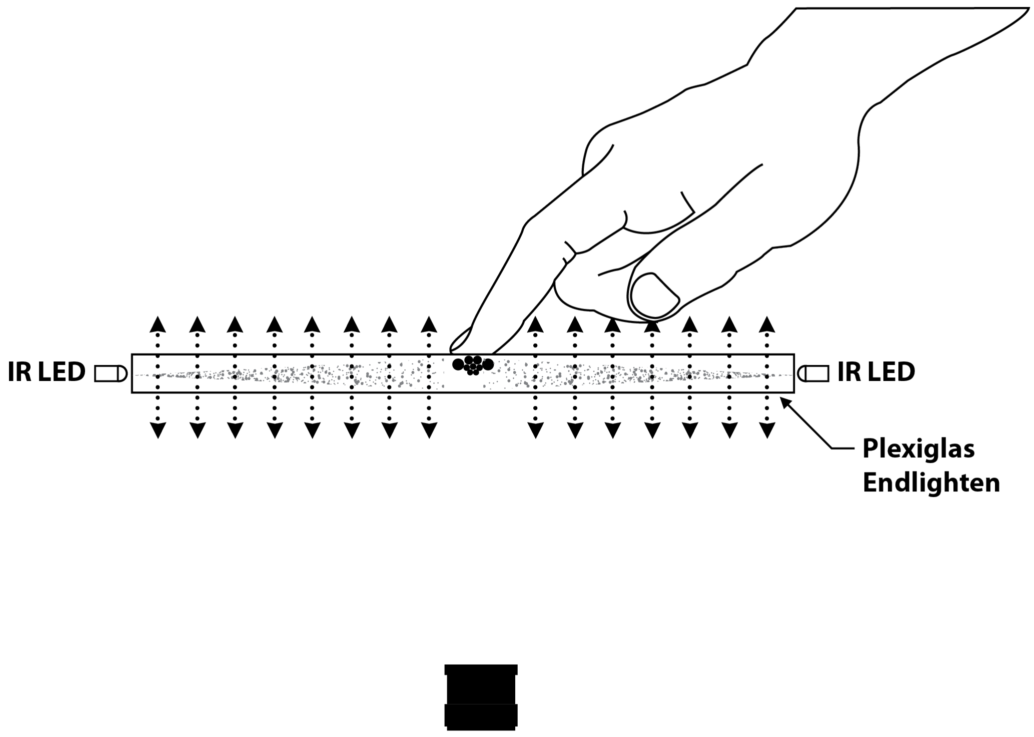


Figure 2.8: Explanation of DSI touch detection [wik11b]

Taking all the advantages and disadvantages into consideration, it is clear why FTIR is the most commonly used technique for quick setups. Assuming it is well built, this technique is the less hazardous to the user because it restricts the use of IR light to the absolute necessary and does not emit IR beams outside the tabletop. Also, it does not require specific environment lighting.

The Rear DI technique may also be considered one that presents some advantages, provided that the light is distributed evenly across the acrylic. This technique may have the upper hand when dealing with environments that, for example, provide constant external illumination, which diminishes the technique's disadvantages.

2.3.4 Touch information protocols

Touch information needs to be sent from the hardware to the processing unit running the application. This will pass on the information relative to the user's action so the application can react accordingly. Information regarding the kind of touch, the shape, the diameter, the centre point, the time elapsed since it was detected, the perimeter or whether it is a marker or a human touch is only part examples that may be relevant when building an application for a tabletop with multi-touch.

There are a number of protocols that transport many of the aforementioned information. Some protocols like TUIO [Com11g] or ones that rely on XML [BPSM$^+$03], are OS-independent protocols that offer great extensibility. Other protocols like Windows Touch and Apple's Touch Protocol are proprietary implementations that require the developer to consider the target OS and are not easily extensible. These protocols are reviewed in the following sections.

2.3.4.1 TUIO

TUIO [Com11g] is a simple yet versatile protocol designed specifically to meet the requirements of tabletop tangible user interfaces. This protocol defines common properties of controller objects on the table surface as well as of finger and hand gestures performed by the user. It relies on a compact binary transport enabled by the OSC protocol [WFM03] and is therefore usable on any platform as it is transport-independent.

This protocol is easily extensible as it defines that objects be sent through a network port, typically port 3333. The only requirements for adding new object information are a device that sends such information and a listener on the application that deals with the object. Its specification supports information regarding fingers and markers. In special cases where the hardware allows it, the TUIO protocol is also able to send out information regarding the 3D position.

2.3.4.2 XML-based messages

Alternate implementations for a touch information protocol are protocols relying on XML messages. These protocols, at the cost of some additional bandwidth requirement and processing overhead - by comparison to the TUIO protocol - have the advantage of being read quite easily by any XML parser. The following is an example of several XML messages sent over a port, containing samples that represent touch information.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<sample>
</sample>

<sample>
</sample>

<sample>
  <finger>
    <id>2</id>
    <age>1</age>
    <loc><x>246.54</x><y>261.02</y></loc>
  </finger>
  <hand>
    <id>1</id>
    <age>1</age>
    <palm><x>320.93</x><y>339.45</y></palm>
    <fingerid>2</fingerid>
  </hand>
</sample>

<sample>
  <finger>
    <id>2</id>
    <age>2</age>
    <loc><x>246.82</x><y>260.60</y></loc>
  </finger>
  <hand>
    <id>1</id>
    <age>2</age>
    <palm><x>331.22</x><y>341.42</y></palm>
    <fingerid>2</fingerid>
  </hand>
</sample>
```

This protocol is of easy customisation and extension as shown in the code snippet presented above. Each sample may contain any information that the developer wants, provided that the device obtaining the information collects and sends out the touch information in the correct format and that the application possesses a parser that processes the extended information.

2.3.4.3 OS and device-dependent protocols

The use of widely used OS-independent protocols is not a requirement when developing an application for a multi-touch device. Using OS-specific protocols like Windows Touch, Windows Pen, Apple's Touch Protocol and Multi-Pointer X may allow applications to receive information quicker. Critical applications may benefit from this advantage. Furthermore, these protocols are often very hard or even impossible to extend due to their embedded status and proprietary nature.

There may be however device-specific protocols. These protocols may be independent on any abstraction layer or not, depending on how they are implemented. Typically, they are closed-source protocols used by commercial products with a specific API and SDK, available for a couple of programming languages.

2.3.4.4 Comparison

Considering all the advantages and disadvantages of all types of protocols mentioned in this section, we have concluded that the needs of this project would only be fully met by taking the final application's extensibility and OS-independence of each protocol into account. Having said that, among the protocols studied and presented, only the TUIO and XML-based protocols meet these demands.

The TUIO protocol was then chosen over the XML-based protocols for bringing more efficiency to the information transfer while still offering extensibility at an easy scale.

2.3.5 Frameworks for tabletops

After having described the hardware techniques associated with image capturing and selected a protocol to work with, this section now provides the reader with a brief explanation of some frameworks that can be used to process the images and obtain gesture-based events. Having chosen the TUIO protocol as means of communication between the final application and the device, we are going to focus on frameworks that already have support for the said protocol.

The framework should be able to interpret the information protocol chosen and provide an API abstract enough so that the development may focus on building the mechanics of the application rather than managing low-level activities of less relevance to this thesis. With this in mind, and in coherence to the previous section, common denominators for comparing the frameworks studied are introduced. Afterwards, frameworks with native TUIO implementations are presented and compared with each other.

2.3.5.1 Relevant comparison factors

Taking into account the end-user and the goal of this thesis, some comparison factors were selected from a wide range [DKW10] to better understand which framework would be more appropriate for the development of applications. Such were the criteria:

- Cross-platform - Frameworks that are able to work on different OS's are more suitable for development as this thesis aims to conduct research for future development of OS and device-independent applications. This may allow future developers to regard this document as a starting point for their own projects, regardless of what OS they choose to implement their application in.

- Gesture manager - Some platforms use a gesture library embedded on the framework to recognise gestures; others use gesture servers that can be by several client applications to query for information regarding gestures. Using a gesture library will allow the application to run on its own. On the other hand, frameworks that use a gesture server are conditioned to a running server application and may have latency issues if running on a separate machine; however, this method may allow an easy update of gestures on several devices at once.

Table 2.2: Programming languages' characteristics

Characteristics	C	C++	Java	Flash	Python
Object-Oriented	No	Yes	Yes	Yes	Yes
Programming level	Low	Low/High	Low/High	High	High
Design-Oriented	No	No	No	Yes	No
OS-independent	No	No	Yes	Yes	Yes
Scripting language	No	No	No	using AS3	Yes
Efficiency	High	Medium	Medium	Low	Medium

- Programming language - Different programming languages may offer different advantages to an application and to the end-user. Choosing the right programming language to suit the needs of the application may be crucial to its development. Programming languages like C, C++, Java, Python or Flash are the most common for offline applications' development. Table 2.2 represents a comparison of the programming languages.
- Custom gestures - Frameworks may have a gesture database where most common gestures are already implemented. However, the need to implement custom gestures may arise. Custom gestures can be implemented by inheritance (using a sub-class) or by access to raw touch data.
- Parameterization - The access to specific parameters may be of some relevance from the developers' point of view. Access to touch-related parameters or more general gesture-related parameters can enable fine-tuning and optimizations.
- Free - Paid software usually has the advantage of being more robust and of possessing more functionality and support. Free software usually allows developers to modify source code as they wish; however, support may not be as complete.

After setting a baseline for comparison between frameworks, some will be presented and described in the next subsection. In subsection 2.3.5.6, these factors will be used for comparing the frameworks.

2.3.5.2 PyMT

PyMT is an open-source library for developing multi-touch applications. It is based on Python, cross-platform, and comes with native support for many multi-touch input devices, a growing library of multi-touch-aware widgets, hardware-accelerated OpenGL drawing, and an architecture that is designed to allow developers to focus on building custom and highly interactive applications as swift and effortlessly as possible [Com11c]. Currently the aim is to allow for quick and easy interaction design and rapid prototype development [Com11d].

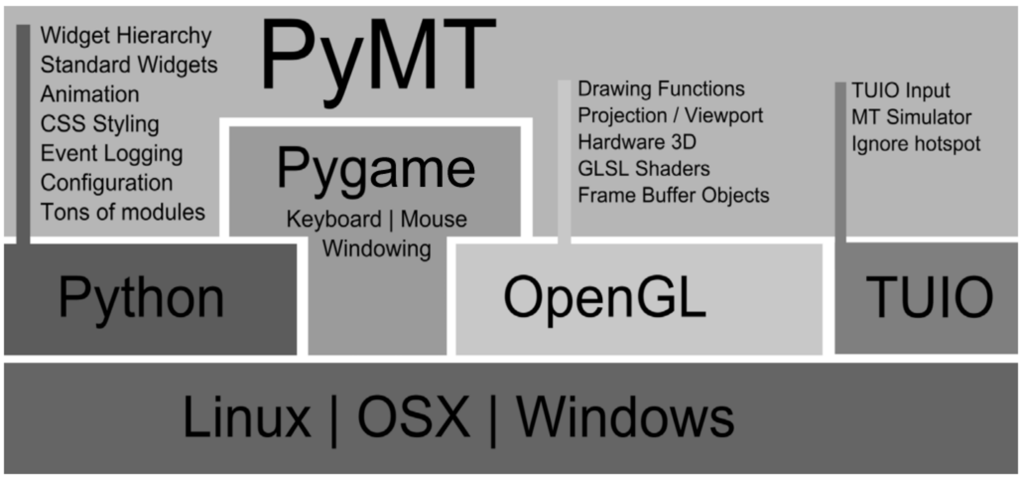


Figure 2.9: Abstraction layers of the PyMT framework [Gro09]

PyMT tries to make dealing with movement detection and graphics output as simple and flexible as possible. For dealing with input, PyMT wraps the TUIO protocol [KBBC05], as well as other protocols, into an event driven widget framework. For graphical output PyMT builds on OpenGL to allow for hardware-accelerated graphics [Gro09]. A structure of PyMT's architecture is shown in Figure 2.9.

Pygame, a cross-platform OpenGL windowing and multimedia library for Python used by PyMT, and its multimedia functionality is prepared to deal with images, audio and video files [Com11b]. PyMT uses similar concepts as other GUI toolkits and provides an array of widgets for use in multi-touch applications as part of the framework. However, the main focus lies in letting the programmer easily implement custom widgets and experiment with novel interaction techniques, rather than providing a stock of standard widgets [Gro09].

- Kivy - a new framework for application development based on the PyMT framework and it is being developed by core developers of PyMT. The Kivy platform is business friendly (licensed under the LGPL 3 license), available for Windows, Mac OS X, Linux & Android, able to run the same code on all supported platforms, compatible with most input protocols / devices natively (such as TUIO, Windows Touch, Windows Pen, Mac OS X multi-touch peripherals - Trackpad, Magic Mouse - Linux Wacom, Kernel HID Input), implemented in Python with some code written in the C language for performance and is easy to extend [MV11]. Unlike PyMT, Kivy has its foundation on an event-based system. It is hardware-accelerated through OpenGL ES 2.0 and the Kivy language enables dynamic class building.

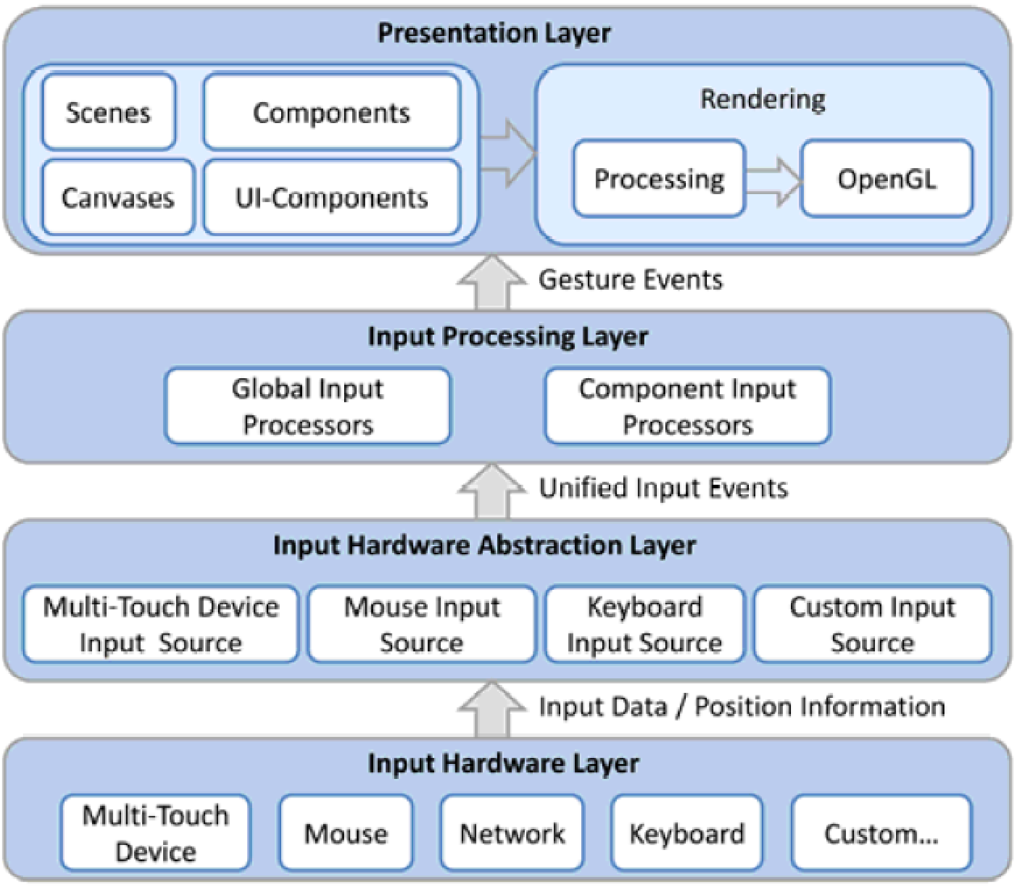


Figure 2.10: Abstraction layers of the MT4j framework [LRZ10]

2.3.5.3 MT4j

MT4j is an open source Java development platform that has been developed for creating extensive graphics applications. Since this framework runs on Java, it is cross-platform, having been reported to run on Windows 7, XP, Vista, Ubuntu Linux and Mac OSX [Com11a]. MT4j has been designed to support different types of input devices with a special focus on multi-touch support. It was created for rapid development of graphically rich 2D or 3D applications and has the most common multi-touch gestures already embedded [Mat09].

The functionality of the MT4j framework architecture is subdivided into different layers communicating through events sent from one layer to the next, as illustrated by figure 2.10. The emphasis on input layers represents the importance of a flexible input architecture, while performance issues are mainly addressed by the presentation layer. By using a hardware abstraction layer, MT4j can support various input hardware with only minimal adjustments in the input hardware abstraction layer [LRZ10].

MT4j comes with a set of implemented input providers including mouse, keyboard and

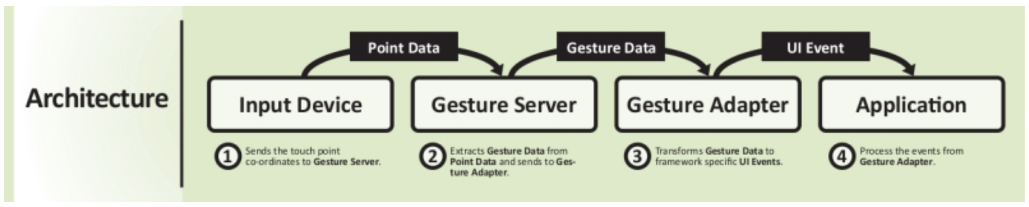

Architecture
Point Data
Gesture Data
UI Event
Input Device
Gesture Server
Gesture Adapter
Application
1 Sends the touch point co-ordinates to Gesture Server.
2 Extracts Gesture Data from Point Data and sends to Gesture Adapter.
3 Transforms Gesture Data to framework specific UI Events.
4 Process the events from Gesture Adapter.

Figure 2.11: Architecture of Sparsh UI [Com11e]

multi-touch input protocols such as the TUIO-protocol. Additionally, MT4j supports the use of multiple mice input on Microsoft Windows and Linux platforms, which facilitates testing of multi-touch functionality [LRZ10].

2.3.5.4 Sparsh UI

Sparsh UI is a multi-touch API that enables users to easily create multi-touch applications on a variety of hardware platforms. The API supports custom hardware drivers and is platform independent [Com11e], enabling applications to be developed in the desired programming language [PRV09]. It consists on a server-sided gesture recognition that handles gesture processing and passes touch-points and/or gesture information to the client application. It also supports basic gestures such as drag, scale, and rotate, and is extensible to support an infinite number of custom gestures. A specific gesture adapter for the aforementioned gesture recognition server is needed for every application framework [Com11e]. The architecture of Sparsh UI is exemplified in figure 2.11.

2.3.5.5 GestureWorks

GestureWorks [Ide11] is a paid framework developed with Flash/AS3 that implements the TUIO protocol. Its library implements the basic TUIO callback API and supports sending TUIO information through both FLC and TCP transport methods. The library also provides a legacy API in order to support existing Flash examples that have been based on the original Touchlib API [Wal06]. Currently, UDP is only recently supported in Flash 10.1 so the implementation of the TUIO/UDP transport method is still in its early stages. The advantage of this framework is that it enables flash to perform a faster connection resulting in smoother touch recognition [Com11f].

2.3.5.6 Comparison

In this section, a comparison according to the aforementioned features is made. Table 2.1 represents said assessment.

An analysis of this table and of the advantages and disadvantages described in the previous subsection reveals that a balance between performance and functionality is achieved

Table 2.3: Comparison between frameworks

Characteristics	PyMT/Kivy	MT4j	Sparsh UI	GestureWorks
Cross-platform	Yes	Yes	Yes	Yes
Gesture manager	library	library	server	library
Programming language	Python	Java	C / C++ / Java	Flash/AS3
Custom gestures	raw touch data	sub-class	sub-class	sub-class
Parameterization	touch only	both	gesture only	gesture only
Free	Yes	Yes	Yes	No

with the Kivy framework, mainly due to its features and extensibility. This framework was chosen to be the used in this project, along with the Python language.

2.3.6 Related work and implementations

Implementations prior to this thesis were made using the techniques and frameworks described in the previous section. The majority use either a custom built tabletop using the FTIR technique or Microsoft Surface to obtain images from the surface [AAG$^+$09, ThT09, FBG$^+$09]. In this section, only projects that refer to which technique and framework is used in the implementation are described.

The Air Touch implementation [AAG$^+$09] is based on the FTIR technique and uses Flex [O'R04] to process hand movement. This project combines existing multi-touch technologies with a suite of new rehabilitation-centric applications to address the mobility issues faced by older adults.

SharePic [AKQ06] is a multiuser, multi-touch, gestural, collaborative digital photograph sharing application for a tabletop. The application incorporated a series of functions to understand how users would work with them and whether they were simple enough to learn. In this project, both young and older adults were used as test subjects. The authors were successful in creating design guidelines and core elements that is learnable and usable by both young and older users.

The Infotouch [KAB$^+$08] project was partly based on the SharePic project. It aimed at a design exploration into how a large multi-touch tabletop display can be used for information visualization. It was concluded that participants generally liked the concept of using multi-touch to casually browse through photo collections. However, participants deemed the multi-touch capability as a complementary feature that was used approximately 10% of the time.

JunctionBox [LF11] is a software toolkit for creating multi-touch interfaces for controlling sound and music based on MT4j. Specially, the toolkit has special features which make it easy to create TUIO-based touch interfaces for controlling sound engines via OSC. Developers using the toolkit have a great deal of freedom to create highly customised interfaces that work on a variety of hardware. The JunctionBox toolkit uses existing libraries for touch tracking. It is able to easily map multi-touch actions to sound and music control messages.

MTVis [AT10] is a multi-touch interactive tree visualization system. Its goal is to display the contents of a file structure by representing their hierarchy according to circles. This project uses a variation of the FTIR technique and a multi-touch tracker and framework that only runs on MacOS.

The TACTUS [VLN09] implementation aims to enable research in multi-touch interaction. It offers insight into the construction of a robust, low-cost multi-touch surface and the development of an extensible software system for the rapid creation of multi-touch applications. It uses the FTIR technology for being a low-cost and viable solution. The only requirement is a platform-specific framework from one of the following: Windows Presentation Foundation (WPF), WinForms or Microsoft XNA.

Other projects based on commercial products are also being studied. Surface [Cor11] is a tabletop implementation developed by Microsoft and widely used on research projects. HERMES [FBG$^+$09] uses Surface to implement a system that aims to reduce or delay the normal cognitive decline that takes place in elderly people. Aside from the type of hardware, frameworks have been chosen to implement and test gestures like copy, drag and select, and test concepts such as the black hole [AKQ06, AA05].

Chapter 3

Methodology

The aim of this thesis, as stated in chapter 1, was to create a framework to enable the development of games to stimulate older adults cognitive and motor-wise. In order to do that, and after gaining knowledge of the technologies available and having chosen which framework and transfer protocol to use, an approach for the development process was devised. It was necessary to understand how games for multi-touch surfaces could be developed and how games could be created or adapted to older adults specifically.

Ideally in software engineering, developers have at their disposal the full specifications of the final application so that, when designing and developing it, no surprises emerge and no patches are needed. However, it is rarely the case. To avoid this scenario, a methodology was followed and will be presented here. It was decided that knowledge of the device characteristics and limitations and of the framework API would be essential for a correct development of the final product and to make sure the requirements of the user could be met. Some design considerations should also be taken into account, thus prototypes were made to study the users in more detail. These two steps were done in parallel as these studies do not influence one another. The implementation and evaluation process then followed, taking into account all the information gathered from the previous steps.

The order in which these tests were performed was chosen as a “low-level to high-level” approach in order to follow the flow of the information and make sure it was being handled correctly. This methodology allowed the sustained development of prototypes, whose objective was to improve the outcome of this project by making the development more effective and help validate the final application. The two stages of this development process, represented graphically in figure 3.1, are the following:

- the exploration and prototyping stage - where tests for the various layers of the product were made, along with prototypes to be tested with the final user

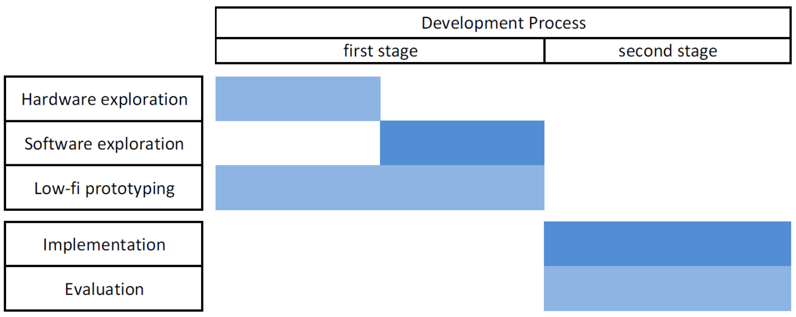


Figure 3.1: Representation of the development stage

- the final application development stage - where an iterative approach was used balancing the implementation and evaluation processes.

The first stage, the testing and prototyping stage, consisted of performing hardware and software tests in order to gain better knowledge of their characteristics, capabilities and limitations along with better understanding of the frameworks' API. At the same time, a number of prototypes were made, implementing selected functionalities that should be, in the future, part of the framework and of the game. While in the second stage of the development process, the framework concept was created. The application's development and evaluation process were also carried out in order to iterate and validate the concepts, gameplay and design the final application. The implementation process was performed by designing the architecture on a first stage and then implementing an actual game on a second stage. This approach was taken due to, in software engineering, avoiding part of the restructuring and rewriting needed when implementation comes prior to the architectural design.

3.1 Hardware tests

To be able to perform accurate hardware tests, the knowledge of the device's characteristics and the limitations that they imply should be gathered. Only then should the system be analysed in terms of its input and output methods.

The study of the input methods that the device is capable of interpreting should have its base on how the tabletop is built, i.e. on what principles the architecture stands. Cameras and light sources were analysed, along with the computer vision capabilities it possesses. Furthermore, the transport methods that the device may use to communicate with the processing unit running the application were also examined and compared in terms of performance and usefulness to the application intended. On the other hand, the study of the device's output system focused on the mechanisms that allow for an image to be seen on the surface.

Through calibration of both the computer vision, associated with collecting input data, and the projection system, associated with the output, an analysis of the tabletop's capabilities and limitations was performed. Adjustments to the calibration were examined so that these limitations may be minimised or, if possible, suppressed. The fact that the users' satisfaction, while using the device, may be affected by how the device is callibrated was also taken into account.

3.2 Software tests

Having decided on the framework for the application development beforehand, software tests were conducted. They were done with the intention of ascertaining how the API handled the input and output data received and sent respectively to the device and of starting to understand the paradigms involved in developing multi-touch applications. Such knowledge would have implications on both the architecture and the iteration process of the final application.

Small applications (prototypes) were created to test the framework's behaviour and the API's robustness and transparency between the classes of the framework. In order to test these small applications the actual device was used, in a controlled environment and by people who had previous knowledge of multi-touch technologies.

3.3 Prototyping process

While performing both hardware and software tests, prototypes were made to evaluate the users' experience with technology - in general and with tabletop systems - and to obtain user characteristics that are relevant to both game design and gameplay. Such prototypes usually fall in one of two categories: low-fidelity or high-fidelity prototypes. Low-fidelity prototypes are tipically made of materials such as paper or cardboard and do not resemble the final product as much as high-fidelity ones [SRP07]. These prototypes allow for more flexibility than high-fidelity prototypes and bring results at a faster rate as well [Ret94]. They are less intimidating to the end-users than a computer, especially considering the target audience of this project, and encourage a more creative feedback because of their "unfinished look" [Sny03].

As such, during this process, low-fidelity prototypes were chosen to support the design process and several were created with different objectives in mind to obtain application requirements that users usually cannot provide on their own.

3.4 Implementation and evaluation process

After obtaining the results from the first stage of the development process, this second stage consisted of developing a game framework and a game prototype, taking into account the information gathered earlier. The implementation and evaluation processes were done iteratively so that when an iteration of the implementation was finished, the evaluation on the same iteration would start. The results from the evaluation would then influence the next iteration.

Firstly, an architecture for the final application was defined to fit the requirements collected. This step resorted to flow diagrams and a class diagram to analyse both the architectural needs of the application and the interaction between main in-game objects. These diagrams were also reviewed in every iteration to guarantee the system's extensibility in the end.

A game was proposed considering the purpose of the game framework created. A prototype of the game was implemented and iteratively evaluated. The evaluation process involved testing each iteration with experienced developers, used to developing for elderly people and accustomed to dealing with their limitations. Due to external constraints, such as portability of the device and users' availability, only one of the final iterations was validated using the final users.

Chapter 4

Analysis and prototyping

After a detailed study in chapter 2 of all subjects involved in this project and along the line described in the previous chapter, an analysis of the hardware, software and their combination is in order before developing the final application. In this chapter, the hardware used, its limitations and calibration methods are explained in detail and in relation to the project at hand. An analysis of software and the combination of both hardware and software is made to detect the existing limitations. Finally, prototypes that have been tested with the end-users are presented and their correlation and contribution to the project are explained.

4.1 Device Specifics

In this section, the device is going to be explored in terms of its particular characteristics, capabilities and limitations. The device calibration and its parameters are explained and the final calibration values are provided.

4.1.1 Device used

The tabletop device used in this thesis is called Cell Advanced [Mul11a] and was manufactured by Multitouch. It is a 46 inch LCD screen that is backlit by a series of LED's. Two IR cameras are placed at a similar distance to detect movement on the surface. The whole system weighs approximately 37Kg. It is stackable with other similar devices and the inclination can be set at will. The device supports Full HD technology and its surface is scratch-resistant. It is a multi-touch multi-user display and can recognise finger tips, fingers, hands and objects. The device is cross-platform as it only requires a computer with USB, FireWire and HDMI ports. Figure 4.1 is picture of the device running a simple drawing application.

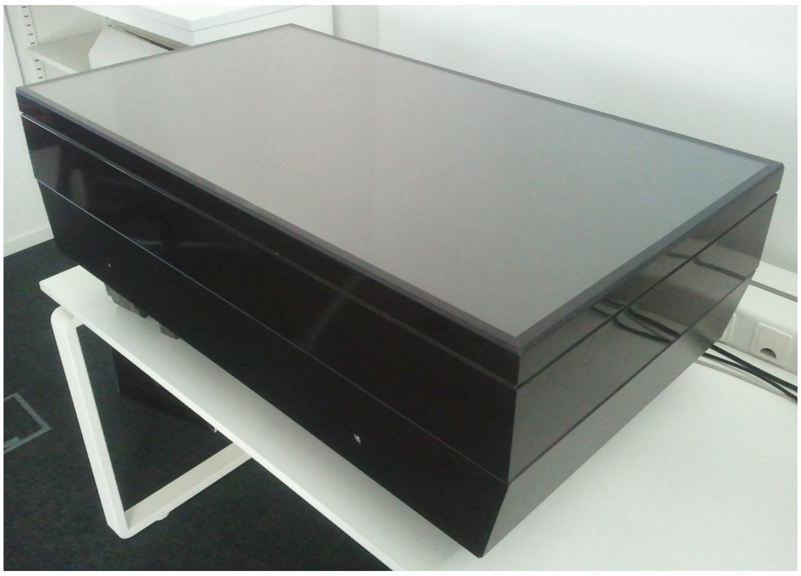


Figure 4.1: Multitouch Cell Advanced system

This device uses the Rear DI detection technique and, because of its size, requires two cameras to capture motion on its surface. The device's middleware interprets the image and is capable of sending touch and marker information through:

- the TUIO protocol
- XML-based messages
- a proprietary transfer protocol

The later requires a specific SDK provided by the manufacturer which is not extensible nor device independent, unlike the first two protocols mentioned.

4.1.2 Hardware constraints

Both cameras are parallel to one another and each one captures half of the surface's reflected IR light. The viewing direction of both cameras makes a 90 degree angle to the surface. The captured - distorted - images can be seen in Figure 4.2. The joint image of the cameras gives the full image of the tabletop's surface and the overlapped areas are used to better track finger motion between cameras. The device's light sources are located around each camera in a square position, as seen in the figure, causing trouble to the computer vision software when detecting motion on different areas of the tabletop - where IR light may be low or the distortion of camera images higher.

The Multitouch Cell Advanced provides computer vision software to track motion. This software is not extensible and is only adjustable through the parameters available.

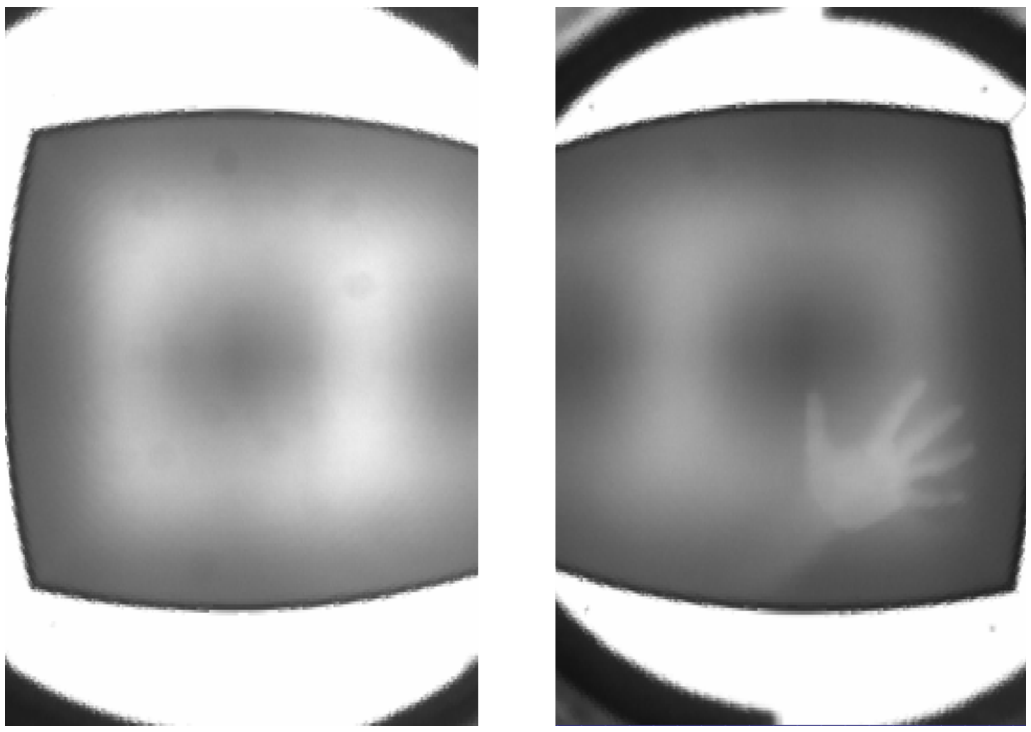


Figure 4.2: Snapshots of the captured images of both cameras

Moreover, regarding hand motion, this device only recognises fingers and palm position locations, so the information sent through the TUIO protocol does not include the shape of the touch, the diameter nor its contour.

In order to detect 2D markers, which can be used to track objects on the screen, the device needs to be told to do so. An extension named “SquareMarkerTracker” needs to be added to the configuration file. Using this extension however will stop the automatic background calibration which regularly captures an image to use as comparison in the touch detection process. This is done to avoid having the marker being burnt into the calibration image, which would result in the marker not being detected through the image’s comparison. In addition, several markers with different sizes were tested. Markers with less than 8 by 8cm were proven to be less effective in the detection process.

4.1.3 Device calibration

Both cameras have separate tracking configurations with separate camera settings, lens corrections and computer vision settings and may require an independent configuration [Mul11b].

The first step for calibrating the device is distorting the camera images and centring them so that they look less rounded. Then, adjustments can be made to the camera and lens using the following parameters:

- Shutter - controls how long is the exposure time for collecting light from the image; the bigger this value, the more light the device captures between frames.
- Gain - amplifier value to control the amount of light of a frame; this may be important for setups that have fewer lighting.
- Brightness and Gamma - these parameters should be modified to adjust the image in case of severe light defects and to minimise the difference between light and dark areas.
- Lens A, B and C - they provide lens distortion corrections to the camera image if the edges are not parallel, by modifying the shape of the surface to be analysed in the computer vision.

After proper camera calibration, the computer vision software can be adjusted through the following parameters:

- Edge Limit - difference between finger tips and the surroundings in pixels.
- Value Limit - absolute minimum value of a blob, that represents a finger tip, should have to be considered a touch.
- Tip sensitivity - how quickly/easily the software reacts to fingers.
- Width CM - the width of the screen in centimetres which tells the software the size of a regular finger in comparison to the size of the image.
- Filter Drag and Filter Radius - they are both smoothing components to eliminate jittering and create smooth interpolations when a finger "jumps" from one position to another, i.e. when motion is not detected between two spots resulting in the two clicks instead of one click followed by a dragging movement.

Finally, some parameters relative to the marker identification needed to be set. The description of each parameter is as follows:

- Threshold - the luminance threshold for starting marker tracking (absolute pixel brightness in the normalized image).
- Max-length - maximum length of the marker edge contour (pixel steps).
- Min-length-rel - the minimum length of the edge contour, relative to the maximum length. For example if the maximum contour length is 210 steps, and minimum is set to 0.5, then the edge contour has to be at least 105 steps long.
- Interval - the contour-walk interval when doing corner detection.

- Division - the number of data rows/columns in the marker.
- KeepAlive - sets the timeout before a missing marker is removed from the marker list. The timeout is expressed as camera frames.
- MinimumSpan - controls the minimum amount of contrast that is required before marker is accepted as a marker. This parameter controls the rejection of false markers that may otherwise be detected around hands (when pure image noise manages to create a perfectly rectangular edge contour, with correct alignment marks).

After these parameters are set, motion detection should be detected correctly. However, the touch information provided from this step may not match the projection. In order to avoid this, one should perform a final calibration step that consists of matching both the input information and the output projection.

4.1.3.1 Projection calibration

The last step in this device's calibration should consist in adjusting the projected image. Even though the cameras' settings are tuned, the lack of this final calibration or its incorrect calibration may lead to a confusing, inefficient and maybe impossible usage of this device. This final step consists of making the input image correspond to the output image in terms of distortion and location so that if a finger touches a certain point in the screen, the result of the action is shown directly below the finger. An example of a badly configured system can be seen in Figure 4.3, where the input touch does not match the output projection.

Because of this device's calibration utilities, this type of calibration only requires that the developer tells the device where the corners and centre lie so it can adjust the positioning. Figure 4.4 shows the calibration procedure for one of the cameras. The five circles needed to be pressed so the device understands the distortion factor and translations needed to match the input information and the output projection.

4.1.3.2 Final calibration

After carefully examining the parameters and their usefulness, experimentation was needed to test several configurations in order to obtain the best results from the device. Due to its manufacturing, this specific device allows both cameras to be callibrated with the same settings. The following are the values for their final calibration, including the marker identification parameters:

- Shutter - the shutter value was set on 0.08.
- Gain - this value was set to zero in order to minimise the noise from the cameras.

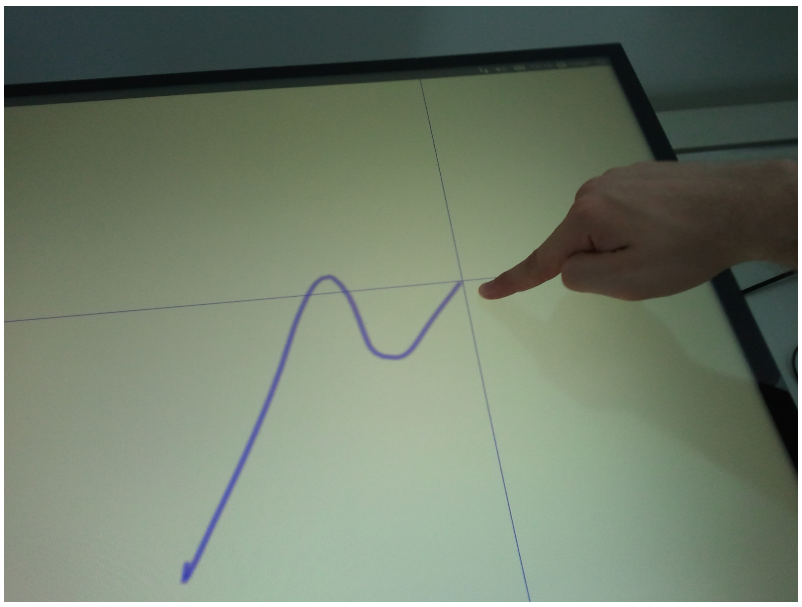


Figure 4.3: Example of the result of an incorrectly configured projection

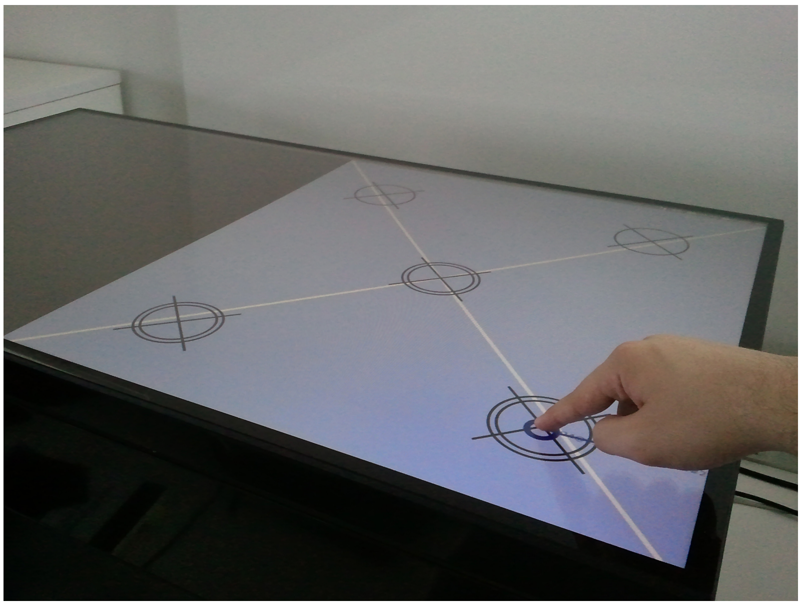


Figure 4.4: Projection calibration procedure

- Brightness - a value of 0.2 in this parameter was needed in order for the image to have values between 170 and 200 in light areas and 150 to 170 in darker areas.
- Gamma - the gamma value was set on 0.5.
- Lens A, B and C - the values were 0, 0.18 and 0.06 respectively; however, they are dependent on the distortion applied to the cameras as these are just correction parameters.
- Edge Limit - the edge limit value was set on 6.
- Value Limit - the value limit was set on 11.
- Tip sensitivity - because this value needs to be high to detect less dead areas, it was set on 1.5; a lower value would be less ideal as it would increase detection of false positives (ghost touches).
- Width CM - the width value was set on 110.
- Filter Drag - the filter drag value was set on 0.9 (maximum).
- Filter Radius - the filter radius value was set on 2 cm.
- Threshold - this value was set to 60.
- Max-length - value was set in 250 in order to detect wide markers.
- Min-length-rel - the minimum relative length of the edge contour was set to 0.06 so to detect a wide range of markers.
- Interval - this value was set to 19.
- Division - the division value was set to 5 as it was considered more efficient, i.e. it was detected across the more surface than other values.
- KeepAlive - this value was set to 20 in order to guarantee no markers were lost.
- MinimumSpan - after experimenting, this value was set to 10.

4.2 Kivy framework and software test suites

After making sure the device had been properly calibrated and was sending correct touch information, tests to verify the framework's behaviour were conducted. The general objective of these tests was to analyse what the API offered in terms of multi-touch interaction, how the graphics engine was handled and to what limitations this combination (device, protocol and framework) was subjected.

The following subsection talks about how information gets processed in Kivy. The subsequent subsection tends to the several test applications created, describing general test conditions and the results gathered.

4.2.1 Programming paradigms in Kivy

The Kivy framework is an event-based framework and relies only on event-calling to interpret the input information. A super-class named Widget is the most general of the multi-touch object classes. It can be used as a container for other Widgets or, by use of its canvas, to display visual information. The Widget class possesses "on_touch_down", "on_touch_move" and "on_touch_up" methods - which represent when a touch is detected, when it moves or when it is lifted, respectively - whose function is to ascertain whether a touch intersects the object and, if so, send its children the same touch information so they can process it as well. From this architectural implementation, we can imply the hierarchical structure of Kivy's event-handling and its usefulness. The Scatter class extends the Widget class and implements usual multi-touch interactions such as translate, rotate and scale by using the 3 methods described above.

Kivy natively supports TUIO with an integrated TUIO client and touch interpreter that transform TUIO's 2Dcur and 2Dobj objects into one single touch variable in order to have transparency between classes and methods. Otherwise, finger touch and tangible object events would need different methods and containers. In addition, Kivy also supports touch emulation, so developers only need a mouse to obtain touch information, the same way the TUIO client would provide them.

4.2.2 Software test suites

All the tests performed during this stage were done on the tabletop, using its TUIO message server and both the Kivy touch emulation (mouse events) and actual TUIO messages with touch information sent from the device. In the case of TUIO messages both hands were used to test the objects' movements separately and independently. Multiple fingers and hands were used to handle a single object as well.

Because of Kivy's transparency in input handling, all tests regarding tangible objects and marker identification were primarily put into practice using simple mouse emulation events and later tested with markers on actual tangible objects on top of the surface.

4.2.2.1 Drag-and-Drop

This prototype was intended to get to know the Python language and its programming paradigms and the Kivy framework. More specifically, the framework was tested to examine the creation of visible, usable objects in the tabletop screen with basic translate, rotate and scale functions and to test the hierarchical dispatch of touch events and their respective handlers. To do that, a simple drag-and-drop program was created where the objective was to drag images from their original (random) location to another.

Results indicated that the python-based Kivy framework handled several fingers at the same time quite well; the event handlers in the Kivy framework are quite reliable; the hierarchy of the objects is very well constructed and makes event handling easy to use; the Scatter and Widget objects in the Kivy language are very useful for creating objects, handling translation, rotation and scale movements and can be easily expanded.

4.2.2.2 Area Drag-and-Drop

The "Area Drag-and-Drop" prototype was developed having the previous prototype as its base. The goals of this prototype were to test the touch handling capabilities of Kivy, by altering its behaviour to a more indirect approach and to test how different objects could interact with each other by intersection detection. In this application, by using more than 3 fingers, the user could drag several objects contained in the space between the fingers and drag them without needing to touch the top of the objects. The objective of this prototype was to drag all the objects from their starting locations to another object.

Results show that touch information can be handled at will and developers can even create an input filter which can modify the input information as well as ignore it so the application does not receive it. Additionally, all objects available in the API have native collision detection handling so a single call to a method of the objects can provide a boolean value indicating whether or not collision occurs.

4.2.2.3 Throw

In order to test animations along with vector handling and general mathematical computations, this prototype was developed. It consisted of an application that attempted to simulate the physics of an actual object with speed and acceleration values. The user was able to touch and move the object and, according to the speed and direction of his finger just before lifting it, the object continued its movement until stopping completely or hitting the corners of the screen, as if it had been thrown.

With this prototype, tests regarding how animations are handled revealed that they are customisable both in duration and in effect over time. Furthermore, any animation could simply be combined with another, thus creating elaborate animations. Vectorial calculations are also native to Kivy and easy to implement and use along with animations.

4.2.2.4 Tangible objects

The ability to track objects on its surface makes this type of interaction a viable choice for applications built for this device and, in this prototype, this capability was explored. The goal of this prototype was to better understand how tangible objects were processed under the Kivy framework and what the differences in handling finger and marker movements on an actual application were. For that effect, the prototype created aimed at a simple tracking application to track only tangible objects while showing associated information in an appropriate location.

The results of the tests run on this prototype indicated that touch information was transparent and that marker handling is no different from finger touch handling. The difference between the touch information came in the form of an object property, whether it existed or not. That property would explicitly inform the application of the identification of the marker and if it was not present, it meant that a finger was touching the surface instead of a tangible object.

4.3 Prototype testing

As previously mentioned, during hardware and software tests, low-fidelity prototypes of simple games were made in parallel to discover what were the requirements of an application such as the one being developed in this project and with this specific target audience.

These tests were performed at a local day care centre with the available subjects at the time of each test. The centre held about 100 people and tests were made with those who felt motivated to help. Tests would always follow a predefined protocol and would always be conducted by at least two people whose roles were to: a) interact with the user and follow the specified protocol in order to obtain the answers to the questions addressed by each prototype - the conductor - and b) observe the user and making notes about answers and behaviour for the purpose of analysing reactions and reaction times for example - the observer.

4.3.1 Arm length prototype

Given the size of the tabletop and by setting a minimum of 2 players, the area of the tabletop was divided vertically (in its width, since it is larger than its height) in two parts - the left and the right - in order to test the worst possible conditions for collaborative games on this tabletop. This particular prototype aimed at testing the arm length in conjunction with a straight torso position to test whether older adults in general had the arm length and gross motor skills needed to reach the tabletop's middle and select objects. This prototype consisted of a cardboard with a set of images placed on top of it and the user had to reach the objects as the conductor would name them.

Results show that very few subjects had difficulty reaching the objects located in the middle of the tabletop and those who did would still reach it without being coerced or employing too much effort. This test shows that on a tabletop application on a device with the given dimensions, one can build a game that uses the tabletop's width and divides it in two to create separate playing areas for different players for example.

4.3.2 Font size prototype

Because the tabletop is such a large object compared to other touch technologies like the smartphone or the tablet, the font size may be an issue when text is positioned away from the user. This prototype addresses this issue and, using the positioning of users from the previous prototype, helps us to understand how small a black font word in the middle of the tabletop can be. Here, the conductor would put words on the centre of the cardboard tabletop and ask the subject to read it; he would then substitute it for a word with a font size that had 2pt less.

Results indicated that, in spite of all the subjects wearing glasses, all of them were capable of maintaining a straight back while reading fonts with sizes between 30pt and 16pt. Fonts with lower sizes would force some users to incline towards the tabletop or cause them to make a bigger effort while reading. The relevance of this test to the goal of this thesis consists in knowing how small objects and letters can be for them to recognise their contours.

4.3.3 Keyboard prototype

The purpose of this prototype was to test the affinity with types of keyboards among older adults. The prototype consisted in asking the users to press some letters on a cardboard and paper keyboard prototype and then ask them to write a full word. Two types of keyboards were used. First, we would perform the test using the "qwerty" keyboard and then a keyboard with letters following the alphabetical order.

With this test it was observed that older people may tend to write better on a "querty" keyboard than on an A to Z keyboard. This result may imply that, if necessary, the use of a "qwerty" keyboard on the final application would be more effective than the other keyboard tested.

4.3.4 Icons prototype

Because perception of an image's meaning [Fre83] differs between individuals and, more generally, between generations as well, icon research was needed to find what kind of perception elderly people have when given certain icons. This prototype aimed to discover which image was a better representation of the magnification effect - should the application need to scale some objects so the user identifies them - and help button. The users were asked to choose between icons that may represent both functions and suggest other possible representations for that function.

Results indicated that the users prefered the magnifying glass when wanting something to enlarge and the question mark for a possible help button. With these results, we may deduct how the tangible objects may have to be processed or manufactured in the final application.

4.3.5 Content prototype

Common knowledge may also vary between generations due to social environment, so content should also be adapted to elderly people as much as any UI. The test performed in this prototype was intended to get to know what type of content the subjects could relate the most. It consisted of a game in which the users needed to throw in a category and, at a turn, say one word related to that category.

Results from this test influenced the topics covered on applications developed with this specific target audience. They indicated that elderly people are more knowledgeable in the following areas respectively: sewing, gastronomy, jobs, music, agriculture, animals, games and healthcare.

4.3.6 Tabletop game prototype

In order to test the ideas that emerged during the test phases of this project, we conceived a prototype of a game that could be played on top of a table and that could be augmented on or recreated in a computer environment. This prototype aimed at comprehending what kind of interaction the elderly have with the table while playing the game and to see whether tangible objects can be implemented; if so, how and in what situation. The game tested is simple enough for the final users to understand and feel motivated to play it. The game would be beneficial to the final users as it requires cognitive skills to deal with the reflection of the disk on the edges of the table, thus stimulating them. This game consisted in throwing a disk along the table's surface and scoring more goals in your opponents' area than your opponent in yours. The disk consisted on a cardboard circle, with a plastic wrapper in order to better slide along the table. Cardboard was added to the borders so that, similarly to hockey, the disk could be reflected back from the wall.

Results of this test indicated that people felt an increasing enthusiasm while playing the game and even try to come up with strategies to beat their opponent, in spite of the limited space and set of rules. This indicates that games similar to the one described may be suitable for implementation on projects with similar purposes. Results also indicated that tangible objects cannot be used efficiently when the players use it frenetically and still require precision. Hence, a tangible object should be seen as a proof-by-possession type of object rather than a gameplay type.

Chapter 5

Design and development

The advantages of games as applications for cognitive and motor stimulation on older adults, described in section 2.1, clarified how games could be beneficial to elderly people.

This chapter is divided in three sections: Game framework, Game design and Game implementation. The first will focus on the game platform developed in this thesis and its architecture. The game design section will describe the concepts involved in a game proposed, the game mechanics and how the user interacts with the application - according to a parallel thesis [Fer11] which states that the characteristics of this particular game concept may foster communication and, if played in teams, collaborative work. The last section will focus on the game implementation itself from a developer's point of view and will illustrate how the architecture is used and what implementation decisions were made.

5.1 Game framework

The platform created in this thesis aims to aid implementation of a certain type of games. Common ground of these games lies on the following characteristics:

- to be played by 2 players or 2 teams or a combination of both
- to be played using finger touch and/or tangible objects to interact with the application
- to be played using multi-touch concepts
- to allow multiple interactions at the same time
- to allow connection between objects of different nature in terms of interaction type

The description given in this section will focus on the architecture and its structure. It aims to present the approach followed and the reasoning behind it along with several of its main components and features.

5.1.1 Architecture overview

Having this type of game in mind, an architecture capable of supporting the game mechanics was drawn. The idea behind the architecture focused on making the game extensible and expandable so that content would be easily added to or removed from the game created and that the game remained flexible enough to support modifications to either game mechanics, UI design or more fundamental game changes.

This platform was then conceived as a layer-based system. Different layers would handle different content, have different tasks and contribute differently to the gameplay. A game engine to manage all the layers and implement the actual game was created as well. The four layers and game engine are depicted in the following sections, along with their communication. The customisable menu and menu button classes, also described in this section, were designed to be part of the framework so that a flexible menu system is available to the developer, one that can be adjusted for any type of game.

5.1.2 Layered approach

The game is composed by four different layers that represent four distinct parts on the UI, all of them overlapping each other - Layer One overlaps Layer Two, and so on. A game engine, which explicitly handles the scoring mechanism and part of the communication between the layers, runs on the background and controls which layers are shown. A simple diagram of the visual and architectural representations of these layers can be seen in Figure 5.1.

Two of the layers are responsible for displaying information and setting the game environment. Layer One handles the information to be shown on top of the rest of the game, such as game statistics, or the scoring mechanism, while Layer Four implements the environment to be shown in the back i.e. background visual information, hence being the layer below. Both layers are designed to ignore user interaction.

Layers Two and Three handle all the game controls and are responsible for the application's interaction with the user. Layer Two is responsible for handing the tangible objects' information, ignoring all other user input information. It also contains all the objects of the UI that are associated with tangible objects' motion. Layer Three on the other hand is meant to ignore the external objects' information and handles the finger motion events of the user input so it contains only objects that react to finger-related input.

By contrast to the four layers described, the Game Engine does not directly take part of the UI; however, it takes part in their communication as it is the link that connects all

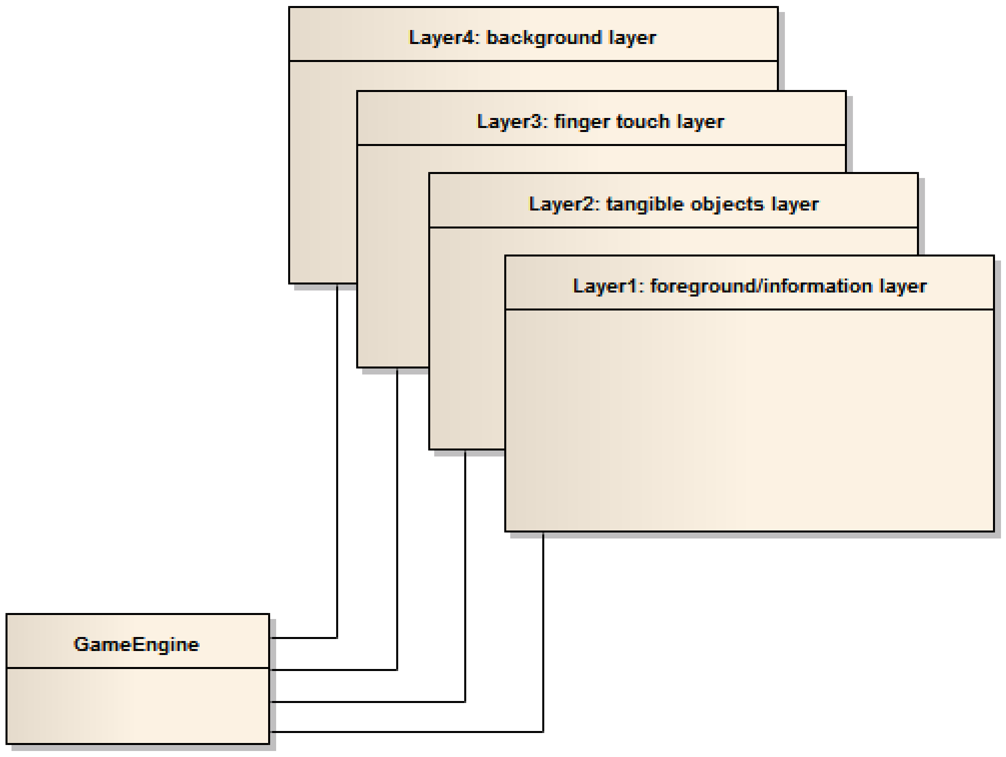

Layer4: background layer
Layer3: finger touch layer
Layer2: tangible objects layer
Layer1: foreground/information layer
GameEngine

Figure 5.1: Visual and architectural representations of the game

layers. In order for a layer to request information contained in another layer, it needs to go through the Game Engine to do it. This guarantees that the information is retrieved from the right layer, in case the game requires it to be replaced, and that any game specification that should run while this communication takes place is ran.

5.1.3 Communication between layers

Bearing in mind the function of each layer, the communications between them will now be formalised.

The Game Engine, whose abstract methods can be seen in Figure 5.2, is responsible for receiving the touch information from Kivy - through the methods “on_touch_down”, “on_touch_move” and “on_touch_up” - pre-process it if needed and send it to all layers, regardless of their functions or goals. It is also responsible for receiving the information related to the users’ actions, i.e. results from the mechanics implemented on layers and objects, and take appropriate action by updating internal game information and asking layers One and Four to create or update objects accordingly.

Layer One should only receive notification - through method calling - for the creation of objects such as labels or images that inform the user of his/her results on a certain action, as seen in Figure 5.3. These abstract methods represented in this figure should be

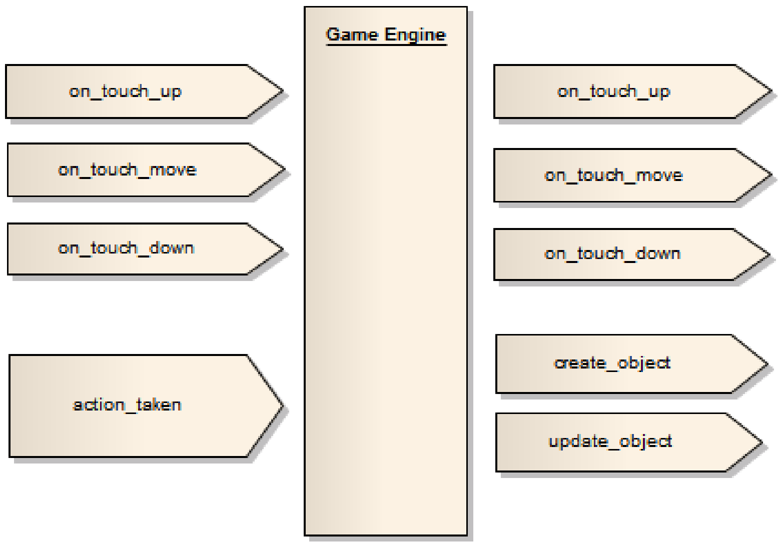


Figure 5.2: Dynamic view of the Game Engine

replicated for each object as they may require specific analysis or implementations depending on the gameplay and on the information to be displayed. The concept of deletion of an object should not come from the Game Engine but from the layer itself as management of these objects should be done inside the layer. Upon creation or update, the objects should behave according to the game mechanics implemented and be deleted automatically by the layer. The same is applicable to Layer Four, with a different goal in mind however, due to its function on the game mechanics. Changes in the environment would require this layer to have the same abstract methods as Layer Two. As previously stated, these methods are called by or through the Game Engine alone.

Likewise, Layers Two and Three share approximately the same structure. Because their task is to receive user input, the "on_touch_down", "on_touch_move" and "on_touch_up" methods of Kivy's Widget class should be implemented according to the layers' goals. The difference lies in the implementation which would ignore input not suited to the

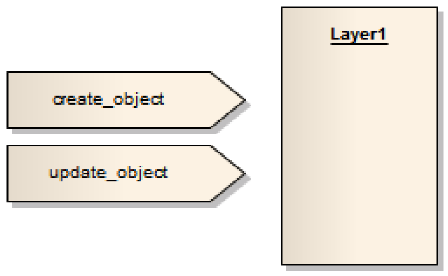


Figure 5.3: Dynamic view of Layer One

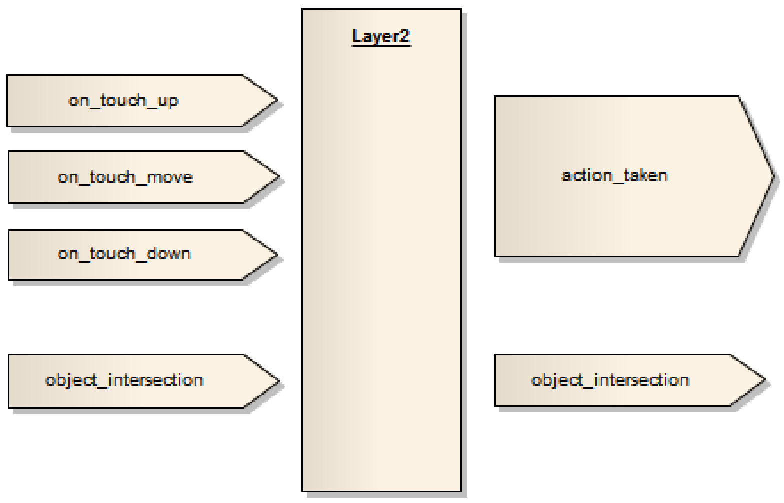


Figure 5.4: Dynamic view of Layer Two

layer's purpose. Both layers also need to communicate with the Game Engine in order to report an action taken on the game so it can process that information and propagate it according to the game mechanics as explained earlier. Furthermore, and due to the fine distinction between these two layers and because they contain the objects that interact with the user, they would need to enquire one another to know whether an object is intersecting with the other layer's objects. These abstract methods can be seen in Figure 5.4 which represents both layers' interaction with the Game Engine.

5.1.4 Menu flow

Generic menus for this framework were developed so that some information can be gathered before starting a game. By default, due to the goal of this framework, the menus are created for half of the screen's size. A menu instance saves the last menus' instance so that it can return to it if the user intends to. The concept behind this frameworks' menu structure is that every time the user advances from a menu to the other, that menu's instance is saved in the next one. If the user wants to go back, then the previous menu's instance is shown and the current menu is deleted - a backtrack approach.

Each menu possesses methods to be called by buttons. Hence, buttons do not have any implementation of the action taken; they simply call the menu's correspondent action method. Menus enable extension of four action methods: "option_one", "option_two", "option_three" and "option_help". The first three are used to call menus defined by the developer while the later is intended to call a help menu specifically. A button may call any of these four methods. The advantage of this approach is that all implementation is done in the menu class.

5.2 Game design

In order to test the architecture described in the section above, a game will be proposed in this section. So that this game may be beneficial to our specific target audience and in order to accomplish the goals of this project, the game needs to meet some requirements in terms of the game concept, mechanics and controls. This section describes these game characteristics and explains how they create an application capable of counteracting motor and cognitive functions decline.

5.2.1 Concept

The purpose of the game is to drag the images related to a certain category to the player's area faster than the opponent. Although the game was designed to be played by two people, several players can form a team and play against another person or team. Hence, the game is divided into two areas along the tabletop's width - Figure 5.5 represents the game as described. Each player, or team, will be assigned a category and will need to drag images that correspond to that category from the centre of the table, where they are initially placed, to a specific area that validates the image, located on their side of the table. The game also includes a magnifying glass so the user can augment the image if it is not clear to him what the picture represents. Each player's score and image count is placed on the border, near the player. The current game state represented in Figure 5.5 is: the left player needs to collect images that represent "Jobs" and the right player images that correspond to "Food"; the left player has one correctly placed image on his area while the right player has two; the left player is using the magnifying glass to amplify an image; of three games played, two were won by the player on the right. During the game, both players can see the amount of games won on the sidebar.

Due to the size of the tabletop and according to a parallel study on games for tabletops for elderly people [Fer11], the division of playing areas can foster gross motor skills as well.

5.2.2 Mechanics

Before the game can be played, the user is prompted whether they would like to play or read the instructions on how to play the game. A series of menus are used so the user can choose the category to play with. In the case of having more than one player per team, the user would also be able to tell the system.

When the game starts, the images of both categories are displayed on a common area in the centre where both players are able to reach them. Each player, or team, can drag any image to any part of the screen they wish. The game ends when both players place all the images associated with their category on their respective areas. When a player places

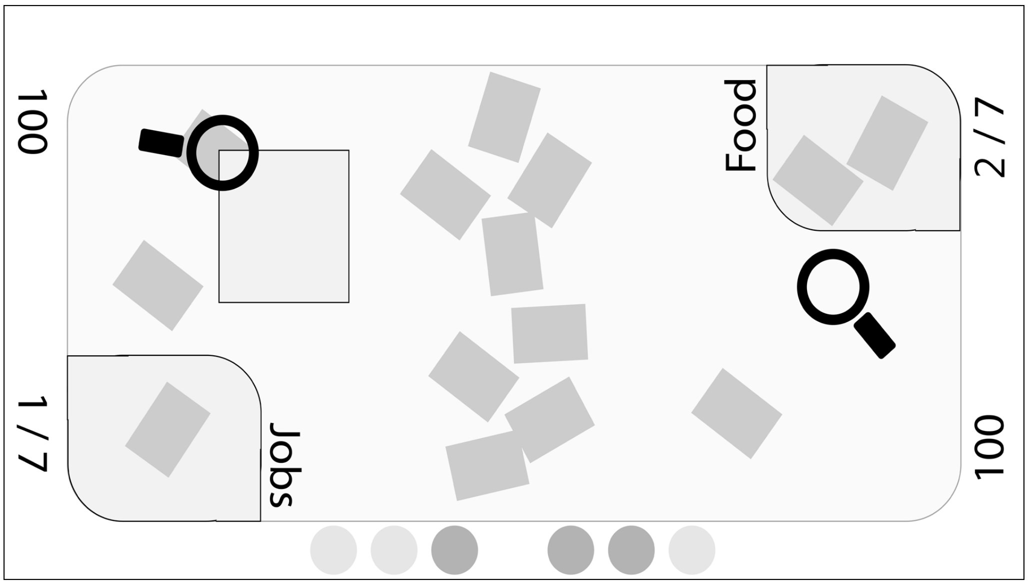


Figure 5.5: Game prototype design

an image on their area that does not belong to their category, the game pushes the image back to the centre of the table so that the other player can reach it. This prevents a player's mistake interfering with the other's chance of winning.

The scoring mechanism involves both time after the other player's completion and the number of consecutive correct images placed on the players' area. If a player places an image that belongs to their category on their own area, the player scores 10 points. On the other hand, if a player places an incorrect image, the score decreases 5 points. When players are able to place several consecutive correct images on their area, the score gained is doubled and a message appears, saying they earned a bonus for it. After the game is over, the player who scored more points wins and this information is represented visually on the side of the table, as seen in Figure 5.5.

This few set of rules would help motivating the players to try to be faster than the opponent(s), promoting gross motor skills, cognitive stimulation of object recognition and "quick thinking". These few rules may also encourage players to search for a weak-link on the set of rules to give them an advantage on the score, thus encouraging reasoning and logical processes.

5.2.3 Controls

The UI of this application was intended to be a natural one, however this does not imply that the user is restricted to using hand movements as previously stated in chapter 2. Therefore, and in an attempt to increase the general motor stimulation on the end-user, tangible objects were used along with finger movement to manipulate objects of the game.

On one hand, to move images from one point of the table to another, users are required to use their finger by touching the surface where the picture is shown. By lifting the finger, the image comes to a halt as it is considered that it was let go. On the other hand, the magnifying glasses do not work through finger motion detection. These objects exist only when a tangible object is placed on the table - as long as the tabletop identifies the marker - and to a maximum of one per player. The recognition of these objects is marked by a visual reaction of the application, identifying that the marker is accepted and the object represents a magnifying glass.

These two forms of interaction provide a greater challenge to the user and richer gaming experience along with the already mentioned benefits to the older adult's health.

5.3 Game implementation

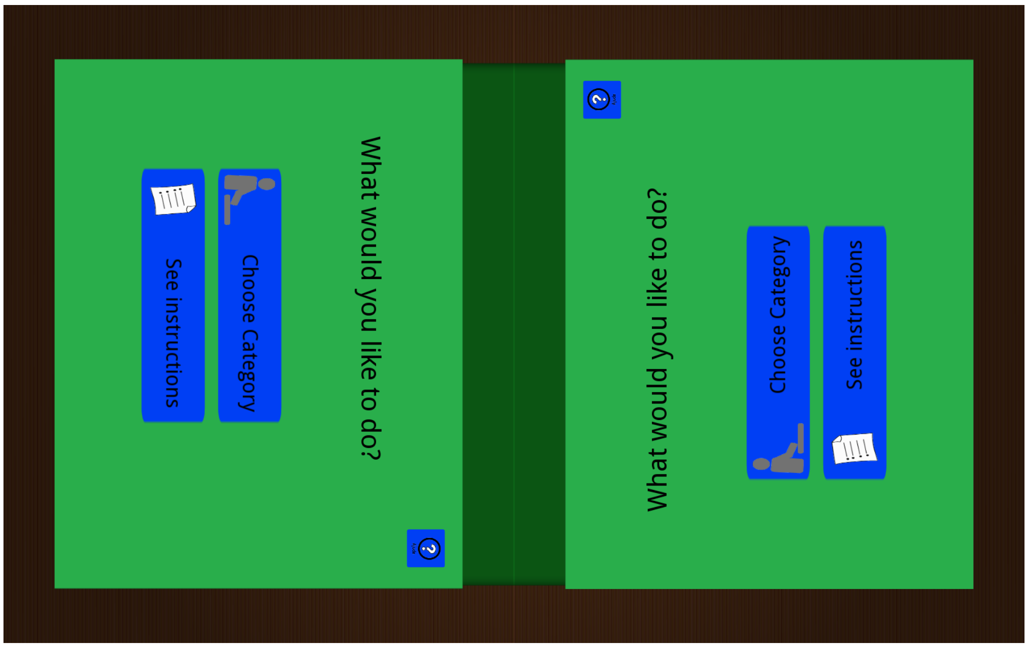
After a specification of how the architecture of the application is built and how different parts of the system interact, the final prototype of the application is shown in detail in this section. The class diagram is presented and explained, relevant parts of the Game Engine are shown and implementation restrictions are described.

5.3.1 Final prototype

Figure 5.6 contains screenshots of both players' menus and of the game implemented using the architecture and mechanics described. Layers One through Four are displayed in the figure. Figure 5.7 shows the Class diagram where all the classes developed in this project are represented.

The two menus shown in Figure 5.6 work on their own accord and when one player works his way through it, both menus synchronise and wait for the other player to finish, to start the game. The prototype has five available menus as shown in the class diagram. The player is first presented with the Main Menu where they can go to the Instructions Menu and back or to the Category Menu where a category will be chosen and the game can start. The user can also ask for help in the help button available where they will be taken to the Help Menu. The Wait Menu was created as means of synchronisation so both players start the game at the same time. Both menus and buttons are created by extending the previously described Menu and MenuButton classes.

Layer Four was implemented with no updatable objects, as it is used for the background environment only. Borders and background are created by the constructor of the class and have been defined as static components. In order for the score, image counter and games counter - located on the border of the image - to be shown above all other interactive objects, Layer One was chosen as their container and both variables and methods for creating and updating them were implemented. Specifically, Layer One contains two



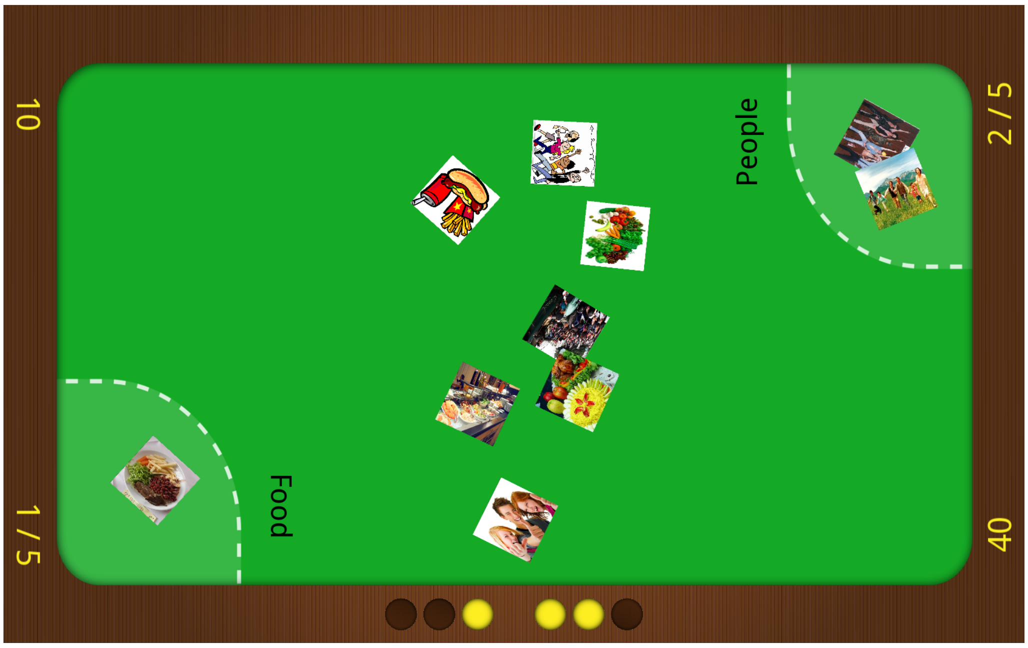


Figure 5.6: Snapshots of menus and of the game prototype

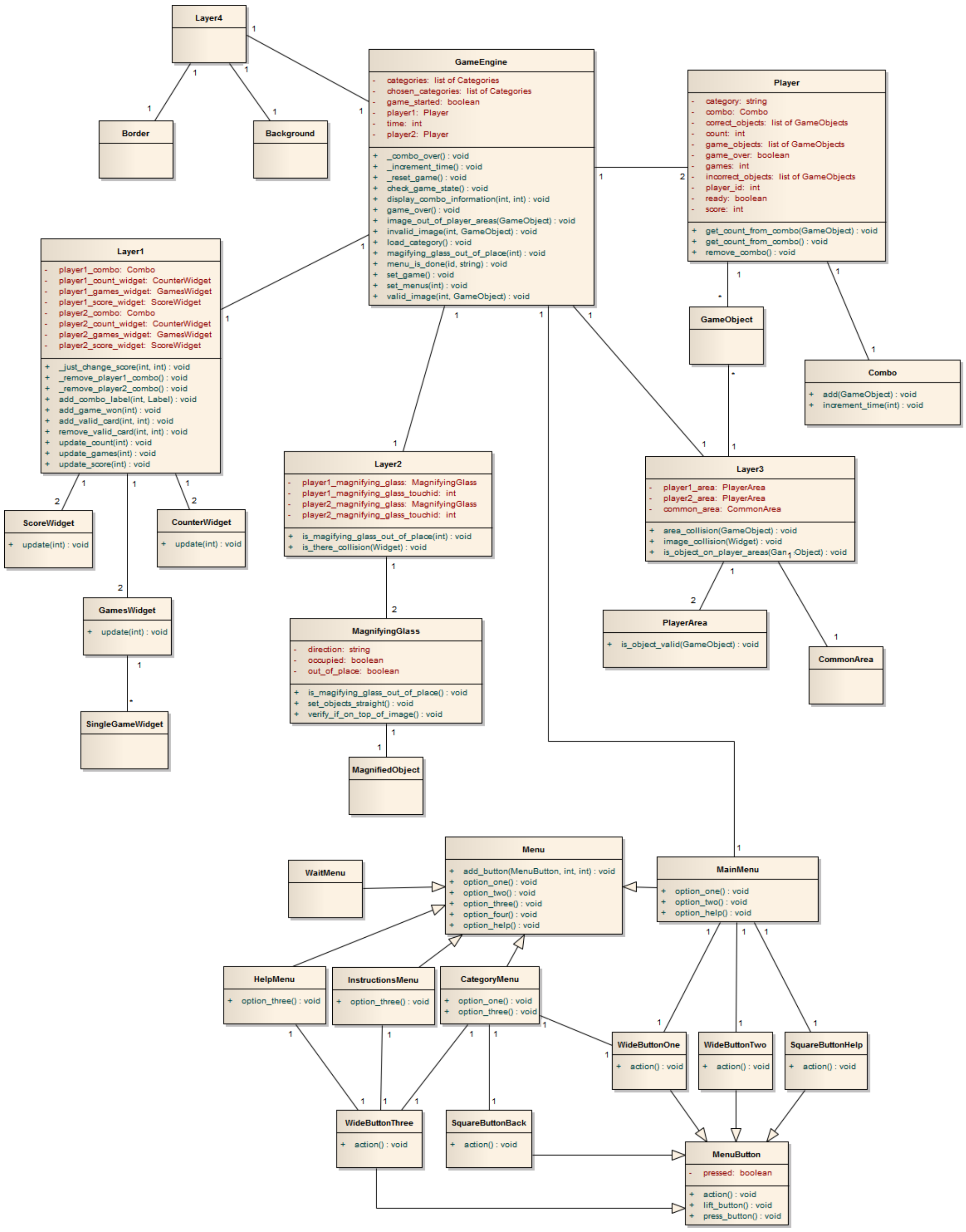

Layer4
Border
Background
GameEngine
- categories: list of Categories
- chosen_categories: list of Categories
- game_started: boolean
- player1: Player
- time: int
- player2: Player
+ _combo_over() : void
+ _increment_time() : void
+ _reset_game() : void
+ check_game_state() : void
+ display_combo_information(int, int) : void
+ game_over() : void
+ image_out_of_player_areas(GameObject) : void
+ invalid_image(int, GameObject) : void
+ load_category() : void
+ magifying_glass_out_of_place(int) : void
+ menu_is_done(id, string) : void
+ set_game() : void
+ set_menus(int) : void
+ valid_image(int, GameObject) : void
Player
- category: string
- combo: Combo
- correct_objects: list of GameObjects
- count: int
- game_objects: list of GameObjects
- game_over: boolean
- games: int
- incorrect_objects: list of GameObjects
- player_id: int
- ready: boolean
- score: int
+ get_count_from_combo(GameObject) : void
+ get_count_from_combo() : void
+ remove_combo() : void
Layer1
- player1_combo: Combo
- player1_count_widget: CounterWidget
- player1_games_widget: GamesWidget
- player1_score_widget: ScoreWidget
- player2_combo: Combo
- player2_count_widget: CounterWidget
- player2_games_widget: GamesWidget
- player2_score_widget: ScoreWidget
+ _just_change_score(int, int) : void
+ _remove_player1_combo() : void
+ _remove_player2_combo() : void
+ add_combo_label(int, Label) : void
+ add_game_won(int) : void
+ add_valid_card(int, int) : void
+ remove_valid_card(int, int) : void
+ update_count(int) : void
+ update_games(int) : void
+ update_score(int) : void
GameObject
Combo
+ add(GameObject) : void
+ increment_time(int) : void
ScoreWidget
+ update(int) : void
CounterWidget
+ update(int) : void
Layer2
- player1_magnifying_glass: MagnifyingGlass
- player1_magnifying_glass_touchid: int
- player2_magnifying_glass: MagnifyingGlass
- player2_magnifying_glass_touchid: int
+ is_magifying_glass_out_of_place(int) : void
+ is_there_collision(Widget) : void
Layer3
- player1_area: PlayerArea
- player2_area: PlayerArea
- common_area: CommonArea
+ area_collision(GameObject) : void
+ image_collision(Widget) : void
+ is_object_on_player_areas(GameObject) : void
GamesWidget
+ update(int) : void
MagnifyingGlass
- direction: string
- occupied: boolean
- out_of_place: boolean
+ is_magifying_glass_out_of_place() : void
+ set_objects_straight() : void
+ verify_if_on_top_of_image() : void
PlayerArea
+ is_object_valid(GameObject) : void
CommonArea
SingleGameWidget
MagnifiedObject
WaitMenu
Menu
+ add_button(MenuButton, int, int) : void
+ option_one() : void
+ option_two() : void
+ option_three() : void
+ option_four() : void
+ option_help() : void
MainMenu
+ option_one() : void
+ option_two() : void
+ option_help() : void
HelpMenu
+ option_three() : void
InstructionsMenu
+ option_three() : void
CategoryMenu
+ option_one() : void
+ option_three() : void
WideButtonOne
+ action() : void
WideButtonTwo
+ action() : void
SquareButtonHelp
+ action() : void
WideButtonThree
+ action() : void
SquareButtonBack
+ action() : void
MenuButton
- pressed: boolean
+ action() : void
+ lift_button() : void
+ press_button() : void

Figure 5.7: Class diagram of the implementation

ScoreWidgets, two CounterWidgets and two GamesWidgets; the later containing multiple SimpleGameWidget - defined by the Game Engine upon creating the layer.

Being responsible for the tangible objects only, Layer Two contains two objects of the MagnifyingGlass class and an ID number for both which is assigned when the tangible object touches the surface. The MagnifyingGlass class has variables to know whether it is magnifying an object and whether it is out of the player's side of the table. Each instance will create a MagnifiedObject when on top of an image and delete it when not. The MagnifiedObject class is responsible for producing a larger image of the one the MagnifyingGlass object is on top of.

Layer Three contains GameObject: a class created for representing an image placed on the table. The GameObject class simply loads an image given to the constructor and places it on its canvas so it can be viewed by the players. These images, due to the game-play established, cannot be scaled. Aside from GameObjects, Layer Three contains two instances of the PlayerArea class and one of the CommonArea class. The first represents each player's area that is located near the edges of the screen. It is used for ascertaining whether the object belongs to the category of the player associated with that area. The later provides a common space where the GameObjects are initially put. Although these may have less pronounced purpose, like other classes already described, they too were created to make the game extensible beyond the mechanics described for this particular game.

As mentioned earlier, during the game, the Game Engine class contains only the four layers previously described and variables to control the flow and implement the mechanics of the game. This class possesses a list of categories available and a list of categories that will be chosen to play with. The process of loading GameObjects and Categories is also managed by this class alone and it is done by scanning the file system for folders and files, inside the "data" folder, containing images; again with the purpose of extensibility in mind. This class also keeps track of time elapsed in the game. It generates two instances of the Player class as well, which are used to store information regarding the player itself, such as their playing category, games won, objects placed correctly, their score and a list of all objects of the players' category - as shown in Figure 5.7. It also contains a variable to store the player's combo scoring system.

5.3.1.1 Engine initialisation

Part of the initialisation procedure will now be shown to demonstrate how the menus exit in order to the game starts, to exemplify how the layer-based system proposed in the framework is used and to show how they interact with each other on this game prototype.

The following piece of code represents the constructor of the class Game Engine:

```
def __init__(self):
  Widget.__init__(self)

  # essencial variables for gameplay
  self.time = 0
  self.game_started = False
  self.categories = []
  self.chosen_categories = []
  self.load_category_names()

  self.player1 = Player(1)
  self.player2 = Player(2)

  # creation of menus
  self.set_menus(1)
  self.set_menus(2)

  # schedules an event to update time-related mechanics
  Clock.schedule_interval(self._increment_time, 0.1)
```

As explicitly shown in the previous piece of code, after creating the variables needed to play, the Game Engine creates the menu system whose initialisation code is shown below:

```
def set_menus(self, pid):
  # create main menu
  menu = MainMenu(pid)
  self.add_widget(menu)

  # set menus' position
  if (pid==1):
    menu.rotation=-90
    menu.pos=(0,0)
  if (pid==2):
    menu.rotation=90
    menu.pos=(Window.width/2,0)
```

When a player finishes going through the menu to play, the following method of the Game Engine class is called to make sure both players are at the same stage before starting the game:

```
def menu_is_done(self, pid, category):
  if (pid==1):
    self.player1.ready = True
    self.player1.set_category(category)
    if self.player2.ready and not self.game_started:
      self.game_started = True
      self.set_game()
  if (pid==2):
    self.player2.ready = True
    self.player2.set_category(category)
    if self.player1.ready and not self.game_started:
      self.game_started = True
      self.set_game()
```

The “set_game” method is presented below:

```
def set_game(self):
  # remove menus from UI
  self.remove_all_widgets()

  # load and set GameObjects from categories chosen
  category1_objects = self.load_category(self.player1.category)
  category2_objects = self.load_category(self.player2.category)

  self.player1.set_game_objects(category1_objects)
  self.player2.set_game_objects(category2_objects)

  # create the 4 layers
  self.Layer4 = Layer4()
  self.Layer3 = Layer3(self.player1, self.player2)
  self.Layer2 = Layer2()
  self.Layer1 = Layer1(len(self.player1.game_objects),
                       len(self.player2.game_objects))

  # add the 4 layers to the UI
  self.add_widget(self.Layer4)
  self.add_widget(self.Layer3)
  self.add_widget(self.Layer2)
  self.add_widget(self.Layer1)
```

Finally, this next piece of code demonstrates how Layer Two asks Layer Three to if a GameObject is placed on top of another object, belonging to Layer Three:

```
## from the Layer Two class
def is_there_collision(self, widget):
  return self.parent.image_collision_Layer3(widget)

## from the Game Engine class
def image_collision_Layer3(self, widget):
  #if necessary, place some code here ...

  return self.Layer3.image_collision(widget)
```

```
## from the Layer Three class
def image_collision(self, widget):
  #for all the widgets that it contains
  for w in self.children[:]:
    unwanted=[self.common_area,self.pl1_area,self.pl2_area]
    #if it is a GameObject
    if (w not in unwanted):
      cx = widget.x+widget.width/2
      cy = widget.y+widget.height/2
      #if the widget collides with the centre of the one given
      if (w.collide_point(cx,cy)):
        return w
  #if not return false
  return False
```

5.3.2 Implementation decisions

During the course of the implementation stage, some decisions were made in order to underline the goal of this thesis. The most significant and constraining decisions taken regarded tangible objects' detection and use, the game's border size, the in-game user interface and the menu design.

When implementing the tangible objects, it was noted that, because a new physical object may be created for some reason - one that uses another marker, for example - the game would have to be adapted to work as the application may not recognise the object as belonging to a specific player. In that sense, the magnifying glass object was implemented to be used freely. When a magnifying glass is placed on the table it becomes property of the user, regardless of its marker or of previous ones. However, a user may not possess two magnifying glasses and the game will not recognise more than two in total. This decision was made due to the increased portability it brings to the application.

In terms of border size, the setting of 5cm borders around the table could not be avoided because a perfect detection could not be guaranteed for finger motion input on those areas. This also affected the design of the application as these dead areas reduced the size of the table interaction and forced the display of information, like the players' score, to take place displayed on the borders rather than occupying valuable space to be used for GameObjects.

The menus implemented in the game's final prototype were not as complete as intended. The baseline for menus was drawn and lack of time prevented from implementing the full range of options tested in collaboration with a parallel study [Fer11]. The same problem affected the overall design of the prototype as well, as it was not iterated enough to be more appealing to the player.

Chapter 6

Validation and Results

The game developed during the implementation stage was iteratively tested, as previously explained. Only one of the last iterations was tested with the end-users. The previous chapter described the game as is its final implementation and considers the changes proposed after the validation of the application with the elderly. Hence, this chapter describes the evaluation performed and how the results are reflected on the gameplay and design of the application.

The validation process was carried out at the day care centre where the initial low-fidelity prototypes were also tested. In this test, an advanced version of the game prototype was used, so the process involved moving the device to the facility. For this evaluation alone, a separate room was required so the device could be safely installed and tested on. Several participants joined the test and stood as spectators. Their presence was also required for multiplayer gaming and social environment during the test. The overall testing phase took approximately one hour and thirty minutes. A representation of the initial participats and setup of the test can be seen in Figure 6.1. The tangible objects used in this prototype can be seen in Figure 6.2. They consisted of a handle on top of the marker, with information of what the object's purpose was.

As a first approach, the participants sat around the tabletop while two users played the game for the first time. Users took turns at playing the game on the tabletop. As time elapsed and the participants grew accustomed to the game, we seized the opportunity to ask them whether they would like to play as a team. This generated interest in multiplayer gaming, active help to both users in identifying images and simple communication and interaction between players and participants. Along the timeline, the tangible objects' usage decreased as the users started to memorise the images and did not need to amplify them.

As a result of this test, several observations were made regarding different aspects of

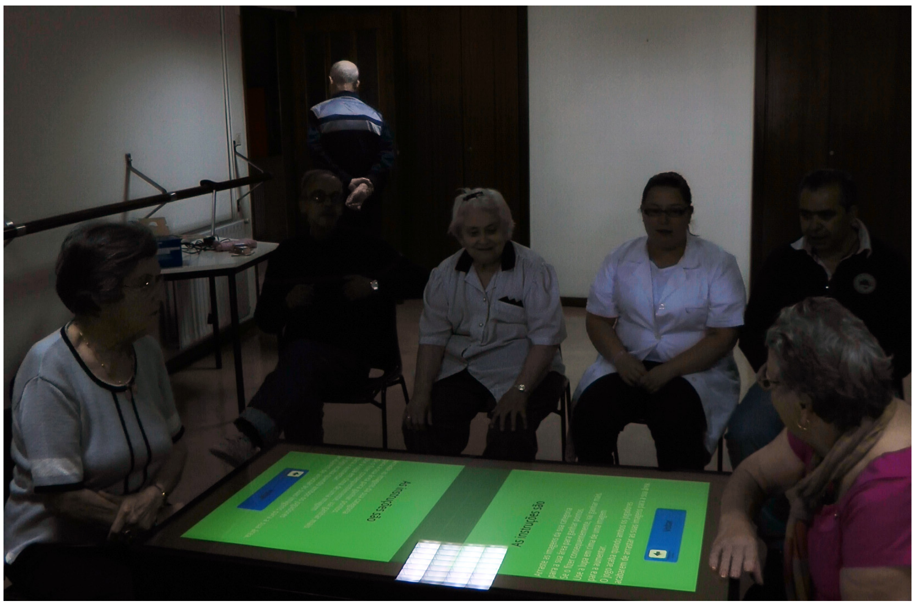


Figure 6.1: High-fidelity test of the game prototype

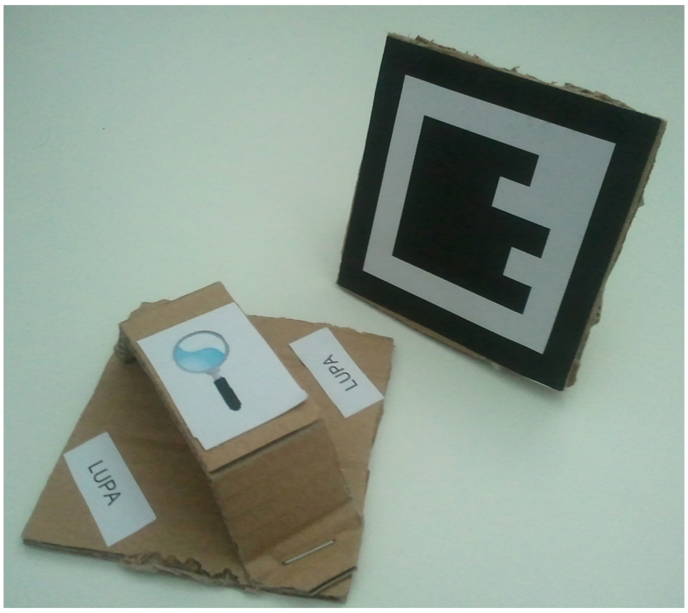


Figure 6.2: Magnifying glass used as tangible object in the high-fidelity test

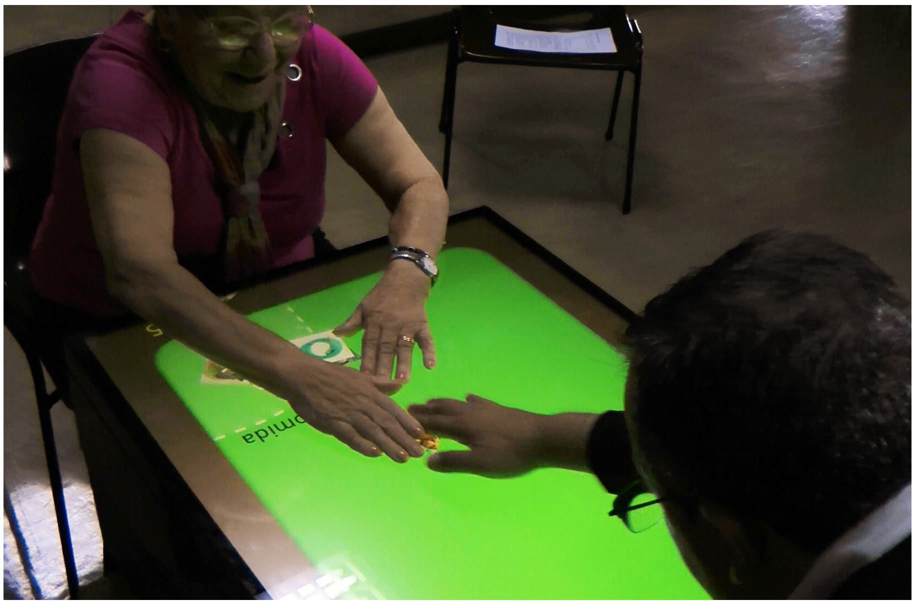


Figure 6.3: Example of poor information positioning

the game. In terms of the menu implemented, it was noticed that users tended to press the icons on the buttons rather than the button itself or the text of the button. Some buttons, on the top right corner were also considered too small and distant. However, the structure of the menu and the purpose of each of them were understood by the users as they navigated through them without difficulty.

Observations from this test also indicate that the initial design of the game was not efficient. Players usually tried to place the entire image on their assigned area; which indicates the validation mechanism could be improved by visually informing the user that the object is validated instead of displaying text for that purpose. Moreover, it was clear that the positioning of the score information was not ideal because - as seen in Figure 6.3 - the users positioned their arms on top of the information displayed on the border.

The effectiveness of the object manipulation was clear although some minor issues were discovered. It was observed that the players used multiple fingers to drag an image, resulting on image jittering due to discrete motion detection, caused by ambience lighting. As a last observation on both the interaction and gameplay, players seemed intrigued by the lack of a time-related point system.

In terms of the magnifying glasses' use, i.e. tangible objects' use, it was observed that their purpose was understood and that the users felt a connection with the object. As a result, the object was being used not only to amplify the game objects but to drag the objects as well due to a minor issue that often allowed the corners of the objects to be considered finger input data. Although the visual representation of the tangible object was somewhat confusing to the users, due to the offset of its amplifying function, a simple redesign of the object and of its visual representation, in order to remove the need for that representation while matching the centre of the object to the amplifying reaction, would dissipate this issue.

Regardless of the issues raised by the observations made during the validation process, all the players remained motivated to play it in a way that even younger adults felt compelled to play and that new participants kept arriving as well. The game concept presented challenges to the players and kept them engaged. Furthermore, no architectural design modifications were necessary, which validates the structure of the framework developed in this thesis, its layer-based approach and its input-related structuring and dispatch. The observations were also a clear indicator that the application was in the right track and the modifications applied were considered useful to its future success.

Chapter 7

Discussion

During the course of this thesis many decisions were made such as choice of platform, device or regarding the implementation itself. In this chapter, these decisions are explained in more detail and alternatives are discussed and analysed.

The literature review revealed techniques more suitable to this project than others along with frameworks and their respective programming languages that could help bring more functionality to the final application. The several criteria for comparing the studied touch detection techniques, explained in section 2.3.2, stood out due to their relevance to the target audience of this thesis. Other characteristics such as "compatible with diffuser", which would influence the hand detection, or "minimum hardware size", were not considered as relevant and therefore would only be suitable as a final deciding factor between two techniques. On the same trail of thought, the comparison between frameworks for building the application also considered the most relevant factors, also described in section 2.3.5.1. The decisions regarding the choice of the framework did not depend on external factors and were mainly based on the comparison of functionality, performance and programming language. On the other hand, the choice for touch detection technique was dependant on the device to implement the project on. Another device aside from the Multitouch Cell Advanced could have been chosen for this project, implementing either one of the chosen touch detection techniques: the FTIR and Rear DI techniques. Although, due to availability constraints, this specific device, that uses Rear DI, was selected.

Regarding the methodology proposed and chosen, it was believed to be the best fit given the resources available to develop this project. The hardware and software tests allowed for a better understanding of future development conditions and revealed hardware and software limitations. The "low-level to high-level" approach taken was proven to be a viable methodology for hardware and software exploration. The most suitable alternative would have been to take a “high-level to low-level” approach which would have allowed a more specific tuning of the device; however, it was decided that the advantage of this approach was not as beneficial, by contrast to the one taken. Prototype testing with the target user was done in parallel to the aforementioned tests in order to make an efficient use of time. Low-fidelity prototypes were chosen over high-fidelity prototypes to allow for flexibility and creativity in the design as explained earlier in chapter 3.

Hardware tests could have been performed under different specific amount of lightning in order to test the response and difference in touch detection of the device. Bearing in mind however, that day-time light may vary within a wide range, it was felt that, due to the nature of the target-audience, only a day-time environment should be considered. As for software tests, they were performed despite the end-users. This could have been done differently as the same tests performed by older adults would reveal more detailed requirements for the final application as they could be considered high-level prototypes, although this would require more time, resources and the users’ availability.

Although the implemented game would benefit from design iterations, its robust architecture and implementation allow it to be easily refined by replacing content and design-related files. The concepts created, such as the layered approach described in section 5.1.1 that successfully divides user inputs and handles them properly, and the portability and extensibility of the Kivy framework, make this game framework usable on other table games. Representative classes such as GameObject and GameEngine can also be reused by altering the game mechanics and creating a different game. The extension possibilities of the work produced in this thesis may lead to the development of new games by altering methods’ names and their implementation. Examples of easy to implement games, on the developed game framework, are “Air Hockey” or Puzzle Games. These would only require small changes to some game interaction and visual design. Moreover, the Menu and MenuButton classes of the game framework can easily be extensible as they were built to be versatile.

Compromises on the actual game developed had to be made as well. Due to balance between pressure required for touch detection and dead areas size on the tabletop, a line needed to be drawn to divide where the user’s input could effectively and accurately be detected and where it could not. This constraint resulted in the creation of borders for the game and a smaller interactive area, as previously mentioned. This decision could have been altered if the system was configured to be more sensitive to finger touches, which would result in hover detection and higher noise sensitivity. Considering elderly people as the end-users and their possibly diminished ability to control fine and gross motor functions, hovering was considered unsecure detection. The opposite, forcing the user to press harder on the tabletop to get a clear touch detection, would be undesirable for that same reason and due to shrinking the interactive area accordingly.

This work was the result of careful considerations and decisions and can considered to be of relevance to developers that aim to create applications for tabletops, particularly games. The reasoning and approach followed to study user-specific design may also be adapted to other studies regarding application development for a specific target audience.

Chapter 8

Conclusions and Future Work

The research conducted and work produced in this thesis led to the development of a game framework for multi-touch devices in general. The extensibility and usefulness of the framework developed are clear, as seen in the prototype developed using the concepts created for the framework. The use of the Python language and the Kivy framework only add versatility and independence to the applications produced, while providing the developer with a fast, robust and flexible way to develop them. In addition, Kivy also provides transparency regarding input methods as it supports OS-dependent protocols besides TUIO. Furthermore, the usage of the TUIO protocol as means of communication of touch-related information ensures that the applications built may operate on a diversity of devices, since TUIO is considered a standard for table based tangible user interfaces. The game framework created meets the expectations as demostrated through a practical implementation of a game.

This thesis successfully proposed and conceived a game using the framework created and the concepts which it is based on. The game proposed was developed with the intent to be beneficial - in health-related issues - to its target-audience and it allowed us to see whether the premises established in the game concept and the approach taken to the design and development of the application were valid. Through observation, it was noted that the elderly can benefit from the game created.

As a perspective of future work, there are some aspects in which the framework and the game could be improved. Regarding the current game implementation and in spite of the positive results obtained with the game at its current state of design and implementation, the menu layout and overall design of both the menus and the in-game environment could be improved through several more iterations, according to the users' feedback.

In terms of work that could be carried out to elevate the current framework, some suggestions have been considered, such as the implementation of:

- different containers, to display and dispose information in the menu system
- background statistics on players and ways to analyse them
- more directly, extendable classes of the framework components

Finally, more games can be developed in the future in order to test the frameworks' capabilities and architectural limitations. Such analysis would eventually provide suggestions on how to improve and widen the framework's functionalities.

References

[AA05] Kay Apted and Quigley A. A study of elder users in a face-to-face collaborative multi-touch digital photograph sharing scenario. *Tech. Rep. TR576*, 2005.

[AAG$^+$09] Michelle Annett, Fraser Anderson, Darrell Goertzen, Jonathan Halton, Quentin Ranson, Walter F. Bischof, and Pierre Boulanger. Using a multi-touch tabletop for upper extremity motor rehabilitation. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, OZCHI '09, pages 261–264, New York, NY, USA, 2009. ACM.

[AEO$^+$04] Aaron Adler, Jacob Eisenstein, Michael Oltmans, Lisa Guttentag, and Randall Davis. Building the design studio of the future. In *Making Pen-Based Interaction Intelligent and Natural*, pages 1–7, Menlo Park, California, October 21-24 2004. AAAI Press.

[AKQ06] Trent Apted, Judy Kay, and Aaron Quigley. Tabletop sharing of digital photographs for the elderly. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 781–790, New York, NY, USA, 2006. ACM.

[All08] J. Allen. Older people and wellbeing, 2008. `http://www.vhscotland.org.uk/library/misc/ippr_older_people_and_wellbeing.pdf`.

[AT10] David Andrews and Soon Tee Teoh. Mtvis: tree exploration using a multi-touch interface. In Jinah Park, Ming C. Hao, Pak Chung Wong, and Chaomei Chen, editors, *VDA*, volume 7530 of *SPIE Proceedings*, page 75300. SPIE, 2010.

[BBL08] Stefano Baraldi, Alberto Bimbo, and Lea Landucci. Natural interaction on tabletops. *Multimedia Tools Appl.*, 38:385–405, July 2008.

[Blo11] Miscrosoft Technet Blogs. Microsoft is imagining a nui future, 2011. http://blogs.technet.com/b/microsoft_blog/archive/2011/01/26/microsoft-is-imagining-a-nui-future-natural-user-interface.aspx.

[BPSM$^+$03] Tim Bray, Jean Paoli, C. M. Sperberg-Mcqueen, Eve, and François Yergeau, editors. *Extensible Markup Language (XML) 1.0.* W3C Recommendation. W3C, fourth edition, August 2003.

[BS02] S.J. Biggs and Srinivasan. Haptic interfaces. *Stanney, K.M. (Ed.), Handbook of Virtual Environments: Design, Implementation, and Applications*, 2002.

[Com11a] MT4j Community. Mt4j homepage, 2011. `http://www.mt4j.org`.

[Com11b] Pygame Community. Pygame homepage, 2011. `http://www.pygame.org/`.

[Com11c] PyMT Community. Python multi-touch framework homepage, 2011. `http://pymt.eu/`.

[Com11d] PyMT Community. Python multi-touch framework notes, 2011. `http://pymt.txzone.net/`.

[Com11e] Sparsh UI Community. Sparsh ui code homepage, 2011. `http://code.google.com/p/sparsh-ui/`.

[Com11f] TUIO Community. Flash/flex for mt homepage, 2011. `http://www.tuio.org/?flash`.

[Com11g] TUIO Community. Tuio, 2011. `http://www.tuio.org`.

[Cor11] Microsoft Corporation. Microsoft surface 2.0, 2011. `http://www.microsoft.com/surface/`.

[Cza96] S. J. Czaja. Aging and the acquisition of computer skills. *Aging and skilled performance*, pages 201–220, 1996.

[Cza97] S. J. Czaja. Computer technology and the older adult. *Handbook of Human-Computer interaction, Second edition*, pages 797–812, 1997.

[DKW10] Georg Freitag Dietrich Kammer, Mandy Keck and Markus Wacker. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. *Workshop on Engineering Patterns for Multi-Touch Interfaces*, 2010.

[DSVA10] Bob De Schutter and Vero Vanden Abeele. Designing meaningful play within the psycho-social context of older adults. In *Proceedings of the 3rd International Conference on Fun and Games*, Fun and Games '10, pages 84–93, New York, NY, USA, 2010. ACM.

[Fab06] M. Fabregat. Development of high therapeutic value ist-based games for monitoring and improving the quality of the life of elderly people. *Information Society Technologies project CN 034552*, 2006.

[FBG$^+$09] David Facal, Cristina Buiza, Mari F. González, John Soldatos, Theodore Petsatodis, Fotis Talantzis, Elena Urdaneta, Valeria Martínez, and José J. Yanguas. Cognitive Games for Healthy Elderly People in a Multitouch Screen. In *Proc. of the International Congress on Digital Homes, Robotics and Telecare for All (DRT4ALL)*, May 2009.

[FEL09] Williams R. Frazier E. L., McCurdy T. Intra- and inter-individual variability in location data for two us health-compromised elderly cohorts. *Journal of exposure science and environmental epidemiology*, 19:580–592, 2009.

[Fer10] Tiago Pinto Fernandes. Game engine for location based services. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2010.

[Fer11] Luís Ferreira. Elderly gaming on tabletop interfaces. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2011.

[FHH06] Carstensen L. L. Fung H. H. Goals change when life's fragility is primed: Lessons learned from older adultsm, the september 11th attacks and sars. *Social Cognition*, 24:248–278, 2006.

[Fis01] Gerhard Fischer. User Modeling in Human–Computer Interaction. *User Modeling and User-Adapted Interaction*, 11(1-2):65–86, 2001.

[Fre83] W J Freeman. The physiological basis of mental images. *Biol Psychiatry*, 18(10):1107–25, 1983.

[GAB$^+$06] L. Gamberini, M. Alcaniz, G. Barresi, M. Fabregat, F. Ibanez, and L. Prontu. Cognition, technology and games for the elderly: An introduction to ELDERGAMES Project. *PsychNology Journal*, 4(3):285–308, 2006.

[GAea09] Luciano Gamberini, Mariano Alcaniz, and Malena Fabregat et al. Eldergames: Videogames for empowering, training and monitoring elderly cognitive capabilities. 2009.

[GB04] C. Shawn Green and Daphne Bavelier. *The Cognitive Neuroscience of Video Games*. 2004.

[GCO$^+$97] Jeffrey H. Goldstein, Lara Cajko, Mark Oosterbroek, Moniek Michielsen, Oscar Van Houten, and Femke Salverda. Video games and the elderly. *Social Behavior and Personality: An International Journal*, 25(4):345–352, November 1997.

[Gre09] Samuel Greengard. Facing an age-old problem. *Commun. ACM*, 52:20–22, September 2009.

[Gro09] NUI Group. *Multi-touch technologies*. NUI Group, 2009.

[Han01] Vicki L. Hanson. Web access for elderly citizens. In *Proceedings of the 2001 EC/NSF workshop on Universal accessibility of ubiquitous computing: providing for the elderly*, WUAUC'01, pages 14–18, New York, NY, USA, 2001. ACM.

[Han05] Jefferson Y. Han. Multi-touch sensing through frustrated total internal reflection. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[HRB$^+$09] Amanda Harris, Jochen Rick, Victoria Bonnett, Nicola Yuill, Rowanne Fleck, Paul Marshall, and Yvonne Rogers. Around the table: are multiple-touch surfaces better than single-touch for children's collaborative interactions? In *Proceedings of the 9th international conference on Computer supported collaborative learning - Volume 1*, CSCL'09, pages 335–344. International Society of the Learning Sciences, 2009.

[HW04] et al. H. Wan, S. Gao. Mivas: A multi-modal immersive virtual assembly system. *ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 113–122, 2004.

[Ide11] Ideum. Gestureworks - multi-touch made easy, 2011. `http://gestureworks.com/`.

[INdKP07] Wijnand Ijsselsteijn, Henk Herman Nap, Yvonne de Kort, and Karolien Poels. Digital game design for elderly users. In *Proceedings of the 2007 conference on Future Play*, Future Play '07, pages 17–22, New York, NY, USA, 2007. ACM.

[Jac10] João Tiago Pinheiro Neto Jacob. Location based videogame. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2010.

[JL06] J. Kim J. Lee. u-table: A tabletop interface for multiple users. *Computational Science and Its Applications - ICCSA*, pages 983–992, 2006.

[KAB$^+$08] Per O. Kristensson, Olof Arnell, Annelie Björk, Nils Dahlbäck, Joackim Pennerup, Erik Prytz, Johan Wikman, and Niclas AAström. InfoTouch: an explorative multi-touch visualization interface for tagged photo collections. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, NordiCHI '08, pages 491–494, New York, NY, USA, 2008. ACM.

[Kam95] C Kamm. User interfaces for voice applications. *Proceedings of the National Academy of Sciences of the United States of America*, 92(22):10031–10037, 1995.

[KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–, Washington, DC, USA, 1999. IEEE Computer Society.

[KBBC05] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. TUIO: A protocol for table-top tangible user interfaces. In *6th International Gesture Workshop*, 2005.

[KJGA06] Martin Kaltenbrunner, Sergi Jorda, Gunter Geiger, and Marcos Alonso. The reacTable*: A Collaborative Musical Instrument. In *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06)*, volume 0, pages 406–411, Los Alamitos, CA, USA, June 2006. IEEE.

[KK96] R. Kjeldsen and J. Kender. Toward the use of gesture in traditional user interfaces. In *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition (FG '96)*, FG '96, pages 151–, Washington, DC, USA, 1996. IEEE Computer Society.

[KKL07] Song-Gook Kim, Jang-Woon Kim, and Chil-Woo Lee. Implementation of multi-touch tabletop display for hci (human computer interaction). In *Proceedings of the 12th international conference on Human-computer interaction: interaction platforms and techniques*, HCI'07, pages 854–863, Berlin, Heidelberg, 2007. Springer-Verlag.

[Lee10] Johnny Chung Lee. In search of a natural gesture. *XRDS*, 16:9–12, June 2010.

[LF11] Sheelagh Carpendale Lawrence Fyfe, Adam Tindale. Junctionbox: A toolkit for creating multi-touch sound control interfaces. In *International Conference on New Interfaces for Musical Expression*, June 2011.

[LRZ10] Uwe Laufs, Christopher Ruff, and Jan Zibuschka. Mt4j - a cross-platform multi-touch development framework. *CoRR*, abs/1012.0467, 2010.

[Mat09] Manuel A. Matos. Kombination von multitouch-gesten und spracheingabe für interaktive oberflächen. Technical report, Institut für Informatik und Wirtschaftsinformatik der Universität Duisburg-Essen, November 2009.

[MS94] Thomas H. Massie and Kenneth J. Salisburg. The PHANToM haptic interface: A device for probing virtual objects. In *Proceedings of the 1994 ASME International Mechanical Engineering Congress and Exhibition*, volume DSC 55-1, pages 295–302, Chicago, IL, USA, November 1994.

[Mul11a] MultiTouch. Multitouch cell advanced, 2011. `http://multitouch.fi/products/celladvanced/`.

[Mul11b] MultiTouch. Multitouch cornerstone, 2011. `http://cornerstone.multitouch.fi`.

[MV11] Christopher Denter Mathieu Virbel, Thomas Hansen. Kivy: a crossplatform framework for creating nui applications, 2011. `http://kivy.org/`.

[NMS10] Meza-Kubo V. Nava-Munoz S., Moran A. L. Context-aware notifications: A healthcare system for a nursing home. *Intelligent Interactive multimedia systems and services*, 6:241–250, 2010.

[NSA10] Francisco Nunes, Paula Alexandra Silva, and Filipe Abrantes. Human-computer interaction and the older adult: an example using user research and personas. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '10, pages 49:1–49:8, New York, NY, USA, 2010. ACM.

[O'R04] Cameron O'Rourk, editor. *A Look at Rich Internet Applications*. Oracle Magazine, 2004.

[OW04] Sejin Oh and Woontack Woo. Manipulating multimedia contents with tangible media control system. In *ICEC*, pages 57–67, 2004.

[PRV09] Thomas Niedzielski Prasad Ramanahally, Stephen Gilbert and Desirée Velázquez. Sparsh ui: A multi-touch framework for collaboration and modular gesture recognition. *ASME-AFM 2009 World Conference on Innovative Virtual Reality (WINVR2009)*, 2009.

[Ret94] Marc Rettig. Prototyping for tiny fingers. *Commun. ACM*, 37:21–27, April 1994.

[RHM$^+$09] Jochen Rick, Amanda Harris, Paul Marshall, Rowanne Fleck, Nicola Yuill, and Yvonne Rogers. Children designing together on a multi-touch tabletop: an analysis of spatial orientation and user interactions. In *Proceedings of the 8th International Conference on Interaction Design and Children*, IDC '09, pages 106–114, New York, NY, USA, 2009. ACM.

[SBD$^+$08a] Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi, Laurence Muller, Patrick Olivier, Tim Roth, and Ulrich von Zadow. Multi-Touch Surfaces: A Technical Guide. Technical report, October 2008.

[SBD$^+$08b] Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi, Laurence Muller, Patrick Olivier, Tim Roth, and Ulrich von Zadow. Multi-Touch Surfaces: A Technical Guide. Technical report, October 2008.

[SCG10] Dominik Schmidt, Ming Ki Chong, and Hans Gellersen. Handsdown: hand-contour-based user identification for interactive surfaces. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 432–441, New York, NY, USA, 2010. ACM.

[Sny03] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (Interactive Technologies).* Morgan Kaufmann, 1 edition, April 2003.

[SRP07] Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction.* Wiley, 2 edition, March 2007.

[ThT09] Hsien-tsung Chang Tsai-hsuan Tsai. Sharetouch: a multi-touch social platform for the elderly. *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference*, pages 557–560, 2009.

[vHB01] Christian von Hardenberg and François Bérard. Bare-hand human-computer interaction. In *Proceedings of the 2001 workshop on Perceptive user interfaces*, PUI '01, pages 1–8, New York, NY, USA, 2001. ACM.

[VLN09] Paul Varcholik, Joseph J. Laviola, Jr, and Denise Nicholson. Tactus: A hardware and software testbed for research in multi-touch interaction. In *Proceedings of the 13th International Conference on Human-Computer Interaction. Part II: Novel Interaction Methods and Techniques*, pages 523–532, Berlin, Heidelberg, 2009. Springer-Verlag.

[Wal06] D. Wallin. Touchlib homepage, 2006. `http://www.whitenoiseaudio.com/touchlib/`.

[WFM03] Matthew Wright, Adrian Freed, and Ali Momeni. Opensound control: state of the art 2003. In *Proceedings of the 2003 conference on New interfaces for musical expression*, NIME '03, pages 153–160, Singapore, Singapore, 2003. National University of Singapore.

[wik11a] NUI wiki. Diffused illumination (di), 2011. `http://wiki.nuigroup.com/DI`.

[wik11b] NUI wiki. Dsi, 2011. `http://iad.projects.zhdk.ch/multitouch/?p=90`.

[wik11c] NUI wiki. Dsi, 2011. `http://wiki.nuigroup.com/DSI`.

[wik11d] NUI wiki. Frustrated total internal reflection (ftir), 2011. `http://wiki.nuigroup.com/FTIR`.

[wik11e] NUI wiki. Llp, 2011. `http://wiki.nuigroup.com/LLP`.

[Wik11f] Wikipedia. History of video games, 2011. `http://en.wikipedia.org/wiki/History_of_video_games`.

[Wik11g] Wikipedia. Serious game, 2011. `http://en.wikipedia.org/wiki/Serious_game`.

[YH07] G.N. Yannakakis and J. Hallam. Modeling and augmenting game entertainment through challenge and curiosity. *International Journal on Artificial Intelligence Tools*, 16:981–999, 2007.