UNIVERSIDADE DE COIMBRA

"Interactive Membranes - rapid interface prototyping for pervasive and augmented reality games"

Pedro Machado Santa

University of Coimbra
Faculty of Science and Technology
Department of Informatics Engineering
August 31, 2012

Advisor:
Prof. Dr. Licínio Roque

Abstract

Interactive Membranes is an approach for rapid interface prototyping for mobile-based, pervasive and augmented reality games. This project extends previous work done on our laboratory on the design of a Pervasive Games platform - that provides an online service for game editing and shared game context management - and aimed to design and build a proof of concept for a mobile touch-based interface building technology. This project follows a Design Science Research approach and tries do address the following questions: How do you help a broad public (age 8+; without any programming skills) to build game interfaces? What compromises will that solution make in terms of game design possibilities? Besides the extension of the aforementioned project such a technology would, on one hand, satisfy the ongoing need by professionals of more flexible pervasive gaming prototyping tools and, on other hand, enable a more participatory culture around this kind of pervasive gaming experiences.

Keywords

game authoring tools; augmented-reality games; pervasive games; mobile interaction; interaction design; design science research; mobile cross-platform development;

Acknowledgments

This research project would not have been possible without the support of many people. I would like to express my gratitude to my supervisor, Prof. Dr. Licínio Roque, who was abundantly helpful, offered invaluable assistance, encouragement and guidance. My deepest graditude and love to Paula, my everything, for her unmeasurable love and support. Gratitude is also due to the all colleagues and friends of the Laboratory, namely the doctoral students Rui Craveirinha and Luís Pereira, for sharing the literature, invaluable assistance and careful review of my work. I would like to express my special gratitude and thanks to all the persons that participated in the usability tests for giving me such attention and time. Special thanks also to my graduate friend David Neto for his tireless support. My love and gratitude to my beloved families; for their understanding and endless love, through the duration of my studies. I would also like to convey thanks to the IPN and the Computer Engineering Department of the University of Coimbra, for providing the financial means and laboratory facilities. A thank you note to the chaps over at Github for providing me a private educational account for free.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Who wouldn't want to make computer games?
It's creative, rewarding, and these days even pretty darn cool too.
You can make them to share with your school friends, your work colleagues,
your grandchildren, or even the entire gaming world.
– Jacob Habgood and Mark H. Overmars [11]

1.1. Context

Johan Huizinga, often considered the forefather of game studies, placed play as an "*free activity standing quite consciously outside 'ordinary' life*" that "*proceeds within its own proper boundaries of time and space according to fixed rules and in an orderly manner*"[13]. This special place in time and space created by games was referred by him as the *magic circle*.

This was in the first half of the XX century when 'game', and 'gaming' was viewed very differently from today standards. Since then with the advent of computer games and and ubiquitous computing - on our lives and culture - the magic circle boundaries have become increasingly thin and blurred - games began to occupy part of our daily reality. Like so, Katie Salen and Eric Zimmerman reinterpreted these boundaries more metaphorically as not being absolute: "*(...) the boundary between the act of playing (...) and not playing (...) is fuzzy and permeable.*"[23]

It's in this context that pervasives games emerge as a "*game that has one or more salient features that expand the contractual magic circle of play spatially, temporally, or socially*"[17] blending, altering or augmenting our reality. Therefore the pervasive game has, for one, specific characteristics that have to be considered in the game design process, and, secondly, many potential applications, such as, physical gaming, augmented story telling, urban exploration, cultural and heritage learning, etc.

The technical complexity of game can be a huge barrier to game creation by end-users limiting the emergence of a participatory culture around games. Game development still needs complex technical skills that only few people master. But for the task of

designing a game, no special equipment or programming skill is *strictly* necessary.[24] It is between the opportunities offered by pervasive games and the problem of low technological accessibility of game development that this project will be developed.

1.2. Project Description and Objectives

In this project we want to extend previous work done on our laboratory on the design of a Pervasive Games platform based on augmented reality techniques that uses mobile devices (smart phones and tablets), QR tagging, geo-positioning and orientation, together with other contextual services, to enable quick design and deployment of game play in real space. Currently, this platform architecture provides a generic mobile interface (for Android), that enables game exploration in open contexts, coupled with an online service for game editing and management of shared game context. On this project was identified the need for a technical solution for quickly drawing and deploying augmented reality game interfaces. Then, the purpose of this project is to design and build a proof of concept for a mobile touch-based interface building technology, aimed at pervasive and augmented reality games, that draws inspiration from interactive membrane concept proposed by Goffman: a "*screen [that] not only selects but also transforms and modifies what is passed through it*".[10] Therefore, the present project has the following stated objectives:

- Enable rapid prototyping of new 2D game interfaces for pervasive and augmented reality games targeted at mobile devices (smart phones and tablets);
- Take advantage of the already existing working Pervasive Game platform infrastructure;
- Portability for the major mobile platforms;
- Make game interface building and configuration as easy as possible and accessible to the broadest user base (ages 8+; without formal programming skills).

1.3. Document Structure

This document details the activities, the circumstances and the results from the development of this project, using the following structure:

- **Introduction** - Give some context for this work, describe the project and it's main objectives;
- **State of the Art** - Explore the state of the art in terms of augmented reality and pervasive gaming, game authoring tools and cross-platform mobile development;

- **Methodology** - Explain the methodology followed in the development of this project and present the project activities plan;
- **Iteration 1 - Technical Demo** - Documentation regarding the process of development of the first design science research iteration;
- **Iteration 2 - First Prototype** - Documentation regarding the process of development of the second design science research iteration;
- **Iteration 3 - Second Prototype** - Documentation regarding the process of development of the third design science research iteration;
- **Conclusion** - Major conclusions of the project and of the design science research process;

Chapter 2.

State of the Art

2.1. Introduction

On the start of this project a State of the Art research was made with the purpose of answering three questions:

1. What kind of pervasive games exist and how they are designed and built?
2. Which game authoring tools exist and what is their approach to the game creation process?
3. Which mobile cross platform tools and environments exist and which features they support?

To answer the first question we explored some literature and case studies on Pervasive Gaming mainly to know which genres of pervasive games exist and how they are designed. We found that this research was important because it allowed us to know more about the type of games and what characteristics of the pervasive games design process our application will have to address. On the second question we gathered, analyzed and described several game authoring tools. In this section we were focused on knowing how these tools address the game creation process and which limitations they have. Finally, for the third question, we gathered and analyzed several mobile cross platform development tools and environments. The results of these activities are documented on the following sections of this chapter.

2.2. Augmented Reality and Pervasive Gaming

Markus Montola et al. placed the Pervasive Game as "*a game that has one or more salient features that expand the contractual magic circle of play spatially, temporally, or socially.*"[17]. Spacial expansion it's about games embracing their environments and contexts, meaning that games can take advantage of the terrain, landscape, buildings,

location, physical objects that surround them and even appropriate cyberspace. Temporal expansion relates to the time appropriation by pervasive games, like non-pervasive games often have somewhat contained play sessions while pervasive games often compete for time and attention with our daily living, therefore blending themselves on our routines. Social expansion is the social phenomena that pervasive games produce, like promoting engagement and interaction with outsiders (or unknown players) or by extending roles to spectators of the pervasive game - in some pervasive games spectators of the game can have a role that helps the pervasive game players accomplish their goals.

The previous authors also propose 8 pervasive game genres which help us to define the features and properties of these games and provide information for the design and analysis of pervasive games. The genres are:

- **Treasure Hunts** - games where players try to find certain objects - uncover a planted prize, find a certain location, locate a very specific everyday object, etc. - in an unlimited game-space. A similarity can be established between this type of games and *geocaching* activities;
- **Assassination Games** - a type of track-and-kill game inspired by the film *La decima vittima* (1965) that usually involve tracking a specific player and (virtually) murder him before being eliminated;
- **Pervasive LARPs** - games that extend the traditional LARPs (live-action role-playing games) in a similar way as Assassination Games but incorporating acting, role playing and character representation elements;
- **Alternate Reality Games (ARGs)** - games that take the substance of everyday life and weave it into narratives that layer additional meaning, depth, and interaction upon the real world;
- **Smart Street Sports** - games that require both physical exercise and tactical thinking. They are usually played outside in urban areas or on university campuses and commonly seen either as updated, technology-enhanced versions of the sporty neighborhood games of youth or as physical variants of popular digital games;
- **Playful Public Performances** - games similar to smart street sports but more geared to creating fun through performing, playing, and creating a spectacle;
- **Urban Adventure Games** - games that combine stories and puzzles with city spaces. These games take the player to areas with some historical or cultural significance to solve puzzles and follow stories, but also to learn tidbits about the history of a place;
- **Reality Games** - games that consciously play with the concepts of real and reality, often more paideic than ludic, seeking to encourage players to see and experience their living area in a new and different way.

Furthermore the authors provide a few insightful tips on issues that game designers need to take care when designing a pervasive game, namely: a) sustaining a critical mass of players; b) carefully setting the pace of the game; c) creating engaging experiences and d) designing pervasive games for casual audiences.

Another interesting essay that explore and map the field of Pervasive Games is the one provided by Magerkurth et al.[15] In it the authors also define pervasive game genres, along with some examples, that serves as a valuable insight into the possibilities of this type of games. The defined genres are:

- **Smart Toys**, traditional toys augmented with pervasive computing technology;
- **Affective Gaming**, a genre that aims to integrate a player's emotional state into the game so that the game environment can adapt to create a "magical game experience";
- **Augmented Tabletop Games**, combine the benefits of computer and tabletop games into a novel type of augmented tabletop game that sets out to provide new and engaging gaming experiences;
- **Location-Aware Games**, similar to the augmented tabletop games but make use of the whole world as it's game board;
- **Augmented Reality Games**, advanced pervasive games that make use of augmented reality, a variation on virtual reality that draws virtual objects into a real-world environment.

Wetzel at al.[26] also analyzed several augmented reality games and compiled a set of useful game design guidelines when considering this types of games. The following recommendations are given by the author:

- **Experiences First, Technology Second** - Design the experience first then consider the relevant technologies;
- **Stick to the theme** - Select technologies which are relevant to aspects such as time period and ambience;
- **Do not stay digital** - Use a combination of real and virtual elements such as paper maps;
- **Use the Real Environment** - Make use of the real world location, beyond simply locating virtual elements in a real space;
- **Keep it simple** - Design interaction schemes which are easy to understand and use;
- **Create Sharable Experiences** - Allow other people to take part in the experience, for example by using tablet PCs and the magic lens metaphor rather than a head mounted display;

- **Use Various Social Elements** - Allow players to interact with virtual characters, other players, non-players and actors;
- **Show Reality** - Do not augment spaces so that the underlying real components are totally obscured;
- **Turn weaknesses into strengths** - Use potential technical problems as elements within the gaming experience;
- **Do not just convert** - Do not simply convert a game to augmented reality;
- **Create meaningful content** - The 3d content in the game should add something interesting to the game;
- **Choose your tracking wisely** - Different tracking methods have different characteristics that should be taken into account.

Following this overview of pervasive games we analysed two relevant case studies to our work as well as to understand eventual hands-on issues and conclusions that arise from pervasive game design and development.

2.2.1. Research Projects

REXplorer

REXplorer[3][4] is an interesting pervasive game project that promotes the exploration and touristic awareness of a city. The game uses location sensing to simulate player encounters with virtual spirits (historical figures) that are associated with historical buildings in the urban setting. The player is then invited to "cast a spell", by making a specific gesture waving a mobile phone through the air, to awake the spirit and communicate with him, or learn more about the site in question, or to receive and resolve quests. The tourists are placed as scientific assistants responsible for examining paranormal activity, recently discovered outdoors in the Regensburg medieval city core, within one hour. This sets the stage for this pervasive game event.

For this project the authors designed a special "paranormal activity detector" - a device composed of a Nokia mobile phone and a GPS receiver in a protective shell - to interact with location-based and site specific spirits. The device, seen on Figure 2.2, was adjusted for simplicity on the protective shell having only a simple set of buttons directly related to the game, a screen to display the spirit cues, and an audio speaker to supply audio feedback.

REXplorer has applied a player-centered iterative design throughout the design process because "*the requirements for an interactive system cannot be completely specified at the beginning of the lifecycle.*"[4] The design was prototyped and tested with real users to reveal any false assumptions or unforeseen problems with the existing design, the problems corrected on the next iteration of the prototype, which was, again, tested to

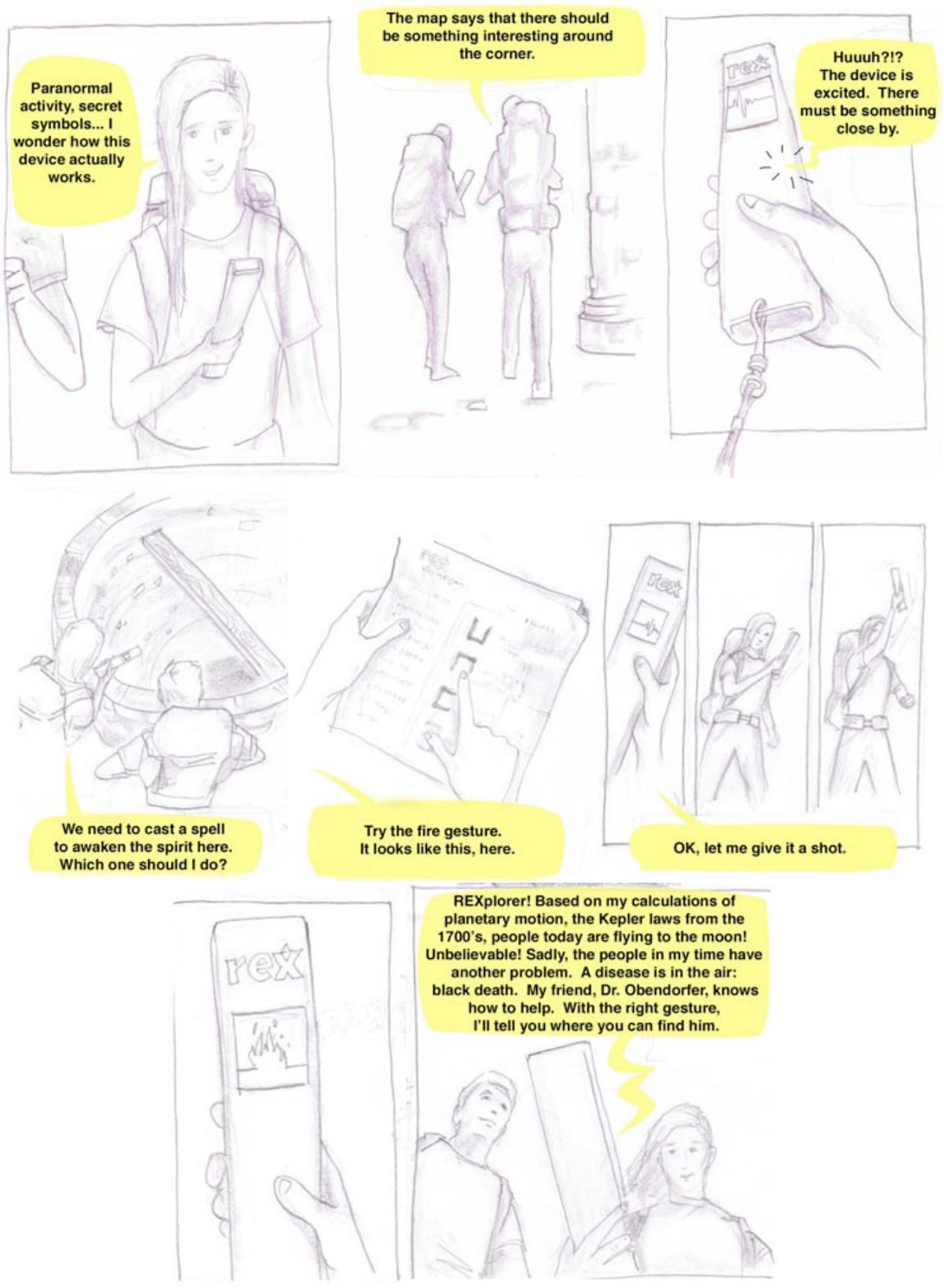

Figure 2.1.: Storyboard conveying the REXplorer game play.

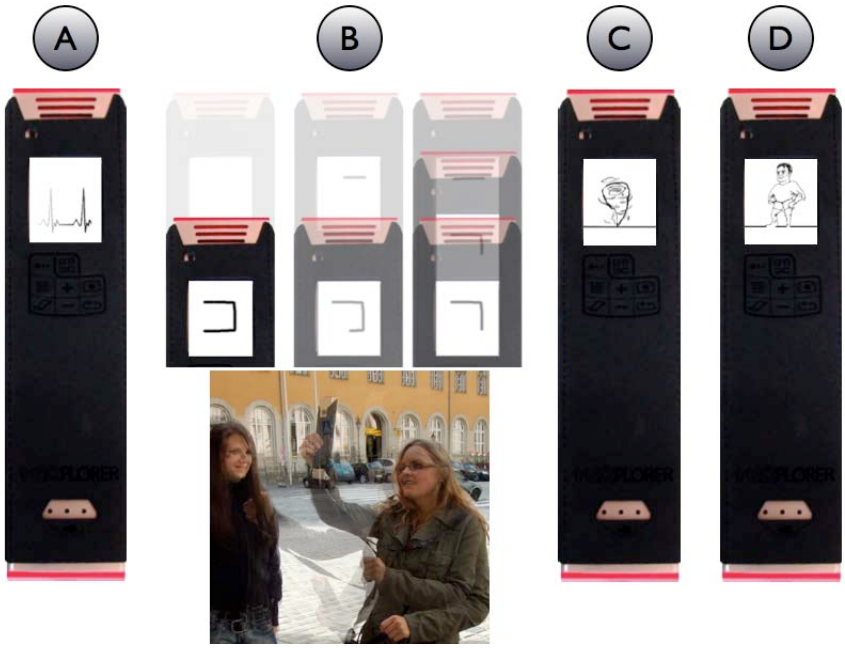

Figure 2.2.: The behaviour of the REXplorer device along the game play steps.

ensure the problems were resolved and so forth. The design process of REXplorer was composed by the following steps:

1. Early Concept Prototyping, with the discussion of several game ideas, stakeholders meetings, scenarions and storyboards creation (Figure 2.1), a first physical prototype in the form of a board game and design document with functional and prototype specifications. This demonstration and simulation material was then used to collect feedback from the target group - two German high school classes 10th grade and 11th grade - using playtesting sessions, a questionnaire and a focus group discussion.

2. Board game Prototyping, that worked both as a demonstration tool and a worl-in-miniature that allowed the gameplay to be easily tested. It was important for early stage content testing, express spaciality, get a feeling for travel times, oversee proximity of sights, achieve narrative consistency and help to ensure that the underlying game was fun.

3. Content Prototyping, with the help of early character sheets that included book-keeping elements, physical characteristics and information for spoken text inspiration (typical quotes, character motivations, content and information checklist, etc.) Then a work of narratice production followed bridging the characters so that they would be connected meaningfully, as well as emotionally, through quests.

4. Game UI Statecharts, helpful to define exactly what text needed to be written for each character, to game state validation and error verification before the script

was written and recorded, and to help the software implementation of the game engine.

5. Hotzone Prototyping, since GPS can have problems in urban spaces due to buildings or even clouds obstructing signals from the satellites, the location system was tested thoroughly to ensure proper functionality. The hotzones were defined iteratively based on GPS measurements.
6. Detector Prototyping, where different materials and designs were tested.
7. Playability tests, on-site and followed by focus group discussions.

At the end, the authors conclude that the significant evolution of the game development, from its initial conception, was driven by the use of iterative design techniques and helpful feedback from tourists. "*Traditional desktop iterative-design techniques, such as paper prototypes, are of limited use for pervasive games because they only capture a very limited part of the player experience.*"[4] "*Our use of these [Human-centered iterative design techniques and Game Design] techniques to pervasive game design helps demonstrate how player-centered iterative design improves the quality of pervasive games.*" This approach also brought advantages over the communication between stakeholders. "*Early conceptual prototypes were important as communication tools that helped convey our ideas to both end-players and other stakeholders. Their feedback was extremely formative and helped identify critical conceptual stumbling blocks that helped us frame controversial elements such as spell-casting, our techno-magical theme, and the threat of gameplay taking priority over tourism.*"

The authors also stress the fact that aren't many tools available to help the iterative player-centered design on pervasive games: "*Currently, we are lacking the tools, and conceptual frameworks to fully support iterative player-centered design in the domain of pervasive games, because existing methodologies for the desktop computing, such as paper prototypes, do not scale to ubiquitous computing applications. A desktop environment is targeted for one user, one set of hardware, and a single point of focus. In pervasive games, complexity is added in every direction; there are multiple players and player backgrounds, dynamic contexts of use, diverse spatial qualities, different metrics for successful interfaces, and varying stakeholder as well as political and economical interests that may change over time.*"

HeartBeat

HeartBeat[12] is an outdoor pervasive game designed for children (ages 8-12) that "*puts outdoors play center stage, combining the benefits of traditional outdoor games with the opportunities for richer experiences and innovation offered by new media.*" Besides, it explores the use of physiological sensing and relies on heart rate measurements of the players as input to the game and as a way to enhance the pervasive gaming experience.

This game can be seen as a *Capture the Flag* type of game combined with elements of traditional games such as '*hide and seek*' and '*tag*'. The game is labelled, by the authors, a Head-Up Game (HUG), a category of "*pervasive games played with minimal but flexible equipment, supporting physical activity, rich social interaction, flexible and open-ended game rules, space for imagination and role play.*" Additionally, one of the game objectives "*is that game-play should resemble playing traditional (non-computer supported) outdoor games of the past as far as the opportunities for physical activity and social interactions offered to the players, while at the same time bringing the benefits of computing and communication technology into the game play (such as applying complex rules, timing, enhanced sensorial and interactive experiences, balanced rewards and challenges).*"

To play the game, each player is given a small portable device (Figure 2.3) and players are randomly allocated between two teams; the attacking team (attackers) and the defending team (defenders). Initially all players are unaware of the role distribution. Once the game starts, players get 30 seconds to hide. After this period of time, their role is displayed to them through their device and one player in the defending team is randomly assigned a virtual treasure. Then the attackers need to seek out the defenders and chase them up in order to tag them. Once an attacker tags a defender, the defender joins the attacking team, therefore the players are not eliminated from the game but remain involved until the end - a problem indicated explicitly by the children on the first prototypes during the iterative design of the game. By physically connecting two of the gaming devices (top to bottom), player roles can be passed on from one player to another. This makes it also possible to pass on the virtual treasure in the defending team. A game is played for a short duration (4 minutes) and if the player with the treasure is not caught within this time, the defending team wins the game. If the attacking team has been able to capture the treasure, they win the game.

The game was designed in an iterative process involving children from the very early stages. Initially a group of 5-7 children from the target age-group play-tested the game rules with paper prototypes. The design process went through 8 iterations, play testing in a suburban outdoor environment and trying out different game rules. Following these play-tests another 6 play test sessions were ran with children in a forested area, before arriving at a fun and playable set of rules.

The evaluation of the playtesting was made with help of video recording and analysis and focus groups with the participants. The authors concluded that the children strongly preferred to play the game outdoors and in contact with nature, like on a forest. Besides, numerous and varied comments were made by the children regarding physical aspects of play - reasons to enjoy tagging: "Then you are most active"; reasons to like or dislike hiding: "I like to sneak around", "I can't stay put"; etc. Another conclusion of the authors is that the players valued the social interaction built in the game - "I think that teamwork is an essential part of a game". Overall, children understood and enjoyed the game, were physically active and in social interactions engendered by the game.

Figure 2.3.: Heartbeat, two players chasing each other in a game session (left) and the game device (right).

2.3. Game Authoring Tools

In this section we analyse several game authoring tools - development environments, game builders, makers, etc - that try to tackle the problem of technical complexity of game development. The tools being analyzed are: *GameMaker*, *Scratch*, *GameSalad Creator* and *Codea*.

2.3.1. GameMaker (1999)

GameMaker is an early graphical game authoring tool for Windows and Macintosh originally developed by Mark Overmars in 1999 - by then still a 2D animation software called "Animo". Current stable version it's 8.1 and since 1999 it gained many new features, notably 3D graphics support, a significant user base and a new developer and publisher, YoYo games, a software company in which Overmars became involved in.

GameMaker provides a simple environment that allows beginners to quickly start building games, using an icon-based system of events and actions. A drag-and-drop programming technique provides an easy way to learn about game development and allows the user to create complete games without going near a traditional high-level programming language. Once the user becomes more experienced GameMaker also provides the Game Maker Language (GML) which offers a more powerful way to program games.[11]

Games in GameMaker are organized into several resources[1]:

- Objects, which are the true entities in the game and can take certain actions when events happen;
- Rooms, the places (or levels) in which the objects live;
- Sprites, (animated) images that are used to represent the objects;
- Sounds, used in games, either as background music or as effects;
- Backgrounds, the images used as background for the rooms;

In addition to those, a number of other types of resources are offered (on an Advanced Mode) for more complex games, such as, paths(used to let an object follow a set path) and scripts (code capable of taking arguments in, and returning a certain return value.), fonts (for different text types), among others.

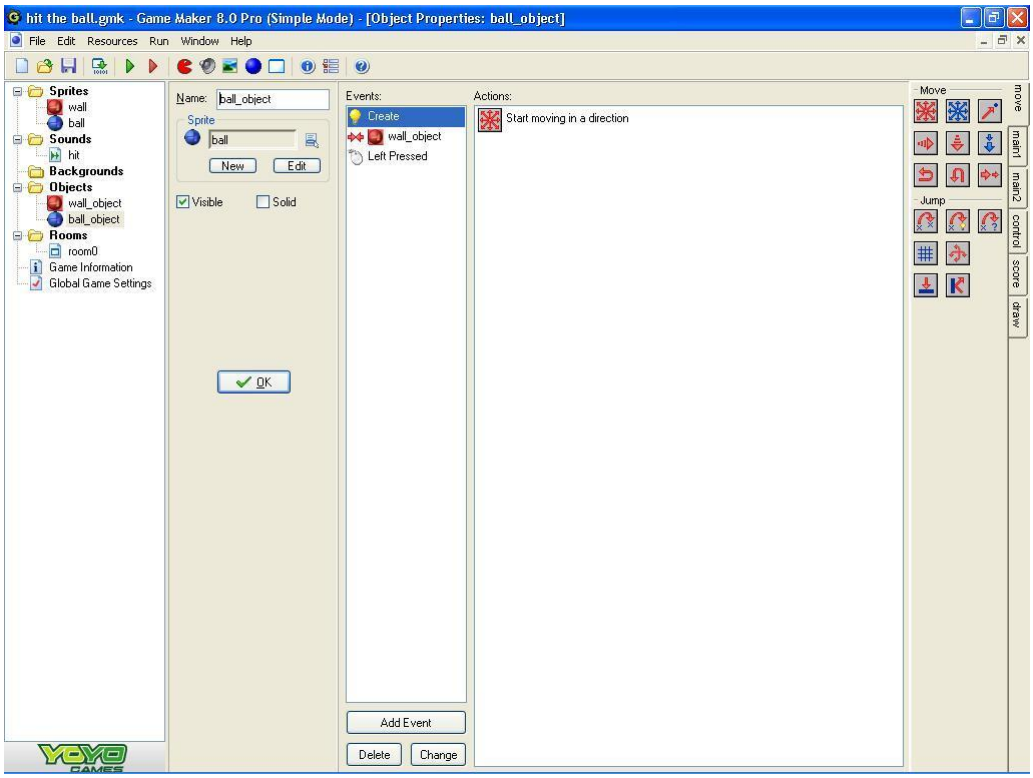

Figure 2.4.: GameMaker user interface.

The GameMaker UI (Figure 2.4) is composed by the usual menu and toolbar (top), the resources explorer (left) and a sort of work area (right). For each type of resource a diferent editor window appears on the work area where you can define and edit the given resource properties.

In the end, the Game Maker does a good job in terms of the possibilities it gives for game making - and since users can resort to a formal game programming language (GML)

there are very few limitations on this authoring tool - but clearly a great compromise is made on the ease of use and acessibility.

2.3.2. Scratch (2006)

Scratch is a LEGO inspired visual programming environment for desktop computers (Windows, Mac OS and Linux) that allows users (targeted at 8 to 16) to learn computer programming while working on personally meaningful projects such as animated stories and games. It started in 2003, as a project of the MIT Media Lab, and the Scratch software and Web site[1] were publicly launched in 2007. As the slogan of the project goes, "Imagine, Program, Share", one of the distinct features of the Scratch project is it's Web site that makes possible the sharing, browsing, running and editing of the projects made on the authoring tool, thus enabling a strong and vibrant community and the emergence of a learn-by-example behaviour using the several readymade projects available on the site.

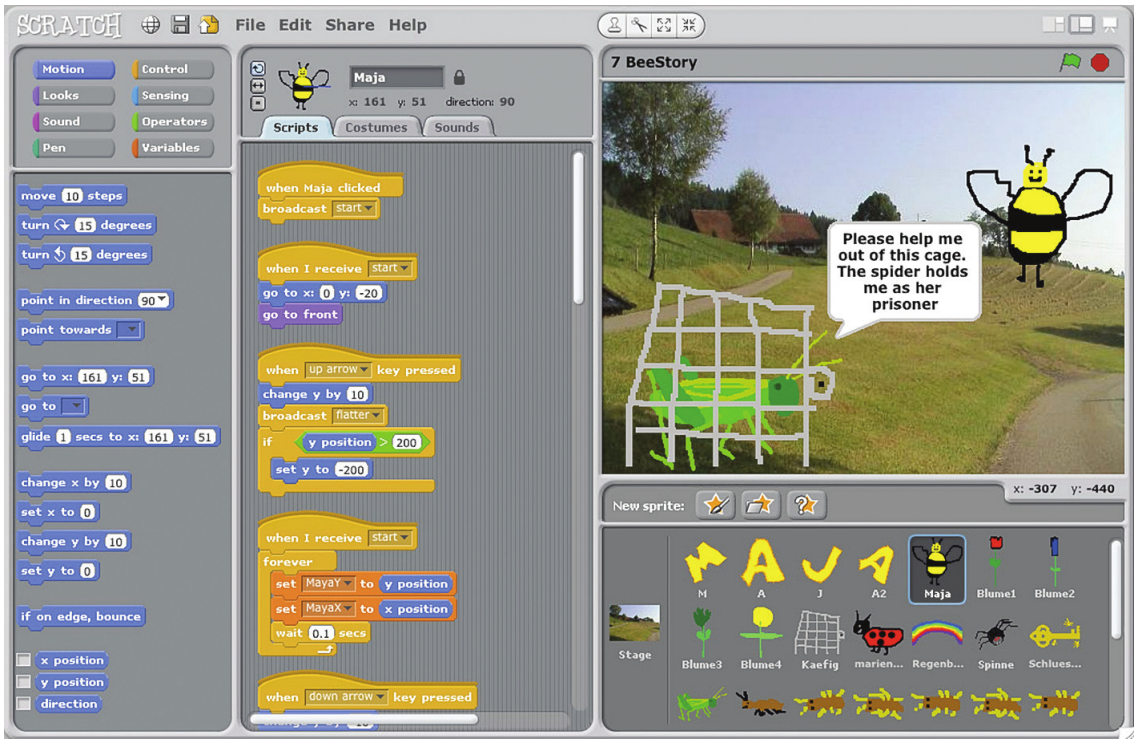

Figure 2.5.: Scratch programming environment user interface.

The Scratch UI is based on a building blocks metaphor, similar to the popular LEGO pieces and assembly, that allows children to explore by dragging, dropping and joining

[1] See `http://scratch.mit.edu`.

blocks of conditions (with parameters) and of consequences ("actions") onto selected agents (called "sprites") and backgrounds (called "stages"). The user interface for the Scratch development environment (Figure 2.5) divides the screen into several panes: on the left is the blocks palette, in the middle the current sprite info and scripts area, and on the right the stage (backgrounds) and "sprites" list. The palette is organized into eight groups of blocks with different colors and shapes: movement, looks, sound, pen, control, sensing, operators, and variables.

Scratch was designed with the "low-floor" (easy to get started), "high ceiling" (opportunities to create increasingly complex projects over time) and "wide walls" triplet in mind; supporting many different types of projects so people with many different interests and learning styles can all become engaged.[22] For this the project had three core design principles: be more tinkerable, more meaningful, and more social than other programming environments.

The design of Scratch, and it's simple albeit limited language, resulted in some limitations and tradeoffs such as no file I/O option, supports for only one-dimensional arrays and limited string manipulation capability. Overall, the Scratch programming environment and language work together to create a system that is exceptionally quick to learn - users can be programming within fifteen minutes - yet with enough depth and variety to keep users engaged for years.

2.3.3. GameSalad Creator (2009)

GameSalad Creator is a game authoring tool for Mac OS X aimed primarily at non-programmers for composing games in a drag-and-drop fashion utilizing visual editors and a behavior-based logic system. It is also used by professionals and game developers for rapidly prototyping, building and publishing cross-platform (iOS, Android, Web/HTML5 and Mac OS X) games and interactive media. Besides the authoring tool, the GameSalad web site[2] works as a complete platform letting users publish and browse games built with the tool, create user profiles, browse extensive documentation, engage with other users through a forum, and buy and sell GameSalad Creator resources through a marketplace - thus establishing a extremely rich social development network around games.

In GameSalad Creator the user starts with a blank project or with a project based on several templates the tool provides (Figure 2.6). The games are thus organized into several Scenes that contain several game objects with rules and behaviours called Actors (Figure 2.7). Scenes can be split into layers (similar to many design programs, such as Photoshop or Illustrator). Layers provide another level of (visual) organization for the game, and they allow the user to group objects within the scene and arrange them in front of or behind other layers/objects. For example, one layer may contain the background, another may contain all the labels, another for actors in the scene that

[2]See `http://gamesalad.com`.

the player can interact with, etc. Actors represent the visible objects within a game (Figure 2.8). Actors can represent the character that the player is controlling or they can be the surrounding objects/characters that the player talks to, collides with, jumps over, or generally interacts with during gameplay. Behaviors are components of an actor that can either instantaneously or persistently affect the actor depending on rules that compose them. The application comes with a library of behaviors (for movement, changing attribute states, affecting collision, saving, etc.) that can be inserted into rules and other behavior groups to create new behavior. In addition to Behaviours, Actors can also have several Images and Sounds.

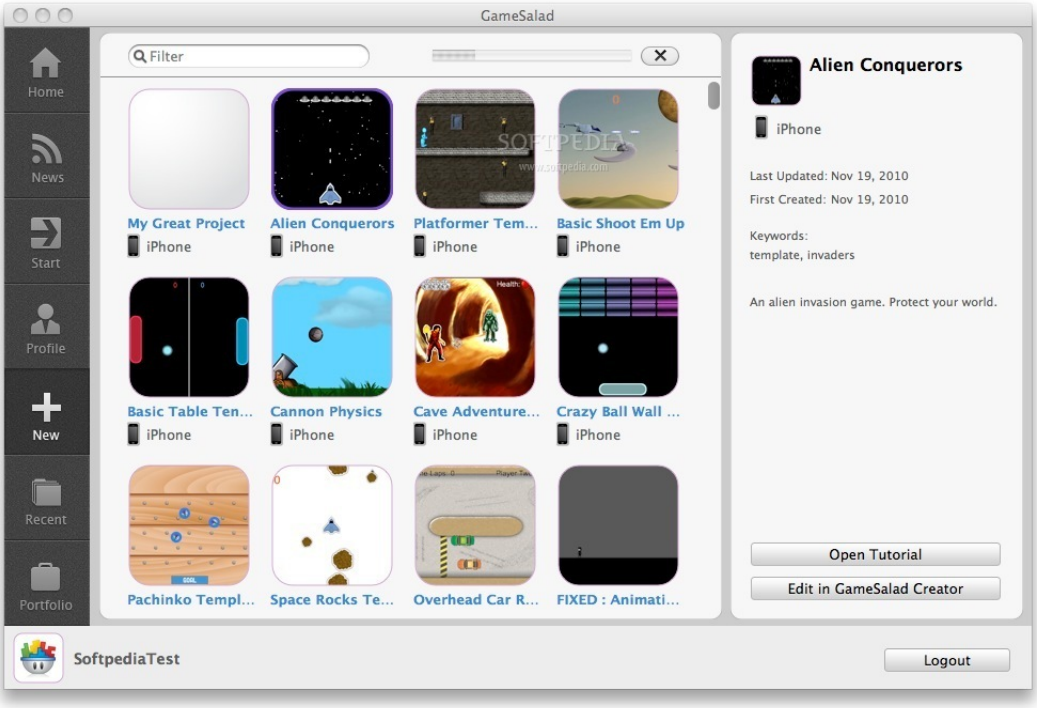

Figure 2.6.: GameSalad Creator user interface when the user creates a new game with several templates to select.

GameSalad Creator today is limited and focused on 2D game authorship, but there are plans to support 3D game authorship in future releases. It is notorious the focus on ease of game development on GameSalad Creator - the tool has a fine polished highly usable interface, presents a clear and consistent interaction conceptual model, etc - but, in terms of possibilities, still presents some constraints because the users are limited to the rules and behaviours that the platform offers - there isn't a feature to extend these like a scripting language.

Figure 2.7.: GameSalad Creator scene edit user interface.

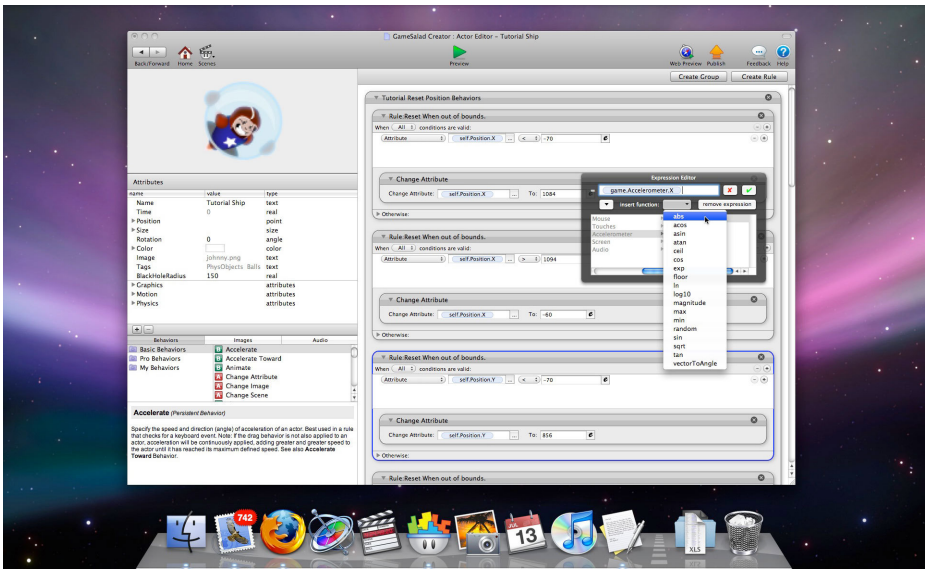

Figure 2.8.: GameSalad Creator actor edit user interface.

2.3.4. Codea (2011)

Codea (formerly known as Codify) is a touch-enhanced code editor for the iPad. It uses the Lua programming language and it's suited for creating games, simulations or visual ideas and programs. The touch-enhanced features of this editor include, among others, tap-and-drag features for changing numbers, colors and images within the app's visual

editor. Unfortunatly, because of Apple Store restrictions, the downloading or sharing of the projects isn't possible - Codify users won't be able to import or export code to or from the app, meaning their iPad creations remain locked on their tablets. Likewise, and again because of Apple Store restrictions, users can't really add custom made resources to their games, having to rely on a big resource library provided by the Codea makers. The app still has built-in extensive documentation - so the users can check any detail while developing.

Figure 2.9.: Codea start screen where you can select examples to play and create new projects.

Without further ado, Codea really isn't far from a Lua code editor (probably the first really good attempt at such an editor on the iPad) with some merit on the novel touch-enhancements it brings to the table. Furthermore, it's notorious the limitations the Apple Store imposes on custom content on it's platform that seriously put in jeopardy, for the time being, any effort of enabling a participatory culture around this application. But for simple simulations and game prototypes Codea fits the job properly: "Due to the limited nature of the environment, the tool is mostly useful for experimentation and prototyping, as you won't be able to send your creations to anyone else. Still, it's an interesting idea and pushes the iPad into more content creation areas."[3]

[3]See `http://www.macrumors.com/2011/10/26/codify-brings-touch-based-programming-to-ipad`

Figure 2.10.: Codea editor screen with a color picker helper interface.

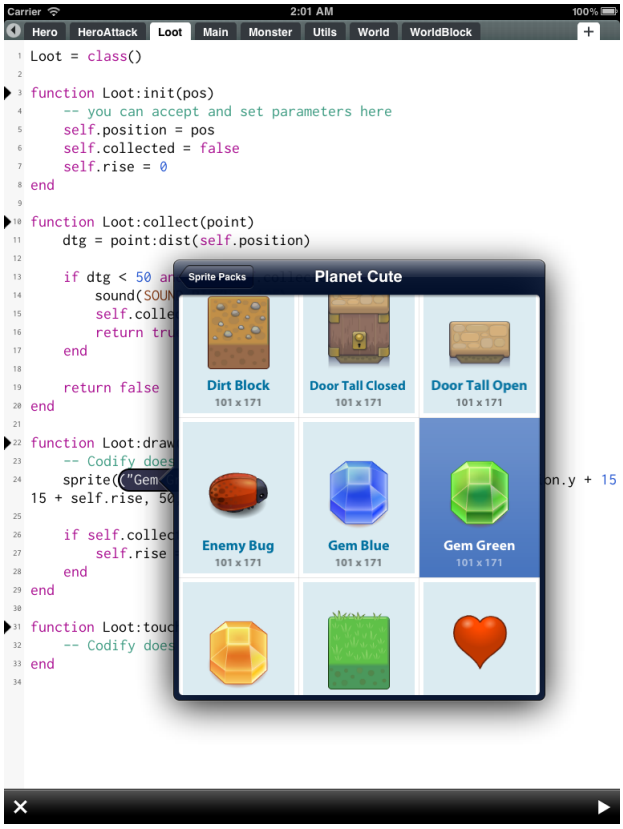

Figure 2.11.: Codea editor screen with a sprite picker helper interface.

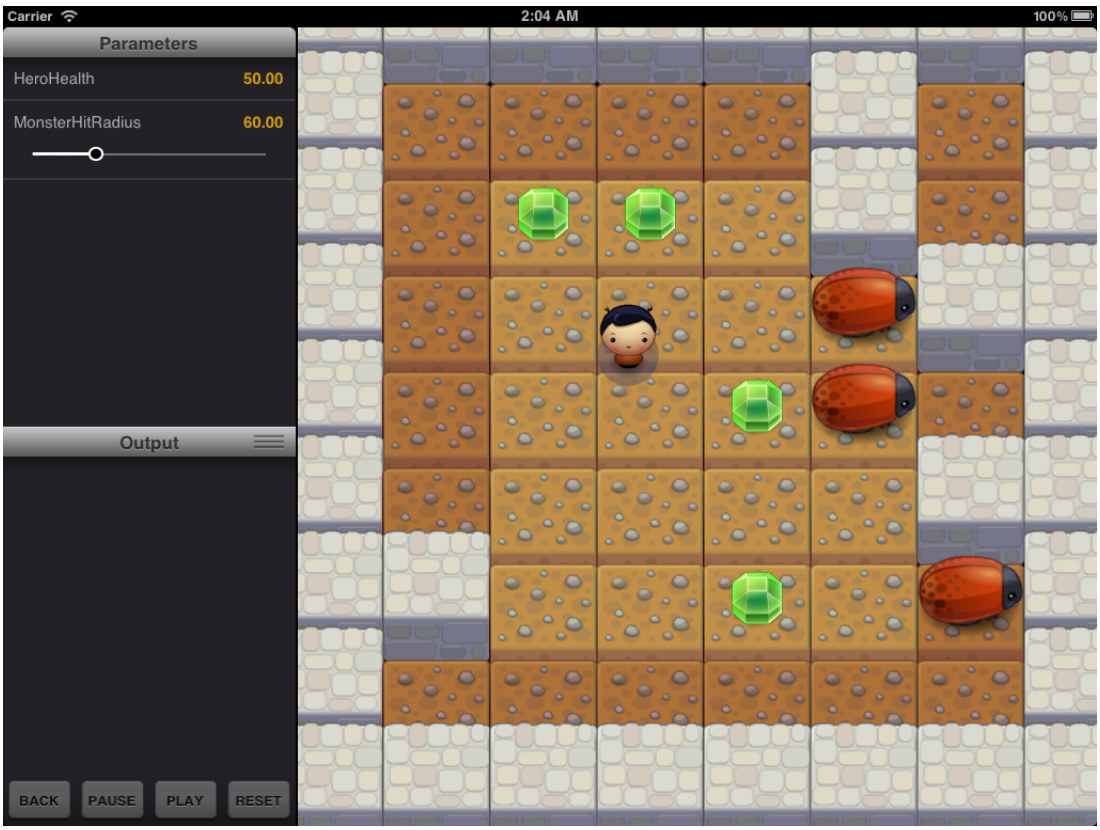

Figure 2.12.: Playing a game in Codea. On the left side bar it's visible the parameter pane, the output pane and some control buttons.

2.4. Crossplatform Smartphone and Tablet Development

In order to achieve the project's objective of a portable mobile application, it was important to do a preliminary research on existing tools and frameworks that could support cross platform mobile development. Therefore, the following frameworks and development kits were analyzed:

- Appcelerator Titanium (`http://www.appcelerator.com`);
- Corona (`http://www.anscamobile.com/corona`);
- EDGELIB (`http://www.edgelib.com`);
- Marmalade (`http://www.madewithmarmalade.com`);
- MoSync (`http://www.mosync.com`);

- PhoneGap (`http://phonegap.com`) and Sencha Touch (`http://www.sencha.com/products/touch`);
- Unity3D (`http://unity3d.com`).

To analyze these tools we defined a set of comparison vectors based on the features and properties relevant to the development our project, namely:

- Supported platforms (Does it support at least Android and iOS?);
- Access to the device features:
  - Touch detection (Does it support single or multitouch handling? Has support for gestures?)
  - Camera (Is it possible to capture/handle photos or video? Has support for QR-codes? Is it possible to display live-video from the camera? Can it be overlayed with graphical elements?)
  - Accelerometer and Gyroscope (How many values/axis can be read from the device sensors? Is it possible to do 1-to-1 mapping?);
  - Compass;
  - GPS/Location (Which other location helpers/services the tool offers?);
  - Sound/Mic (Is sound recording supported? How many channels supports? Does it have 3D sound libraries? Does it support OpenAL[4]?);
  - NFC (Near Field Communications).
- Maps Support (Is possible to display Maps? Can Google Maps or a similar service be used to display maps?)
- Game helper features:
  - 2D/3D graphics support/manipulation (Which kind of 2D/3D manipulations are available? Does it support OpenGL ES?)
  - Networking/Multiplayer (How is the network communication handled? Is there any kind of multiplayer helpers?)
  - Embedded game engines (Which type of engines does the platform offer?)
- Performance (Does the platform compiles or interprets the code at run-time? Does examples exist of a smooth execution?)
- Programming Languages (In which programming languages the development can be done?)
- Cost (What's the cost of the plaftorm for commercial use?)

[4]See `http://connect.creativelabs.com/openal/default.aspx`.

- Famous Products/Showcase (Which are the most famous products built with the platform?)

Because of time constraints it was not possible to perform a more deep analysis and benchmarks to these tools, therefore, in each vector, a more brief and qualitative approach was made to compare the tools. The result of this research is available on the comparison tables D.2 and 2.2. The information contained in these tables was collected from the tool's official sites, official documentation and official discussion forums.

Frameworks	Supported Platforms	Device Features						
		Touch Detection&Handling	Camera	Accelerometer/ Gyro	Compass	GPS/Location	Sound/Mic	NFC
Appcelerator Titanium	iOS, Android (Windows, Mac, Linux)	Single-touch (multi-touch may be possible with plugin, Gestures support in beta)	Capture video and images. Live video support unknown.	Only Accelerometer x,y,z values.	Supported.	Supported, with additional features like reverse geocoding, etc.	Basic playback and recording.	Not supported.
Corona	iOS, Android	Multi-touch. No direct gesture support, implemented by hand through multitouch events.	Capture video and images. Live video support unknown. Overlays not supported.	Full 6-axis accelerometer and gyroscope (1:1 position mapping).	Supported.	Supported.	Playback with OpenAL support. Audio recording depends of native calls to OS.	Not supported.
EDGELIB	iOS, Android, Maemo 5, Symbian and 7 more (Windows, Mac, Linux)	Single-touch. Multitouch can be implemented by hand.	Capture video and images (frames). Live video overlays support probable, implemented by hand, access to raw video.	Only gyroscope/angle of the device, x,y,z angles.	Support unknown.	Not supported.	Neat "sound engine" for playback (Hekkus Sound System). Recording unknown.	Not supported.
Marmalade	iOS, Android, Blackberry, Bada, Symbian, Palm/HP webOS	Multi-touch. No direct gesture support, implemented by hand through multitouch events.	Capture video and images. Live video and overlays supported. AR games/applications showcase.	Only Accelerometer x,y,z values. Access to gyro maybe possible with plugin/extension.	Supported.	Supported.	Playback (24-channel software sound mixer). Record sound supported. Live record/analysis unknown.	Possible with extension.
MoSync	iOS, Android, Blackberry, MeeGo, Symbian, Windows Mobile, Java ME	Multi-touch. No direct gesture support, implemented by hand through multitouch events.	Capture video, live video and images. Overlay support is unknown.	Full 6-axis accelerometer and gyroscope (1:1 position mapping).	Supported.	Supported.	Basic playback supported. Recording unknown.	Supported.
PhoneGap (with Sencha Touch)	iOS, Android, Blackberry, Palm/HP webOS, Windows Phone 7, Symbian, Bada	Single-touch. Multitouch depends on the device browser support. Simple gestures out-of-the-box (pinch, slide, etc). Gestures have to be coded trough touch events.	Capture images. Video, live video and overlays is unsupported.	Only Accelerometer x,y,z values.	Supported.	Supported.	Basic playback and recording.	Possible with plugin.
Shiva3D	iOS, Android, Maemo 5, Wii, Symbian and much more (Windows, Mac, Linux)	Multi-touch. Gesture support unknown.	Acess to AR functionality through plugin.	At least accelerometer support.	Supported.	Supported.	Basic playback supported. Recording unknown.	Not supported.
Unity3D	iOS, Android, Web, Flash	Multi-touch. Some gesture support.	No access to camera.	Full 6-axis accelerometer and gyroscope (1:1 position mapping).	Not supported. Can be supported with plugin.	Supported.	High selection of filters and mixers for audio playback. Recording not supported.	Not supported.

Table 2.1.: Cross-Platform Mobile Development Tools Comparison Table, part 1.

Frameworks	Maps Support	2D/3D graphics support/ manipulation	Game-enabling Networking/ Multiplayer	Embedded engine	Performance	Programming Languages	Cost	Famous Products / Showcase
Appcelerator Titanium	Built-in.	Simple HTML/CSS manipulations.	Network access, HTTP client, sockets.	-	A mix of webviews and native code.	HTML5/CSS/Javascript	Free (49$/month for premium support)	-
Corona	Built-in.	Focus on 2D graphics manipulation. No direct 3D support.	Network access, Async LuaSockets/HTTP.	Physics.	-	Lua	199$/year (1 platform), 349$/year (2 platforms)	Magic Defenders (Clueless Ideas),
EDGELIB	Unsupported.	2D and 3D manipulation. OpenGL ES access.	TCP Sockets, HTTP Client, Bluetooth.	Colision detection.	Good performance based on the showcases.	C++	Free (with startup EDGELIB screen), 5000$ one-time license for Corporate	Syberia 2, Star Defense (ngmoco)
Marmalade	Not built-in. Could use the Google Maps API to download images and display them.	2D and 3D manipulation. OpenGL ES access.	Sockets and HTTP Client.	Physics engine.	Produces native optimized code. High performance.	C++, Lua (scripting)	Starts at 139$/year (with splash screen), 399$/year (no splash)	Need For Speed Shift (EA Mobile), Pro Evolution Soccer 2011 (Konami)
MoSync	Built-in.	2D and 3D manipulation. OpenGL ES access.	TCP Sockets, HTTP Client, Bluetooth.	-	Compiles to C++/C#.	HTML5/CSS/Javascript	Free for Open Source, 199$/year Pro commercial license	-
PhoneGap (with Sencha Touch)	Not built-in. Can use Google Maps API directly.	Simple HTML/CSS manipulations.	No information.	-	Uses web view. Low performance.	HTML5/CSS/Javascript	Free.	None.
Shiva3D	Unkown.	2D and 3D manipulation. OpenGL ES access.	Good Network stack. Offers Shiva Server for Multiplayer management.	Physics, audio and many more.	Produces native optimized code. High performance.	C++, Lua	Basic version at 169$	-
Unity3D	Unkown.	2D and 3D manipulation. OpenGL ES access.	Real-time networking support.	Physics, audio engine and more.	-	C#, Javascript, Boo/Python	From Free to 1500$ (Pro)	Battlestar Galactica Online (Bigpoint), Marvel Superhero Squad Online (Amazing Society & Gazillion), Rochard (Recoil Games)

Table 2.2.: Cross-Platform Mobile Development Tools Comparison Table, part 2.

2.5. Discussion

Discussing on the presented state of the art, and regarding to pervasive gaming, the genres detailed by Markus Montola et al.[17] are closer to the type of games we consider for our proof of concept. We envision that all of the described genres could be built with our approach to pervasive game authoring. In contrast, Magerkurth et al.[15] makes a more broad approach to pervasive games. From these genres we feel that our approach

is only suited for Augmented Tabletop, Location-Aware and Augmented Reality Games. We exclude Smart Toys and Affective Gaming because, the former, often needs custom-made physical objects on which the whole experience relies, and the latter, needs special equipment and sensors in order to function properly that, presently, go beyond those usually found on mobile devices.

The analyzed pervasive gaming literature also provides several recommendations about the design and development of pervasive games. We feel that some of those could be incorporated in our proof of concept, thus helping and guiding the game authors to create better games. For example, Markus Montola et al.[17] stresses the importance of sustaining "*a critical mass of players*". Then, having a general purpose technology that can run in the mobile devices of the potential players, along with game promoting and discoverability features, could be a helpful and valuable feature of our proposal. The remaining recommendations of Markus Montola et al.[17] are highly influenced by the specific decisions the game author makes about the game so, we will leave them at the game author consideration.

Regarding the Wetzel at al.[26] recommendations some have to be ignored due to the nature of our proposal. "*Experiences First, Technology Second*" can hardly be considered since we're proposing a very mobile devices and general purpose focused approach, hence (at least part of) the technology is already defined. Others not only could be used but even incentivated by our technology such as, "*Do not stay [only] digital*", "*Use the Real Environment*", "*Use Various Social Elements*", among others. One final note to two specific recommendations, "*Create Sharable Experiences*" and "*Show Reality*" which are, precisely, two challenges of pervasive game design our proposal.

On the two pervasive game design case studies analyzed, REXplorer[4] and HeartBeat[12], it's notorious the importance of prototyping and iterative approaches for these kind of projects. Moreover, the authors also feel the need for more flexible and ubiquitous pervasive game prototyping tools, "***Currently, we are lacking the tools, and conceptual frameworks to fully support iterative player-centered design in the domain of pervasive games, because existing methodologies for the desktop computing, such as paper prototypes, do not scale to ubiquitous computing applications.***"[4], which is also a purpose our approach clearly wants to satisfy.

On our analysis of the authoring tools we felt that, altough the authoring tools help the development process of a game, they still employ some kind of programming (or logical sequencing of steps/actions) that still needs (1) some motivation for prior devising more complicated sequences and (2) it needs familiarity with procedural and logical thinking, skills that children or the most technological iliterate adult usually do not have and even, sometimes, find that particular requirement unappealing. Therefore an even more accessible approach (probably making some more compromises on game design possibilities) to game development could be attempted.

Discussing the more technical part of this chapter, the comparison of cross-platform mobile development libraries and frameworks, three tiers could be identified:

- Powerful industry-proven frameworks (Unity3D, Marmalade & Corona);
- Early-stages (Shiva3D) or retiring outdated frameworks (EDGELIB);
- HTML/Javascript/Web to Mobile App libraries (Appcelerator Titanium, PhoneGap + Sencha Touch & MoSync).

This categorization was based mainly on the graphic and augmented reality capabilities and from relevant show cases of these frameworks. From these three tiers the most interesting to us is obviously the first one. In this first tier are the industry most used frameworks, still they have some distinctive characteristics worth analysing. For instance, Unity3D doesn't have, presently, acess to the video camera of the mobile device. Moreover, since it relies on it's proprietary IDE doesn't leave much room for further framework extension through custom made code. That left us with the Corona or Marmalade option. We opted for the latter because it has better 3D capabilites and better extensability - through Marmalade EDK (Extensions Development Kit). These are our remarks to the performed state of the art research work.

Chapter 3.

Methodology

3.1. Design Science Research

For the development of this project an approach similar to the Design Science Research in Information Systems proposed by Vaishnavi, V. and Kuechler, W.[25] was choosen. Design Science Research is a methodology that promotes the design of novel and innovative artifacts and it's analysis of use or performance in order to improve and understand the behaviour os certain aspects of Information Systems. Such artifacts might include algorithms, human/computer interfaces and system design methodologies or languages. The purpose of this methology is learning and generating new knowledge through building. "*Knowledge is generated and accumulated through action. Doing something and judging the results is the general model (...) a cycle in which knowledge is used to create works, and works are evaluated to build knowledge.*"[20] Although this process might seem to lack rigor, the process is not unstructured, the knowledge using and building is made through "*systems of conventions and rules under which the [desigh] discipline operates*"[20] that "*embody the measures and values that have been empirically developed as 'ways of knowing' as the discipline has matured.*"

In this methodology (Figure 3.1) the process begins with the *Awareness of a problem* - since design science research is called "improvement research"[25], this designation emphasizes the problem-solving/performance-improving nature of the activity. *Suggestions* for the problem solution are abductively drawn form the existing knowledge/theory base for the problem area. Then an attempt to implement an artifact according to the proposed solution is performed (*Development*). Partial or full implementations are then *Evaluated* - according to functional specification implicit or explicit in the suggestion. *Development*, *Evaluation* and further *Suggestion* are then iteratively performed in the course of the research design effort. The *New Knowledge* flow from partial completion of the cycle back to Awareness of the Problem is made through a *Circumscription* process. *Conclusion* indicates termination of the design project. The *Circumscription* process is especially important to understanding design science research because "*it generates understanding that could only be gained from the specific act of construction*"[25] that is precisely the nature of the problem we're trying to tackle with this project.

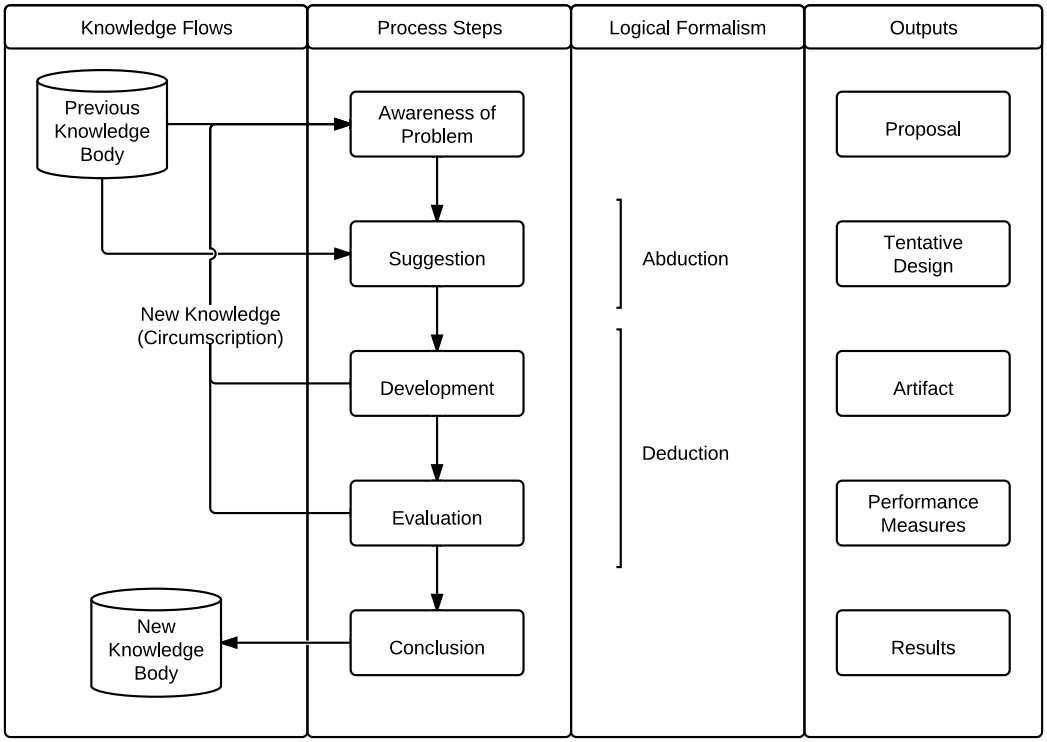

Figure 3.1.: The general methodology of design science research.

3.2. Planning and Milestones

For the planning of this project the following activities were defined:

- State of the art research (September 2011 - November 2011)
- Writing of a detailed proposal (December 2011 - 24th January 2012)
- **Milestone:** Intermediate delivery (24th January 2012)
- Iteration #1: (January 2012 - February 2012)
  - Design (February 2012 Week #1)
  - Development (February 2012 Week #2-#3)
  - Evaluation & Learning Statement (February 2012 Week #4)
- **Milestone:** Demo (29th February 2012)

- Iteration #2: (March 2012)
  - Design (March 2012 Week #1)
  - Development (March 2012 Week #2-#3)
  - Evaluation & Learning Statement (March 2012 Week #4)
- **Milestone:** 1st Prototype (31st March 2012)
- Iteration #3: (April 2012)
  - Design (April 2012 Week #1)
  - Development (April 2012 Week #2-#3)
  - Evaluation & Learning Statement (April 2012 Week #4)
- **Milestone:** 2nd Prototype (30th April 2012)
- Field testing and evaluation data collection (May 212)
- Data analysis and final tunings (May 2012 Week #3 - June 2012 Week #2)
- **Milestone:** Field testing report (June 2012 Week #3)
- Dissertation writing (June 2012 - 12th July 2012)
- **Milestone:** Final delivery (12th July 2012)

3.2.1. Adjustment

During the first Iteration some unexpected challenges regarding the technology being used forced us to revise the initial planning to accommodate a greater time period for each iteration. In that sense, the field Testing activity and it's milestone was discarded and the rest of the iterations increased in time. The revised milestones became:

- State of the art research (September 2011 - November 2011)
- Writing of a detailed proposal (December 2011 - 24th January 2012)
- **Milestone:** Intermediate delivery (24th January 2012)
- Iteration #1: (January 2012 - April 2012)
  - Design (February 2012 Week #1-#4)
  - Development (March 2012 Week #1 - April 2012 Week #3)
  - Evaluation & Learning Statement (April 2012 Week #4)
- **Milestone:** Demo (30th April 2012)
- Iteration #2: (May 2012 - June 2012)

  - Design (May 2012 Week #1-#2)
  - Development (May 2012 Week #1 - June 2012 Week #3)
  - Evaluation & Learning Statement (June 2012 Week #3-#4)
- **Milestone:** 1st Prototype (30th June 2012)
- Iteration #3: (June 2012 - August 2012)
  - Design (July 2012 Week #1)
  - Development (July 2012 Week #1 - August 2012 Week #1)
  - Evaluation & Learning Statement (August 2012 Week #2-#3)
- **Milestone:** 2nd Prototype (15th August 2012)
- Dissertation writing (August 2012)
- **Milestone:** Final delivery (31th August 2012)

Chapter 4.

Iteration 1 - Technical Demo

This first iteration focused on the design, development and testing of a first interaction model proposal and a first suitable software architecture that could support this particular interaction model.

4.1. Proposal

In this section we present the interaction concepts and deliverables developed for this first iteration of the application.

We began by developing a concept for the application based on a theatre metaphor that setted the tone and vocabulary for the next steps of the design. Next we planned the task and work model of the application. Afterwards a interactive low fidelity paper prototype was designed. Finally, a layout system optimized for touch interfaces of tablets that was developed and used to guide the more detailed design of the screens and the application user interface.

4.1.1. Concept

The aim of this interface technology is the construction of 2D game interfaces. So there are several particular features of games that we have to address - such as animations, mechanics and behaviours, etc - when developing such a technology.

Since we're aiming for pervasive and augmented reality game design, our starting point was the concept of interactive membrane: a membrane/layer between the user and the world which alters/augments the world view and with which the user could interact. The membranes can have transparent areas and move in the membrane plane. Thus, the device becomes an interactive window to an alternate world. (One could also consider a group of membranes as a stack, layered to build a particular interface.) A sketch explaining this concept is presented in the Figure 4.1.

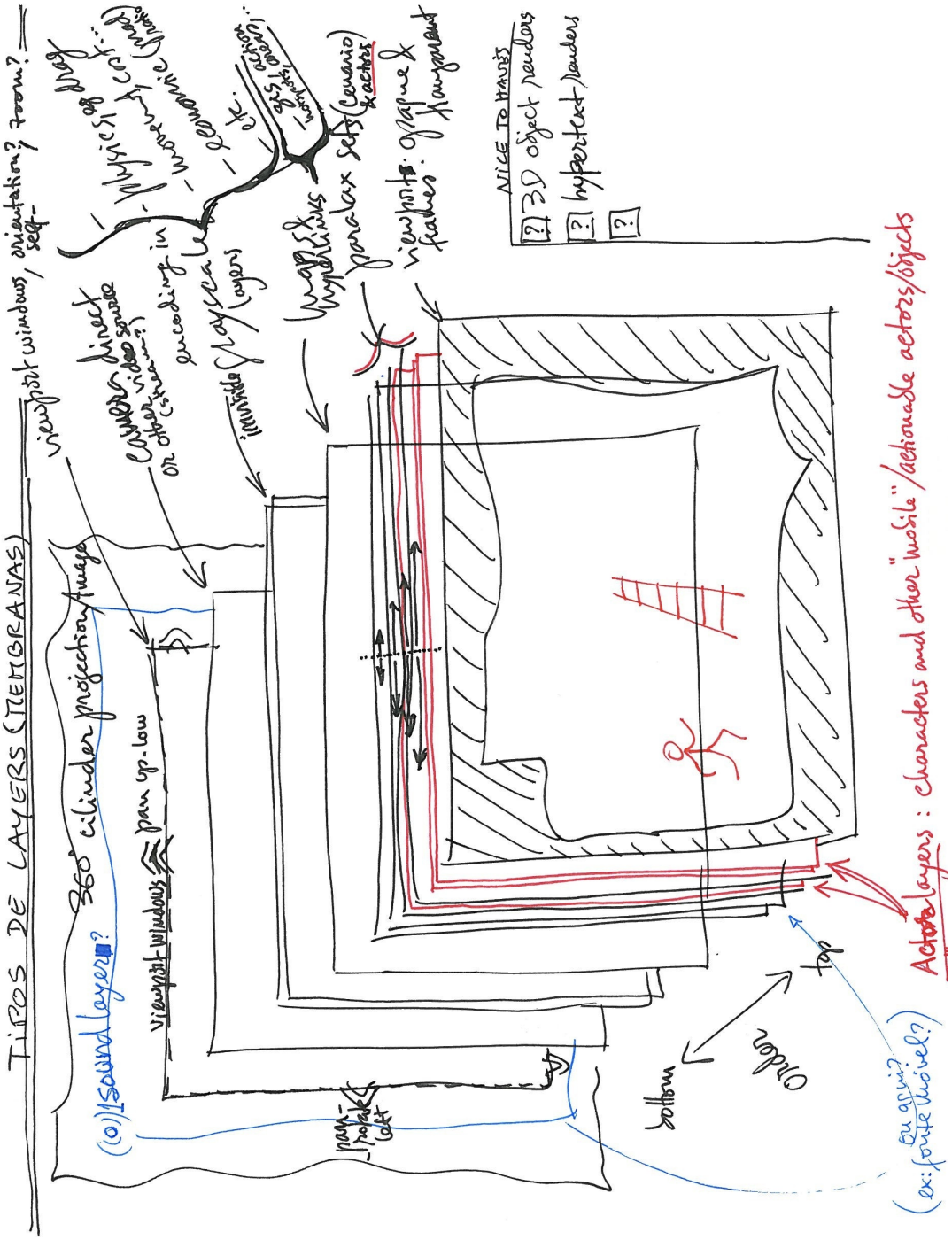

Figure 4.1.: A sketch of the Interactive Membranes concept.

Given this scenario, the metaphor of theatre - with it's several panes and planes - was considered. This metaphor was chosen because it fits with this particular exploration of the interactive membrane concept within the context of pervasive and augmented reality game design.

A concept map, detailing the terms and the relationships of this concept proposal, is provided in Figure 4.2.

By using the theatre metaphor we can consider a game analogous to a play with several scenes. Although, in a game, there isn't a fixed storyboard, or scene flow, but a directed graph based on the outcomes or the conclusion of a given scene. By organizing a game in several scenes we can therefore set up a game flow where a scene can be either a level, a particular part of the narrative or a game interface like a menu or options screen.

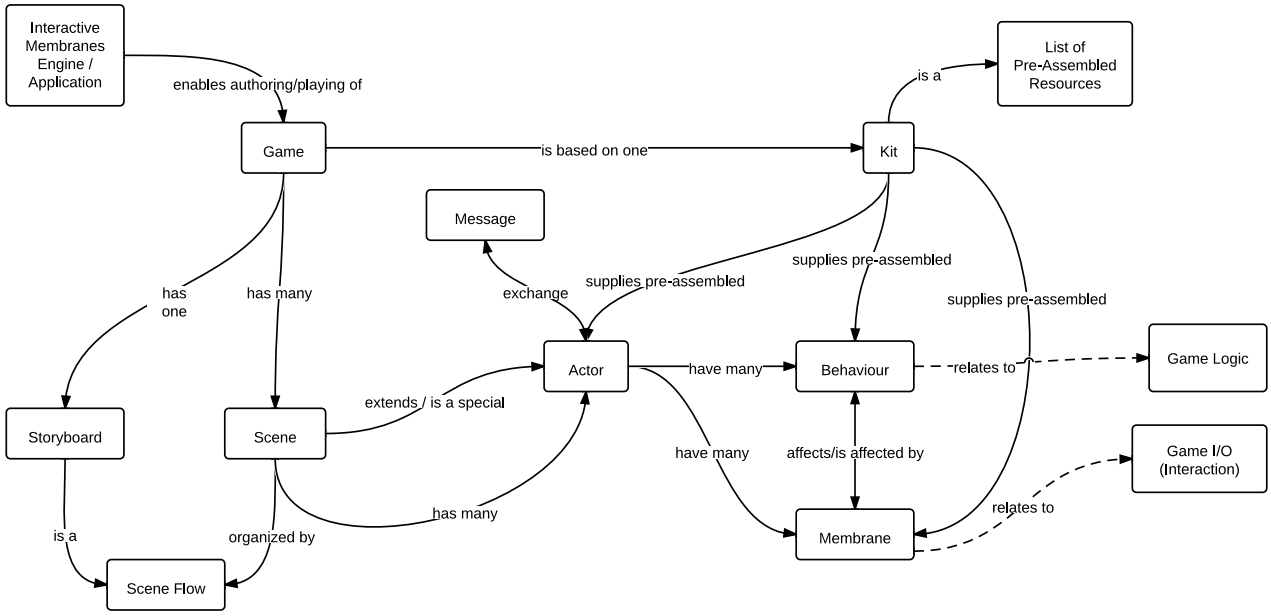

Figure 4.2.: Concept map with the vocabulary of terms and their relationships.

A scene is then composed of several actors, which in turn, are composed by several membranes and several behaviours. Also, a scene is considered an extension of an actor so it can have membranes and actions aswell. Regarding to theatre, these scene membranes and actions can be similar, respectively, to stage scenery/props and to backstage actions and control (lights, panes, scenery movimentation, etc.). Thus, a relation can be established between membranes and game interaction, or I/O, and between behaviours and game logic or mechanics.

Furthermore, actors can communicate by means of messages - to trigger certain behaviours and other relevant action - functioning like theatrical cues in a play. Finally, in

order to ease the task of game creation, one is based on a kit that provides pre-assembled actors, membranes, behaviours and scene outcomes. (The kits will be provided by the applicationand could be defined using XML.) With this concept the game creator is seen as the director of a game as interactive theatrical play.

4.1.2. User Environment Design

After the definition of the concept of the interaction mode we set to detail the tasks that would be performed with the application. Based on the previous work and from this analysis resulted a hierarquical task list depicted on the Figure 4.3.

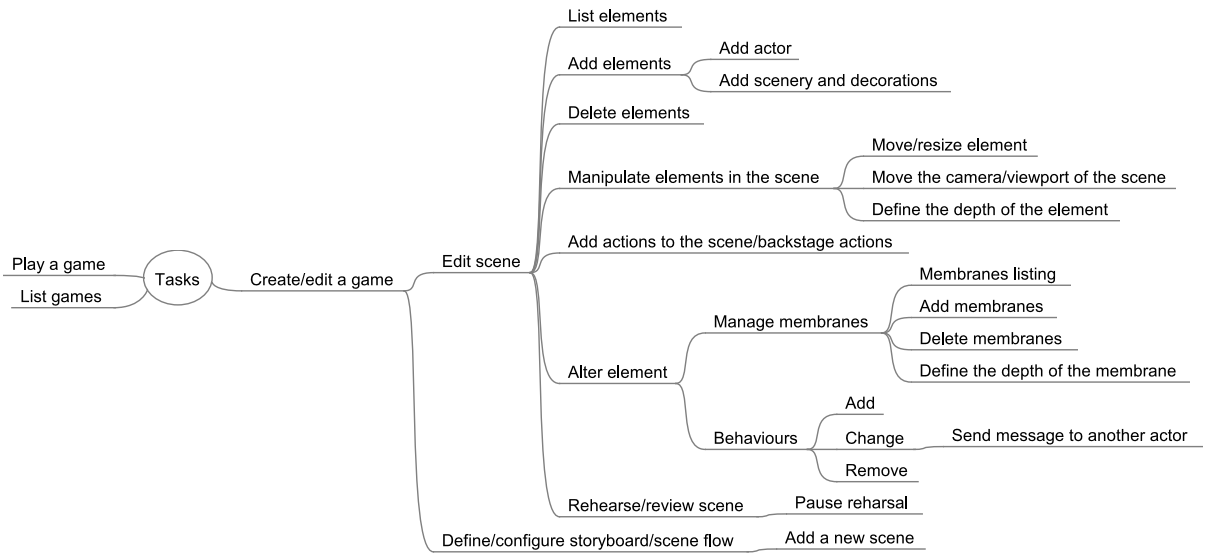

Figure 4.3.: Application hierarquical task list.

Next, a User Environment Design[5] was developed to document work model proposal for the application to guide the prototyping process. The resulting UED (depicted on Figure 4.4) has two major groups of focus areas: one for game creation and one for gameplay.

The gameplay encompasses the Game List (1) and Gameplay (2) areas. In these areas the user can view detailed information about a game, select a game to play and return back to the game list, either by exit the game normally, through a link provided by the game, or by a force exit mechanism.

Game creation revolves around the Game List (1), Kit List (3), Scene Edit (4), Actor Edit (5), Backstage Edit (6) and Storyboard (7). The game creation starts by following the link on the main focus area, Game List (1), to the Kit List focus area. This area is responsible for displaying all the game kits availabe to the user, along with a detailed description and links to game examples built with this kit. After selecting a kit a user

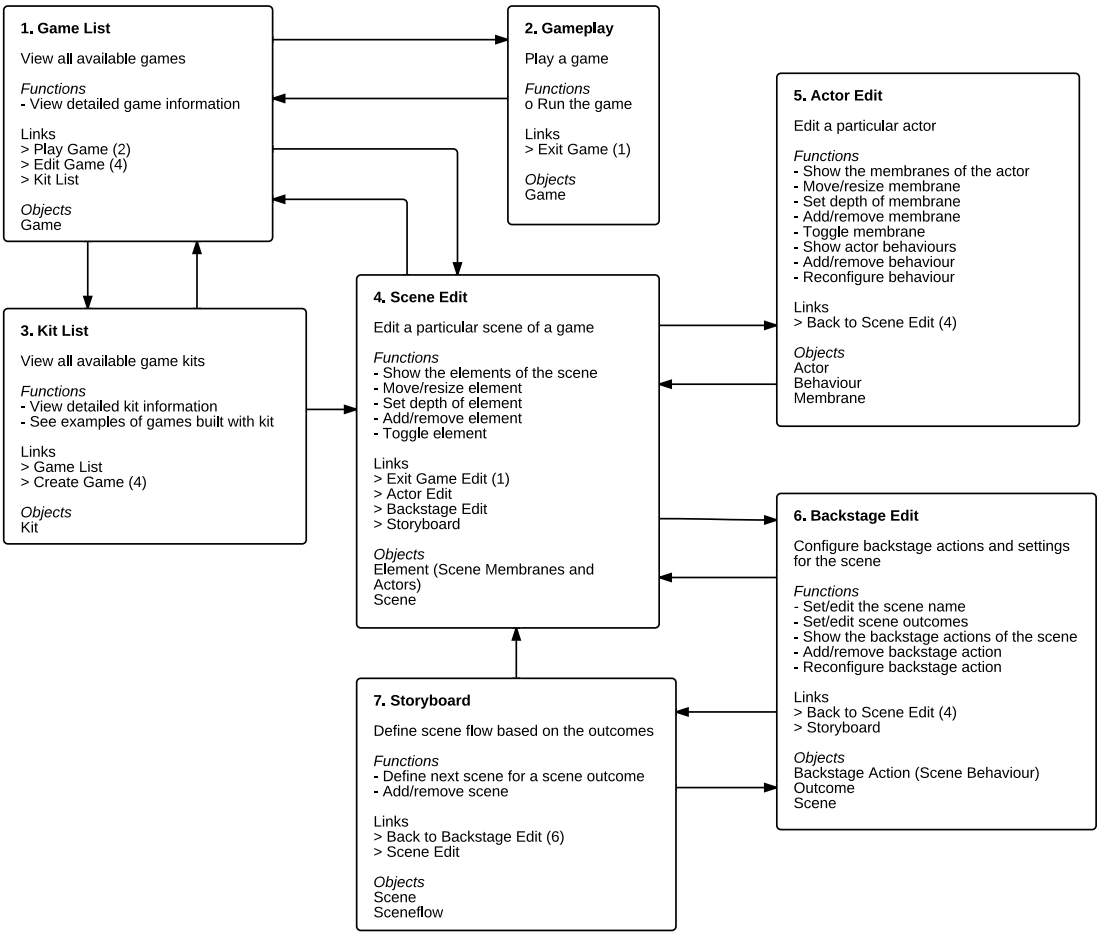

Figure 4.4.: User Environment Design for the proposed interaction model.

can create a new game based on it which will lead him to the Scene Edit (4) focus area. Although one may think that it would make more sense to go to the Storyboard (7) focus area first, since the game several scenes are organized on a storyboard, going to Scene Edit (4) first has relevant advantages: it puts the user manipulating objects and building games faster and hides the added complexity of processing the storyboard mechanism, that will be considerably easier to understand after the user grasps and experiments with the concept of scene first. From the Scene Edit (4) area the user can move to the Actor Edit (5) area, if it wants to set up and configure an actor while abstracting itself from the rest of the scene, or to the Backstage Edit (6) if it wants to perform configurations that relate to the scene itself.

4.1.3. Paper Prototyping

To conclude the interaction model a interactive low fidelity prototype was produced. The prototype was made using the Balsamiq prototyping tool (http://www.balsamiq.com) and published as PDF with links between pages thus simulating the interaction. Below the prototype several screens are presented grouped by focus area along with a brief explanation of the screens.

Game List

The Game List focus area is composed by screen depicted on Figure A.1. The user can swipe with finger through carroussel of available games to view a game description, play the game or edit the game. The user can also tap the create button on the bottom bar to go to the Kit List screen/focus area.

Gameplay

The Gameplay focus area is composed by screen depicted on Figure A.2. It's important that this screen contains a force quit mechanism, for cases such as game "freezing". For this prototype this mecanism was stated as the user tapping two oposite corners of the game with two fingers that leads the application to display a force exit confirmation.

Kit List

The Kit List focus area is composed by screens depicted on Figures A.3 and A.4. The Kit List screen functions similarly to the Game List screen (Figure A.1). After the user taps the create game button the second screen is displayed to set a name for the game.

Scene Edit

The Scene Edit focus area is composed by the following screens:

- **Scene Edit (Figures A.5 and A.10)** - This is the main editing screen. In it the user can manipulate the various visible elements that compose the scene, namely, moving a element by tapping it and dragging it around the screen and scaling/rotating it with a two finger pinch gesture. The user can also move, zoom in and zoom out the view of the scene. This is done through the same dragging and two finger pinch of the elements manipulation. But since these two interactions pose a conflict a switch or a toggle button is provided on the top right corner of the screen to toggle between elements manipulation and viewport manipulation. On the top left corner you can tap the arrow to exit the edit mode and go back

to game list. On the bottom there's links to add a new element to the scene (Add Element screen), list the scene elements (Scene Elements Listing screen), review the game (Scene Review screen) and to backstage tasks (Backstage focus area);

- **Scene Review (Figure A.6)** - This is the screen that appear when the user reviews the scene (the play icon on the Scene Edit screen). Tapping the bottom pause button takes the user to Scene Edit screen;
- **Add Element (Figure A.7)** - In this screen the user can add new elements to the scene. These elements are predefined with by the Kit.;
- **Scene Elements (Figures A.8 and A.9)** - In this screen the user can list all the elements ot the scene, order them by view depth, remove them or set their visibility. Tapping an element highlights it in the same way as the Scene Edit screen (Figure A.9). To order an element the user has to long-tap the element on the list and then it can drag it around the (depth) stack.

Actor Edit

The Actor Edit focus area is composed by the following screens:

- **Actor Edit (Figure A.11)** - This is the actor edit screen. In this screen appear all the membranes that belong to the actor while all the other scene elements are grayed out for the sake of focus the actions on the actor. Similar to the Scene Edit screen, it has a scene or elements toggle on the top right corner, a back to Scene Edit button on top left corner and, on the bottom, links to add new membranes do the actor (Add Membrane screen), list all membranes (Actor Membranes screen), scene review and actor behaviours management (Actor Behaviours screen);
- **Add Membrane (Figure A.12)** - In this screen the user can add new pre-defined membranes to the actor and get back to the Actor Edit screen;
- **Actor Membranes (Figure A.13)** - This screen functions similarly to the Scene Elements screen (Figure A.8) but only for the membranes that compose the actor;
- **Actor Behaviours (Figure A.14)** - In this screen the user can view the behaviours that the actor has and assign new ones. If a user clicks a particular assigned behaviour it goes to the behaviour edit screen where it can parameterize the behaviour triggers and effects;
- **Actor Behaviour Edit (Figure A.15)** - In this screen the user can parameterize a particular behaviour. The user can change and feed triggers to the behaviour by dragging the connectors from the membranes to the connectors of the behaviour. The same goes with the effects of a particular behaviour. When a connector is connected it will display a connection line. When the user tries to connect or change a connector the list of the membranes gets updated to only show compatible membranes with that connector (this compatibility is defined on the Kit). If

the number of certain connector is variable then when a user connects the first connector, immediately a new grayed out one appears below - is possible to see a greyed out example as the second "Walking Animation Frame" effect connector. Scene outcomes will appear as membranes aswell so the user can conclude a scene on specific outcomes.

Backstage Edit

The Backstage Edit focus area is composed by the following screens:

- **Scene Actions (Figure A.16)** - This is the screen where the user can assign or remove behaviours of the scene, since the scene conceptually is an actor. We opted to call scene behaviours "scene backstage actions" because semantically and in the metaphor context it makes more sense. If the user clicks a behaviour it will go to a screen similar to the Actor Behaviour Edit (Figure A.15);
- **Scene Properties (Figure A.17)** - In this screen the user can edit the scene name and assign or remove outcomes of the scene.

Storyboard

The Storyboard focus area is composed by screen depicted on Figure A.18. In it the user defines the game scene flow based on the scenes' outcomes by dragging the outcome connectors to other scenes. He can also create new scenes and change the scene being edited by tapping the "Edit" button.

4.1.4. Graphical Interface Design

Before the drawing of the detailed graphical interfaces we looked for a suitable grid to serve us as a guide which we could build and compose the interface upon. The iOS Human Interface Guidelines state that "*The comfortable minimum size of tappable UI elements is 44 x 44 points.*"[2], but using the iPad with 52 points per cm as a reference, that equates to a touch target of 1.18cm x 1.18cm which is less or as wide as a pointer finger width. Meaning that when the user taps the area the edges of that same area become invisible. So for our grid we used an area 50% bigger than that minimum (62 x 62 points). This allows the user's finger to fit better inside the target while the edges of the target are visible when the user taps it. This provides them with clear visual feedback that they are hitting the target accurately and they are also able to hit and move to their targets faster due to its larger size. This is consistent with Fitt's Law[9], which says that the time to reach a target is longer if the target is smaller.

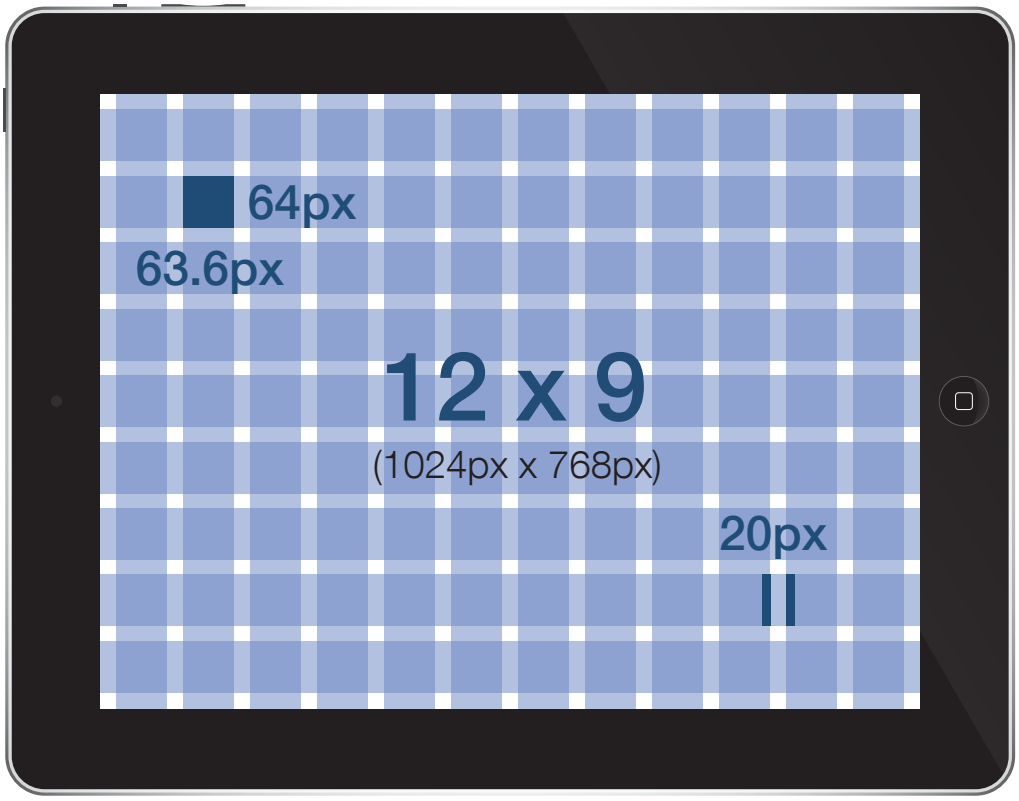

Figure 4.5.: The developed 12 x 9 grid used on the design of the graphical interfaces.

With this in mind and using the iPad's 1024 points per 768 points landscape resolution as a reference we designed a 12 columns by 9 rows grid with a 20 point gutter (Figure 4.5), which left us with the size targets of 64 points per 63.6 points to the minimum target area. The size gutter and the number of columns and rows was determined by trial and error to assure that the minimum touch area was squared. Using this grid we designed and composed detailed graphical interfaces mockups pictured on the Appendix B.

4.2. Architecture

After the work on interaction models we started the work on the architecture that would support our application. In this task we favored several best practices of software development such as the use of **software design patterns**, the consideration of higher-level **architectural patterns** and concern with **flexibility** and **reusability**.

The developed architecture is composed of three components: the domain logic of the application, a input manager and an output manager. The purpose of this was to create

an **abstraction layer** between the application and and the SDK being used - so that the application could be flexible enough to be more easily ported to other technology if need be. To increase the **reusability** of the architecture, extensive use of **interfaces** was made, namely by using interfaces to access the managers. This way if in the future the need to implement a different manager occurs, or to separate one of the managers into many, that could be possible while the application continues to work in a transparent way. In this way we can even implement several managers, which could be useful, for example, in a scenario where one would code a more fast module for physics processing. With this pluggable and modular approach the change would be seamless.

Regarding the higher-level architecture we wanted to try a different approach from the common Update/Render Game Loop model and experiment the feasibility of a more event-driven architecture using the **Model-View-Controller** architectural pattern[14]. This experiment is driven by the assumption that this game loop based architecture model, common on desktop gaming where power and energy are 'cheap', may not be the best approach to a game engine for mobile devices, where battery power is limited. A permanent cycle of processing even in static interfaces, when constant update and drawing isn't required, puts a constant strain on the battery of the device.

To implement such an architecture some design patterns proposed by Gamma et al.[8] we're also considered of assistance, namely the **Observer** and **Singleton** patterns - for example, the Observer pattern was used on the managers since they adopted several publish/subscribe interactions.

Regarding data persistence we have another component (Persistence) that is used to retrieve and save persistently the data needed to the function of the application. This component also makes use of an interface available to the application so that if we want to change the persistence method later a seamless transition would occur. For this iteration we planned to use a persistence component that persisted the data in XML files on the device's filesystem.

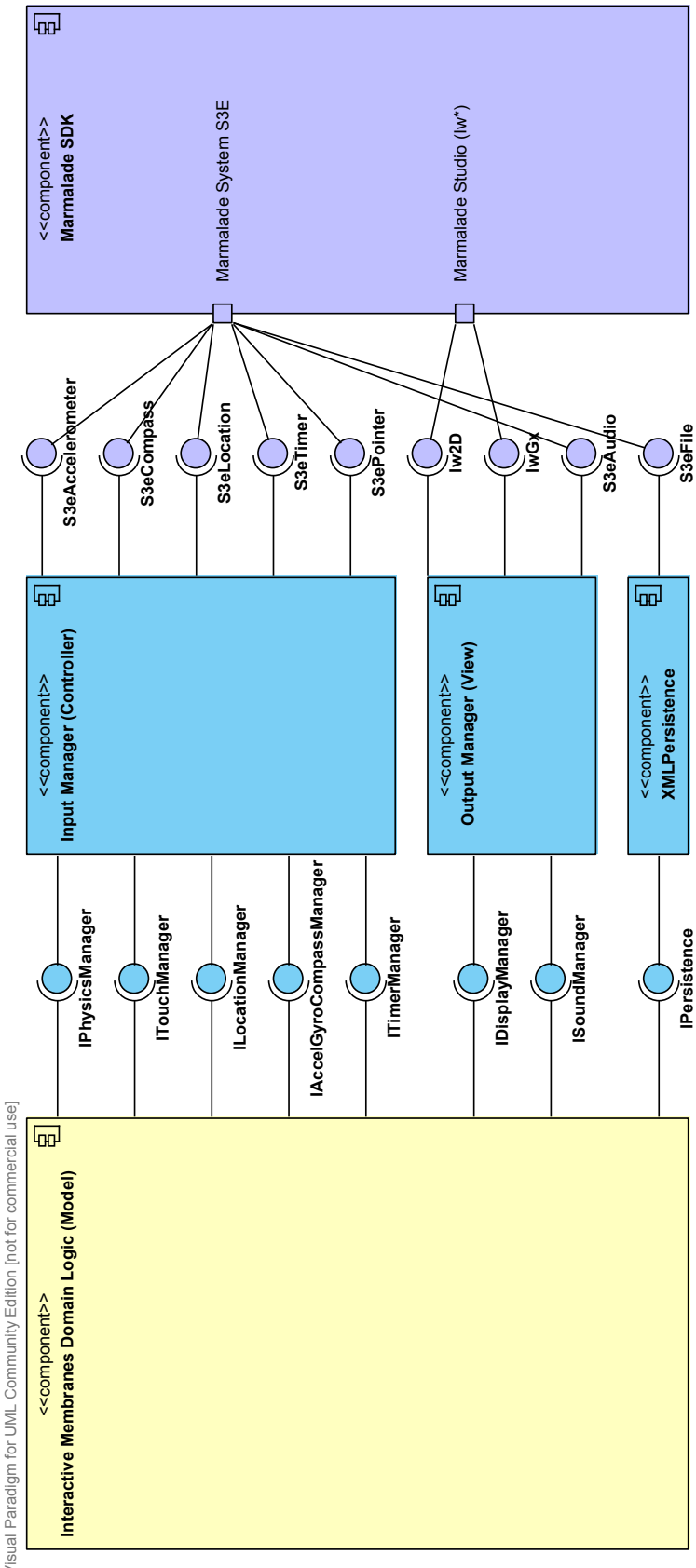

Figure 4.6.: The UML Component Diagram of the components of the proposed Interactive Membranes application architecture.

4.3. Implementation

The development of the first iteration artifact went from the first week of March to the third week of April. During that time the following sprints were performed:

- **Sprint #1:**
  - Preparation of the development setup;
  - Implementation of simple example with touch and sound;
  - Implementation of adding membranes to the game;
- **Sprint #2:**
  - Implementation of dragging of membranes on the scene;
- **Sprint #3:**
  - Implementation of UI stack mechanism on Display Manager;
  - First version of stack view for the membranes;
- **Sprint #4:**
  - Namespaces refactoring;
  - Managers support for Planes;
  - Simple membrane position animation;
- **Sprint #5:**
  - Select and highlight membranes on scene;
  - First version of virtual canvas;
  - Implementation of Compass Manager;
  - Binding of actors to compass;
  - Crosshair, button, and remove on touch added;
- **Sprint #6:**
  - Bugfixing on virtual canvas/camera mechanism;
  - Auto-stretching to devices resolution;

4.4. Testing

To test the interaction model an exercise of **design walkthrough** was performed along the advisor. From this exercise several issues and suggestions came out:

- Kits could also have some pre-configured scenes;
- Game author could annotate some metadata text along with the scenes to be used as a rough draft of the script of the game;
- It must be possible to clone scenes;
- It could be useful a actor inventory so that some actors could persist between scenes if need be;

As an exercise of validation of the developed architecture a simple game was defined to be implemented upon that architecture. The game was built around a theme of a archer that had to protect a castle wall from several advancing enemies that could appear from every direction, thus having to grab the tablet as a eyesight and spin to take on the enemies with a button. Since we only wanted to validate the feasibility of the architecture to run a augmented reality game, the edition interface was developed in the next iteration.

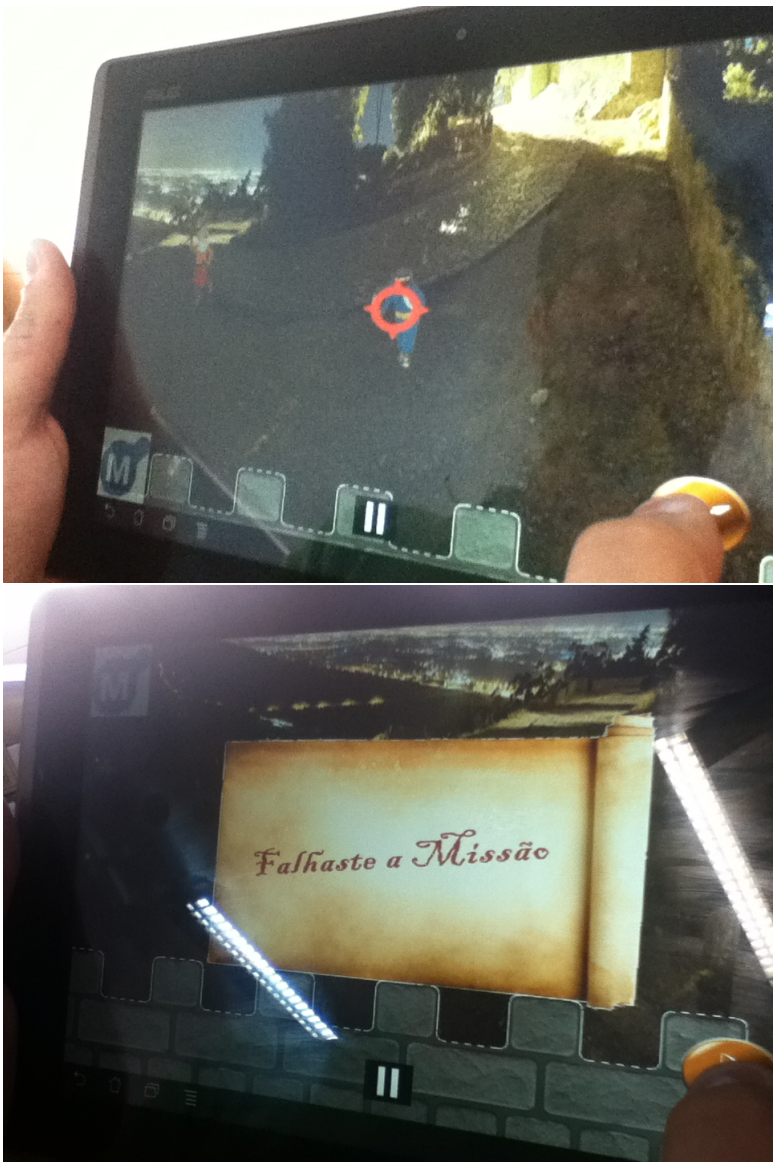

Figure 4.7.: The test game developed on the iteration 1 deployed and running on a Android tablet.

From this exercise, we concluded that our proposal was lacking a mechanism to assign and target a group of related actors - by the behaviours for example - as well as some sort of way to group several membranes in a Plane other than creating an Actor - to use in the editing application UI.

In this iteration, it also came to our attention how cumbersome can be the development in C++ with the Marmalade SDK. The platform lacked several needed features to power a more agile process, which lead to a unexpected big iteration time span and that affected the following iterations.

Chapter 5.

Iteration 2 - First Prototype

The main objective of this iteration was to develop an artifact that could offer some level of game edition and that would address the issues identified on the first iteration. In that sense, some changes were made to the proposed concept for game authoring. Also, given the increasing difficulty of coping with the planned abstraction layer between the application and the Marmalade SDK resulting in the dropping of this requisite. Moreover, facing the identified limitations of the Marmalade's cross-platform technology, a layer of middle-ware that could function as a Game Engine Library was designed and developed based on the previous code base already developed. Finally, some limitations of the MVC architectural pattern (as originally proposed by Krasner and Pope [14]) were identified and a custom approach at the pattern was developed.

5.1. Proposal

Based on the knowledge acquired on the previous iteration we felt the need to make some changes to the proposed concept. The updated concept map is presented in the Figure 5.1.

In this revision we introduced the concept of **Category** used to define a group with a similar role - like 'enemies', for example. Although it may seem that 'role' would be a more appropriate designation for this concept we opted to to choose 'category' as the name because the word 'role' is often associated with the individual role of an actor, that in our case is mostly determined by the behaviours of the Actor than anything else. This concept was needed because sometimes we want behaviours not only targeting or affecting a finite number of actors but a dynamic group of actors - projectiles, enemies, etc. We also introduced the formal concepts of **Triggers** - the elements that activate the behaviours - and the **Results** - the elements that the behaviour affects.

We also felt the need to add the **Plane** concept, a container/group of several membranes that when moved/resized affected it's membranes in the same way. Moreover, the Plane itself extends a Membrane, so that a Plane could contain other Planes (posing as Membranes) thus establishing a hierarchic mechanism for the membranes, that allowed us to

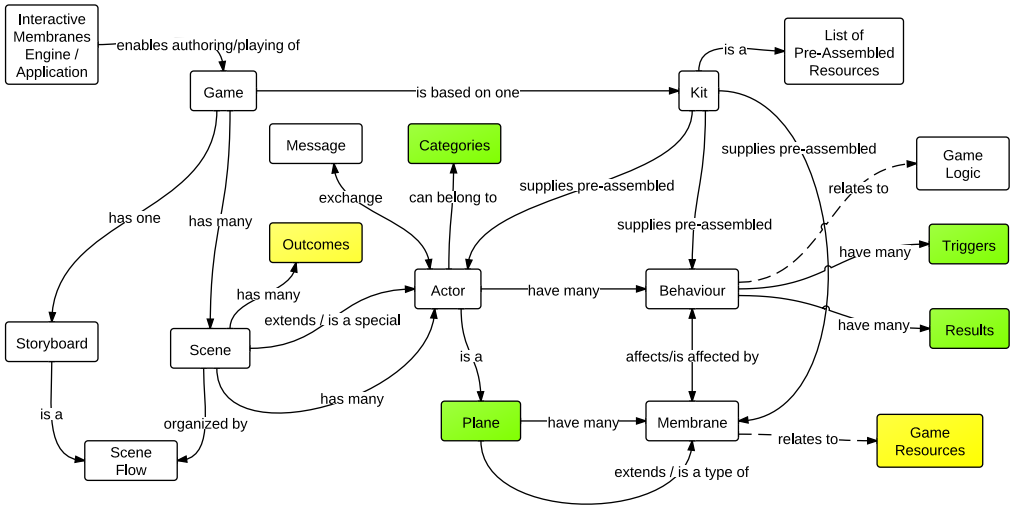

Figure 5.1.: Concept map with the vocabulary of terms and their relationships with added elements in green and changed elements in yellow.

have a flexible way to do more complex manipulations. For example, as we will discuss in the following Architecture section, it was particular useful to enable the support of sub-Views in a View hierarchy.

Although already considered on the previous iteration, the Scene **Outcome** concept wasn't properly identified on the map so we've added it to this version.

Finally, we reinterpreted the way we looked at the role of the **Membranes** in the overall concept. When in the previous version we made a connection of the Membranes with the Game Input/Output - and Behaviours with Game Logic and Mechanics - in this iteration we looked at the Membranes more as the **Game Resources** that could be used by the Behaviours to convey or setup the interaction.

In this way, we then defined the list of **Membrane Types** that we would consider on our application (and, as we'll explain later, also on our engine):

- **Visual**
  - Shape
  - Image
  - Animation
  - Repeating Membrane (horizontal or/and vertical repeat)
  - Live Video (live camera feed)

  - Text/Dialogs
  - Camera (viewport)
- **Non-Visual**
  - Touch/Gesture
  - Sound
  - Soundscape
- **Plane**

5.2. Architecture

The architecture proposed in the first iteration was developed with the intention of having the **Managers** as an abstraction layer between the Interactive Membranes application domain logic and the Marmalade SDK. However, in the development of the first iteration this objective was taking us too much effort versus gain, and given that, at the present, the portability requisite between cross-platform technologies was a nice-to-have feature and not a formal requisite. And even in that case, the platform to port to would have to support C++ as well. So we decided that from now on the application would be *targeted specifically for the Marmalade SDK*.

Furthermore, in the light of the limitations of the Marmalade SDK found in the previous iteration we decided to separate the technology into two distinct parts: the **Interactive Membranes Game Engine**, the **reusable** middle-ware built upon Marmalade SDK to help the development of games and rich applications based on the Interactive Membranes concept (Figure 4.1) and composed by the code of the managers, the membranes and generic extensible base classes for Models, Views and Controllers; and the **Interactive Membranes Mobile Application** used to build and edit the game interfaces built upon Marmalade SDK and the previous Game Engine.

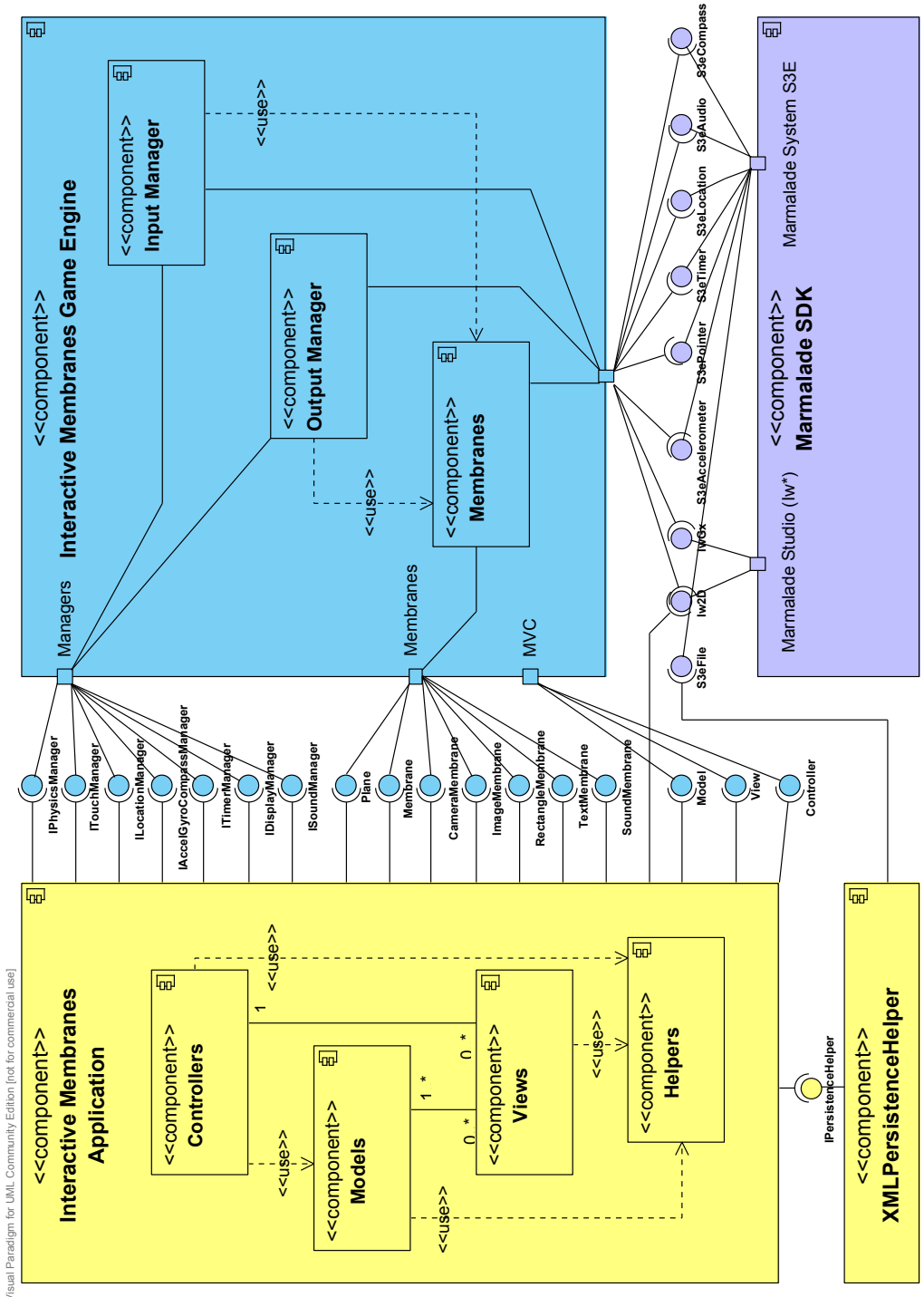

Figure 5.2.: The UML Component Diagram of the components of the proposed Interactive Membranes application architecture.

5.2.1. Interactive Membranes Game Engine

The Interactive Membranes Game Engine is a middle-ware library suitable to help build 2D games and rich applications on the Marmalade SDK. In this iteration it offered the following features:

- Virtual 2D Canvas
- Extensible Membrane System
- Extensible Touch and Gesture Manager
- Basic MVC framework

In the next sections we will cover each one of this features.

Virtual 2D Canvas

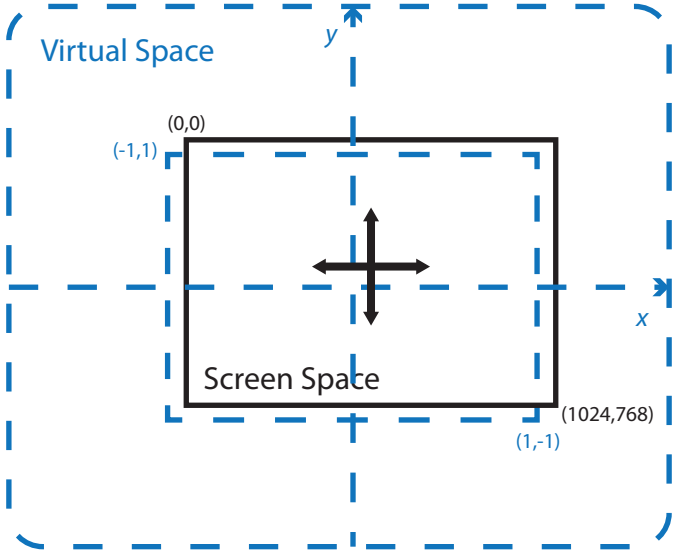

Figure 5.3.: The Virtual 2D Canvas of the Interactive Membranes Game Engine.

Since the Iw2D module of the Marmalade draws to screen space, the Game Engine implements a virtual 2D screen space. This way we can position the membranes in an virtual infinite space and move or zoom the camera along this space. The current camera is defined through a **static pointer** on the Camera Membrane class - along with the static methods **setCurrentCamera(camera)** and **getCamera()**. To speed up the Engine the screen coordinates and screen size aren't calculated at runtime (or when drawing) but in **real-time**, which means that the Membranes always maintain a virtual position/size and a screen position/size. The screen position/size is only updated

when their virtual position/size is changed. If our camera were always fixed on the same position and size, this mechanism would be enough, **but**, since out camera can move as freely as any other membrane this can lead to positions and size inconsistencies between virtual and screen spaces. So, we had to fix this problem on this feature. The way we did that, was through the **Observer pattern**[8], which means that, every membrane that gets registered on the display or touch manager gets automatically registered as a Camera Observer, and in the Camera Membrane class a static list of Observers was kept. Each time the Camera was resized or moved, it notifies the observers to update their position/size accordingly. This way we guarantee the consistency of the virtual-screen mapping.

But, another issue remained unsolved by this mechanism which was the support for **fixed membranes** - membranes that remain in the same position even when the camera moves. The way we solve this is, besides having a fixed attribute on the membranes, by updating only the virtual position when a particular membrane is fixed. This way the membrane appears to remain in the same place, but internally is travelling with the camera along the virtual space - this would be useful, for example, if we wanted a block to be solid but fixed thus affecting other solid membranes when the camera is moved. This concludes the explanation of the Virtual 2D Canvas feature.

Membrane System

The Membranes are the core of the Engine. They act as resources that you can register on the managers to enable a particular manipulation over the membrane. For example, you can simultaneously register the same membrane on the display manager to be displayed, but also on touch manager associated with a callback to be called whenever that membrane is touched.

On the top of this mechanism is the Membrane base class. This class stores the common information to all membranes (id, name, type, position, size, fixed to camera, visibility, touch blocking) and common methods needed to all membranes (getters and setters for these attributes). The derived classes then store and provide functionality only to their data. Some of the methods of the Membrane base class are virtual and can be overridden - like in the case of the setTopLeft() method of the Plane class.

Thinking on extensibility the type of the Membrane was defined as an enum type meaning that it's possible to add more membrane types provided that you implement the membranes classes and make the necessary adjustments to the managers to handle that new type. Even if you don't make the desired changes on the manager, the Engine would still work seamlessly since we were careful to enclose all the specific code to a type in switch statements with non blocking "default:" escape conditions. The worse that could happen, if you don't do those changes is your new type of membrane being ignored by the managers in methods where specific code is defined. The adding of the new type is then transparent and don't affects the rest of the code for the other membranes.

Touch and Gesture Manager

Another of the Marmalade SDK limitations discovered was the fact that in terms of touch functionality, the events were transmitted as raw. The s3ePointer module of the SDK[1] only detects button/touch events (up and down) and motion events and transmits them to the registered callbacks along with the screen coordinates of the event This means that if we were to register a membrane directly to this module, the membrane would be receiving several button and motion events, each time the screen was touched, while the majority wouldn't really be intended for that particular membrane. Furthermore each registered membrane would have to process the x and y values of the touch to see if they matched their screen area (not the virtual one, since the events contain x,y information in the screen space).

Therefore we coded a Touch Manager that would act as middle-ware and accept membrane registration for touch events and then manage the touch events handling, firing only the callbacks that were connected to an membrane whose area contains that specific point. Furthermore, the membranes also can be touch blocking, so that if a callback of a touch blocking membrane is fired, it stops there. Otherwise the touch event relays to the membranes below that one. This way we minimize the processing of the touch events. This solution also allow us to provide a more rich touch model for the underlying application, one with more touch vocabulary such as longpresses and other gestures through gesture analysis. Below is the state model, as suggested by Wigdor and Wixton[7] for our particular touch manager implementation.

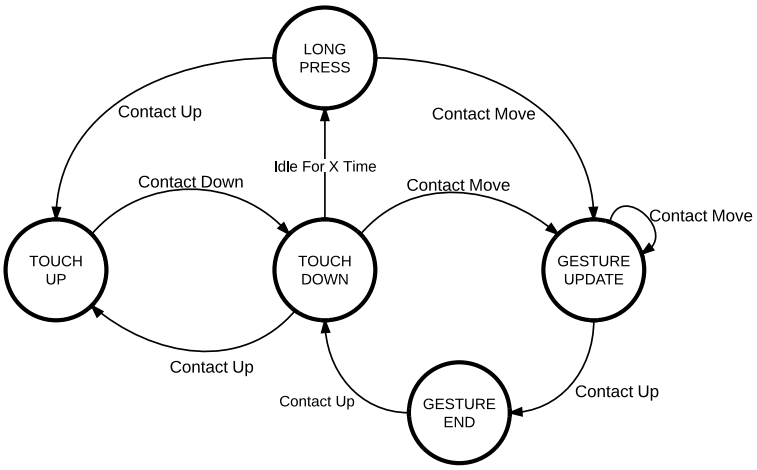

Figure 5.4.: The State Model of the current Touch Manager implementation.

It's important to note that in this model the gesture processing is delegated to the callback registered - the callback receives the gesture x,y updates and could employ a

[1]More information at: http://goo.gl/q6psl.

gesture analysis process if necessary. Unfortunately the limited time didn't allow us to explore and support custom or more complex gestures. Although it's addition would be fairly easy, again, thanks to a series of architectural decisions that allow to extend the type of events the touch manager processes and responds to. First, we employed the same mechanism of declaring the touch event types as an enumeration that allows for easy adding of new types of touch/gesture events. Seciondly the callbacks registering always have to carry the type of event to fire that callback. So if we wanted to add a new custom gesture - let's say a 'V' shaped one - we would have only to add the new type, and change the touch manager to accomodate this new gesture, which would be like to having another state on our touch state model connected to the Gesture Update state which would be reached if a particular gesture was detected while on the Gesture Update loop and left again to the Gesture Update state after more movement were detected.

Basic MVC framework

The Game Engine also provides simple Model, View and Controller base classes that could be used to build games and applications around the MVC architectural pattern[14]. In this case the View also extends the Plane so it can be used easily with the Display Manager of the engine.

5.3. Implementation

The development of the second iteration artifact went from the first week of May to the third week of June. During that time the following sprints were performed:

- **Sprint #1:**
  - Implementation of Text Membranes;
  - Implementation of XML Persistence Helper;
  - Use of background music on the game;
- **Sprint #2:**
  - Bugfixing with font rendering;
  - Implementation of Scene membranes and actors listing;
- **Sprint #3:**
  - Refactoring and proper MVC implementation and organization;
  - Implementation of first version of scene behaviours view;
- **Sprint #4:**

  - Implementation of Behaviour XML parsing;
  - Implementation of Kit XML parsing;
  - Finish work on scene behaviours view;
- **Sprint #5:**
  - Implementation of repeat membranes;
  - Implementation of Accelerometer Manager;
  - Implementation of Camera Observers;
- **Sprint #6:**
  - Implementation of add elements view;
  - Support of gestures on Touch Manager;
  - Implementation of visibility toggle buttons on elements listing;

5.4. Testing and Evaluation

For the purposes of testing the artifact developed in this interaction a simple usability test was prepared. A test script, presented in the Appendix C was written to detail a series of tasks to build a game similar to the one produced on the first iteration to validate the architecture.

For this test 5 test subjects with ages between 20 and 42 years were invited to perform the test, as suggested by Nielsen[18].

In this section we present the results of the evaluation of the prototype produced on the second iteration of the project. We begin with a quick description of each test highlighting the most significant and important cues and issues that have emerged during the interaction of the tester with the Ending. artifact the appendix will be a conclusion of the test and the summary of the results.

5.4.1. Subject A

This test began well with the tester grasping immediately the interface of the two first tasks.

On the **task 3** several issues emerged. First the tester tried to **add a behaviour using the add element button ('+' sign)**. Finally after successfully finding the button and the interface to add behaviours we felt uncomfortable with the fact that **the arrow buttons used to closing the interfaces had different directions over different interfaces** (the button to add elements on the left of the screen had a left arrow to

close/collapse the interface and the button to add behaviours on the right of the screen had a right arrow to close/collapse the interface). Still on the third task the tester tried to **touch on the behaviours to add them** - mimicking the same gesture to add elements.

On the rest of the test the tester also: entered wrongly on the behaviours interface on **task 10** arguing some confusion regarding the ambiguity of task description regarding behaviours versus elements; made **confusion with the back button of the permanent Android UI bar** on **task 13**; and demonstrated the expectation of the automatic elements addition when one adds a behaviour on **task 13/15**.

5.4.2. Subject B

On this test the tester, again, demonstrated **confusion about the direction of the arrows used to close/collapse the interfaces** as well as **confusing the add elements button ('+' sign) to add behaviours** on **task 3**. Additionally also occurred: **confusion with the controls on the Android UI bar** occurred (**task 1**), difficulties on positioning the elements (several tasks), frustration because the task descriptions didn't include the literal designations of elements/behaviours (several tasks). Finally the tester also suggested that should be visible on the add elements interface what you have already added (or even locked for elements that you only have to add once) and that the play button should be in reach of the hands resting/holding the tablet (like the other buttons).

5.4.3. Subject C

This test began with some issues regarding the responsiveness of the interface right on the **task 1**. Like Subject A, the tester tried to **click on the behaviour to add it** (**task 3**) - mimicking the gesture to add elements -, and like Subject A and B he made **confusion with the controls on the Android UI bar**. The tester also had some difficulties on positioning the elements (**task 8**), confusion between behaviours and elements (**task 13**)

The tester also made the suggestion of a way to add more than one element (**task 1**). A last note, a curious remark by the tester: *"once you understand the workings of the add elements/behaviour it very easy to use"*.

5.4.4. Subject D

On this test, again, the tester tried to **add a behaviour using the add element button ('+' sign)** (**task 3**) and had difficulties on positioning the elements (several tasks). Apart from that the test went quite well with nothing to note.

5.4.5. Subject E

On this test the problem with the **add behaviour with the add elements button ('+' sign)** occurred again (**task 3**). Apart from that there weren't anything to note.

5.4.6. Conclusion and summary of results

In terms of task completion this particular test was a success, all test subjects completed all the tasks and achieved the test goal. Nonetheless, there were several non-critical problems that were valuable to identify problems.

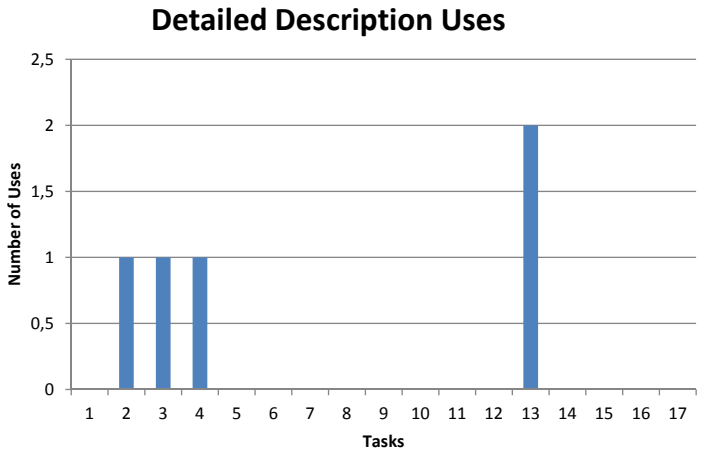

Figure 5.5.: Chart of the number of uses of detailed descriptions per task.

First, looking into the chart of the number of detailed description uses (Figure 5.5) we can spot two distinct problems. The uses on the tasks 3 and 4 relate to the confusion over using the add element button ('+' sign) to add a behaviour. The uses on task 2 and 10 were related to the particular wording of the tasks (not using the literal names on purpose to test the concept comprehension) and some ambiguity of an element and behaviour having similar names.

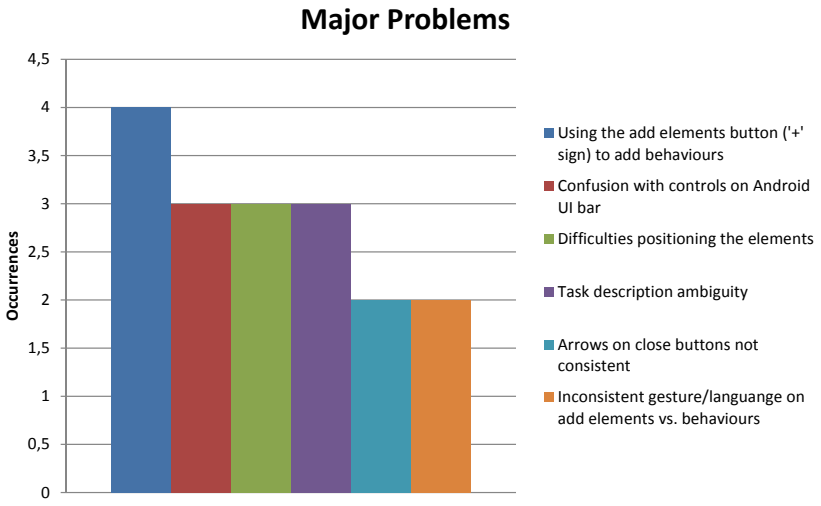

Figure 5.6.: Chart of the major problems occurred in all the tests.

When we look to the major problems occurred in the tests, the problem of using the add elements button, depicting a '+' sign often associated with a add action, to add behaviours clearly stands out. It occurred on 4 of the 5 tests. The button having a '+' sign is misleading when the testers wanted to add behaviours and identifies a major UI consistency problem with the add interaction. Potential solutions for this would be **joining the add interaction together for actors, membranes and behaviours** or **changing the icon of the add element button**.

Relating to this problem is the problem of inconsistent gesture 'languange' on the add interaction, that test subjects found uncomfortable and awkward: when adding elements one must simply touch the element to add it and the interface disappears, while adding the behaviours one must drag a behaviour from a palette to an area and the interface remains in place. This suggests that **further work must be done in following iterations regarding the consistency of the gesture 'language' actions**.

The confusion with the native Android UI controls is a tricky problem, since it's beyond our scope of development, and calls for a **closer attention to these sort of problems when designing interactions on multiple platforms**. Namely, **the acknowledgment of different mobile OS UI paradigms and the subsequent normalization of our interaction designs to them**. A possible solution for this problem, without compromising the cross platform deployment, would be to define a additional code target solely to Android that could bing the native UI back button with the back buttons of our application.

The difficulties with elements positioning is a identified bug that should be fixed on following iterations. Task description ambiguity, which was on purpose to test the level of understanding of the concepts, presented too much difficulty and frustration on the testers. A suggestion, in fact given by test Subject B, is do **describe literally the name of elements/membranes/actors/behaviours on the task descriptions**.

Finally, we have the problem of arrow direction inconsistency. Although the initial intention was to give some sense of collapsing panels - hence the add elements panel on the left collapsed to the left with a left arrow and the behavious panel on the right collapsed to the right with a right arrow - the users didn't understood it that way resulting in strangeness on testers and compromising this intention. One solution would be that all arrows on go back icons point to the same direction - preferably left which is commonly linked on western cultures to go back.

Chapter 6.

Iteration 3 - Second Prototype

The objective of this third iteration was, not only to propose a solution that addressed the usability problems identified in the last iteration and develop a new artifact that implemented that proposal, but also to implement new features deemed important: live video, animations and positional membranes.

6.1. Proposal

In this iteration we decided to approach positional membranes, which are membranes bound to a particular geographic point. This posed a challenge since some kind of mapping between a spherical coordinate system and the virtual coordinate system of the 2D Virtual Canvas had to me made. Beginning to work on this issue it became apparent to us that our referential in this mapping **had to be always the current camera**, for which points to the current direction and represents frame of the front view of the world. From there we agreed on the ratio of size of the current camera - 90^{o} degrees corresponding both to the height and to the width - and used a simple cylindrical projection to represent the sphere in 2D space. The positional membranes are then positioned onto this projection based on the their bearing delta with the current direction and it's height angle delta. The Figure 6.1 details this reasoning.

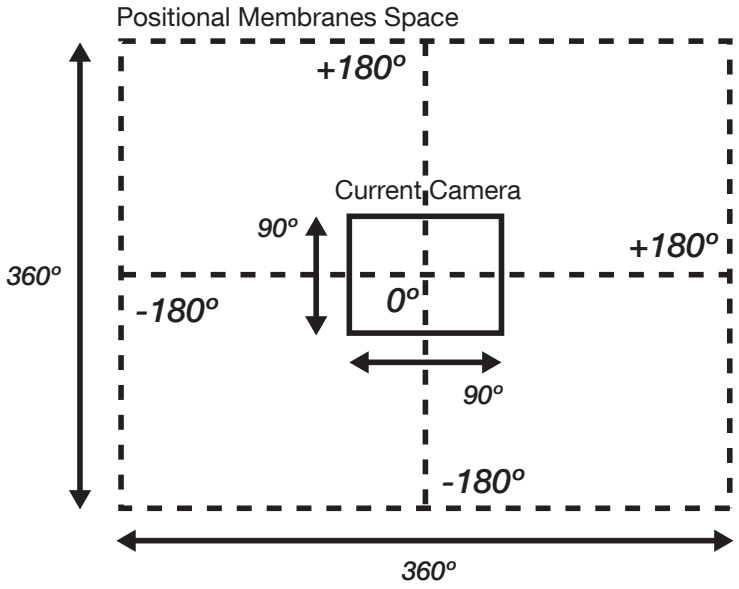

Figure 6.1.: The space where positional membranes are drawn regarding the current camera.

6.1.1. Interaction Adjustments

The test of the previous iteration highlighted several problems with the user interface and the interaction with the artifact, namely: testers using the add elements button ('+' sign) to add behaviours to the scene (Figure 6.3), the inconsistent direction of the arrows used to close the interfaces and the inconsistent gesture/language between adding elements and behaviours.

In the Figure 6.2 below it's visible the above mentioned inconsistency regarding the directions of the arrows and regarding the add interaction for elements and behaviours. While on elements a touch on the desired element would immediately add the element to the scene and close the add element interface, on the behavious interface the user was presented with a grid where the added or configured behaviours would appear, a list of the available behaviours to add to the scene and to add a behaviour the user would have to drag the behaviour icon from the list to the grid area. After that the view would reload and the behaviour interface remained open - until the user touched closing arrow to dismiss it. As we can see, a radical different approach that confused and caused discomfort on the users.

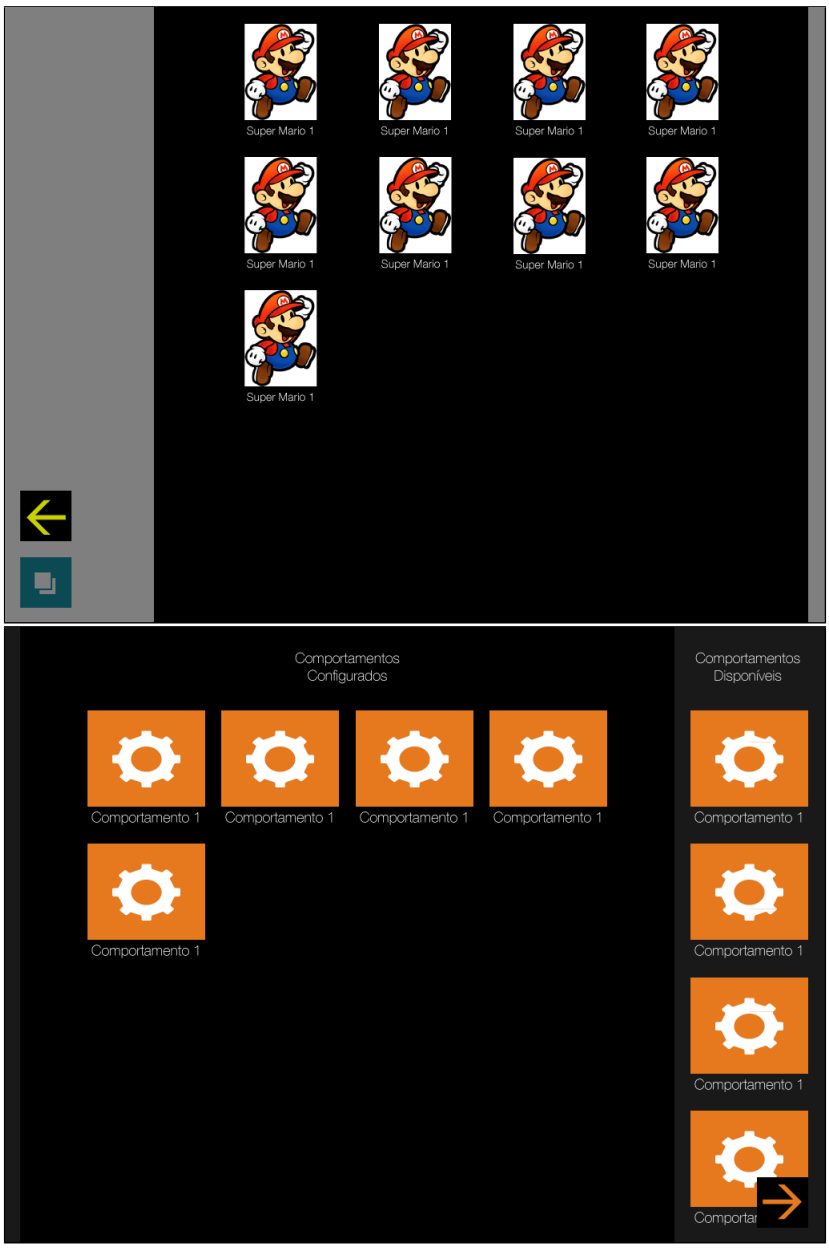

Figure 6.2.: The adding of elements and behaviours on the second iteration artifact showing the two very different approaches to add elements and behaviors, and the two opposing arrows to close the interfaces.

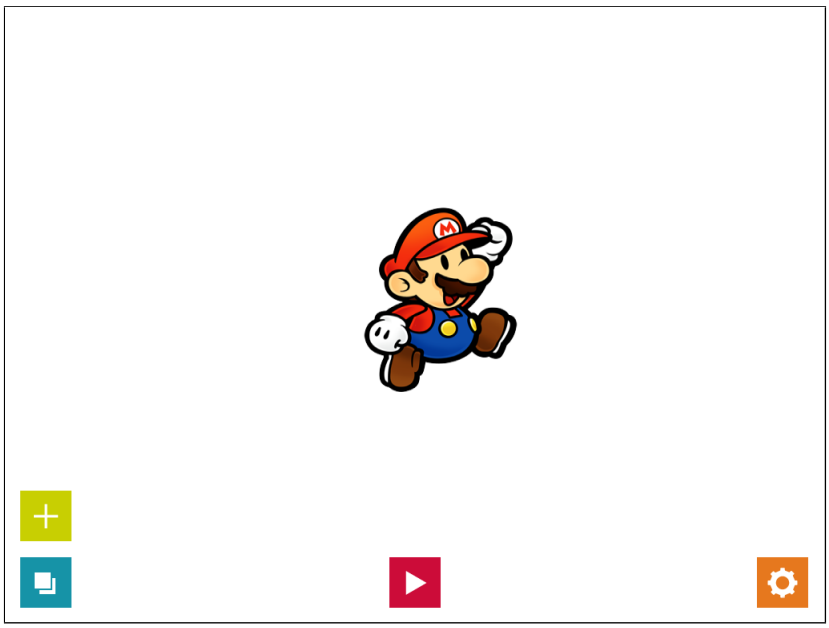

Figure 6.3.: The scene edit screen of the second iteration artifact. Most of the users pressed the add elements button ('+' sign) when asked to add a behaviour to the scene.

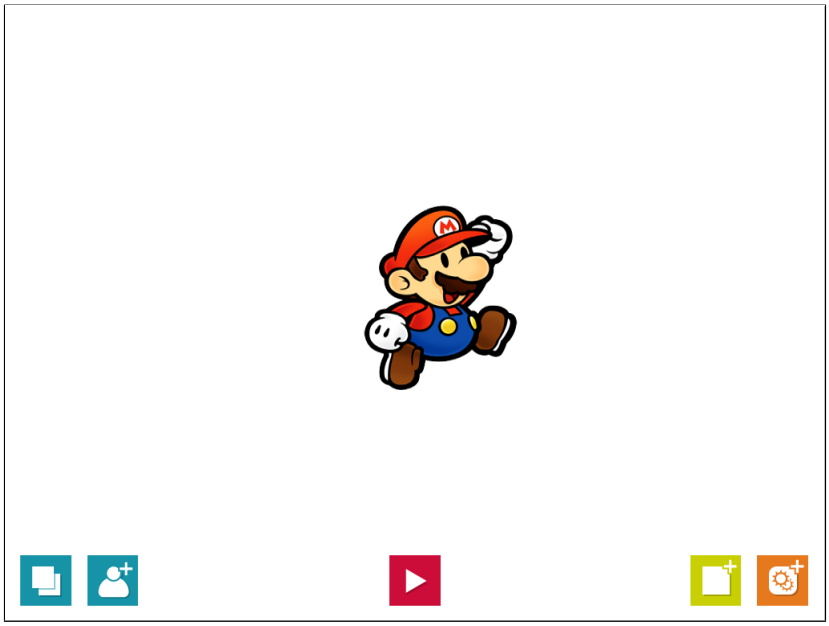

Figure 6.4.: The scene edit screen of the third iteration artifact with the new buttons layout.

For this iteration we proposed a new direction to the interface, depicted on the Figure 6.4, that didn't had these problems and test it. Namely we decided to stop treating the actors and membranes as a whole under the name 'element' and bring upfront to the user that distinction. So in our proposed Scene Edit screen feature five distinct buttons: the list elements button, that we opted to maintain (since we need a way to list the elements of the scene in a single depth stack) while giving it a little refresh to emphasize the layers iconography; the add and manage actors button; the play button, also maintained with a little refresh; the add and manager membranes button; and the add and manage behaviours button.

We also implemented the actor edit/focus mode, where the user could focus on an actor to add membranes and behaviours to it. In this mode the add and manage actors button would disappear - since we don't allow adding actors to actors - thus reserving the right side of the interface to buttons relative to actor actions: adding membranes and behaviors.

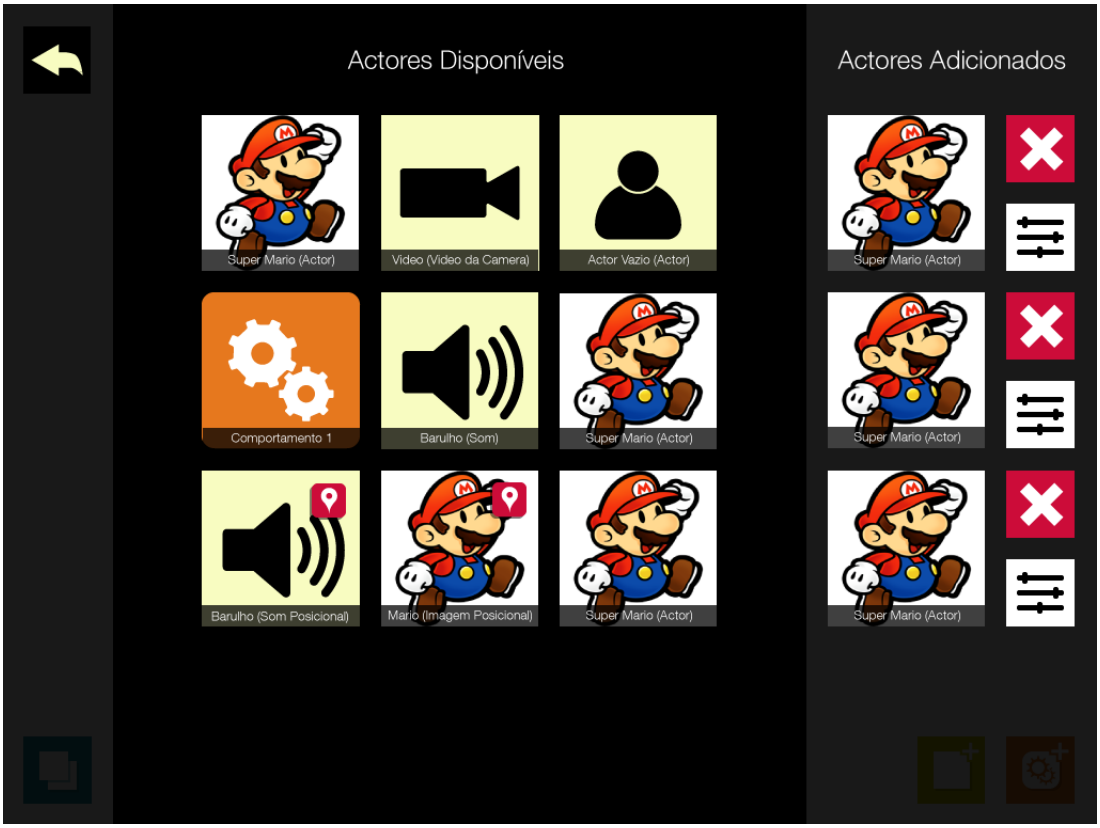

Figure 6.5.: The new reusable add/manage elements view with the new back button on the top left corner.

So, based the behaviours screen of the previous iteration we made a new generic "add and manage elements" view, displayed on the Figure 6.7, that could be reused for actors,

membranes and behaviours. In this view we swapped the areas because you would end up having much more elements of the available area (palette) than added elements. So the grid is now for the available elements display and the list for the added/configured elements. Moreover this new view could also be used to remove elements and to setup a special edit button below the remove button to call other interfaces - which in the case of actors was used to focus/edit that particular actor.

Finally, we replaced all the back buttons on the screens with a single equal back button that appeared always at the top left corner of the screen, thus maintaining consistency over the user interface.

6.2. Architecture

For this iteration the following features were added to the Interactive Membranes Game Engine:

- **Camera Manager and Live Video Membrane** - for the augmented reality functionality working with overlaying of elements (images, positional markers, etc);
- **Frame Animation Membranes and Animation Manager** - frame animation membranes based on sprite sheets for the purpose of having animated actors on the scene;
- **Positional Membranes and Location Manager** - for the positional image membranes, positional frame animation membranes and positional sound membranes, according the previous explained concept of positional mapping;

6.2.1. Animation Manager

To provide our Game Engine with animation functionality we devised an extensive and flexible animation framework. First we approached the animations from 3 perspectives plus a support one:

- **Frame animation**: animation of several sprites in sequence;
- **Position animation**: animation of the position of a membrane, either with movement determined by a movement vector, or by a destination position;
- **Scale animation**: animation of the size of a membrane;
- **Composite animation**: group of animations to be played as one.

After this we developed a Frame Animation Membrane, very similar with an Image Membrane but having a spritesheet as an image, several helper vectors to help with the offset rendering of the correct frame (number of columns and rows, frame size and current frame) and a time between frames attribute.

On the animations several attributes were defined: duration, delay (time to wait until the animation start, useful for composite animations sequencing), loop, number of loops, current loop and state (paused, started and stopped). (We assumed an infinite loop as a looping animation with number of loops defined to 0.)

For simplicity no easiness of the animation was considered, so all animations progress linearly.

6.3. Implementation

The development of the third iteration artifact went from the first week of May to the third week of June. During that time the following sprints were performed:

- **Sprint #1:**
  - Refactoring of Managers as Singletons;
  - Longpress support on Touch Manager;
  - Elements reordering on scene elements listing;
- **Sprint #2:**
  - Implementation of Live Video Membranes;
  - Implementation of Camera Manager;
  - Implementation of Live Video Membranes XML parsing;
- **Sprint #3:**
  - Implementation of Frame Animation Membranes;
  - Implementation of Animation Manager;
  - Reimplementation of Sound Membranes;
- **Sprint #4:**
  - Implementation of Positional Membranes;
  - Implementation of Location Manager;
  - Work on Behaviour Edit;
- **Sprint #5:**

- Implementation of Add Manage Elements view;
- UI refresh;
- Preparation of Test;

6.4. Testing and Evaluation

For the purposes of testing the artifact developed in this interaction another simple usability test was prepared. A test script, presented in the Appendix D was written to detail a series of tasks to build a game that able to test the functionality provided by the Adventure platform through the mapping of it's actions through behaviours.

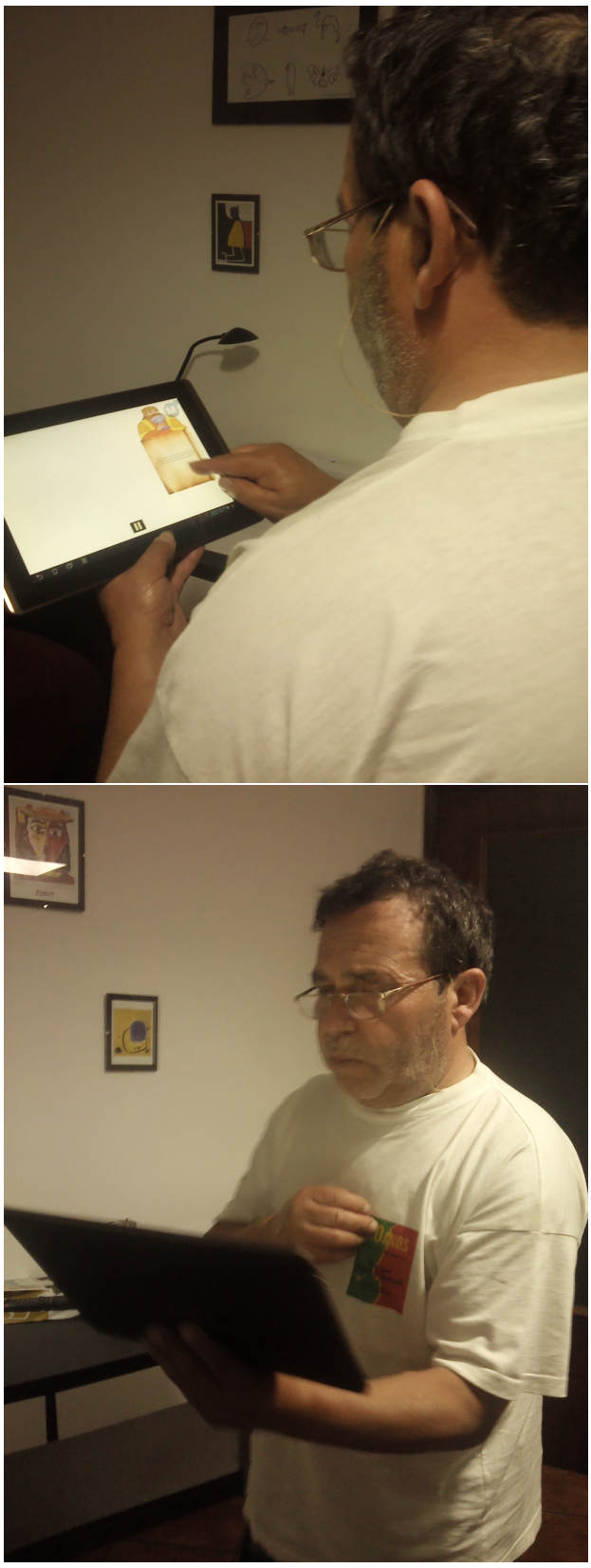

Figure 6.6.: Testing of the artifact by a tester 60 years old.

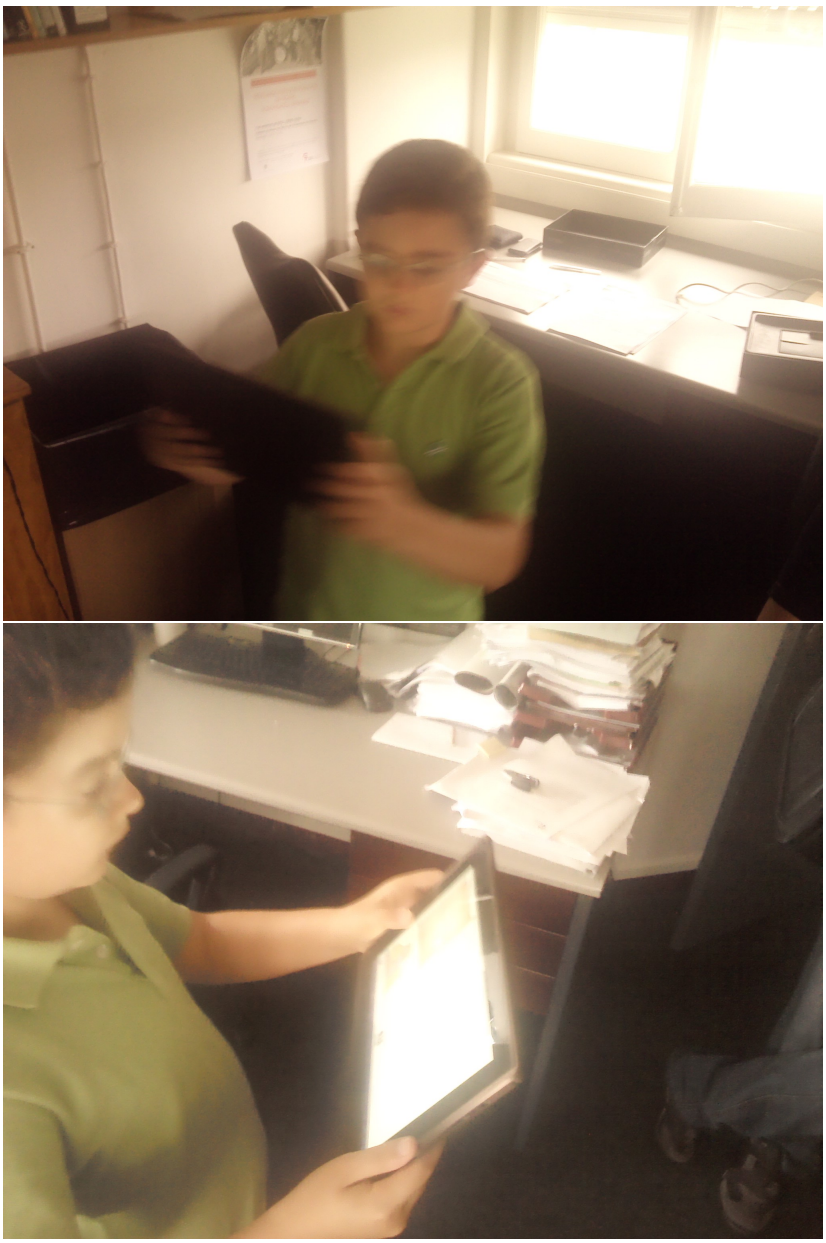

Figure 6.7.: Testing of the artifact by a tester 8 years old.

For this test another 5 test subjects with ages between 8 and 60 years were again invited to perform the test, as suggested by Nielsen[18].

In this section we present the results of the evaluation of the prototype produced on the third iteration of the project. As with the report of the second iteration, we make a quick description of each test highlighting the most significant and important cues and issues emerged during the interaction of the tester with the artifact. Ending the appendix will be a conclusion of the test and the summary of the results.

6.4.1. Subject A (Age: 8)

This test went fairly well with the tester - the youngest of all testers - grasping the basic concepts of the artifact from early on and showing a above average availability, predisposition and focus to the proposed tasks - regularly without any major assistance and with no need to, formally, use the detailed descriptions at all. All the tasks of the test were completed by the tester with no major issue to note. Only one remark was made by the tester regarding the size of the text that was "*too small*". Because this was our younger tester the test was placed no so much as a formal testing procedure but more like an directed play testing activity taking place between the artifact, the tester and supervisor.

After the test several curious remarks we're made by the tester, namely the desire for "*more stuff*" - that could confirm tester engagement with the artifact - and a remark given to the supervisor as "*now we should add here a message saying we've won*" leading us to believe that the tester understood not only the purpose of the tool but already assimilated the underlying concepts by demonstrating the will to use them to advance the test scenario further. A final curious remarks by this tester is the apparent need on a game for an avatar of the player (role playing a knight), "*how does the knight looks like? don't we have to put a knight in the game?*".

Finally, taking advantage of the whole play testing context the supervisor asked the test which other games he would like to build with the tool, which resulted in the following 'story' told by the tester: "*I would like to build the game of the Monica's Gang*[1]*. She would have to save all of his friends from Captain Ugly*[2] *[And how would she do that?] She would have to deal with several monsters along the way... and she would use the Bunny Samson*[3] *to get rid of them... along with a magic trick she does with both hands...*". This episode is particularly interesting since it goes along our assumptions that there are vast creative potential untapped in children to design and detail games lacking proper tools and ways to materialize those stories and games.

6.4.2. Subject B (Age: 35)

This test started with some difficulties by the tester including the confusion with the Android UI bar controls (**task 1**). On the following tasks the tester still felt a little disoriented, feeling that was being dissipated as each task passed and the tester got a

[1]Monica's Gang (Turma da Mônica in Portuguese) is a popular Brazilian comic book series which happens to be one of the tester's favorite series.

[2]Captain Ugly (Capitão Feio in Portuguese) is character from Monica's Gang, generally appearing as a villain that lives in the sewers and tries do leave the world ugly and dirty, but Monica and his friends always break the Captain plans.

[3]Bunny Samson (Coelho Sansão in Portuguese) is a blue bunny that Monica brings all the time. In most of the stories, she uses him as a "weapon" to hit the street boys who bully her.

grasp on the interface and the concepts. The tester showed some difficulties on the **task 13** but was able to perform the task without needing to see the detailed description.

After the test small talk occurred between the tester and the supervisor which had some interesting insights. The tester stated that "after that you grasp the interface and the concepts it's easy to use" The tester pointed out that the constructed game wasn't very comfortable to play from an ergonomic point of view, since to play the game the player would have to hold the tablet in front of him and that causes strain on several arm's muscle after a while.

Asked about the theater metaphor, if he felt some resemblance with that metaphor or not the tester replied: "*I felt the theater metaphor more in the sense that elements manipulation and actors building is like to* ***stage/assemble a scene, mark positions, cues, etc.***".

6.4.3. Subject C (Age: 27)

On this test some confusion emerged on the tester regarding the **task 2** - the tester wandered on several interfaces - but in the end it was able to complete the task without looking to the detailed description. On the **task 5** stated that "*talways confuses these two [rightmost] buttons of the membranes and* ***definitions***". At the **task 8** the tester made some suggestions: clicking outside of an interface could dismiss it (e.g., clicking on the scene area while having the elements listing open dismisses the listing) and that the several screen of the application could be dismissed to the side by swiping them away to the side of the screen and pulling them again when needed. On **task 13** the tester made an interesting exclamation in surprise: "*That's cool! We can move the actor group as a whole and all it's membranes move along as well!*".

6.4.4. Subject D (Age: 60)

Again, on this test the tester became confused with the Android UI bar controls on **task 1**. On **task 5** the tester became intrigued stating "why don't the behaviours show up in the scene?". After completing the test, the tester made several remarks about the testing experience: it said that the application "is intuitive", complained about the small text size, suggested an 'X' mark (remove) more small (possibly to avoid mis-touching) and, finally, "the interface could show if and how many invisible membranes there is in a given moment".

6.4.5. Subject E

On this test, again, some interference of the Android UI bar controls was evident (**task 7**). Noting that on the **task 1** the tester clicked the actor to add it and only after that

it dragged them to add, and didn't repeated that error again. The tester also showed some difficulties on **task 13** and made a suggestion: "having to press a button that has a plus sign (+) to edit an actor confuses me a bit" **task 14**.

6.4.6. Conclusion and summary of results

As the previous iteration 2 test, this one was also a success regarding to task completion with all the tasks completed in each test. Nonetheless, some minor problems appeared that didn't compromise neither the test nor the usage of the application and in lesser number and extent than on the iteration 2 test. This test, in general, had fewer and smaller problems.

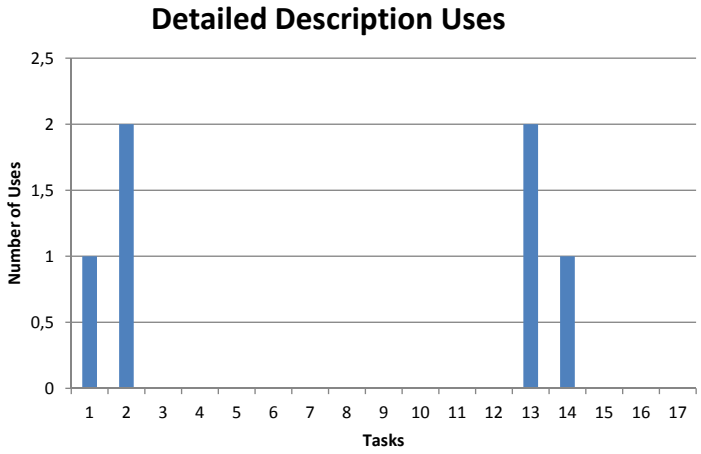

Figure 6.8.: Chart of the number of uses of detailed descriptions per task.

Looking at the graph of the number of uses of detailed descriptions in the iteration 3 test (Figure 6.8) we observe that the usage of this help is still low but, it's worth watching closely what caused these usages in more detail. We think that the detailed description usage on the first tasks is normal given that the testers are facing a new interface and paradigm - detailed description uses on tasks closer to the end could signal that the tester wasn't learning to use the interface. Still, on the **task 13** the detailed description was used twice and several testers also had problems on that particular task. This task is about reordering a element on the list of elements, and, effectively, there isn't a good *affordance*[19] to do so, but since that particular interface already has other gestures to perform other actions a assigning a simpler gesture for that action is difficult without

causing ambiguity on the 'touch language'. So this particular issue remains to be further addressed.

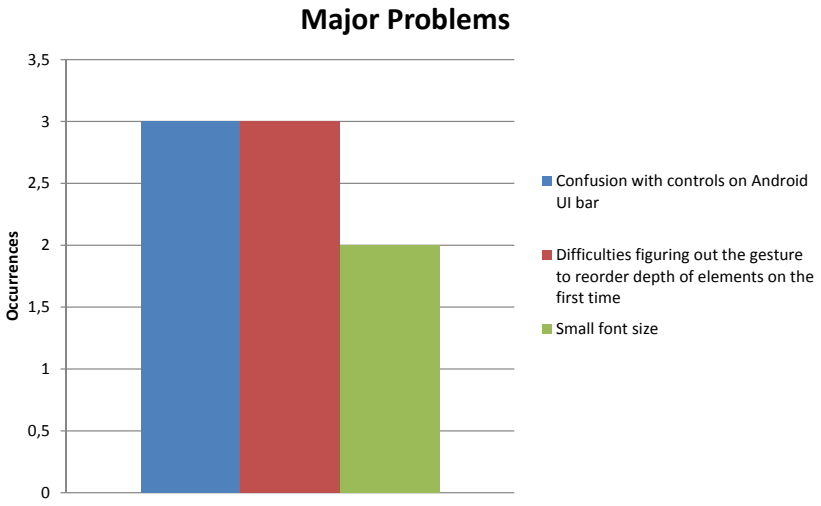

Figure 6.9.: Chart of the major problems occurred in all the tests.

On the major problems that surfaced on the test of this iteration we have, for one, the reordering difficulties that we already addressed above, and, again and like on the test of the previous iteration the problem with the native Android UI and bottom bar controls affecting the interaction with our artifact. Again, the problem of the native Android UI controls affecting the interaction is a tricky one that calls for a **closer attention to these sort of native UI interferences when designing interactions on multiple platforms**. At least, the acknowledgment of different mobile OS UI paradigms and the subsequent normalization of the interaction designs to them. The fact that this problem persists on this iteration is because it isn't a critical problem and because when faced with other important functional features to design and implement, and in the light of limited time, a greater importance was given to the former.

It remains the problem of the small font size that arises from the fact the application was only packaged with a global font size. Although the problem had been spotted before, we choose to set a low priority to it and focus on other higher priority tasks since the inclusion of another font on a Marmalade application isn't properly trivial.

Overall, the test went very well. To only big problem to notice would be, in our opinion, the problem of the reordering elements difficulties that deserves further development.

DRAFT

As the previous iteration we prepared a simple usability test with 5 participants with now with ages from 8 to 60 years old (insert reference to Don Norman 5 users 'is enough'). A test script was produced (which is in the appendix).

The problems from the second iteration didn't appeared on the test of this iteration which leads us to believe that were somewhat on the right track regarding those ambiguities and inconsistencies found on the second iteration.

Chapter 7.

Conclusion

Recapping, the data and the knowledge we have gathered during the development of this project - either by the state of the art study as well as by the process of the three design research iterations - have revealed us with many interesting clues, ranging from several perspectives - technology, pervasive game design, interaction design, etc.

In this closing chapter we try to tidy and group them by major perspectives for better synthesis of the acquired knowledge. Finally we end with a section on potential future work for this project.

7.1. Game Authoring Proposal

One of the stated objectives of this projects was to enable rapid prototyping of new 2D game interfaces for pervasive and augmented reality games targeted at mobile devices (smart phones and tablets). The usability tests performed on the second and third iteration portrayed examples that fall under this category (making use of the compass, location, overlaying markers over live video, etc.) and have fairly encouraging results. But, in our project, we merely scratched the surface in terms of empowering the player, or end-user, with the tools to openly build and assemble a fairly range of games and interfaces for games. The reliance of our approach on the previous assembling of the game kits by more technical literate individuals still leaves the end users powerless and dependent facing the video game creation activity. Using the three domains of participation proposed by Raessens[21] - interpretation, reconfiguration and construction - is fair to say that this project stands on the second domain, reconfiguration, with a solid developed work to proceed into the third proposed domain, construction.

The results of the tests far from giving us a clear answer for the problem at hands, give us encouraging hints that, with the theater metaphor allied to the greater technological inclusion and accessibility that the new mobile touch computing devices bring, we might be on the right track to a novel game authoring technology that could help, even more, the development of a participatory culture around the video game medium. One of these hints was given by the Subject B of the third iteration testing, "*I felt the theater*

metaphor more in the sense that elements manipulation and actors building is like to ***stage/assemble a scene, mark positions, cues, etc.***" and the remark of "*easiness of use*", after you grasp the basic concepts, by several testers. Other of the hints is the test of eight year old Subject A that not only went pretty flawlessly as it was notorious that he not only became engaged with the tool, but also was already proposing new ways to use it and to build new games.

7.2. Augmented Reality Games Design

Regarding the design of augmented reality games an interesting conclusion was observed, that can work as a cautionary tale for anyone designing these type of games, on how little ergonomic is holding the tablet in a upright head facing position, typical of several augmented reality applications and games, for too long. After a while this interaction causes too much strain on the arm muscles of the user/player. Often marveled with the inclusion of the novel gimmickry of augmented reality, designers can overlook these basic and important issue. That's why we suggest that this type of interaction, using a mobile computing device to have a augmented view of the world, should not be too long - perhaps mixed between other type of interactions.

7.3. Cross Platform Development

On the development of this project we learned that even though there are a great (and ever increasing) number of cross-platform development toolkits/SDKs/libraries, few have the necessary support for novel reality of augmented-reality games - case-in-point, the chosen technology was the only one that could access the live video feed of the camera and overlay objects on it. We soon found out that this augmented reality support came at the cost of being a relative low level technology. From our own experience there still is a lack of proper augmented reality middle-ware to support an easy development of cross-platform augmented reality applications.

7.4. Future Work

With this platform as a base, one could now further experiment on ways to empower end-users to reconfigure games could be made, in an attempt to reach a more profound level of participation closer to the construction domain defined by Raessens[21].

Furthermore, since the devised system architecture guarantees that the Interactive Membranes Game Engine is reusable, this particular component could be independently developed as a Augmented-Reality ready Game and Rich Applications Engine.

Bibliography

[1] YoYo Games Wiki, GameMaker Documentation. `http://wiki.yoyogames.com/index.php/Documentation:Main_Page`.

[2] Apple. ios human interface guidelines, Aug. 2012.

[3] Ballagas, R., Kratz, S., Borchers, J., Yu, E., Walz, S., Fuhr, C., Hovestadt, L., and Tann, M. REXplorer: a mobile, pervasive spell-casting game for tourists. In *CHI'07 extended abstracts on Human factors in computing systems* (2007), ACM, pp. 1929–1934.

[4] Ballagas, R., and Walz, S. REXplorer: Using player-centered iterative design techniques for pervasive game development. *Pervasive Gaming Applications 2* (2007).

[5] Beyer, H., and Holtzblatt, K. *Contextual design: defining customer-centered systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[6] Bichard, J., and Waern, A. Pervasive play, immersion and story: designing interference. In *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts* (2008), ACM, pp. 10–17.

[7] Daniel Wigdor, D. W. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann, 2011.

[8] Erich Gamma, Richard Helm, R. J. J. V. *Design patterns: elements of reusable object-oriented software*, vol. 206. Addison-Wesley, 1995.

[9] Fitts, P. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47*, 6 (1954), 381–391.

[10] Goffman, E. *Encounters: two studies in the sociology of interaction*. The advanced studies in sociology series. Bobbs-Merrill, 1961.

[11] Habgood, J., and Overmars, M. *The game maker's apprentice: game development for beginners*. Technology in action series. Apress, 2006.

[12] Herbst, I., Braun, A., McCall, R., and Broll, W. TimeWarp: interactive time travel with a mobile mixed reality game. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services* (2008), ACM, pp. 235–244.

[13] Huizinga, J. *Homo ludens: a study of the play-element in culture*. Beacon paperbacks. Beacon Press, Boston, MA, USA, 1955.

[14] Krausner, G. E. S. T. P. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *System 1*, 3 (1988), 26–49.

[15] Magerkurth, C., Cheok, A., Mandryk, R., and Nilsen, T. Pervasive games: bringing computer entertainment back to the real world. *Computers in Entertainment (CIE) 3*, 3 (2005), 4–4.

[16] Magielse, R., and Markopoulos, P. HeartBeat: an outdoor pervasive game for children. In *Proceedings of the 27th international conference on Human factors in computing systems* (2009), ACM, pp. 2181–2184.

[17] Montola, M., Stenros, J., and Wærn, A. *Pervasive games: theory and design*. Morgan Kaufmann Game Design Books. Morgan Kaufmann Publishers, 2009.

[18] Nielsen, J. Why you only need to test with 5 users, Aug. 2012.

[19] Norman, D. *The Design of Everyday Things*. Basic Books. Basic Books, 2002.

[20] Owen, C. Design research: Building the knowledge base. *Journal of the Japanese Society for the Science of Design 5*, 5 (1997), 36–45.

[21] Raessens, J. Computer Games as Participatory Media Culture. In *Handbook of Computer Game Studies*, J. Raessens and J. Goldstein, Eds. MIT Press, 2005, pp. 373–388.

[22] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Others. Scratch: programming for all. *Communications of the ACM 52*, 11 (2009), 60–67.

[23] Salen, K., and Zimmerman, E. *Rules of play: game design fundamentals*. MIT Press, 2004.

[24] Schell, J. *The art of game design: a book of lenses*. Morgan Kaufmann. Elsevier/Morgan Kaufmann, 2008.

[25] Vaishnavi, V., and Kuechler, W. Design Science Research in Information Systems. `http://www.desrist.org/desrist`, 2004.

[26] Wetzel, R., McCall, R., Braun, A., and Broll, W. Guidelines for designing augmented reality games. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share* (2008), ACM, pp. 173–180.

Appendix A.

Iteration 1 Paper Prototype

In this appendix we present the figures of resulting from the Paper Prototype of the first iteration of the application. The listed screens of the paper prototype are:

- **Game List screen**, A.1;

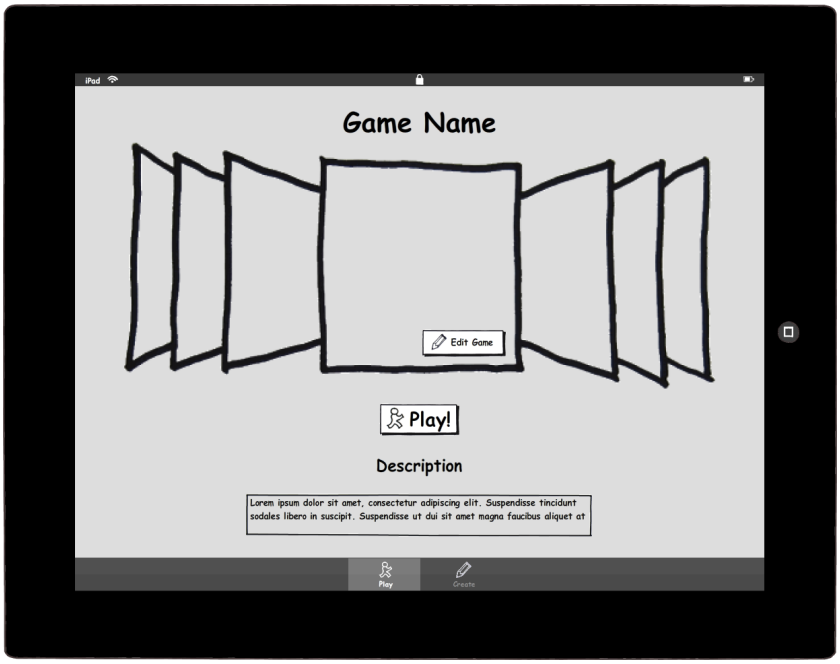

Figure A.1.: Paper Prototype, Game List screen, where the user can view, play and edit games.

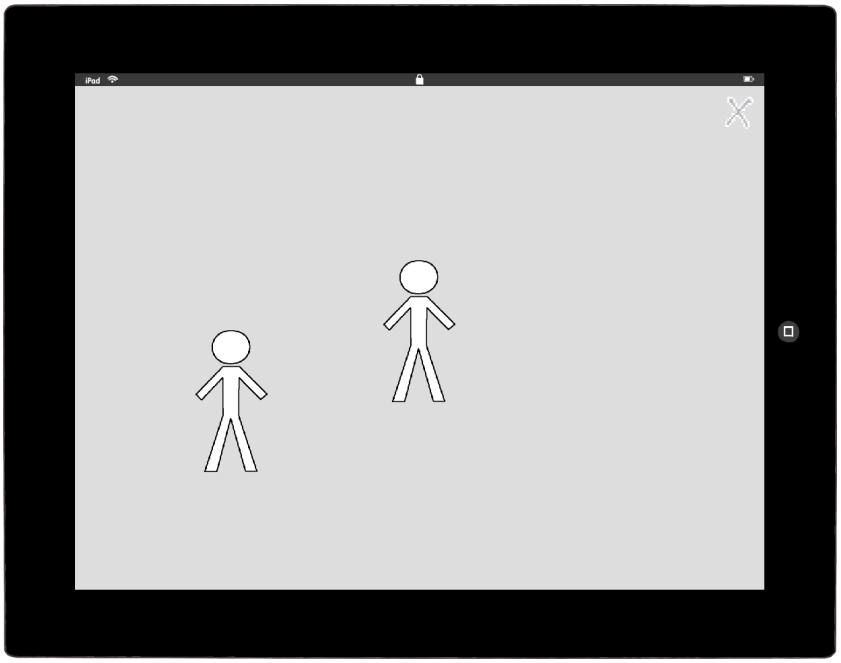

Figure A.2.: Paper Prototype, Gameplay screen, the screen where the game runs.

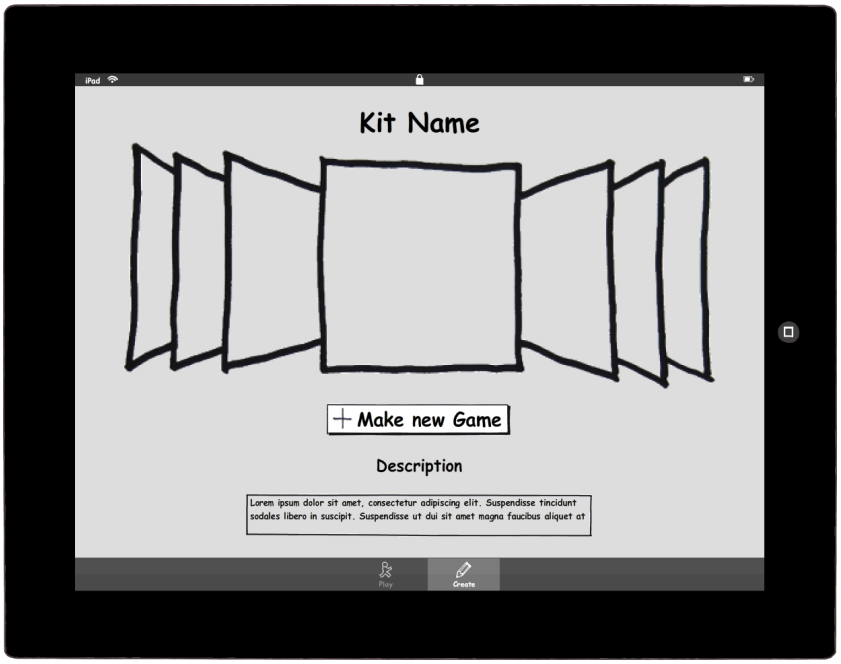

Figure A.3.: Paper Prototype, Kit List screen, displaying the available kits to build games from.

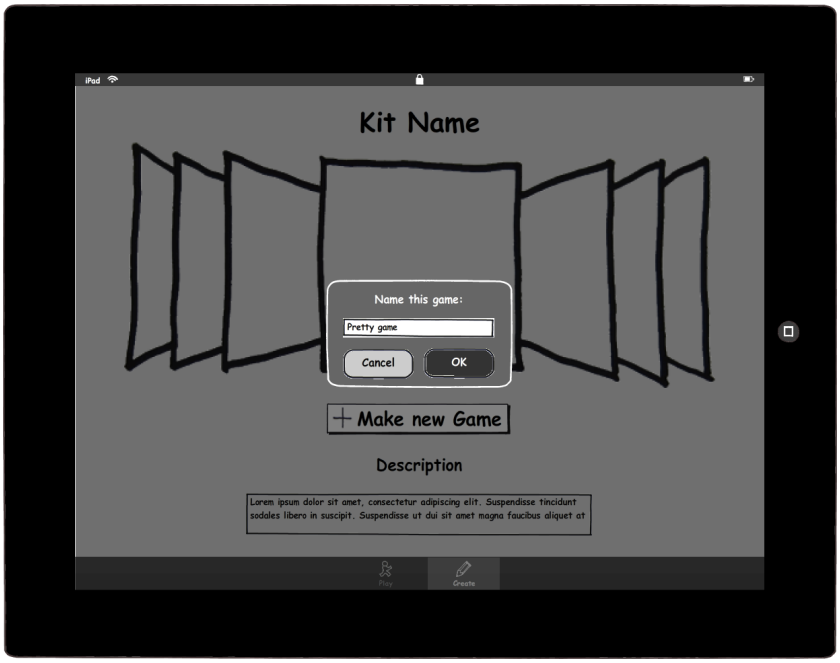

Figure A.4.: Paper Prototype, Game naming screen, displaying a modal window to set a name to the new game.

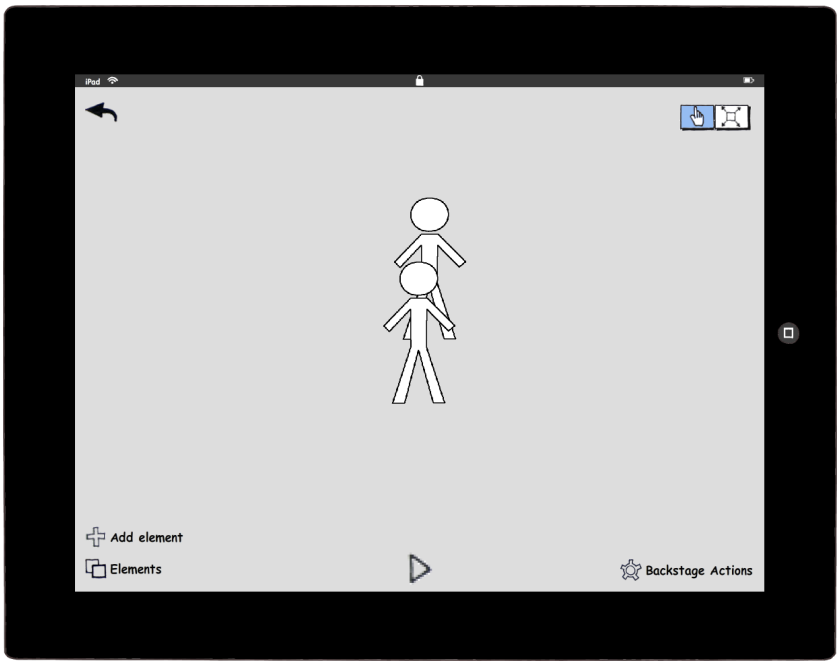

Figure A.5.: Paper Prototype, Scene Editing screen.

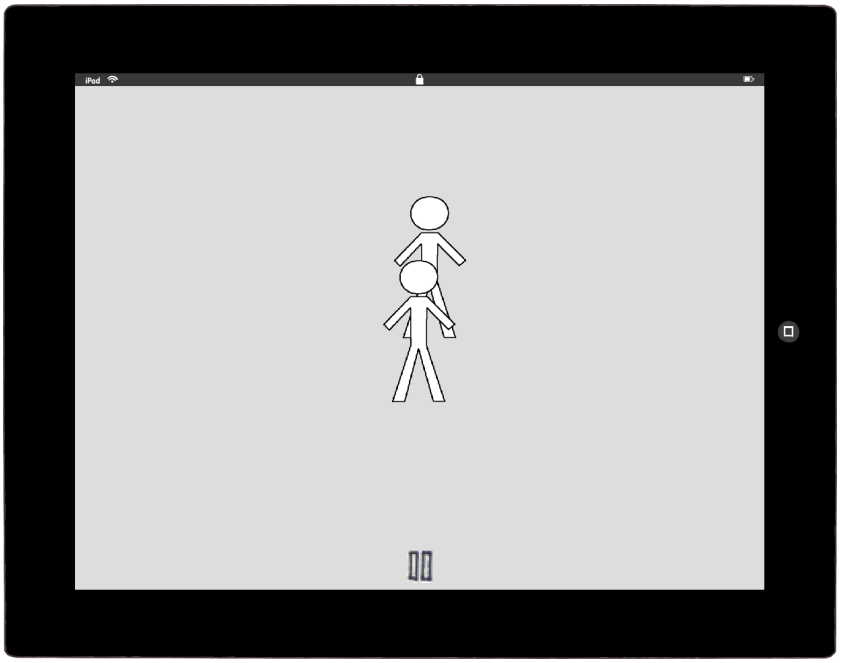

Figure A.6.: Paper Prototype, Scene Review screen.

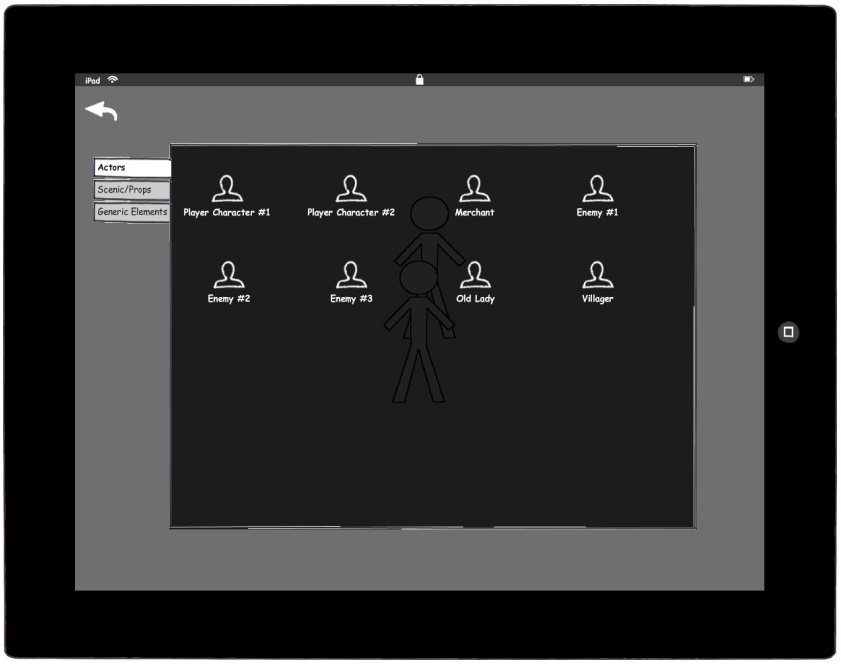

Figure A.7.: Paper Prototype, Add Element screen, where the user can add an actor or other elements pre-defined in the kit.

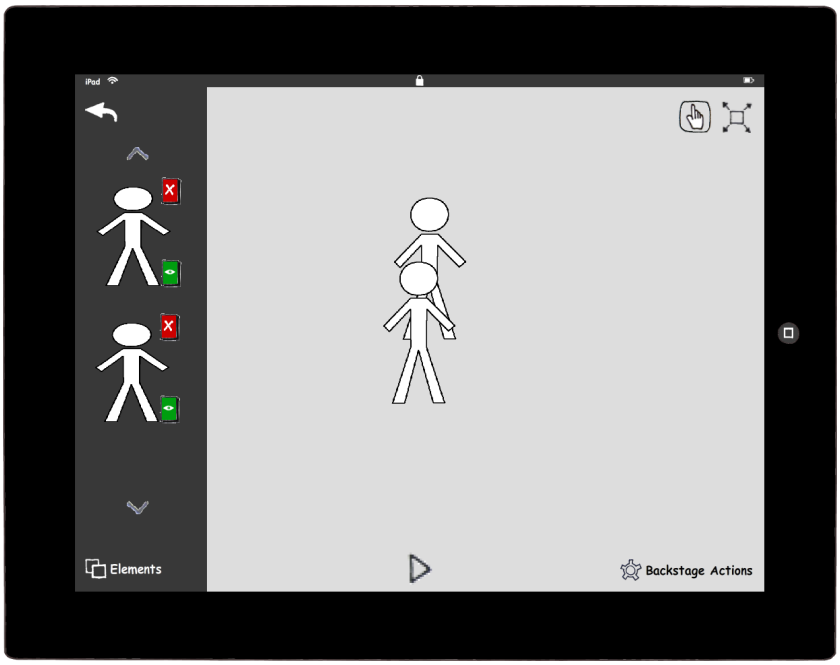

Figure A.8.: Paper Prototype, Scene Elements screen, displaying the elements list in a scrollable stack.

Figure A.9.: Paper Prototype, Scene Elements screen with an element selected and highlighted.

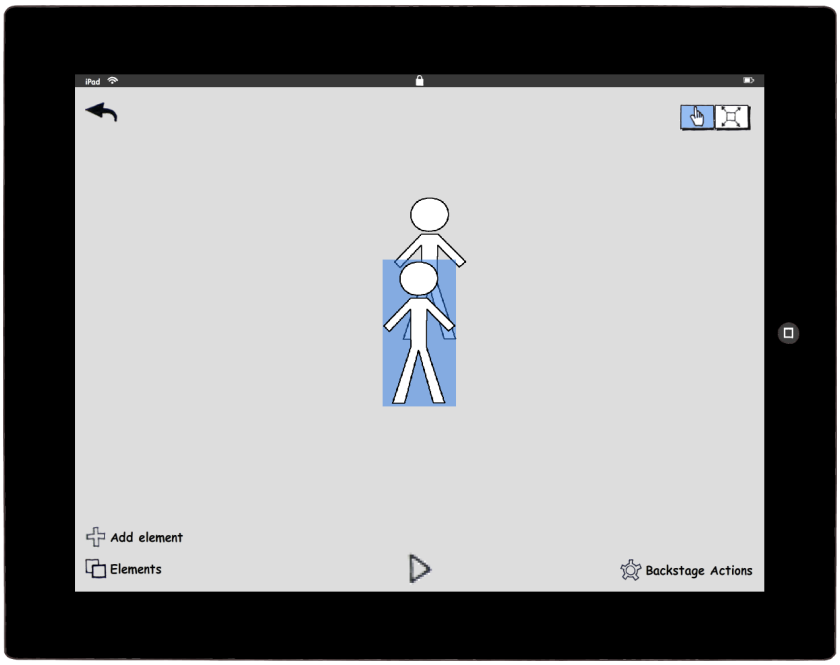

Figure A.10.: Paper Prototype, Scene Edit screen with an element selected and highlighted.

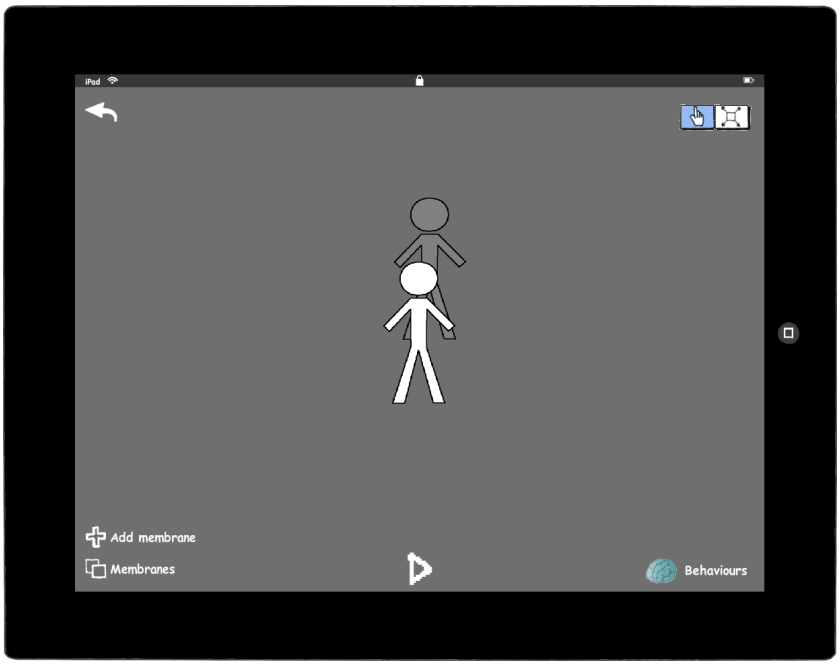

Figure A.11.: Paper Prototype, Actor Edit screen with the actor membranes visible and the other scene elements grayed out.

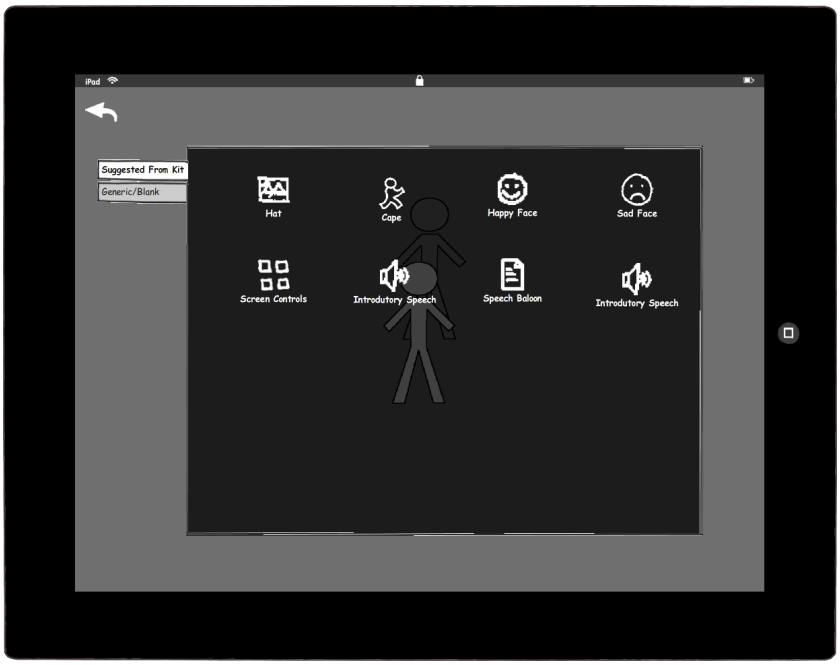

Figure A.12.: Paper Prototype, Add Membrane screen where the user can add more membranes to an actor.

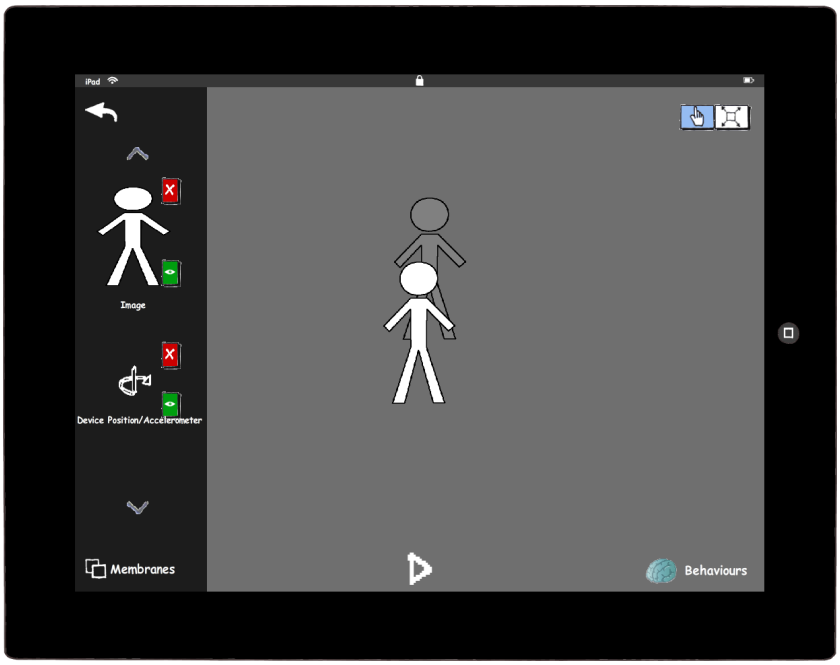

Figure A.13.: Paper Prototype, Actor Membranes screen, displaying the list of the actor membranes in a scrollable stack.

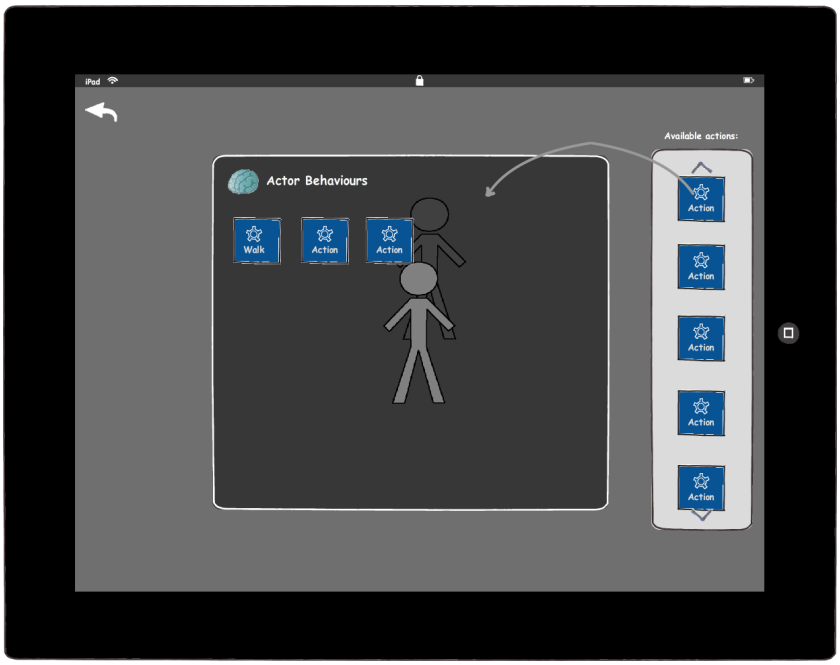

Figure A.14.: Paper Prototype, Actor Behaviours screen, where the user can set the behaviours associated with the actor.

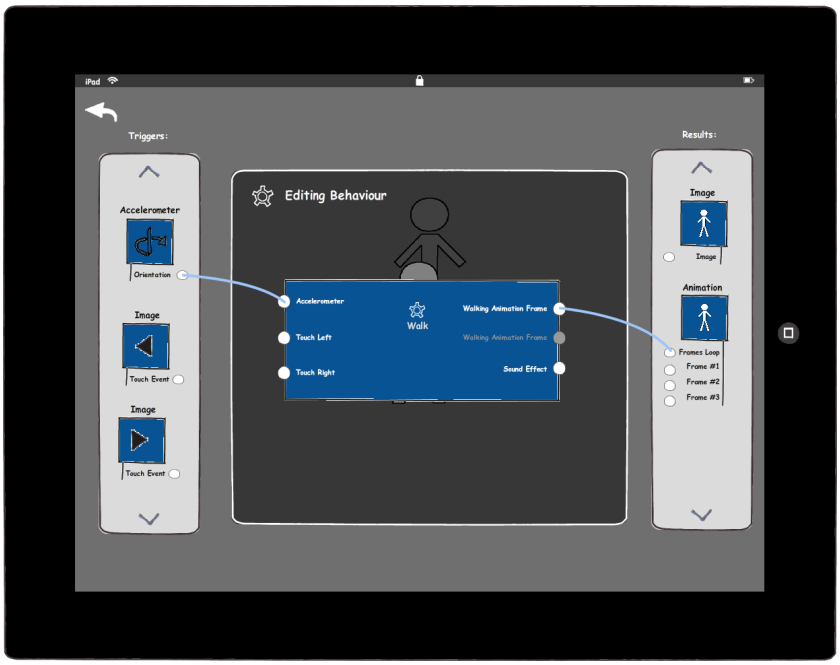

Figure A.15.: Paper Prototype, Actor Behaviour Edit screen, where the user can parameterize a behaviour.

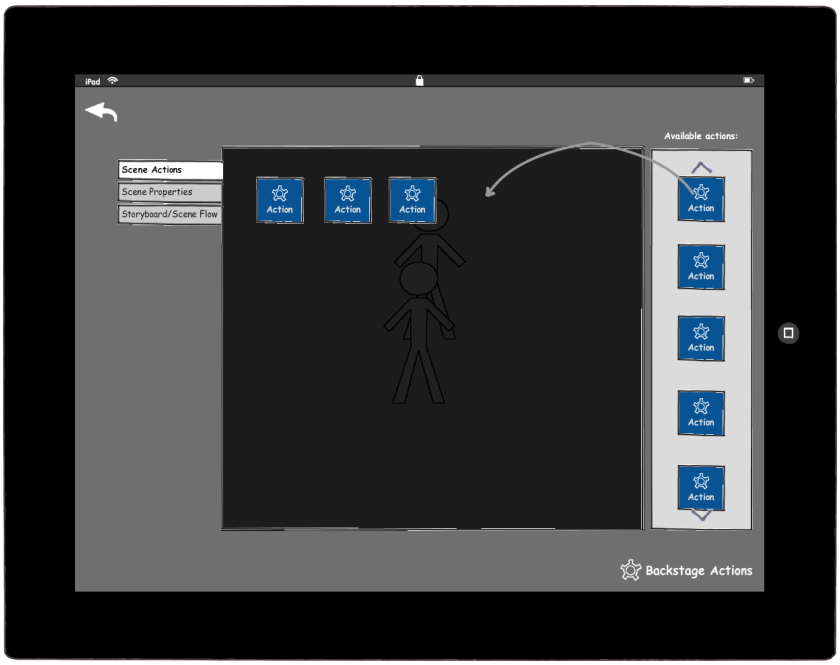

Figure A.16.: Paper Prototype, Scene Actions screen, showing the actions assigned to the scene.

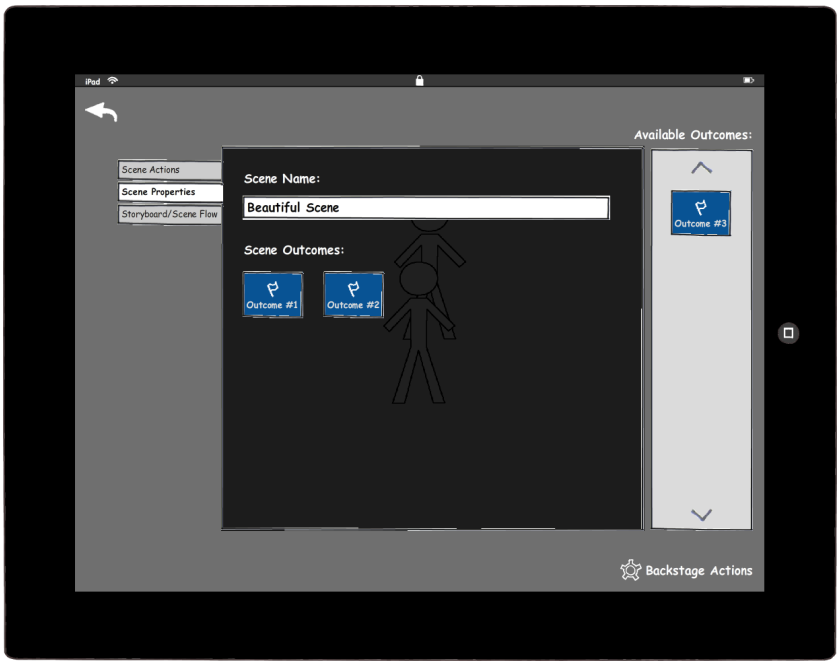

Figure A.17.: Paper Prototype, Scene Properties screen, where the user can set the scene name and possible outcomes.

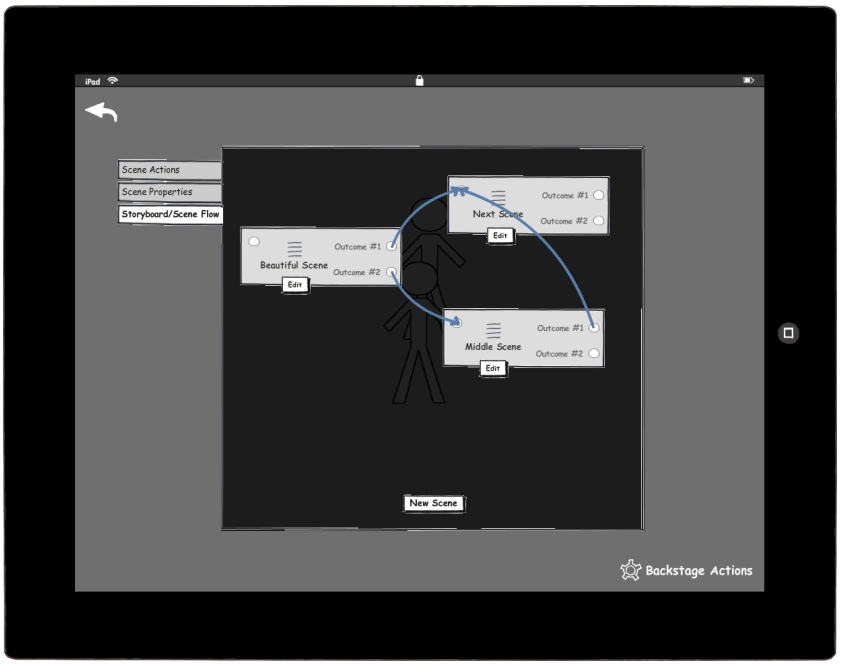

Figure A.18.: Paper Prototype, Storyboard screen, where the user add new scenes and defines the game scene flow.

Appendix B.

Graphical Interface Designs

B.1. Iteration 2

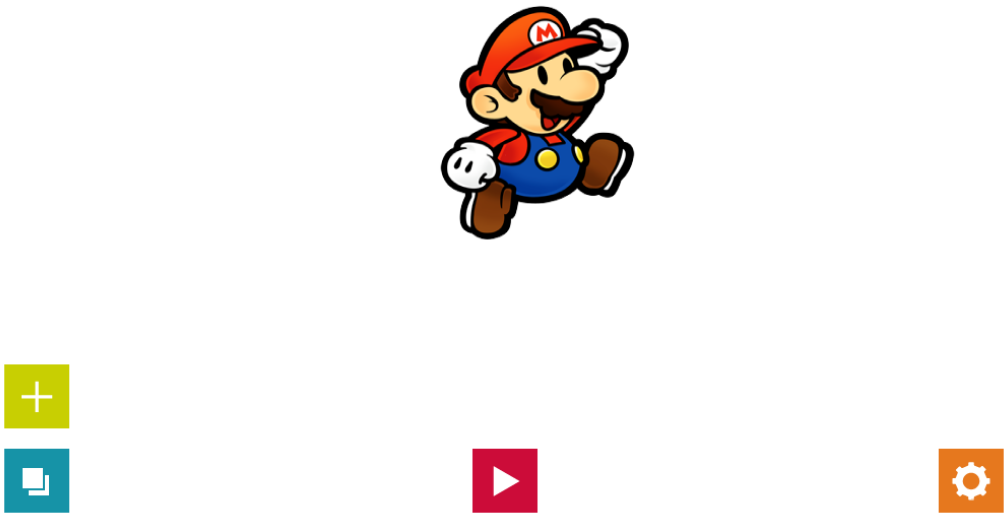

Figure B.1.: Detailed design of the Scene Edit screen.

B.2. Iteration 3

Figure B.2.: Detailed design of the Play screen.

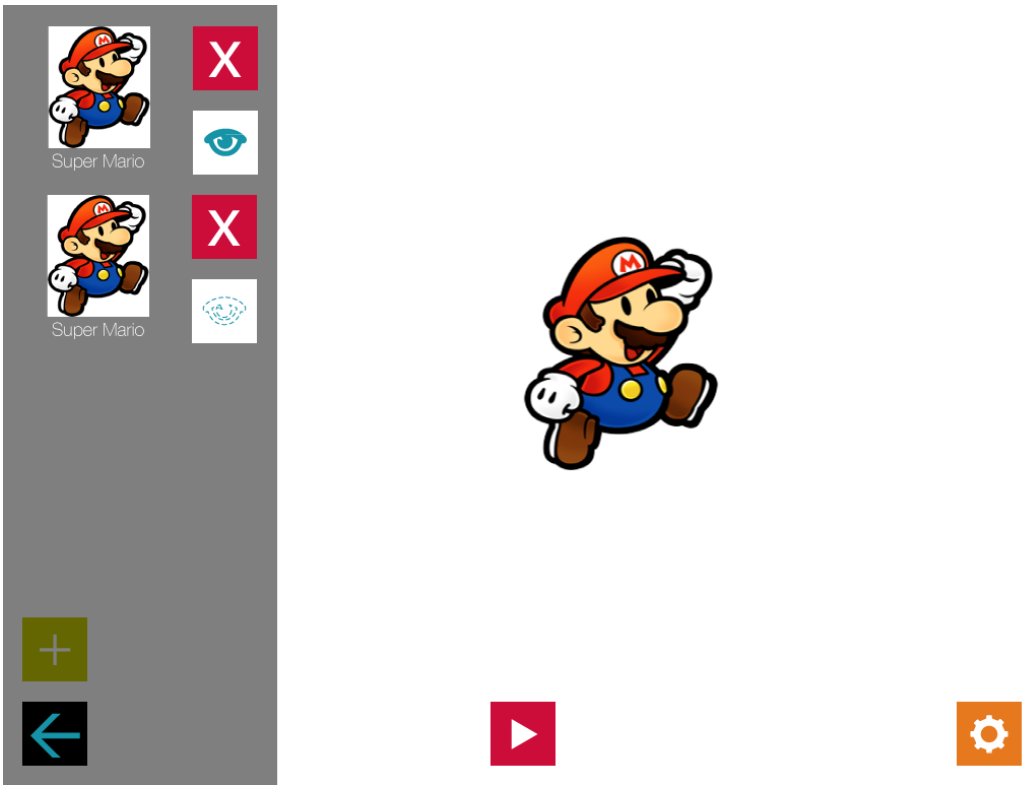

Figure B.3.: Detailed design of the List Elements screen.

Figure B.4.: Detailed design of the Add Elements screen.

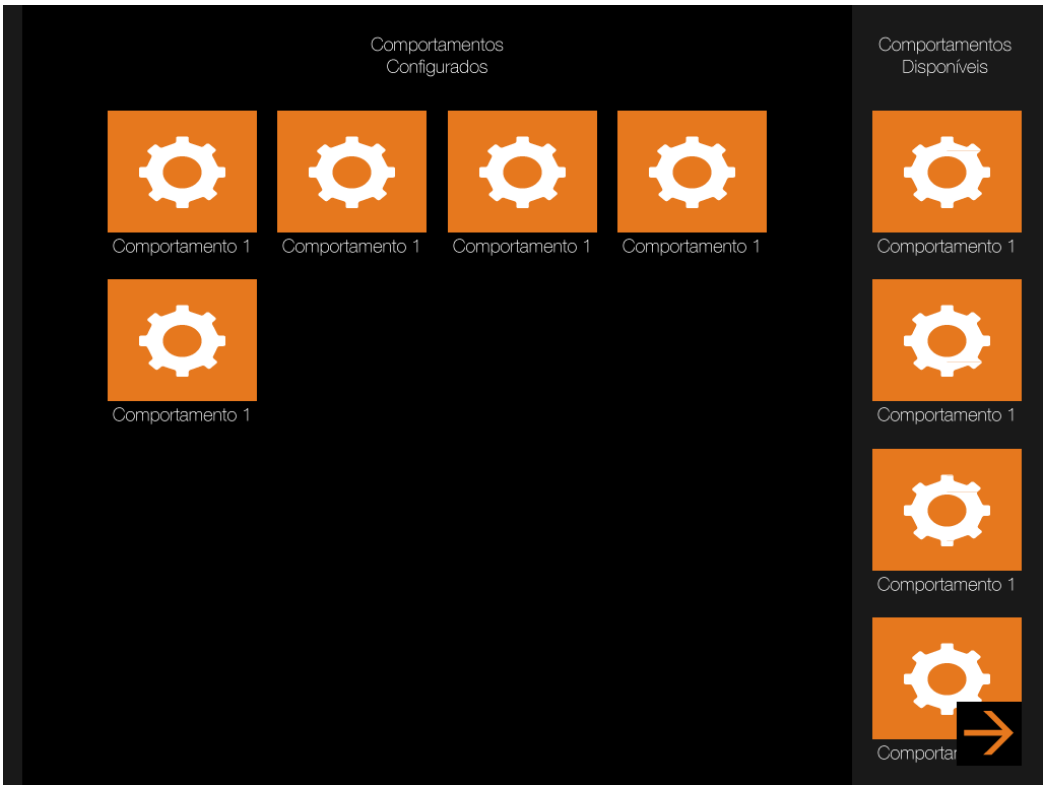

Figure B.5.: Detailed design of the Scene Behaviours screen.

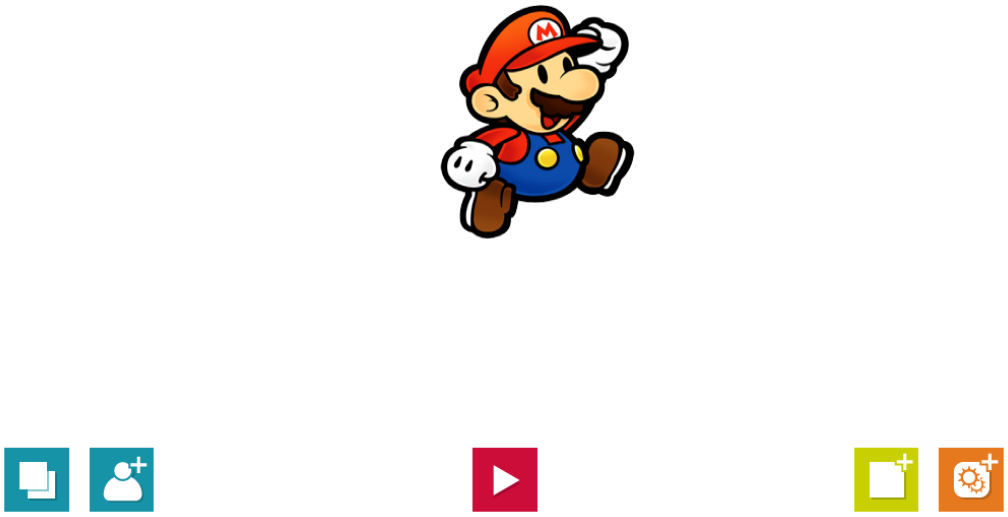

Figure B.6.: Detailed design of the Scene Edit screen.

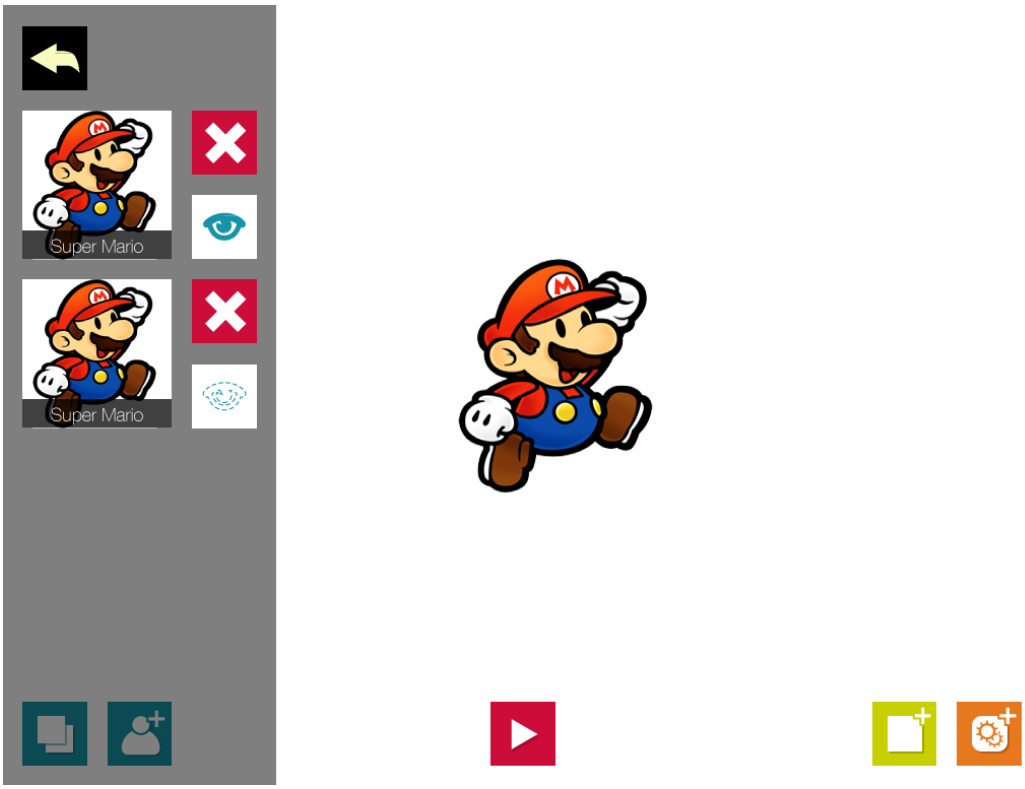

Figure B.7.: Detailed design of the List Elements screen.

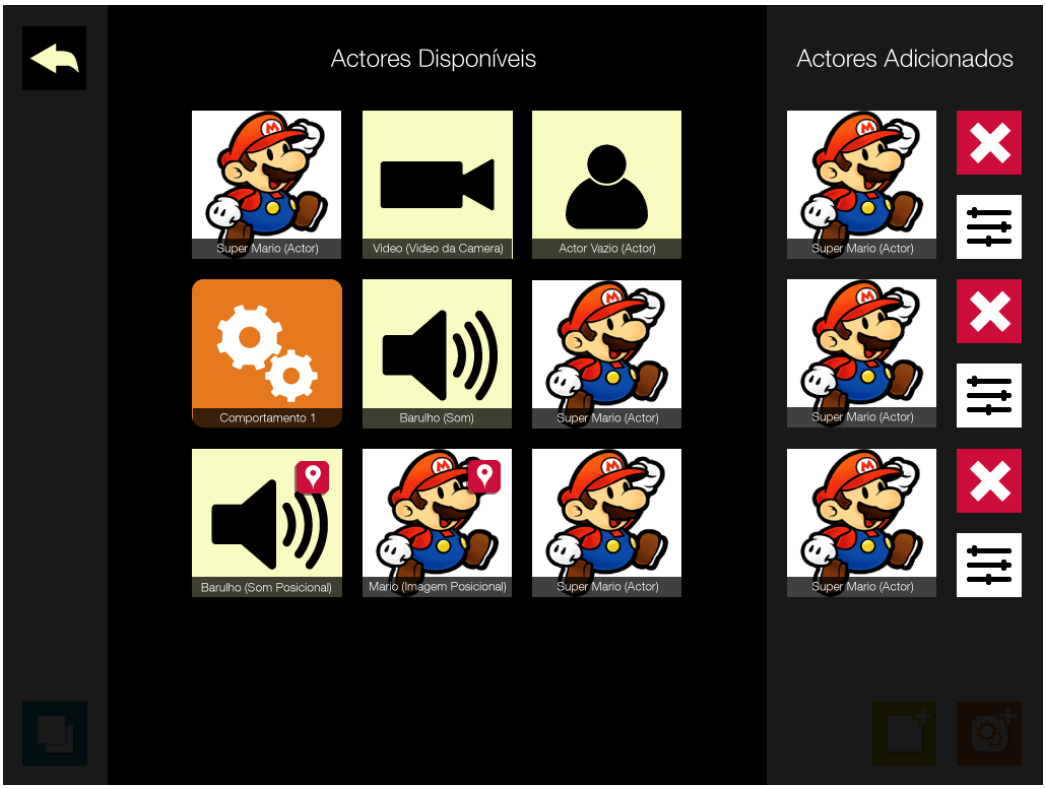

Figure B.8.: Detailed design of the Add Manage Elements screen.

Appendix C.

Iteration 2 Test Script

In this appendix we reproduce the test script that was given to the testers of the second iteration artifact.

C.1. Introduction

We would like to ask you to take part in a usability test for the dissertation project "Interactive Membranes", within the Master in Computer Engineering of the University of Coimbra. It aims to produce a tool (for tablets or pads) for the creation and fast prototyping of 2D, ubiquitous and augmented reality videogames, by a public without programming expertise. In this test you will use a prototype of the developing tool in order to solve/complete a specific case/example. For this you will have a list of written tasks. Please fell free to take as much time as you need to complete each task. In the end of each task please write down: 1) if you have completed the task; 2) if you used the detailed description; 3) problems that you had during the task; 4) and suggestions for the improvement of the interface . Your participation is extremely useful for the assessment and evaluation of the objectives of this interface and We would be very grateful for your help.

C.2. Test Goal

The main goal is create a game in a tablet, where you will play the role of a defender of a wall being attacked by Moors from any direction (180º degrees). The tablet allows you to move the crosshair in any direction so that you can find and shoot (by pressing a button) the attacking Moors. Then the goal is to eliminate every Moors before they reach the Wall, otherwise you will lose the game. In order to build this game you will use the prototype tool. In this tool the game builder play the role of a theatre director, therefore in the interface is represented the visible part of the scene (proscenium arch) with a virtual stage where the action of the play/game will take place. Additionally

you can add behaviours to the scene, that will be activated upon certain conditions and produce certain results.

C.3. Task List

Task Number	Task Description	Detailed Description	Completed (Y/N)	Used detailed description(Y/N)	Problems	Suggestions
1	Add enemy (actor) to the scene.	Press the button to add elements ("+"); from the available actors, click an enemy (one of the Moors) for adding it to the scene; select and drag the actor to wherever you want in the scene.				
2	Add crosshair (aim and button) to the scene.	See Task 1 (similar). The crosshair should be in the middle of the scene.				
3	Add behaviour for moving the crosshair with the compass to the scene.	Press the button of setting the scene ("cog"); from the available behaviours, select and drag "Crosshair movement" to add it to the behaviours of the scene; return to the main edition screen.				
4	Add behaviour to the scene for eliminate enemies with shoot.	See Task 3 (similar).				
5	Test the game.	Press the play button to launch the game; shoot an enemy (Moor); Press the pause button to return to the edition screen.				
6	Remove all scene elements	Press the button list of element (blue); remove each element by tapping the corresponding button.				
7	Add a background to the scene	See Task 1 (similar). Select one of the available backgrounds.				
8	Add multiple enemies to the scene (3 or more).	See Task 1 (similar). Drag the enemies to wherever you want in the scene.				
9	Add Wall to the scene.	See Task 1 (similar). Select the Wall.				
10	Add crosshair (aim and button) to the scene.	See Task 1 (similar). The crosshair should be in the middle of the scene.				
11	Add behaviour to the scene for the enemies go forward the wall.	See Task 3 (similar).				
12	Add behaviour to the scene for the enemies go forward the wall.	See Task 3 (similar).				
13	Add victory message to the scene and make it invisible.	See Task 3 (similar).Add victory message and drag it to the middle of the screen.				
14	Add defeat message to the scene and make it invisible.	See Task 3 (similar). Add defeat message and drag it to the middle of the screen.				
15	Add "Show victory message" behaviour to the scene.	See Task 3 (similar).				
16	Add "Show defeat message" behaviour to the scene.	See Task 3 (similar).				
17	Test the game.	See Task 4 (similar).				

Table C.1.: Iteration 2 Test Script Task List.

Appendix D.

Iteration 3 Test Script

In this appendix we reproduce the test script that was given to the testers of the third iteration artifact.

D.1. Introduction

We would like to ask you to take part in a usability test for the dissertation project "Interactive Membranes", within the Master in Computer Engineering of the University of Coimbra. It aims to produce a tool (for tablets or pads) for the creation and fast prototyping of 2D, ubiquitous and augmented reality videogames, by a public without programming expertise. In this test you will use a prototype of the developing tool in order to solve/complete a specific case/example. For this you will have a list of written tasks. Please fell free to take as much time as you need to complete each task. In the end of each task please write down:1) if you have completed the task; 2) if you used the detailed description; 3) problems that you had during the task; 4) and suggestions for the improvement of the interface . Your participation is extremely useful for the assessment and evaluation of the objectives of this interface and We would be very grateful for your help.

D.2. Test Goal

In order to build the game you will use the prototype tool. In this tool the game builder play the role of a theatre director, therefore in the interface is represented the visible part of the scene (proscenium arch) with a virtual stage where the action of the play/game will take place. Thus you can add and handle actors and other elements - images, animations, sounds and text, designated herein membranes - to the scene in order to build this theatre/game. Additionally you can add behaviours to the scene or actors, that will be activated upon certain conditions and produce certain results. These behaviours congregate all the required logic for a fast and easy way to build the game and for the operation of the game.

The main goal is creating the interface for part of a location-based game and is divided into three sequential objectives:

Objective 1 - Reading an introductory dialogue; Objective 2 - Listen and discover the direction of a specific geographical point through the sound; Objective 3 - Check in

Based on these broad objectives we drew up a game scenario where the player assumes the role of a knight who is suddenly called to return to the castle by the King and fulfills the command (Obejctive 1), however before the knight will have to find the lost horse thruogh the sound that emits horse (Objective 2). Once the knight get near the castle the knight use the available key to open the gate of the castle (Objective 3) and thus fulfill the order of the King. This is the scenario of the game whose interface is being built. and upon arrival at the castle using the key available within the castle to open the gate of the same (Objective 3) and so fulfill the order of his above. This is the scenario of the game whose interface is being built.

D.3. Task List

Task #	Task Description	Detailed Description	Completed (Y/N)	Used detailed description(Y/N)	Problems	Suggestions
1	Add the actor “King” to the scene and place it as you like.	Press the button to add actors (blue button with a figure and a plus sign “+”); from the available actors, select and drag the King to the added actors area in the right; Return to the main edition screen pressing the button return (arrow in the upper left corner). Select and drag the King to wherever you want in the scene.				
2	Edit the actor “King”.	Press the button to add elements (blue button with a figure and a plus sign “+”) (similar to task 1) and press the edition button of the king (button with three regulators below to the remove button) to enter the edit mode of that actor.				
3	Add the following membranes to the actor ”King”: “Scroll (image)”, “Objective 1 - Introduction dialog (Text)” and “Objective 1 - Button (Image)”.	Press the button to add elements/membranes (green button with a white square with a plus sign); select and drag each membrane to the added membranes area in the right; Return to the main edition screen of the actor pressing the button return (arrow in the upper left corner). Select and drag the King to wherever you want in the scene.				
4	Relocate the membranes added in the previous task in order to form a dialogue box near the image of the ”King”.	Select and drag the membrane to the desired place.				
5	Add behaviour “Adventure Objective 1 - Dialogue” to the actor ”King”.	Press the button to manage behaviours (orange button with cogs); select and drag the behaviour “Adventure Objective 1 - Dialogue” to the added behaviours area in the right; Return to the main edition screen of the actor pressing the button return (arrow in the upper left corner).				
6	Hide the membranes “Objective 1 - Introduction dialog (Text)” and “Objective 1 - Button (Image)”.	Press the button of the elements list (blue button with two overlapped squares); press the visibility button (blue eye) of the membrane and close the elements list by pressing the return button (arrow in the upper left corner).				
7	Add the following membranes to the actor ”King”: “Neighing horse (Positional sound)”; “Objective 2 - Dialogue Find Horse (Text)”; “Objective 2 - Dialogue Try again (Text)”; “Objective 2 - Dialogue Horse Found (Text)”; “Objective 2 - Button (Image)”.	See task 3 (similar).				

Table D.1.: Iteration 3 Test Script Task List Part 1.

Task #	Task Description	Detailed Description	Completed (Y/N)	Used detailed description(Y/N)	Problems	Suggestions
8	Relocate the membranes added in the previous task (except the positional sound) in order to form a dialogue box near the image of the King (Sugestion: After relocate each membrane hide it so you can relocate the others).	See tasks 4 and 6 (similar).				
9	Hide all the membranes of the actor King, except the followings: Neighing horse (Positional sound); "Scroll (image)", "Objective 1 - Introduction dialog (Text)" and "Objective 1 - Button (Image)" and "King talking (animation)".	See task 6 (similar).				
10	Add behaviour "Adventure Objective 2 - Listen and Click" to the actor "King".	See task 5 (similar).				
11	Leave edition of the actor "King".	Return to the main edition screen of the scene pressing the button return (arrow in the upper left corner).				
12	Add the actor "Castle" to the scene.	See task 1 (similar).				
13	Move the actor "King" to the front of the actor "Castle".	Press the button of the elements list (blue button with two overlapped squares); press and hold (for about 2 seconds) the miniature of the actor "King" and drag it to relocate it in the list of elements; close the elements list by pressing the return button (arrow in the upper left corner).				
14	Edit the actor "Castle".	See task 2 (similar).				
15	Add the following membranes to the actor "Castle": "Open Castle (Coimbra) (Positional image)"; "Key (Image)"; "Tinkling keys (Sound)"; "Unlock door (sound)".	See task 3 (similar).				
16	Relocate the membrane "Key (Image)" to the upper right corner.	See task 4 (similar).				
17	Add behaviour "Adventure Objective 3 - Check in" to the actor "King".	See task 5 (similar).				
18	Hide the membranes "Open Castle (Coimbra) (Positional image)" and "Key (Image)" of the actor "Castle".	See task 6 (similar).				
19	Leave edition of the actor "Castle".	See task 11 (similar).				
20	Test the game.	Press the play button to launch the game; follow the instructions in the dialogues until open the castle; press the pause button to return to the edition mode.				

Table D.2.: Iteration 3 Test Script Task List Part 2.