Utrecht University

Game and Media Technology
Department of Information and Computing Sciences

Master Thesis
ICA-4098927

RGB Slemmings: An augmented reality game in your room

Alexandros Panayiotou
Student Number: 4098927

Supervisor 1: Dr. Ronald Poppe
Supervisor 2: Prof. Remco Veltkamp

Utrecht, February 2016

Acknowledgments

This Thesis, would have not been possible without the contribution and help of some people, so I would like to express my gratitude to them.

First, I would like to thank my supervisor dr. Ronald Poppe for his help, patience and insights he offered me in our weekly meetings, without which I would still have been stuck in the first obstacle of this project. Also I want thank my friend and roommate Nikos for all his help throughout the project, without him I wouldn't be able to find any participants for the user test. Last but not least I would like to thank my family and friends for their never-ending support throughout my studies, which helped alleviate all the stress and tension that appeared in the last 2.5 years.

Contents

Chapter 1

Introduction

1.1. Motivation

Augmented reality is the direct or indirect view of reality supplemented with computer generated content such as graphics, audio or GPS data. It has been around for many years, it started with the first head mounted display prototype by Ivan Sutherland in 1968 [1] (see Figure 1) but wasn't until the early 90s for the term augmented reality to be introduced by Caudell and Mizell [2]. Today, augmented reality has become a distinct field of research and is starting to turn into a household name. Augmented reality applications apply in various fields and big companies such as Microsoft and Google have shown major interest in this field, investing large amounts of money on augmented reality related hardware and software such as the Microsoft HoloLens [3] and the Magic Leap [4] (See Figure 2).

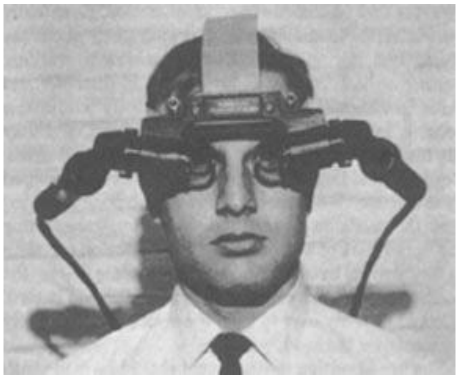


Figure 1: Early head-mounted display device developed by Ivan Sutherland at Harvard University [1].

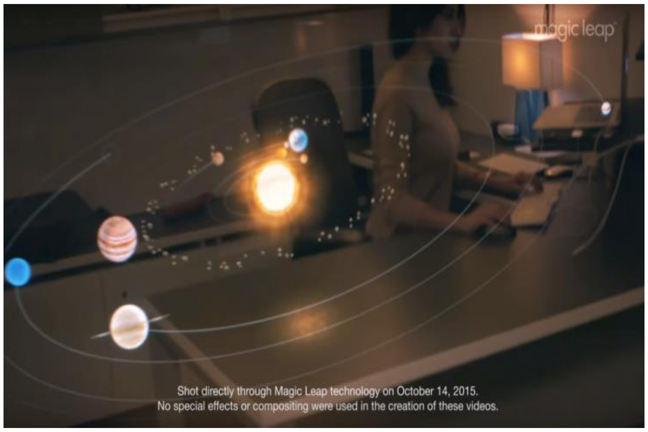

magic leap
Shot directly through Magic Leap technology on October 14, 2015.
No special effects or compositing were used in the creation of these videos.

Figure 2: Early tests of the Magic Leap technology. Screenshot taken from YouTube video Magic Leap Demo [5].

Furthermore, there has been an increase in attention on augmented reality applications that augment the whole physical environment that surrounds the user, mostly due to the research and development of mobile augmented reality headsets and technologies by Microsoft [3] and Google [4]. An example of such an environment can be a room. In an augmented room the user explores the virtual environment by moving around his actual surroundings. In order to achieve fully augmentation of the user's surroundings, the shape and dimension of the environment should be known beforehand to the application. This requirement makes it is necessary to have a 3D representation of the environment such that the application can map the virtual world to the real world so to achieve its augmentation. A common representation is a 3D model of the environment which can either be made by a 3D modeling tool or by an automatic reconstruction process consisting of scanning the environment and reconstructing it into a 3D model.

The upcoming shift to augmented reality applications which use the environment makes it relevant to research methods which automate the reconstruction of 3D environments and document user experiences from this type of applications. Examples of augmented environment applications are entertainment games, serious games for training, education, urban planning or even room renovations.

1.2. Scope

The goal of this project is to develop and evaluate an augmented reality game which uses the environment, in this case a room, as its levels. The game, RGB Slemmings is inspired by the 90s platform puzzle game Lemmings [6] and it will be played on a handheld mobile device.

To use the room as a level in the game, an accurate 3D model is needed from which the levels will be designed. The steps taken during the development of the game were assembled into a pipeline which will automate the room reconstruction. The pipeline steps consisted of scanning and mapping the room using a Microsoft Kinect [7], reconstructing the resulting scan as a 3D model and use the model as a stage for our augmented reality game.

To evaluate the game, we conducted a user test in which the participants had to play the game and answer a questionnaire. The two aspects we want to evaluate are first the concept of an augmented reality game in which the player must move around the room to play it, and second, the amount of immersion the game offers due its adjustment to the player's physical environment.

The steps we will follow will be to create a pipeline which will automate as much as possible the procedure of creating the model of the environment and use the model in the development of the augmented reality game. Finally, we will test the game with a group of volunteers, who will later fill out a questionnaire about their experience.

More particular, in this thesis we have two research goals. First, we want to create a game which uses its environment and gives the player the illusion that the real world and the virtual world are the same. To do so we will measure the amount of immersion a player has while playing the game through his responses on the questionnaire. Secondly we will measure the quality of the reconstructed room our pipeline. If the quality of the output is good then we have succeeded in creating a pipeline which automates the room reconstruction.

1.3. Approach

First we must create the 3D environment that will be used in our game. We want to automate this process, as it will make it easier to use different environments for our game. For that we designed a pipeline. The pipeline consists of two stages: the Offline and Online Stage. The Offline part is typically performed once and consists of the development process of the game. More specifically, in the Offline stage we have the room scanning, the 3D model reconstruction and the game development. The Online stage consists of playing the game, i.e. the augmented reality part, which is handled by the augmented reality library. An overview can be seen in Figure 3.

The first step of the Offline part is the scanning phase for which we use a 3D mapping tool along with the Kinect to scan the room and create a point cloud. Next we use a 3D reconstruction algorithm for converting the point cloud to a 3D model. The resulting 3D model will then be used in the development of the game.

As our goal lies in the development and evaluation of RGB Slemmings, hence some steps in the pipeline were completed with the use of pre-existing tools. The reason is firstly because the scope of this work is not to implement our own tool for each step. Secondly because of knowledge and time limitations e.g. implementing an augmented reality library from scratch is not a trivial task. As a result for most steps we were limited by the available technology and techniques. The steps that were implemented, were the 3D reconstruction algorithm and the game development.

For the game development part we have found different augmented reality tools and analyzed their features in order to find which are more useful for developing an augmented reality game which uses its environment.

The levels were designed in a way so that the whole room was used during gameplay hence “forcing” the player to move around the environment. For the evaluation phase we designed the levels and tested the game with a group of participants. We analyze both objectively and subjectively how they played the game.

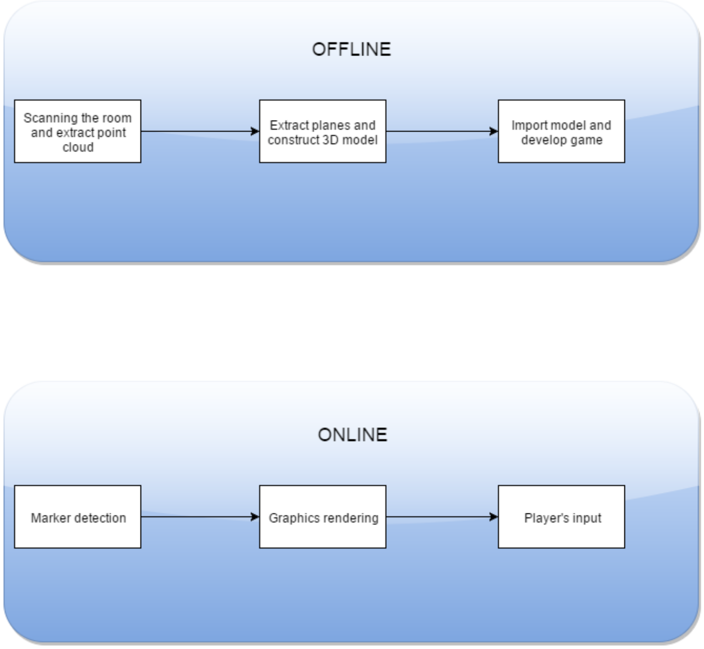

OFFLINE
Scanning the room and extract point cloud
Extract planes and construct 3D model
Import model and develop game
ONLINE
Marker detection
Graphics rendering
Player's input

Figure 3: Offline and Online parts of the pipeline

1.4. Overview

Chapter 2 will discuss the background of the project. We will start by discussing some augmented reality games currently in the market in order to show the current state of commercial augmented reality games. Next, we will discuss the possible technologies that can be used for each step of the pipeline. Moreover, we will introduce and explain concepts so to help the reader to further understand the technical details for each technique.

Chapter 3 will discuss in detail the techniques we have used for each step, and motivate our choices. We will explain how each technology works and how the pipeline steps are connected with each other. We will start with the how we did the room scanning using the Kinect and how we created a model from these scans. Next we will make an evaluation of the results. Finally, we will discuss about the augmented reality SDK we used for our game.

Chapter 4 will discuss the RGB Slemmings game. We will discuss about the game mechanics and components and give the reader an idea of how the game looks like and is played.

Chapter 5 will discuss the game evaluation. First we will discuss about the user test procedure and the questionnaire used. Next we will analyze the data gathered from the game sessions and the questionnaire and draw conclusions about the participants’ experience.

Chapter 6 concludes the Thesis by discussing the contribution, conclusion and future works.

Chapter 2

Background

In this part we will describe the background of this project. First we will discuss about some commercial augmented reality games in market and the technologies used in each of them. Next we will mention the different technologies used for each step of the pipeline. First we will discuss about the mapping phase which is the phase where the room is scanned and how these scans were connected to each other into one complete scan. Next we will discuss about the 3D reconstruction algorithm and finally, we will explain what augmented reality is and the tools we used for developing our game.

2.1. Augmented Reality Games

In this part we will discuss about some notable augmented reality games which have been published in the past years. All of the following games run in handheld mobile devices unless it is stated otherwise.

Table Zombies AR by SRG United Solutions [8] game is a shooter game which goal is to shoot the zombies that appear on the marker which is placed usually on a table (see Figure 4). Table Zombies uses marker detection technology which means that the player must print a game specific marker which must be detected by the device's camera. The game does not use its physical environment since the game takes place entirely on the top of the marker and it does not matter if the marker is either on a printed paper on the table or an image shown on a computer screen. While Table Zombies were working only with marker detection, the game Temple Treasure Hunt by Thought Shastra Solutions [9] uses both marker and location based augmented reality. Temple Treasure Hunt is a puzzle Scavenger hunt-like augmented reality game which features different modes like outdoor mode in which the game becomes a location based game using GPS or the indoor mode where the game uses its own markers (see Figure 5). The innovation of this game is that it gives the player the possibility to create his own levels in the indoor mode and position the markers wherever he desires. The previous games use marker based tracking while AR Invaders by Soulbit7 [10] works solely by using the device's accelerometer and compass. AR Invaders is a shooter game in which the user shoots alien UFOs which fly around the player's screen. The game needs an initial calibration from which it determines the player's pose along with the orientation and direction of the ground and sky. In that way the game can make the UFOs fly around the player giving the illusion of an actual attack from flying saucers (see Figure 6). However, due to the instability of the accelerometer and compass, a bad initial calibration makes the game unplayable.

In the last years Sony and Nintendo have made an attempt to enter the augmented reality market with games on their latest portable game consoles, the PS Vita and Nintendo 3DS.

An example of a PS Vita augmented reality game is PulzAR^{tm} by Exient Ltd [11]. PulzAR is a typical laser mirror puzzle game where the user must place the mirrors in a certain position in order to direct the lasers by deflecting them towards a desired destination (see Figure 7). The game uses the PS Vita camera and the AR Cards which act as markers which come bundled with the PS Vita. The mirrors' position are mapped by the markers' physical position on a surface. The player's interaction with the game is achieved by moving the markers.

Similarly to the PS Vita, the Nintendo 3DS uses its own cards as markers in order to play games. However, the Nintendo 3DS has two outer cameras used in the augmented reality apps, the combination of which provide very impressive visual results. The game Archery [12] (see Figure 8) for example manipulates the real world with the creation of holes in the furniture on which the marker is on.

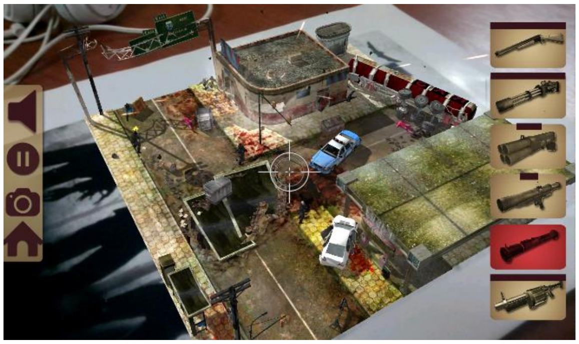


Figure 4: Screenshot of the game Table Zombies AR. Image taken from Google Play [13].

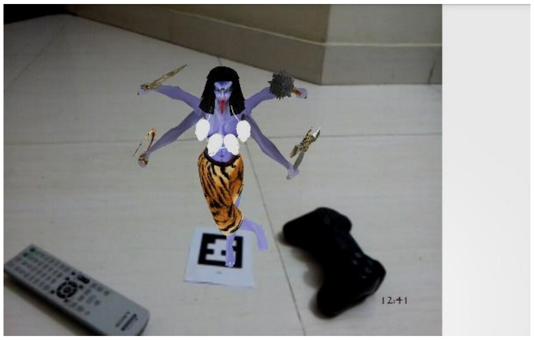


Figure 5: Screenshot of the game Temple Treasure. Image taken from Google Play [14].

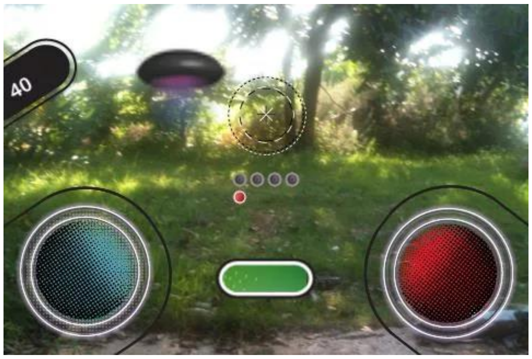


Figure 6: Screenshot of the game AR Invaders. Image taken from Google Play [15].

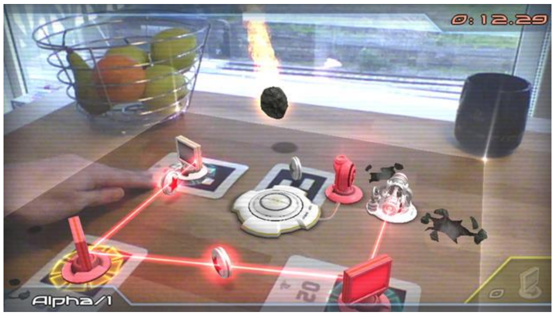


Figure 7: Screenshot of the game PulzAR. Image taken from Kotaku.au.com [16].

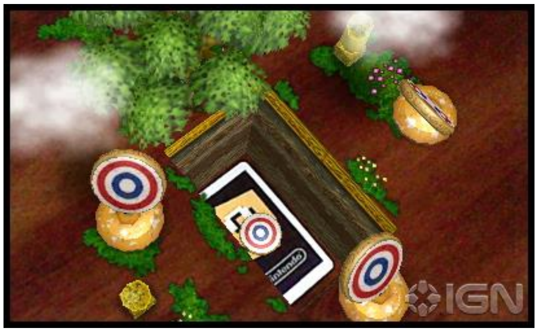


Figure 8: An example of surface manipulation in Nintendo 3DS game Archery. Image taken from IGN [17].

2.2. Pipeline Technologies

In the next section we will discuss some technologies used in the pipeline followed during the development of the game.

2.2.1. Room Scanning

A common algorithm used when scanning a room is a SLAM (Simultaneous Localisation and Mapping) algorithm. SLAM is a process by which an agent can build a map of an environment and at the same time use this map to deduce its location. The agent can be a robot, a visual sensor such as a laser scanner which can be held and moved by a person, or even a self-driven car. The agent must be equipped with a sensorial system capable of taking measurements of the relative location between landmarks and the agent itself. The most commonly used sensors can be categorized into *laser-based*, *sonar-based*, and *vision-based* systems [18]. Laser ranging systems usually follow the time of flight principle, which work by sending a laser pulse in a narrow beam towards the object and measuring the time taken by the pulse to be reflected off the target and returned to the sender. Sonar-based systems use inertial sensors, such as odometers, to provide measurements and recognition capacities hence they lack of appearance data. Its dependence on inertial sensors implies that a small error can have large effects on later position estimates. Vision systems use solely vision based data, so they need high resolution in order to extract and match features. The computation cost is considerably high as features need to be extracted in every frame. Vision is used to estimate the 3D structure, feature location and robot pose.

In our case for an agent we used a Kinect laser-based sensor [7] connected to a computer and mapping our surroundings by moving in the room. During scanning, the Kinect's position and orientation in the room is estimated while the surroundings are been constructed through the scans.

The scans are represented by what is recorded by the Kinect. Kinect captures both a regular image and depth image which is represented by a point cloud. A point cloud is a set points in space, in our case three-dimensional space, each of them having their own X, Y, and Z coordinates. The point cloud represents the set of points measured by the Kinect using time of Flight technology. Time of Flight technology estimates distances by measuring the time a light signal takes to return to the camera for each point of the image [19].

A challenge faced when scanning the room is to properly "stitch" all scans together into one complete scan. To face this challenge each new scan is compared with all previous scans in order to check if it is part of any of the previous ones and merge them together if it is. The process involved when trying to find a match between the current and a previously visited locations in SLAM is called loop closure [20]. To find a match the system compares the current location with the previous ones, if they are similar enough then they are merged together. The comparison of the different scans is done with either visual criteria or by using geometrical criteria. An example of visual criteria is the use of SURF visual descriptors in combination with bag-of-words. Speeded Up Robust Features (SURF) descriptor was developed by Bay et al [21], and is a robust rotation-invariant interest point detector and descriptor. The goal of their work was to develop a descriptor which is fast to compute while not sacrificing performance. It relies on integral images which is an image representation developed to reduce any computation time [22]. Moreover, only 64 dimensions are used, reducing the time for feature computation and matching, and increasing simultaneously the robustness. Bag-of-words works as a vocabulary of words, where words are the extracted features are quantized into clusters. Whenever a new image appears, its features are extracted and compared with the ones in the visual words for classification. On visual criteria along with RGB information from which the SURF features are extracted, there are 3D information captured along the images such as a 3D point cloud. During loop closure the points are connected together based on the result of the SURF features comparison, and they use some transformations to refine the quality of the point cloud. An example of geometric criteria comparison is the use of Iterative Closest Point (ICP) [23] which compares the smallest possible transformations needed for matching one point cloud with another.

A loop closure technique can be seen in Real-Time Appearance-Based Mapping (RTAB-Map) tool by Labbé and Michaud [24] which uses SLAM and loop closure and outputs a point cloud. A common problem in loop closure is that as the number of previous scans gets bigger the comparison of each new one with the previous ones gets more time consuming thus making it impossible to run in real-time for a large number of scans. In RTAB-Map a novel memory management system has been developed which deals with this problem. They keep the most recent and frequently observed locations in the Working Memory (WM), and transfer the rest into a Long-Term Memory (LTM). When a match is found between the current location and one stored in WM, associated locations stored in LTM can be remembered and updated. Real-time SLAM is accomplished by limiting the number of the scans used in the comparison to the size of WM, hence the WM's size limit is the maximum number of scans that can be compared before a new scan arrives. RTAB-Map is fully parameterized to the user's needs. For example the size of WM can get bigger by reducing the frequency in which new scans are captured. The scan comparison is done using SURF descriptors in combination with a bag-of-words. SURF was used in RTAB-Map due its high performance and speed. Bag-of-words works as a visual dictionary where extracted visual features. The bag-of-words is stored in the WM, hence each time a new frame appears its features are extracted and compared with words already in the dictionary and makes a loop closure check.

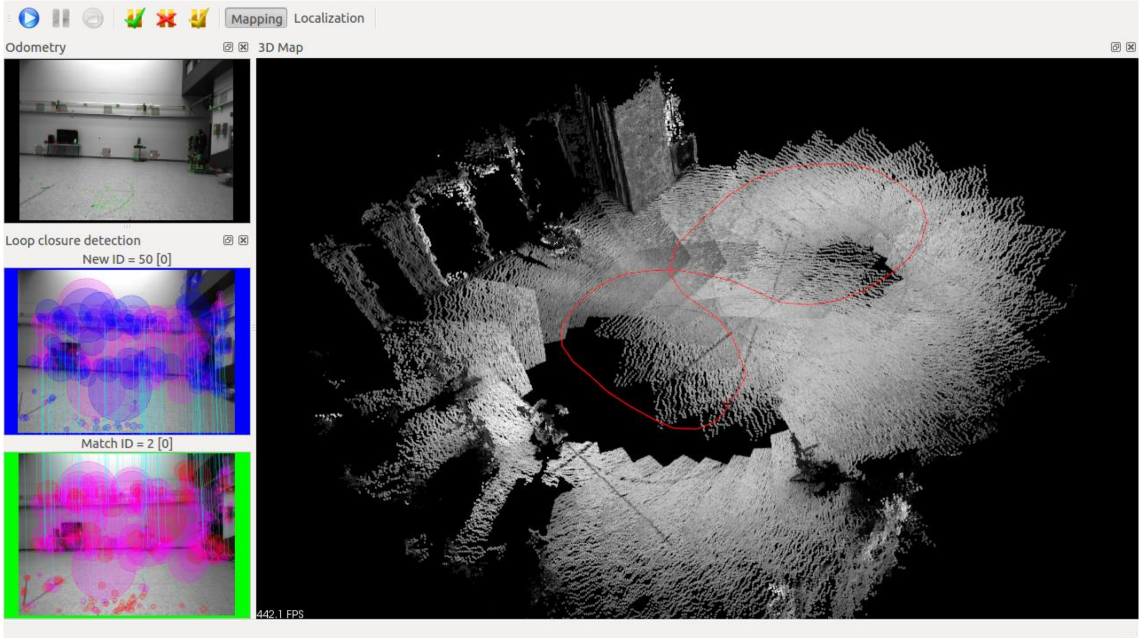


Figure 9: SLAM and loop closure example. On the right we see the robot's movement with a red line. On the left we see the sensor's view. If there is match with a previous scan we have a loop closure. Image taken from RTAB-Map page on GitHub [25].

Another option for the room scanning would be A. Newcombe et al [26]. In this work scanning and 3D reconstruction is done simultaneously, they also use SLAM with geometric criteria for extracting information. They use Kinect for estimating the depth data. Their system uses depth maps captured from Kinect and perform real-time dense SLAM, building the 3D scene model incrementally while simultaneously tracking the sensor's agile motion using all of the depth data in each frame. The system is consisted by three components: Surface measurement, Surface reconstruction, Surface prediction and Sensor pose estimation. The *surface measurement* is the pre-processing stage, where a dense vertex map and normal map pyramid are generated from the raw depth measurements obtained from the Kinect device. On *surface reconstruction* it updates the global scene process, determines the devices pose by tracking the depth data from the next frame. *Surface prediction* closes the loop between mapping and localization by comparing the live depth data with the already reconstructed parts. They use ray-casting into the estimated frame to provide a dense surface prediction against which the live depth map is aligned. In *Sensor pose estimation* a live sensor tracking is achieved by using a multi-scale ICP alignment between the predicted surface and current sensor measurement. Their implementation on the GPU makes sure that that they can handle the data at frame-rate.

The advantage of this work is that it creates a mesh on run time without the need of a 3D reconstruction process and does not face problems that appear in vision-based SLAM and loop closure such as low light conditions. However, it is difficult to control the level of detail of the resulting scan and remove any possible noise due to the fact that the output is already 3D model. Having the scan as a point cloud makes it easier for post-procession.

2.2.2. 3D Reconstruction

A point cloud is a common and descriptive way of visualizing 3D objects but are not used in 3D games as most of them use 3D models which consist meshes. As a result for our game we need to convert the point cloud into a polygon mesh representation. A polygon mesh is a collection of vertices, edges and faces that define the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals, or other simple convex polygons, since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes. An important challenge that should be faced in this step is noise occurred during the scanning phase. Noise reduction is handled by either a preprocessing noise removal step such as using a statistical outlier filter or by the 3D reconstruction algorithm itself.

In the work of van Lankveld et al [27] they created rectangular meshes by from the extracted planes from the input point cloud. Their goal in this work was to develop a one-parameter algorithm that can identify point sets on a plane for which a rectangle is a fitting boundary and that there is no large part of the rectangle which is empty of points. More particularly there are two main steps in this work, first to find all the planes in the point cloud given as input and second to find the best fitting rectangle for each plane. The output of this algorithm will be the scanned area represented by rectangular planes. Furthermore, any other scanned objects that cannot be represented by planes will be considered as noise and be removed, which in a way works as a filter from any occurring noise by the scanning phase.

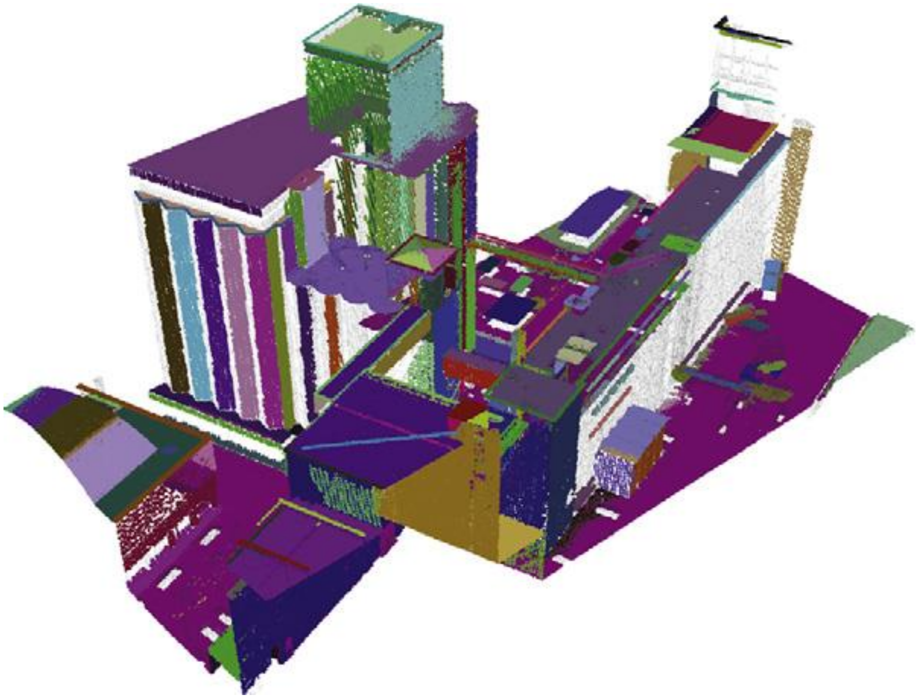


Figure 10: A resulting example from the work 'Identifying rectangles in laser range data for urban scene reconstruction' by Lankveld et al [27].

2.2.3. Augmented Reality

According to Azuma et al [28] augmented reality is the enhancement of the perception and interaction of the user with the real world by the supplementation of 3D objects that appear to coexist with the objects of the environment. Also Azuma et al [28] stated that an augmented reality system should have the following three properties:

1. Blends real and virtual in a real environment
2. Real-time interaction
3. Registered in 3D

The first property means that computer generated graphics and real world scenes should coexist in the same view of the user. Real-time interaction means that the user can interact with the augmented reality objects and they should respond the to the user's actions on real-time. Registration refers to the accurate alignment of real and virtual objects. The coexistence of graphics with real world in a screen is not enough to create an augmented reality experience, i.e. a UI that appears on screen along with real world objects is not an augmented reality element. Without accurate registration, the illusion that the virtual objects exist in the real environment will fail.

Augmented reality applications need specific devices which are equipped with enabling technologies. Enabling technologies are display technologies that combine the real and virtual worlds, followed by sensors and approaches to track user position and orientation for correct registration of the virtual with the real, and user interface technologies that allow real-time 3D interaction [29].

The display is usually a visual display in which the user can see the augmented graphics. There are two main types of display the optical see-through and video see-through display. The *optical see-through display* allows the user to see both the real world and the augmented content through the same display. *Video see-through display* uses the device's embedded cameras and blends the real-time video captured with virtual information and returning the result to the user via a screen. [30]

The user's position and orientation can tracked by sensors. Sensors can be device components such as gyroscope and accelerometers which track the device's angle and orientation. Moreover, another way to track the user's position is by vision based methods. Vision based methods use visual information retrieved from the display extract features and use them for tracking and the device's pose estimation.

Furthermore, there are two prevalent types of tracking, marker and marker-less tracking. Marker tracking uses images or 3D objects, in case of object tracking, which are previously known to the application before been tracked in the real world by the device. Marker-less tracking is when the target is previously unknown to the application and found during runtime [31]. The target is chosen by the user which can be either an image, a surface, an object or even an area in more advanced situations.

Finally, depending on the device there are different ways of interaction, an example of interaction in handheld mobile devices can be through the touch screen or interaction with finger tracking in which the user's finger is been tracked through the camera and uses gestures to interact with the virtual components.

To develop an augmented reality application there are numerous augmented reality SDKs with their own API and documentations.

An example is Vuforia SDK by Qualcomm [32]. Vuforia offers many different features such as tracking of 3D objects, images or markers, has an active community and many tutorials which facilitates the development process. In addition Vuforia runs in most handheld devices so there is no need for any special hardware.

Another augmented reality SDK is Metaio [33]. Metaio offers the same features as Vuforia and some unique features such as marker-less tracking, GPS for location based AR, SLAM, CAD tracking, which is object tracking using CAD data, and even face tracking.

Chapter 3

Pipeline

In this part we will discuss the development pipeline, i.e. the Offline section of the pipeline introduced in section 1.3., which we followed for the game implementation. We will discuss each step separately and explain the technologies used.

The pipeline consists of 3 steps:

1. Scanning the environment which will be done using a Kinect and the RTAB-Map Tool by Labbé et al [24]
2. Converting the point cloud form the previous step to a polygon mesh using the work of Lankveld et al [27].
3. Using the 3D model to design and develop the augmented reality game.

We will motivate and explain the design below.

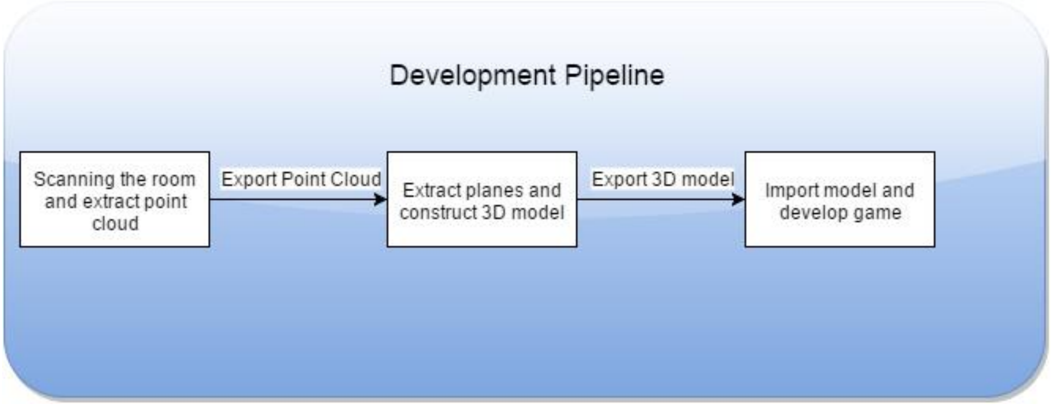

Development Pipeline
Scanning the room and extract point cloud
Export Point Cloud
Extract planes and construct 3D model
Export 3D model
Import model and develop game

Figure 11: The development pipeline.

We need to scan the room to get a 3D view of the room, the Kinect's depth camera is capable of capturing 3D data like the furniture's and room shape dimensions. The output of the scanning's 3D data are in a point cloud format, which cannot be used in our game. For that reason we convert the point cloud into a 3D model which can then be imported into our game. Finally, we develop the augmented reality game around the model which is the same as the room in which the game will be played in order to achieve a real world virtual world correspondence which is what creates the illusion of the augmented room.

3.1. Scanning

This is the first step of the pipeline which is responsible for scanning the room that will be used for the game. The output will be a point cloud representation of the room, which will be used on the next step of the pipeline. In this step we use the Real-Time Appearance-Based Mapping technique. We chose this work due to the fact that it is easily parameterized to the user's needs, it offers multiple sessions which means that the scanning can be continued later from where it stopped. This is useful because if the resulting scan has some missing information, like holes, instead of a new scan the user can just scan the missing area. Furthermore, the result of the scanning is in a point cloud format which is the required format for the next step of the pipeline.

3.1.1. Real-Time Appearance-Based Mapping

The Real-Time Appearance-Based Mapping (RTAB-Map) tool [24] is based on three papers: [20], [34], [35] by Labbé and Michaud. All three works focus on loop-closure detection during SLAM and the creation of a map from the visited locations. The first work presents a novel data management method in order to achieve long-term and large-scale SLAM. The second work improved upon the loop-closure hypothesis selection and the third work extends into multi-sessions which deals with the kidnapped robot problem and the initial state problem which is the problem where the robot is stopped, moved and restarted at a different position into a previously created map. This work allows multiple sessions of the same environment even if the new session starts in a previously unexplored area of the environment.

The main concept of this work is the memory management it uses which helps handling a large number of scans in real-time. The memory management system consists of four memory components inspired from works in psychology, Working Memory (WM), Long-Term Memory (LTM), Sensory Memory (SM), Short-Term Memory (STM), each responsible for its own task (see Figure 12).

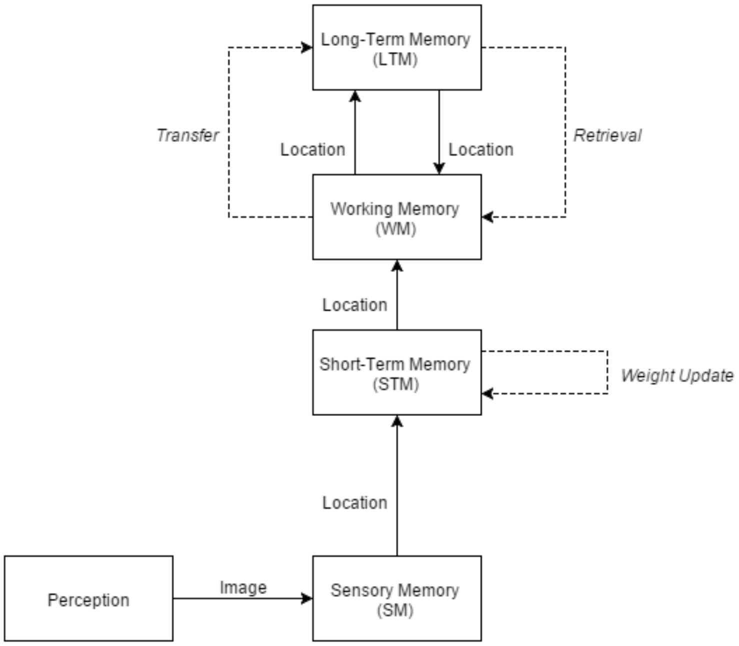


Figure 12: Memory management diagram. Image taken from [34].

Working memory (WM) keeps locations with higher probability for loop closure. Whenever a new location is scanned it is compared with the locations on WM for a loop closure. Each location is represented by an image signature represented by a bag-of-words, a time index (or age) and a weight, and locations are linked together in a graph by neighbor or loop closure links. When a loop closure is detected, neighbor locations can be retrieved from LTM and brought back into WM to be considered in future loop closure detections, this process is called *retrieval*. When the processing of the locations stored in WM gets too time-consuming for real-time, the locations with the lowest weight are transferred to the LTM, this process is called *transfer*. Sensor memory SM is responsible for evaluating the image signature, reducing data dimensionality through the extraction of image features for loop closure detection. Next it creates a new location through the image signature and sends it to STM. STM updates the aforementioned created locations through a process called *weight update*. If the weight update process considers that the new location is similar to the last one in STM, it merges them into the new one, and it increases the weight of the new location. In other words, the role of STM is to observe similarities through time between consecutive images for weight updates, while the role of the WM is to detect loop closures between locations in space. Furthermore, no locations in STM are considered in loop closure to avoid loop closures on locations that have just been visited. The size of STM needs to be under a threshold to keep the system working in real-time. If the size of STM reaches the threshold then the oldest location is moved into the WM.

3.1.2. The Algorithm

The main algorithm consists of 6 interconnected components each of them responsible for one segment of the algorithm. These are:

1. Location Creation
2. Weight Update
3. Bayesian Filter Update
4. Loop Closure Hypothesis Selection
5. Retrieval
6. Transfer

In *location creation*, a bag-of-words is created which contains every signature z_t of an image acquired at time t. This signature is represented by SURF features extracted from the retrieved image by the scanner. These features are clustered into similar words for the creation of the visual vocabulary i.e. the bag-of-words. The vocabulary is created in runtime from the retrieved images. Furthermore, along with the SURF data, a depth image is registered which contains the 3D position in space for each 2D point in the image. When a loop closure is detected, the rigid transformation between the matching images is computed by a RANSAC approach using the 3D visual word correspondences. The extracted SURF feature has a strength referred to as feature response which is used in order to select the most important features in the image. A threshold $T_{response}$ is used which rejects features with strength lower than the threshold. Furthermore, a second threshold $T_{maxFeatures}$ is used which represents the maximum number of features with the highest feature response that can be kept so to have nearly the same number of words across the images. If not enough high strength features are found then the image is not considered for any loop-closures. This happens with images with no discriminative features like white walls. To find matches between the good features and words already in the vocabulary, the SURF features are compared using the distance ratio between the nearest and the second-nearest neighbor. Two features are considered to be represented by the same word if the distance with the nearest neighbor is less than T_{NNDR} times the distance to the second-nearest neighbor, T_{NNDR} is a user defined threshold. In order to reduce comparison time and due to the fact that SURF features have high dimensionality, a randomized forest of four kd-trees is used. The kd-trees decrease the query time, thus makes the search for the two closest neighbors faster. The kd-trees are built from the SURF descriptors contained in the visual vocabulary. Hence, whenever a new image is retrieved, its SURF descriptors are extracted and searches to find its two closest neighbors in the kd-trees. If the T_{NNDR} criterion is not satisfied, a new word is created with the feature's descriptor and then added to the vocabulary and z_t. If it is satisfied then it merges with the closest neighbor and a new location $L_t$is then created with signature $z_t$and time index t; its weight initialized to 0 and a bidirectional link in the graph with L_{t-1}.

The *weight update* runs only in cases where we have a merge of similar locations. More particular, the weight of the acquired location L_t is updated by comparing the weights and similarity of the location with the most recent location L_c in STM. If the similarity is higher than a user defined threshold $T_{similarity}$ then L_c is merged into L_t and keep the older signature z_c since they are similar. To complete the merging process, the weight of L_t is increased by the weight of L_c plus one, the L_c is deleted from STM and its neighbor and loop closure links are redirected to L_t.

The discrete *Bayesian filter* keeps track of loop closure hypotheses and estimations. The probability estimation considers the states of all loop closure hypotheses at time t, the probability that the current L_t closes a loop with a past location L_i and the probability that L_t is a new location. For each new location the system processes and updates the Bayesian filter.

Next comes the *Loop closure hypothesis selection* in which a loop closure hypothesis is accepted if the probability of loop closure of a new location L_t with an old location L_i is lower than the loop closure threshold T_{loop}. Next, weight and neighboring links are adjusted where the weight of $L_t$is increased by the one of L_i, the weight of L_i is reset to 0, and a loop closure link is added between L_i and L_t. The loop closure link is used to get neighbors of the old location during Retrieval and to setup the transition model of the Bayes filter.

Retrieval takes place after a detected loop closure. Neighbors not in WM of the location with the highest loop closure hypothesis are transferred back from LTM to WM. When the neighbors are retrieved from LTM, the visual vocabulary is updated with the words associated with the corresponding retrieved signatures.

Transfer takes place when the processing time for a new image gets larger than a user defined threshold T_{time}. In *transfer*, the oldest and least likely locations to achieve loop closure, i.e. the locations with the lowest weight, are transferred from WM to LTM. The threshold T_{time}denotes the maximum processing time an image can have without needing to execute a *transfer*. If T_{time} is set to 0 or very big then the algorithm intrinsically uses an image rate corresponding to time, with 100% CPU usage. Depending on the case the user can set T_{time} accordingly. Higher T_{time} means that more locations (and implicitly more words) can remain in WM, and more loop closure hypotheses can be kept to better represent the overall environment. An overview of the algorithm can be seen in Figure 13.

```
Algorithm: RTAB-Map
$time \leftarrow TimeNow()$    ▻ $TimeNow()$ returns current time
$I_t \leftarrow$ acquired Image
$L_t \leftarrow LocationCreation(I_t)$ ▻ $LocationCreation()$ Creates location $L$ with image $I$
if $z_t$ (of $L_t$) is a bad signature (using $T_{bad}$) then
    Delete $L_t$
else
    Insert $L_t$ into STM, adding neighbor link with $L_{t-1}$
    Weight Update of $L_t$ in STM (using $T_{similarity}$)
    if STM's size reached its limit ($T_{STM}$) then
        Move oldest location of STM to WM
    end if
    $p(S_t|L^t) \leftarrow$Bayesian Filter Update in WM with $L_{t-1}$
    Loop Closure Hypothesis Selection ($S_t = i$)
    if $S_t = i$ is accepted (using $T_{loop}$) then
        Add loop closure link between $L_t$ and $L_i$
    endif
    Join $trash's$ thread     ▻ Thread started in $Transfer()$
    $Retrieval(L_i)$          ▻ LTM → WM
    $pTime \leftarrow TimeNow() - time$     ▻ Processing time
    if $pTime > T_{time}$ then
        $Transfer()$
    endif
endif
```

Figure 13: The RTAB-Map algorithm. Taken from [34].

3.1.3. Setting up RTAB-Map

The RTAB-Map tool is available online [24]. Along with RTAB-Map with used a Kinect v2 [7] sensor. The Kinect sensor is equipped with both an RGB camera with 1920 x 1080 resolution on 30 FPS and depth camera with 512 x 480 using Time of Flight technology.

The user defined parameters are the detection rate of the Kinect, the images buffer size, the maximum processing time T_{time}, the loop closure threshold T_{loop}, the similarity threshold $T_{similarity}$ and the size of STM; all of which have predefined default values.

The scanning process consists of moving the Kinect around the room and collecting images along with point cloud data. During scanning, when the current location does not have enough features to extract or if the movement of the Kinect is too fast, the system loses track of the position and pose of the Kinect and pauses. The user must then point the Kinect to the position of the previous frame which when detected triggers the reconstruction process to restart from there. As mentioned in *location creation* RTAB-Map faces difficulties extracting SURF features in simple plain colored environments. A solution for reducing the frequency of track losses, was putting posters on the walls to reduce the amount of plain colors on the environment. Furthermore, after the completion of the scanning it is possible to for further post-processing and the recalculation of neighbors and loop-closures, which improves the quality of the result.

The result of the scanning process can be exported as either a point cloud or a polygon mesh format. Even though the final goal is to have a polygon mesh format of the room, at the current stage, a point cloud format is preferable as it is easier to manipulate scanning process and there is no need of amount of detail the exported mesh has.

3.2. 3D Reconstruction

For the next step following the scanning, we used the work of van Lankveld et al [27] to reconstruct the output point cloud from the previous step into a polygon mesh which consists of a set of planes. The reasons for using the work of van Lankveld et al is that since we are making a platform puzzle game, a representation of the environment by planes is more suitable as platforms in 3D games are usually represented by planes. Even though the focus of this work is on reconstruction of urban scenes such as buildings, most furniture can be represented by a set of planes.

One of the contributions of this work is a one-parameter algorithm that can identify point sets on a plane for which a rectangle is a fitting boundary. This parameter is called δ which represents a geometric shape called δ-coverage region U_δ.

The δ-coverage region of a point set S in a plane is the union of disks in the plane with radius δ and center $c \in S$. Point set S is a random set of points that lie on a plane. A polygon P is δ-covered by point set S if and only if it is inside the δ-coverage region of S. The role of δ-coverage is to determine the correctness of the rectangular boundary of a plane. By correctness we mean the amount of empty space within the rectangle. The δ-coverage region doesn't give a rectangle but a region which the rectangle should be inside. To calculate the most fitting rectangle we need to calculate 4 types of shapes: the δ-coverage region, the convex hull CH, the α-shape and the trajectory region T.

A convex hull (see Figure 14) $CH(S)$ of a set S is the smallest convex set that contains a point set S [36]. If part of the convex hull CH is not in δ-coverage region U_δ, then no bounding rectangle exists that is δ-covered. Calculating the δ-coverage region and the convex hull is faster than calculating the bounding rectangle thus it saves time to check if CH is part of U_δ.

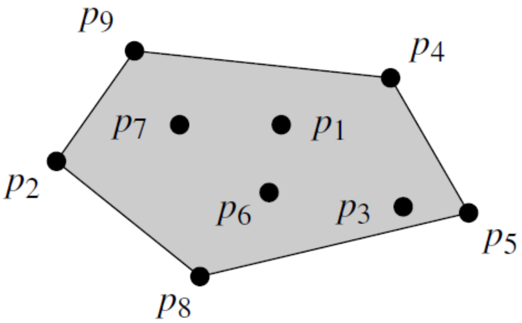


Figure 14: Convex hull of a set of points. The hull consists of the points $p9$, $p4$, $p5$, $p8$ and $p2$. *Image taken from [36]*

The α-shape (see Figure 15) of a point set is the collection of edges between two points that share the boundary of an empty disk with radius α [37].The role of the α-shape is to calculate the δ-coverage region faster as we can calculate the union of disks (see previous definition of δ-coverage region) of the points which belong to the boundary of the α-shape instead of the discs of all the points in the plane. More specifically we calculate the δ-coverage region of the points $p \in a$-shape where a-shape $\subseteq S$.

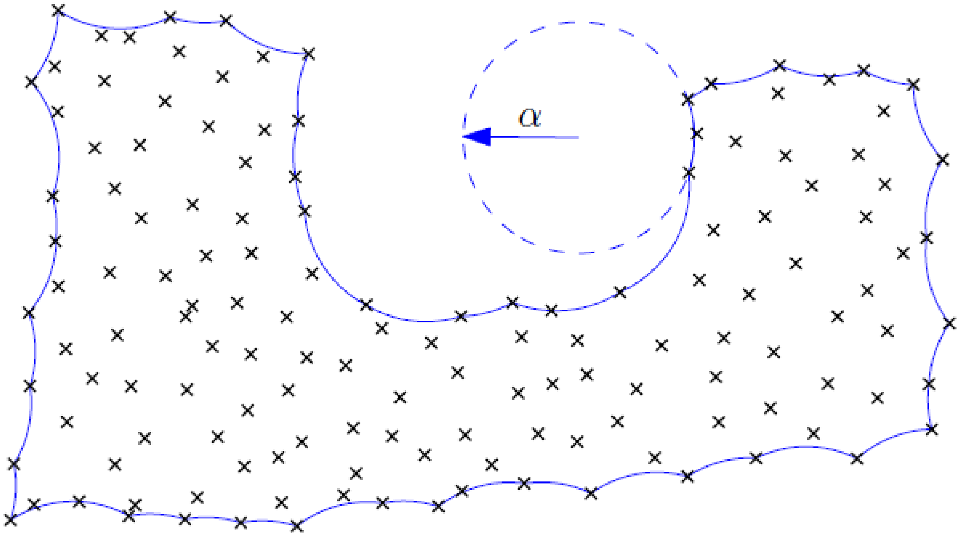


Figure 15: The α-shape. The points that belong to the α-shape are the ones on the boundary and have maximum distance with each other α. The smaller the α the more points will belong on the boundary. Image taken from [37].

The trajectory region T (see Figure 16) is the union of minimal bounding rectangles over all rotation angles. During rotation all corners of the minimal bounding rectangle remain on the boundary of this region.

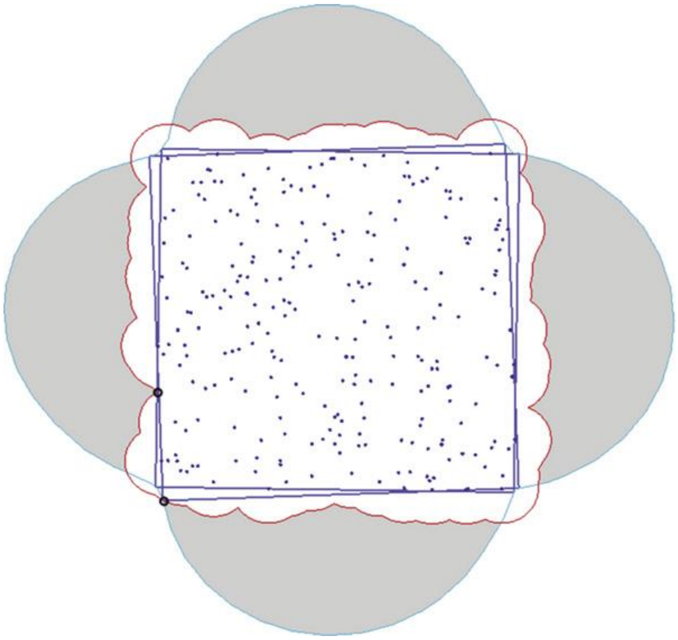


Figure 16: The different structures used and the extrema of the range of allowed rectangles. The blue points were sampled uniformly in a unit square. The δ–coverage region is bounded by the red arcs, the trajectory region is bounded by the blue arcs. The grey regions show where the rectangle is outside the δ-coverage region. Image taken from [27].

These geometric shapes are 2-dimensional and calculated from the points which are on a plane. The points given as an input in the algorithm are in a 3-dimensional point cloud as they are scanned with a depth scanner. Thus, the points of the point cloud must first be clustered into 2-dimensional planes before the rectangles can be calculated. The clustering was done using RANdom SAmple Consensus (RANSAC) algorithm. RANSAC is an iterative algorithm formulated by Fischler and Bolles [38] which takes as input a set points and outputs all of which that fit into a model, in our case the model is a plane. The algorithm follows these 3 steps:

1. Select a random sample of minimum required size to fit model

2. Compute the desired model from sample set

3. Compute the set of inliers to this model from whole data set.

Repeat 1-3 until model with the most inliers over all samples is found (See Figure 17).

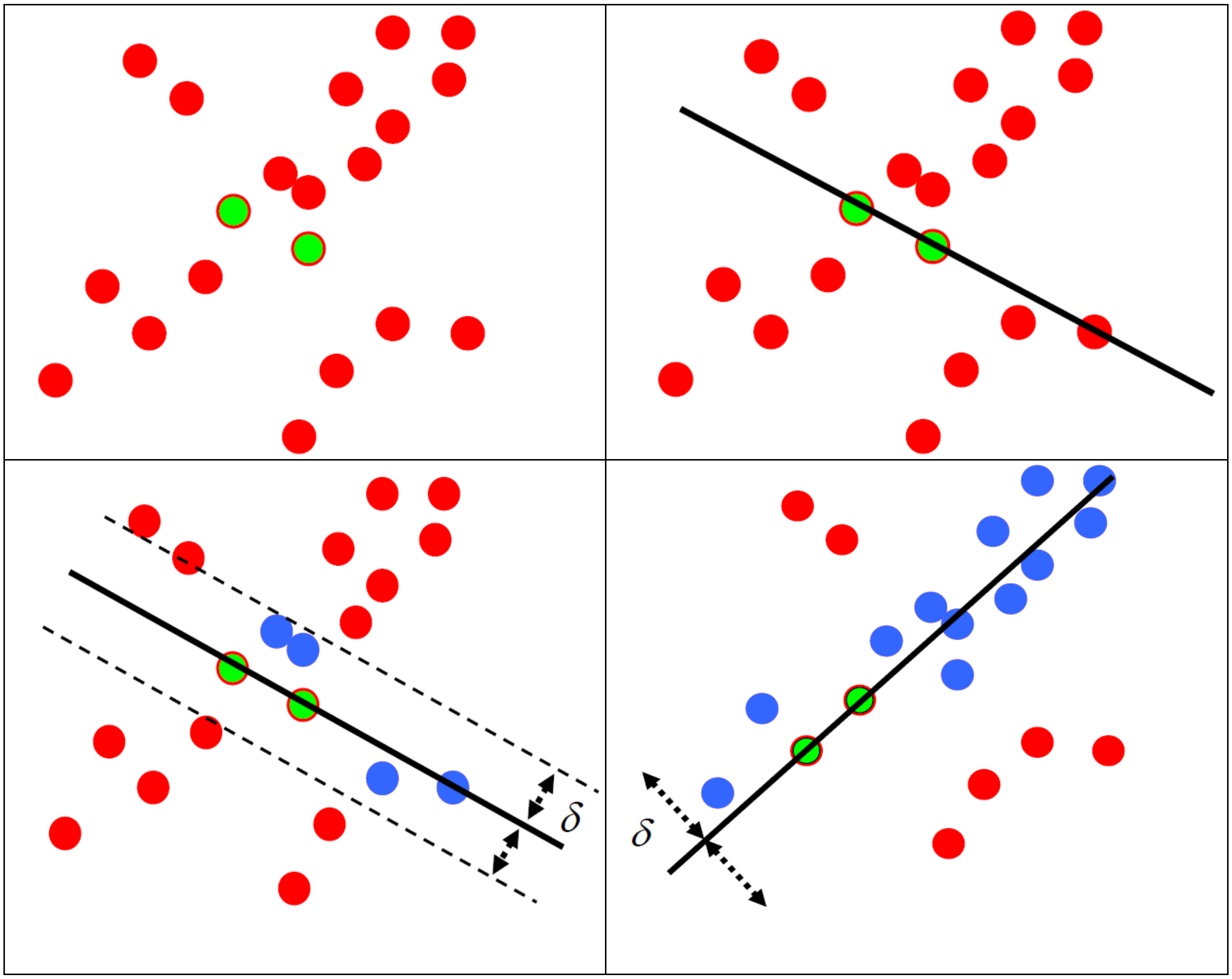


Figure 17: The steps of RANSAC algorithm for a line model. On the top left the minimum points needed are picked. On the top right the model is been computed. On the bottom left the set of inliers from the whole data set is estimated. On the bottom right the output of the algorithm after all iterations are done. The letter δ denotes the tolerance ϵ. Images taken from Peter Meer's lecture from the course Robust Computer Vision at Rutgers University [39].

The iteration is repeated until a model with a satisfying amount of inliers is found. By satisfying amount we mean above a percentage of the total number points. RANSAC also uses a user defined constant number ϵ (see Figure 17) which denotes the tolerance i.e. the maximum distance a point can have from the model and still be considered an inlier. Extracting only the points which fit a plane model also works as a noise filter since any points that do not fit to the model will not be used when the rectangles are been identified. To extract all planes in a point cloud we applied RANSAC iteratively on the cloud. After every iteration RANSAC is executed on the remaining points where there has been no plane detection. The iteration continues until the number of remaining points is less than a predefined threshold, which in our case was the

30 percent of the total points. After plane extraction the Identifying rectangles algorithm uses the δ parameter to extract the best fitting rectangle for every plane.

3.2.1. The Algorithm

The algorithm uses a rotating rectangle R which is a rectangle which bounds the plane points and rotates around them. The rectangle R is rotated around the points as tightly as possible, by tightly meaning the smallest possible oriented rectangle that bounds the points of the plane within, and handles events. Events are the important occurrences that happen during the algorithm's execution. Their number is previously unknown to the algorithm and the only thing known is their nature so that they can be identified when they happen. To identify these events the δ-coverage region and trajectory region T are used. As mentioned, the best fitting rectangle should always be inside the δ-coverage region U_δ thus during the rotation of the rectangle R only the orientations in which R is inside U_δ are the ones that should be further examined. It is proven that there are two ways for R to enter or exit the δ-coverage region during rotation, i.e. be contained wholly or partially by the region. Either during rotation, an edge of R passes over a vertex of U_δ or a corner of R crosses the boundary of U_δ. Each vertex of U_δ will be inside the rectangle at some rotation angle, if and only if it is inside T. Each vertex produces at most two events: when it enters and when it leaves R. Because the corners of the rectangle follow the boundary of T, a corner enters or leaves U_δ at an intersection of the boundaries of U_δ and T. Most of these intersections produce one event: the corner either enters or leaves U_δ, but some intersections produce two events if the boundaries of U_δ and T touch, but do not intersect. All events are then collected sorted by the angle of R when they occurred. After determining the situation before rotation, the events are handled sequentially. The algorithm keeps track of the number of vertices of U_δ inside R and the number of corners of R outside U_δ. The angles at which R becomes or stops being δ-covered are collected. Finally, when all events have been handled, all rotation angles for which R is δ-covered are known. The summary of the algorithm can be seen in Figure 18.

```
Algorithm: Identifying $\delta$-covered rectangles
Identify the range of angles of rotation for which the minimal bounding rectangle is $\delta$-covered.
  in $P$: The set of points in the plane that should be bounded.
  out $A$: The set of angles for which the minimal bounding rectangle is $\delta$-covered, or $\emptyset$ if there is no such rectangle.
Construct $CH$
Construct $U_\delta$
Construct $CH \setminus U_\delta$
if $CH \setminus U_\delta \neq \emptyset$ then
  return $\emptyset$
end if
Construct $T$
Construct $T \setminus U_\delta$
Identify events $E$ from $T \setminus U_\delta$
Sort events $E$
Handle events $E$ in order to fill allowed angles $A$
return $A$
```

Figure 18: The identifying rectangles algorithm. Taken from [40].

As input of the algorithm we used the point cloud extracted from RTAB-Map and two constants both which influence the output of the algorithm. The first constant is the RANSAC tolerance constant ϵ and the second is δ which represents the radius of the discs of the δ-coverage region. Small ϵ on RANSAC lowers the tolerance leading to more plane detections but smaller planes. Smaller δ will lead to fewer or no angles in which the rectangle is δ-covered resulting to less identified rectangles. We ran the whole process, the plane detection using RANSAC and the identifying δ-covered rectangles through a number of iterations with different numbers and compared the results with each other to find the constants which gives the best result.

3.2.2. Point cloud library

The Identifying Rectangles algorithm was implemented using the Point Cloud Library (PCL) [41]. Point Cloud Library is an open source library for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. It is widely used by researchers and is free for commercial and research purposes.

3.3. Evaluation

After running the algorithm with various ϵ and δ we see that the results are not satiable for any ϵ or δ, an example of which can be seen in Figure 19. From the result we see that only the walls are reconstructed with some major inaccuracies, like multiple walls, and also that no furniture is reconstructed. More reconstructed results can be seen in Appendix A.

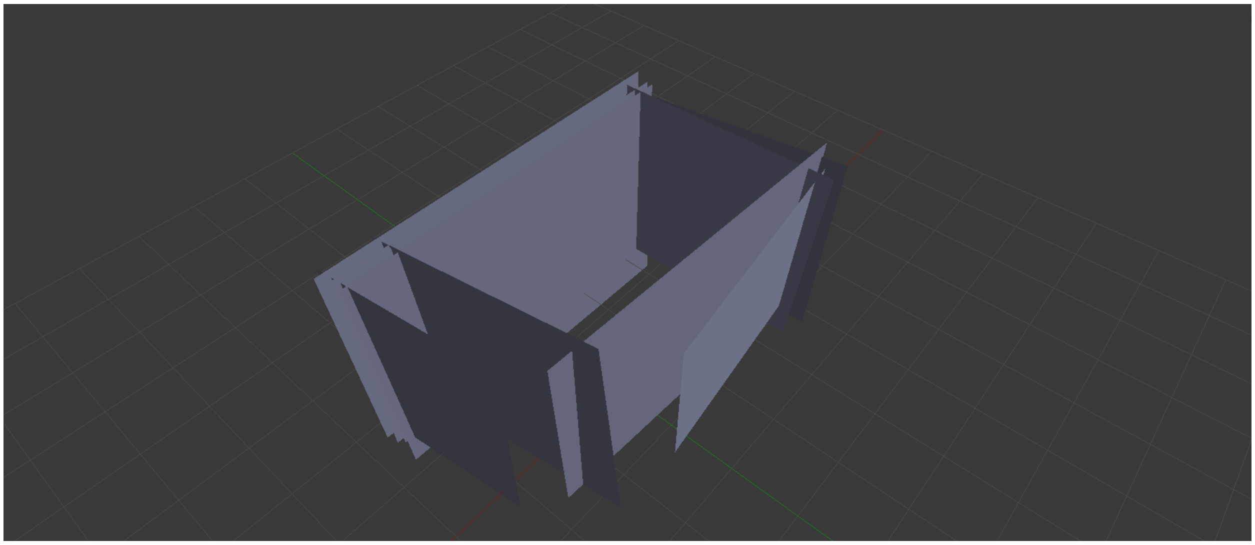


Figure 19: Reconstructed room.

Next, we find and explain the reasons we have this type of results.

3.3.1. Noise in scanning

As expected, some noise from the scanning is unavoidable, we will see the causes of the noise and how it affects the reconstruction.

Noise during the scanning phase was produced due to the sensor's (i.e. the Kinect) and RTAB-Map's inaccuracies along with environmental limitations which means that the algorithm does not have the same performance in all type of environments. More specifically, Kinect offers high accuracy static scans but it is not very efficient when combined with movement for example scanning the same area from a different distance in order to fill gaps from the previous scan of the area. This results in a scan with multiple layers of the same object which is caused by to inconsistent estimations of the distance between the sensor and the scanned object hence adding a new layer for every new estimation. In other words every time the object is scanned, instead of supplementing the current cloud with the newly scanned information it adds a new layer at a different position due to miscalculations. An example of the multiple layer effect can be seen in Figure 19, where we see a corner wall of the room represented by multiple layers instead of one. Multiple layers result in multiple plane detections during RANSAC which will result to having multiple walls in our reconstructed model see Figure 19.

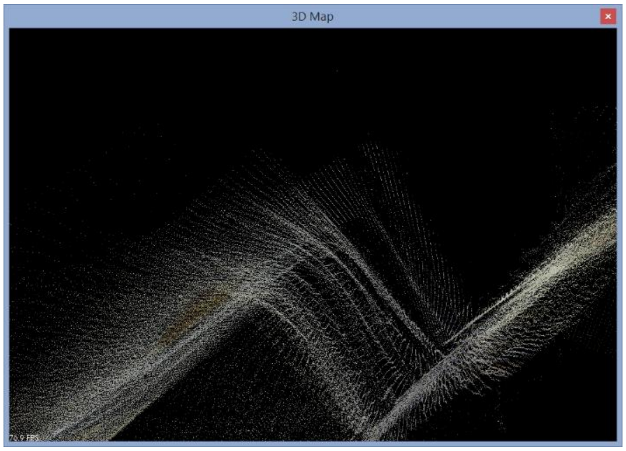


Figure 19: Multiple layer effect seen on the scan of a room corner. The view is from the top of the wall which can be seen to have many layers.

Furthermore, scanning of cloth objects such as curtains or bed sheets produces incomplete scans as the cloth wrinkles behave like "waves" i.e. they hide parts of the cloth behind them making it complicated to make proper scans of the soft objects (See Figure 20). Also such complicated objects cannot be fit into a plane model during RANSAC. As a result they will be filtered out.

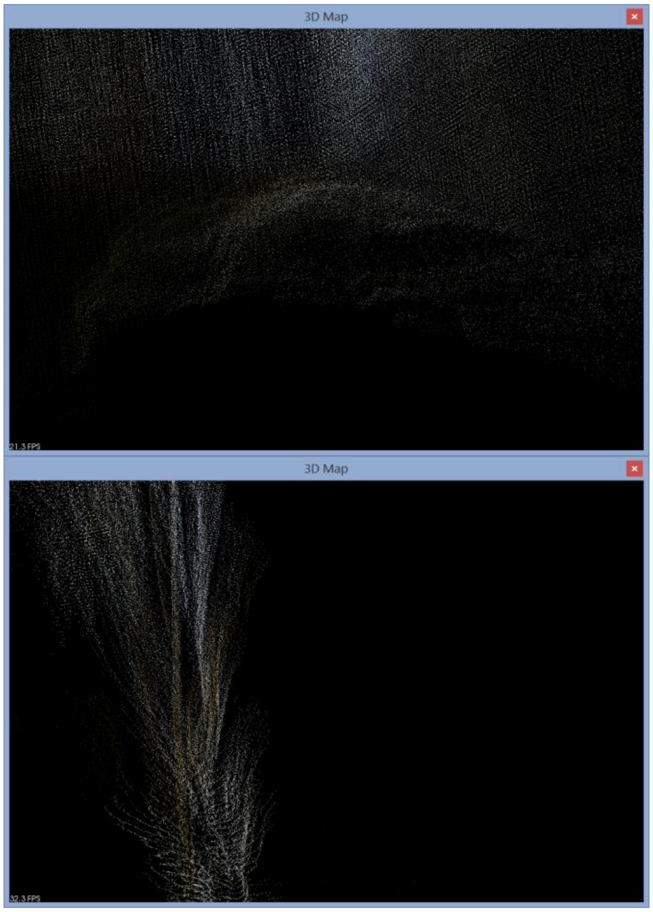


Figure 20: Results of scanning cloth material objects. On the top we see part the scan of a bed in which the wrinkles are hiding parts of the view. On the bottom we see the scan of the curtains on top view and the "waves" of the points, making it impossible to recreate with planes.

Another RTAB-Map limitation appears when trying to scan windows or other transparent objects. As mentioned in section 3.1, RTAB-Map mapping uses SURF descriptors extracted from RGB images from the RGB camera and depth information obtained by the depth camera. As a result, when the sensor points at a transparent object, the SURF features are extracted by the visual information obtained by objects behind the transparent object while the depth information is obtained by the objects itself which leads to inconsistencies between RGB and depth information. This also occurs when connecting images of locations near the transparent object with the object. Moreover, SURF descriptors need visually complicated environments to and extract features with high response (see Section 3.1.2) that will not be rejected by the algorithm. This makes scanning parts of the room such as the floor, walls or even plain colored closets difficult.

To conclude, this leads to a certain type of environments that RTAB-Map can work since rooms with windows, cloth objects and plain colored objects offer too many obstacles for the algorithm to produce the desirable results.

3.3.2. 3D Reconstruction evaluation

Next we evaluate the Identifying rectangles algorithm. For a complete evaluation we needed a way to measure quantitatively the quality of the reconstructed room. Hence, to measure the quality we did two things, first we created a ground truth and second we introduced a similarity measure. As ground truth we used a 3D model of the room. To create the room we measured the dimensions of the furniture, the distance between them and the room's dimensions and then we used a 3D modeling software to build the 3D model (see Figure 21). We wanted the 3D model to represent the perfect result that could be outputted by the Identifying rectangles algorithm, hence it was made using only planes and cube shapes, which can be represented by 6 planes. The similarity measure is an algorithm which measures the similarity between the ground truth and the output of the Identifying Rectangles algorithm and returns a similarity error. The lower the similarity error the best the output from the algorithm. The similarity measure algorithm uses a point to plane distance as a dissimilarity measure. The reason for using point to plane distance is that both the ground truth and the output of the Identifying Rectangles algorithm are represented by planes. There are three steps of the algorithm. First, sample each detected rectangle with a number of random points, second, project these points on every plane of the ground truth and third, find the ground truth plane with the smallest distance to each projected point from the second step. The goal of the point to plane projection is to find to which real world object does each identified rectangle belongs to by projecting each sampled point to all the ground truth planes. By summing all the point to plane distances of every point we get the dissimilarity error. Furthermore, apart from the error distance we keep the number of identified rectangles that do not belong to any real world object, in other words of identified rectangles that have no correspondence with any of the ground truth planes. These are called false detections. False detections do not influence the dissimilarity measure but are instead used as an indication of the result's quality. The steps of the algorithm can be seen in Figure 22.

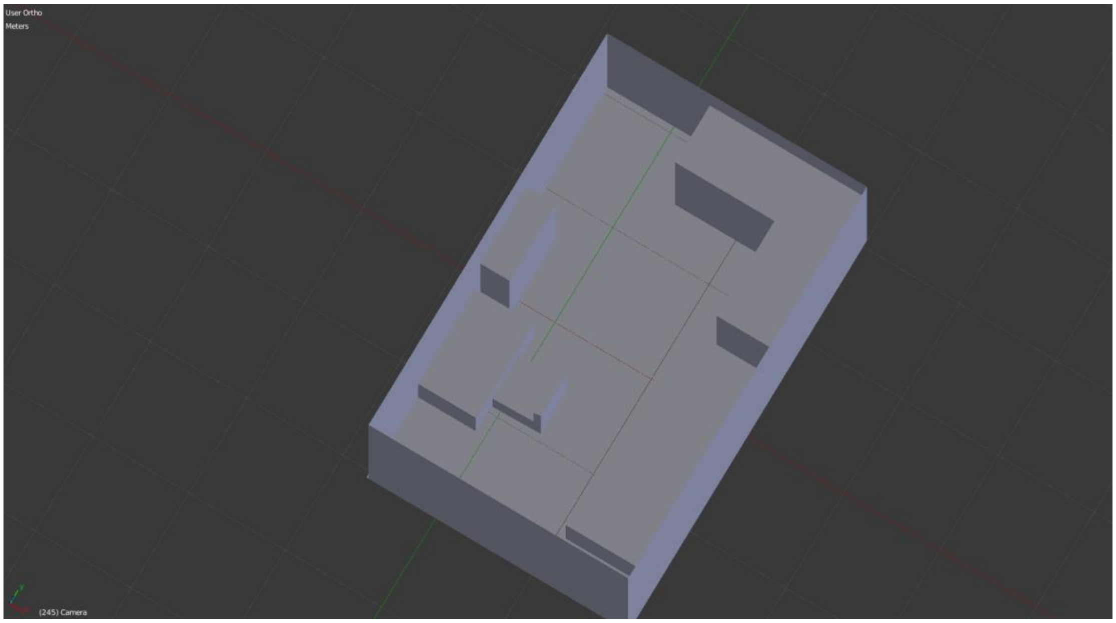


Figure 21: The ground truth of the room, created in a 3D modelling software.

```
Algorithm: Error Measurement of the identified rectangles
Measure the error of the identified rectangles by comparing them to the ground truth.
   $in\ R$: The identified rectangles.
   $out\ E$: The error of the identified rectangles and the false positives.
Randomly sample points $P$ for every identified rectangle $R$
Set $error = 0$
Set $falsePositives = 0$
foreach rectangle $r$ in rectangles $R$
Set $prevError = error$
   foreach point $p$ in sampled points $P$ of $r$
      Set $min = +\infty$
      foreach plane $pl$ in ground truth
         project $p$ in $pl$ to get $proj$
         if projection $proj$ is inside $pl$ then
            Calculate point to plane distance $p2pl$ between $p$ and $pl$
            if $p2pl < min$ then
               Set $min = p2pl$
            endif
         endif
      endforeach
      if $min < +\infty$
         Set $error = error + min$
      endif
   endforeach
   if $prevError = error$
      Set $falsePositives = falsePositives + 1$
   endif
endforeach
return $error, falsePositives$
```

Figure 22: The algorithm used to measure the similarity of the identified rectangles and the ground truth.

Now that we have an evaluation algorithm and a ground truth, we need an input to test the algorithm's efficiency. As input we gave the ground truth room and measured the similarity error of the output. We reiterated the process adding more noise to the ground truth to find how the error changes with the added noise and how it affects the quality of the visual result. As the algorithm's numerical input we used 0.9 for δ, 0.01(1 cm) for the ϵ and during the RANSAC phase in the case with zero noise we reiterated until the remaining points were less than the 0.1% of the initial number of points since no points are considered as noise. In the cases with added noise the percentage was 5% of the total points. We chose the δ and ϵ values that offered the best visual results with the scanned input. The resulting error in the case with no noise was 1458.56, which is smaller than the rest of the cases as shown in table 1. Along with the lowest error it offered the best visual result we could achieve for this setup (See Figure 23). As we see from Figure 23 the errors are mostly missing planes which were either not detected or wrongly detected and rejected by the Identifying rectangles algorithm as it could not find any angles in which the minimum bounding rectangle was δ-covered. Furthermore, we see the problem on reconstructing the chair due to its smaller size and the amount of detail it has, as it contains more planes than the rest of the furniture.

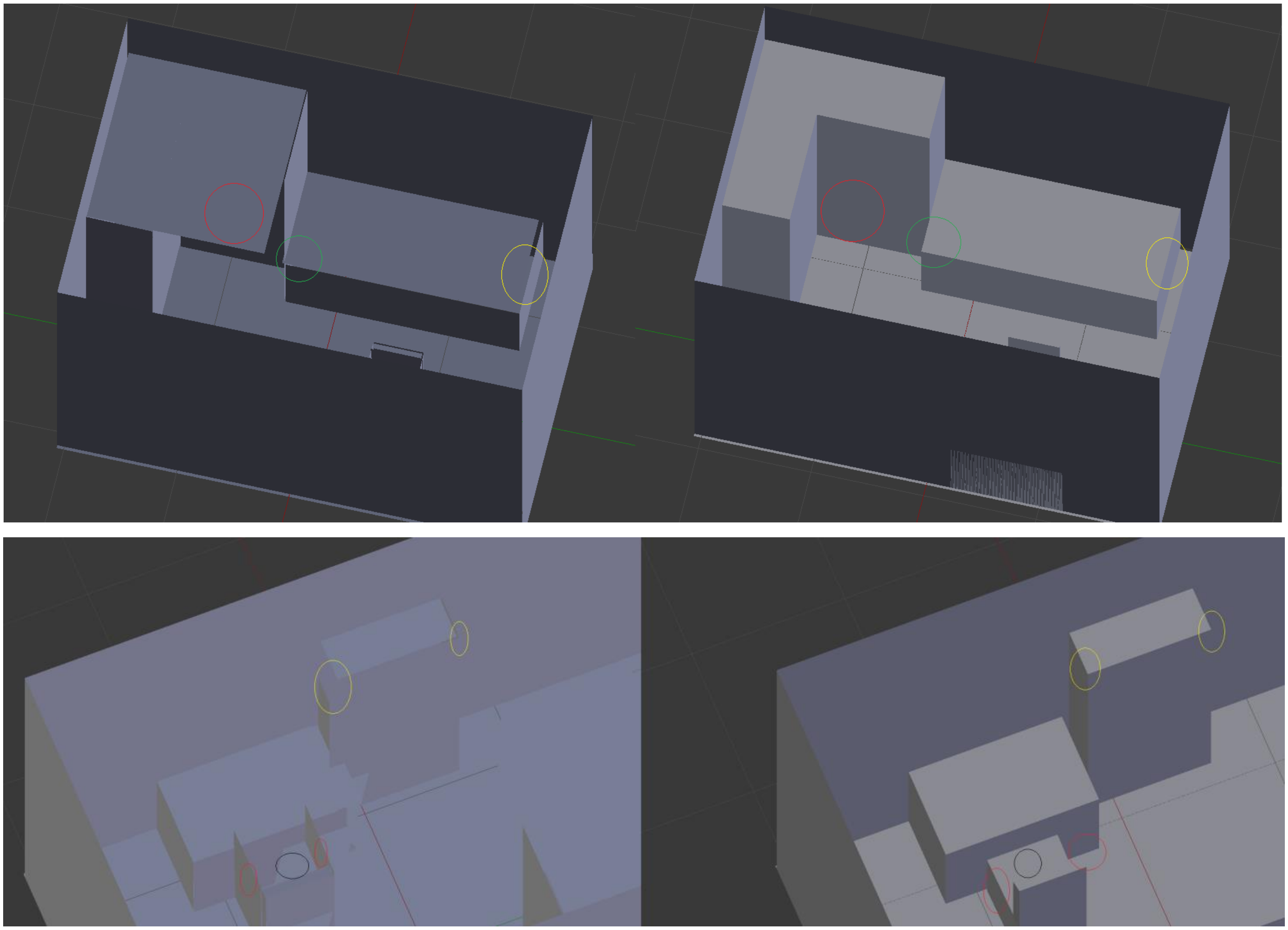


Figure 23: Here we see the visual difference between the result of the algorithm giving as input the ground truth and the ground truth itself. The ground truth is the right room and the reconstructed room is the left darker room. The inconsistencies between the ground truth and the resulting model are been highlighted by same colored circles on the two rooms.

More particularly on the Top image we see the limitation of the Identifying rectangles on detecting L-shapes which is highlighted by the red circle. There we see that the L-shape is detected as a rectangle. The green circle shows the difference on the sizes of the detected rectangle with the ground truth one, the detected has the same size as the bed while the ground truth is thinner. Finally we see on the yellow circles a small gap on the edge of the bed.

On the bottom two images we have the reconstructed room on the left and the ground truth on the right. There we see a hole in the furniture on the top left yellow circle and a gap on the right one. Furthermore, there are many problems with the detection of the chair as it has a hole in the seat with a wrongly detected plane instead on the dark blue circles. Finally, at the red circles we see that the detected rectangles at the two sides of the chair are bigger than in the ground truth ones.

Furthermore, the results of the rest of the reconstructions can be seen in the Table 1. We added three types of noise, the first type was adding noise only on different axes, either only on the Z axis or in all three axes. This type of noise will show the difference between non-uniformly and uniformly added noise. The second type of noise determined the amount of noise we would add, the amount varied from 5% of the points to be noise up to 20%. The third type determined the offset of the noise i.e. how much the points are drifting from the plane model, the offset was measured in centimeters which varied from 1cm to 10cm.

Axis	Percentage	Offset	Error
Z	5%	1cm	1555.76
XYZ			1629.72
Z	5%	5cm	2833.86
XYZ			4887.47
Z	5%	10cm	2191.99
XYZ			7896.26
Z	10%	1cm	1625.49
XYZ			1504.63
Z	10%	5cm	20744.9
XYZ			5110.2
Z	10%	10cm	212871
XYZ			8985.41
Z	20%	1cm	1573.85
XYZ			1502.82
Z	20%	5cm	100872
XYZ			20629.6
Z	20%	10cm	473042
XYZ			37208.9

Table 1: The error of the reconstructed room using the ground truth as input and adding some artificial noise. The axis denotes the axes in which the noise is applied to, the percentage is the amount of noise to the number of total points and offset how far away from the rest of the points will the noise points be moved.

From the results we see that the smallest error is found with offset 1cm and noise percentage 20% on all axes and the worst result with offset 10 cm on just the Z axis with 20% of the total points as the error (See Figure 24). We can see from the results that offset influences the result much more than the percentage of error and also that adding error non-uniformly, i.e. only on the Z axis, creates more error as the number of error points increases along with the offset. The low error scores with high noise percentages although it seems counter-intuitive has a reasonable explanation. The high error percentage along with the small offset can eventually help RANSAC during plane detection as it increases the amount of inliers since the noise points between 0 and 1cm could also fit the plane model due to the tolerance threshold ϵ which was set to 1cm. Finally, we see that as the percentage of noise and offset increase, which can be considered as simulating inaccuracy in scanning, the amount of error gets much bigger. In real scanning situations where the amount of noise is bigger and non-uniform, the error would be too big to have a satiable 3D model to use for our game.

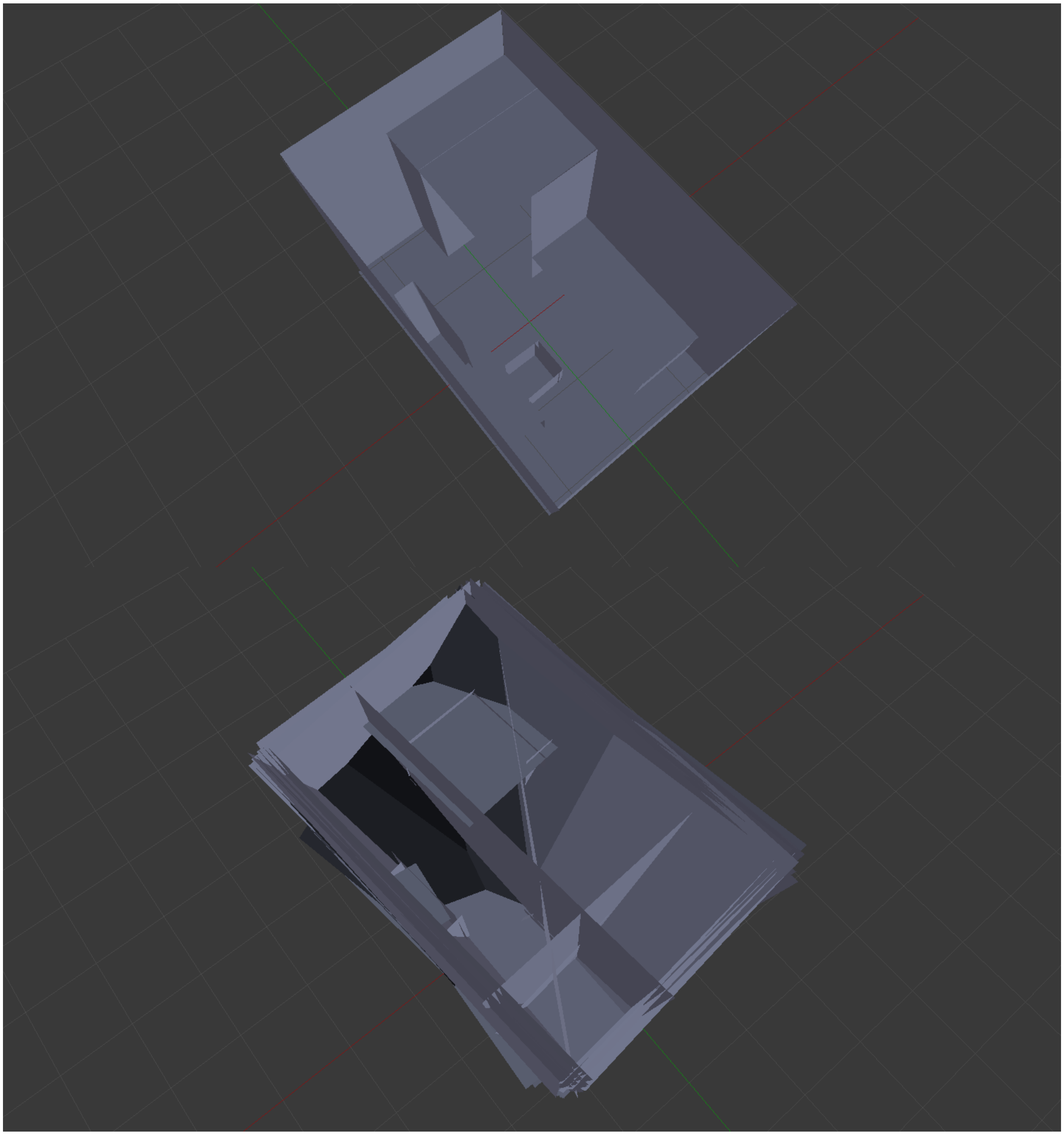


Figure 24: The reconstructed results. On the top we see the result with the smallest least error. On the bottom we see the result with the biggest error.

Also, we see that apart from the scanning noise there are some other elements that showed us that ‘Identifying δ-covered rectangles’ algorithm was eventually not the best choice for our case.

Firstly, trying to represent everything as planes is not a realistic goal for every type of room, as there are furniture of different shapes and rooms filled with small items e.g. a desk with a small lamp and computer on it or a bookcase with a picture frame on the shelf. As a result these objects will be considered as noise and be rejected along with a parts of the furniture under the object, therefore limiting the output model’s accuracy and similarity with the real world one. This is an important limitation as our goal is to be able to play the game independently of the room in which the player is as long as you have the 3D model and the levels designed for the room.

Secondly, even if the scanning was perfect and every object within the scan can be represented by a plane, RANSAC's randomness and design does not guarantee that all planes will be detected, especially for smaller objects, such as a chair's seat, as we can see in Figure 23. As described in section 3.2, at the end of every iteration we have a set of inliers that have fitted into the model, for that set to be considered valid their number must be over a percentage of the total amount of points in the set. This percentage can be set beforehand and through the reiterations the total number of points will get smaller, however some points will still be left out. In our case the largest amount of points is gathered on the walls which means that smaller furniture such as chairs will not be detected by the RANSAC due to the smaller number of points they have. Finally, we wanted this technique to work in even more complicated rooms with more furniture, but we saw that even with the simple room we gave as input there were problems.

3.3.3. Pipeline alteration

The result of the evaluation and the low quality of the scan had showed that no matter how many times we reiterate the pipeline by trying to make a better scan and trying to find even more suitable δ and ϵ we cannot have satiable results from the 3D reconstruction. As a result we decided to use the ground truth for our game. The development pipeline eventually has changed into the one in Figure 25.

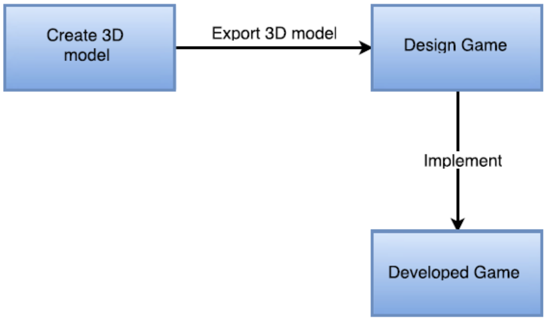


Figure 25: The new pipeline

3.4. Vuforia SDK

For the augmented reality game we decided to use Vuforia SDK. Ideally, we would like to use object tracking instead of image tracking since we have a 3D model of the room. Both Metaio [33] and Vuforia [32] offer object detection. Metaio offers CAD detection where the user can give as input the 3D model and the Metaio can track it in the real world by using edge-based tracking. This approach has its limitations as the object should be entirely in the field of view of the camera which is impossible since the user is surrounded by the room. Vuforia has the same limitation as well but also does not work with given 3D models, but they need to be scanned by its own object scanner application [42]. This application uses markers to estimate depth, hence making it impossible to scan a room. As a result we have chosen to use image tracking instead of the object tracking approach. Both offer image tracking but Vuforia was chosen due to the fact the fact it was more user friendly.

3.4.1. The architecture

The architecture of Vuforia consists of 5 main components: camera, image converter, tracker, video background renderer and application (see Figure 26).

The *camera* component ensures that every preview frame is captured and passed efficiently to the tracker. The developer only has to initialize the camera to start and stop capturing. The camera frame is automatically delivered in a device-dependent image format and size.

The *converter* converts from the camera format (e.g., YUV12) to a format suitable for OpenGL ES rendering (e.g., RGB565) to a format suitable for rendering and for tracking internally. This conversion also includes down-sampling to have the camera image in different resolutions available in the converted frame stack.

The *tracker* component contains the tracking algorithms that detect and track real-world objects in camera video frames. Based on the camera image, different algorithms take care of detecting new targets or markers. The results are stored in a state object that is used by the *video background* renderer and can be accessed from application code. The tracker can load multiple datasets, i.e. collection of targets, at the same time and activate them.

The *application* component consists of the application's logic through the developer's code. For each processed frame, the state object is updated and the application's render method is called. So in each frame there are three steps executed:

1. Query the state object for newly detected targets, markers or updated states of these elements
2. Update the application logic with the new input data
3. Render the augmented graphics overlay

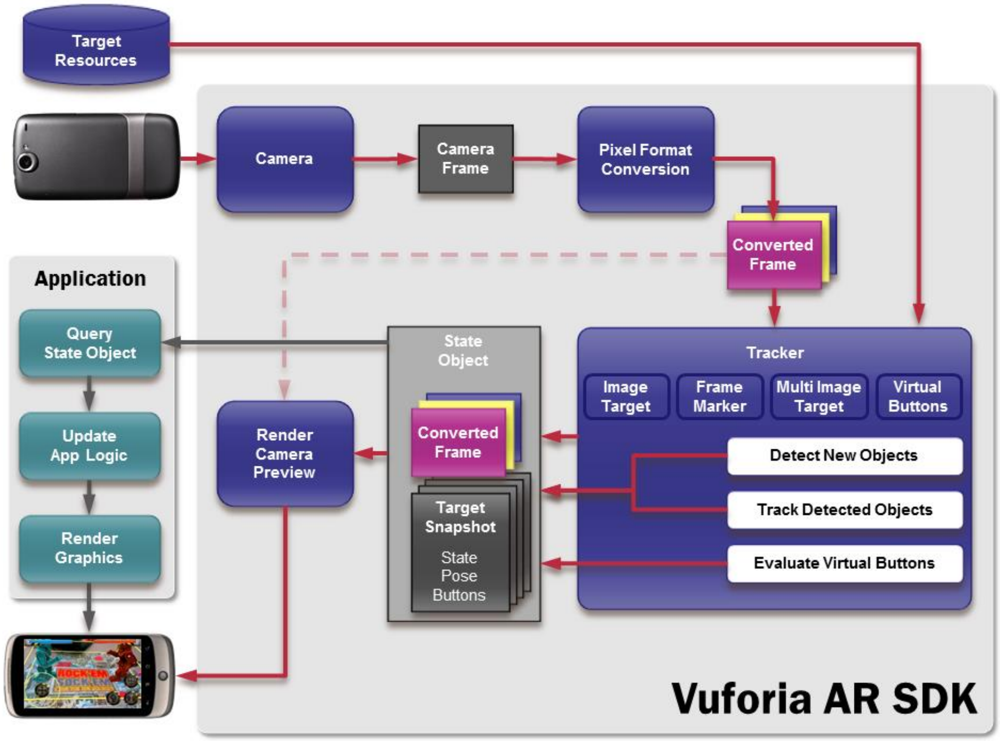


Figure 26: Vuforia SDK architecture. Image taken by Vuforia developer portal [43].

Chapter 4

The Game

The game RGB Slemmings is a 3D augmented reality platform puzzle game inspired by Lemmings and Star Wars: Pit Droids [44]. The goal of this game is to guide the Slemmings, which are slime-like creatures, from their initial spawned position to their goal. The Slemmings (named after the word slime and Lemming) have different colors, some of them have special abilities and each of them must go to their respective goal of the same color, which is a circular disc, in order to complete the level. The game consists of four small levels within the room that will be augmented. Each level is a small puzzle that needs to be solved by the player in order to complete it. The room is a 3D model of the room created on a 3D modeling software by using actual measurements of the room.

RGB Slemmings was developed using the Unity game engine [45]. It was prototyped as a computer game before it was converted to an augmented reality game. In that way we were able to test the game mechanics and design the levels without having to face any challenges, such as marker detection, from the augmented reality part yet. As soon as the game was finalized the game was converted into an augmented reality game, where the 3D model of the room was replaced with the actual room.

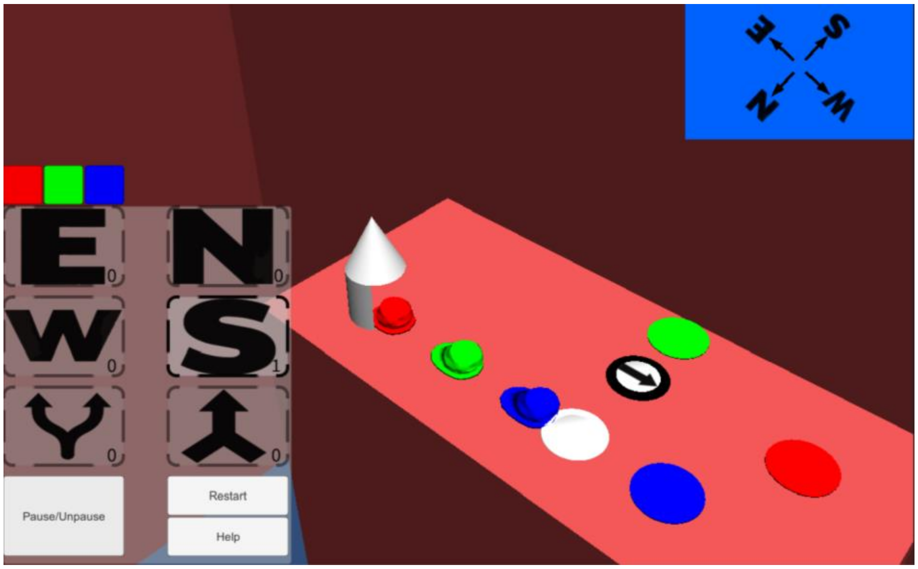


Figure 27: Screenshot of the computer version of the game

4.1. Game Components

There are four game components. The Slemmings, the goals, the starting position and the action bases. The game objective is to guide the Slemmings to their respective goals. The goals are the positions where the Slemmings must go. The starting position is the spawning position of the Slemmings i.e. where the Slemmings start at each level. The action bases are where the user must assign the actions in order to guide the Slemmings.

4.1.1. The Slemmings

The Slemmings are the protagonists of the game. They are mindless slime-like creatures that only move forward unless the player changes their direction in order to guide them to their goal. The goal is represented by a colored disc (see Figure 28).The starting position of the Slemmings is represented by a house (see Figure 29). The Slemmings have different colors such as the primary colors of the RGB color model (red, blue, green) and their combinations. This means that different colored Slemmings should be guided to different goals which is the main challenge of the game and also its puzzle element. When all Slemmings have reached their goal then the level has been complete. Depending on the level the Slemmings are spawning in different order, colors and numbers. The Slemmings are not spawned simultaneously but with a small delay one after another.

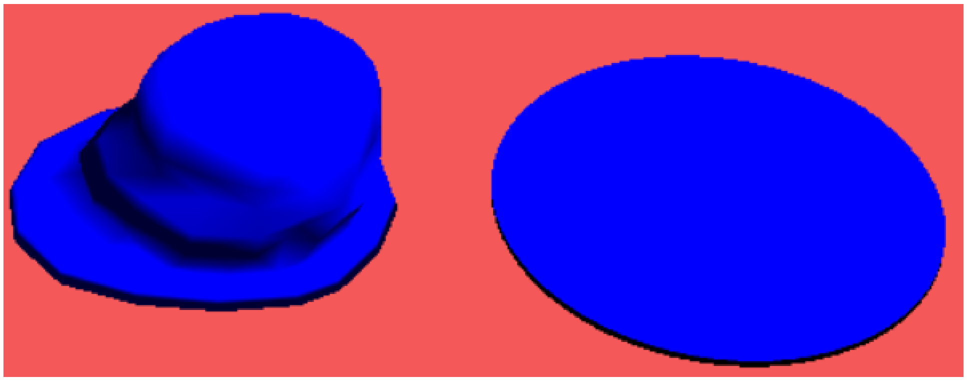


Figure 28: A blue Slemming moving towards a blue goal.

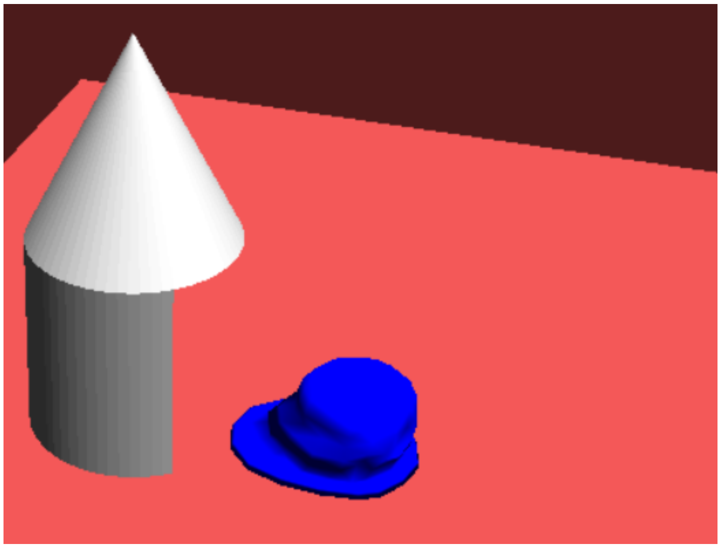


Figure 29: The spawning point.

4.1.2. The actions

To player must use some actions to guide the Slemmings around the levels. These actions however, are not applied directly on the Slemmings but on circular bases called *action bases* (see Figure 30). The action bases are part of the level and are positioned accordingly to help the player complete the level. When a

Slemming moves over an action base with an action on it then it follows that action indicated on it, if there is no action on the action base the Slemming continues its movement and passes over the base.

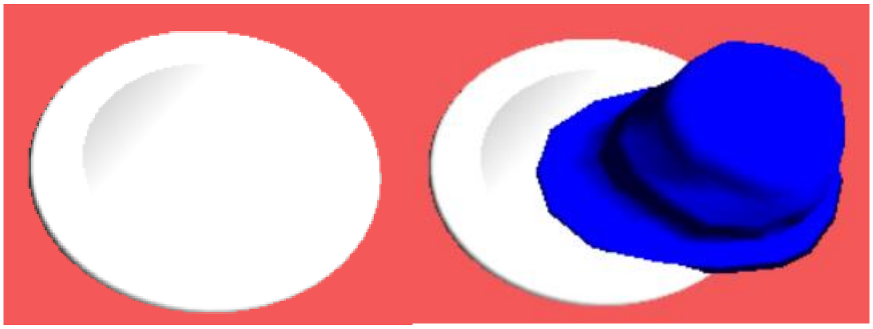


Figure 30: The action base on the left. On the right a blue Slemming passes over the base since no action is assigned on it.

The available actions are shown in the action bar (see Figure 31). The action bar is a UI component from which the player can select the action he wants to apply on the action base. To apply the action, the player first selects the action base and then selects the action from the action bar. The actions are represented by the buttons on the action bar, and the numbers next to the actions represent how many times the action can be used on the current level. Each level has just the exact amount of actions needed to be completed, this also works as a hint for the player as to which actions he should use to complete the level. There are six actions on the game. Four of these actions are the redirection arrows which change the direction of each Slemming that steps on the action base with the redirection arrow action. The redirection arrows represent the cardinal directions i.e. North, East, South and West and are symbolized by their respective symbol on the action bar N, E, S and W. To help the player understand where the arrows point we have a compass tool on the top right of the screen (see Figure 27 and 31). The compass rotates accordingly to the player's view so that the cardinal points are always on the same direction, independent of his view.

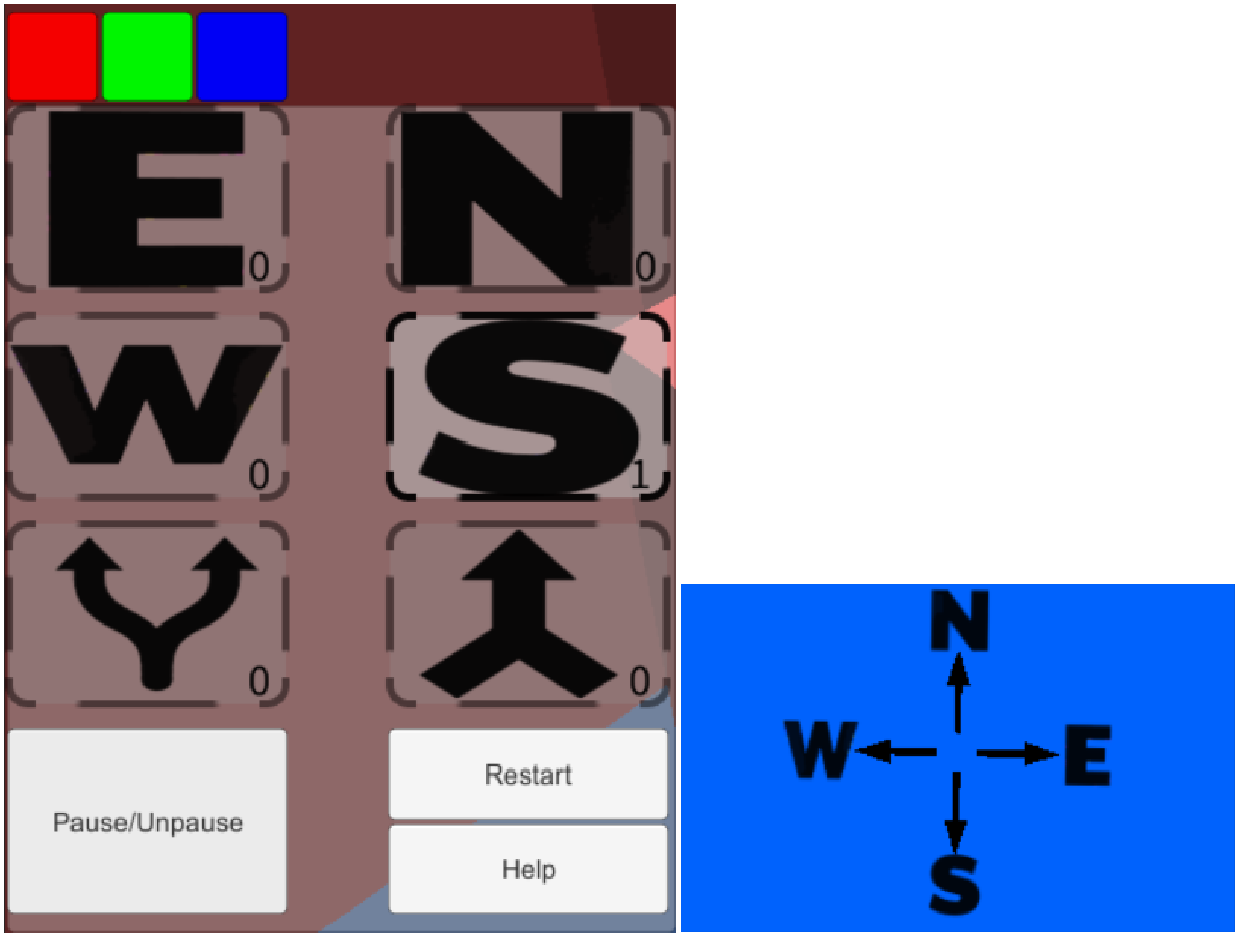


Figure 31: The action bar. The top four actions are the redirection arrows and the other two are the Split (which is represented by the line split to two arrows) and Merge (represented by two lines merged into one arrow) actions. The action bar also contains 3 buttons the Pause/Unpause button which pauses and Un-pauses the game, the Restart button which restarts the current level and the Help button which shows helping instructions for the current level. On the top we see the order in which the Slemmings will be spawned from right to left, e.g. in this case we will have a blue Slemming spawned and then a green and then a red one. On the right we see the compass which helps in the use of the redirection arrows.

The next two actions of the game are the *merge* and *split* actions. The merge action merges two Slemmings into one and combines their color just like in the RGB color model, for example combining a blue and red Slemming gives a purple Slemming. The merge action is used when there are goals with colors other than the primary RGB colors. Furthermore, there is a special rule when merging a black Slemming with any other Slemming; the resulting Slemming always stays black but it increases in size. The black Slemming is used in levels with holes where the smaller Slemmings fall inside and cannot move anymore while the bigger black one moves over the hole as it is too big to fall inside. The split action splits a Slemming into two same colored Slemmings, this action is used in levels in which the number of spawned Slemmings is not enough in order to complete it.

On top of the action bar (see Figure 31) there is a line of colored squares each of which represent a Slemming. The spawning order is represented by the order of the squares from right to left.

Finally there are 3 more buttons in the action panel. The Restart button which restarts the current level, the Help button which shows some hints for completing the current level and the Pause/Unpause button. The Restart button is used when the user has made a wrong action and the level becomes unsolvable with the remaining actions or the when the Slemmings has passed over the action base in which the user did not assign an action on it. The Help button shows some information such as the possible color combinations when merging the Slemmings. Pausing the game is really important as it gives the player time to think before choosing which actions to apply next on the action bases. Furthermore, due to the nature of the game where there are multiple Slemmings constantly moving, it is difficult to do everything when the game is un-paused as the Slemmings may have moved too far away from all the action bases.

4.1.3. Level Settings

For the game we designed 4 levels within the room. Each level is on a different part of the room "forcing" the player to have to move around the room to play the game. The levels were designed so that they can use their surroundings and the room's different characteristics such as difference in height, as much as possible helping the user to get more immersed to the game and adding more on the "illusion" of the augmented reality. The levels can be seen in Appendix B.

The levels were stored in XML files. A level editor was made in which the levels were created and then serialized into XML files. For each level we stored the spawning position of the Slemmings, the positions of the goals and action bases, the number and color of the Slemmings and the number available actions of each type. The game's levels are played in a certain order, the reason is so that the player will start with the easier levels so that he can get used to the mechanics. Each level is generated separately, once a level is complete its components are removed from the game and the next level is de-serialized in the game.

4.2. Augmented Reality

The next step after the prototyping was to convert the game to an augmented reality game. The game will be played on an Android tablet and will use its camera to capture the markers and render the results on screen.

4.2.1. Vuforia SDK

We used Vuforia SDK for developing the augmented reality part of the game. Vuforia has a Unity compatible add-on which facilitates the process of the game conversion.

For tracking we used an Image target. Image targets are predefined images printed on a paper which when detected by the device, it renders the graphics linked with these markers. The linking was done during development, where we set which graphics should be rendered when each marker is tracked. These images should have a good amount of detail which will help the device detect them more easily and in greater distance.

For RGB Slemmings we used one image marker for every level. The image markers have been virtually placed in the 3D model of the room in Unity at the same position that will be placed in the real room in order to achieve correspondence between the virtual and the real world (see Figure 33). As a result whenever a marker is detected it renders the exact position of the virtual room where the virtual marker is at. The 3D model of the room however is not rendered so that the game components will seem as they are on top of the actual furniture and not on the virtual room (see Figure 32).

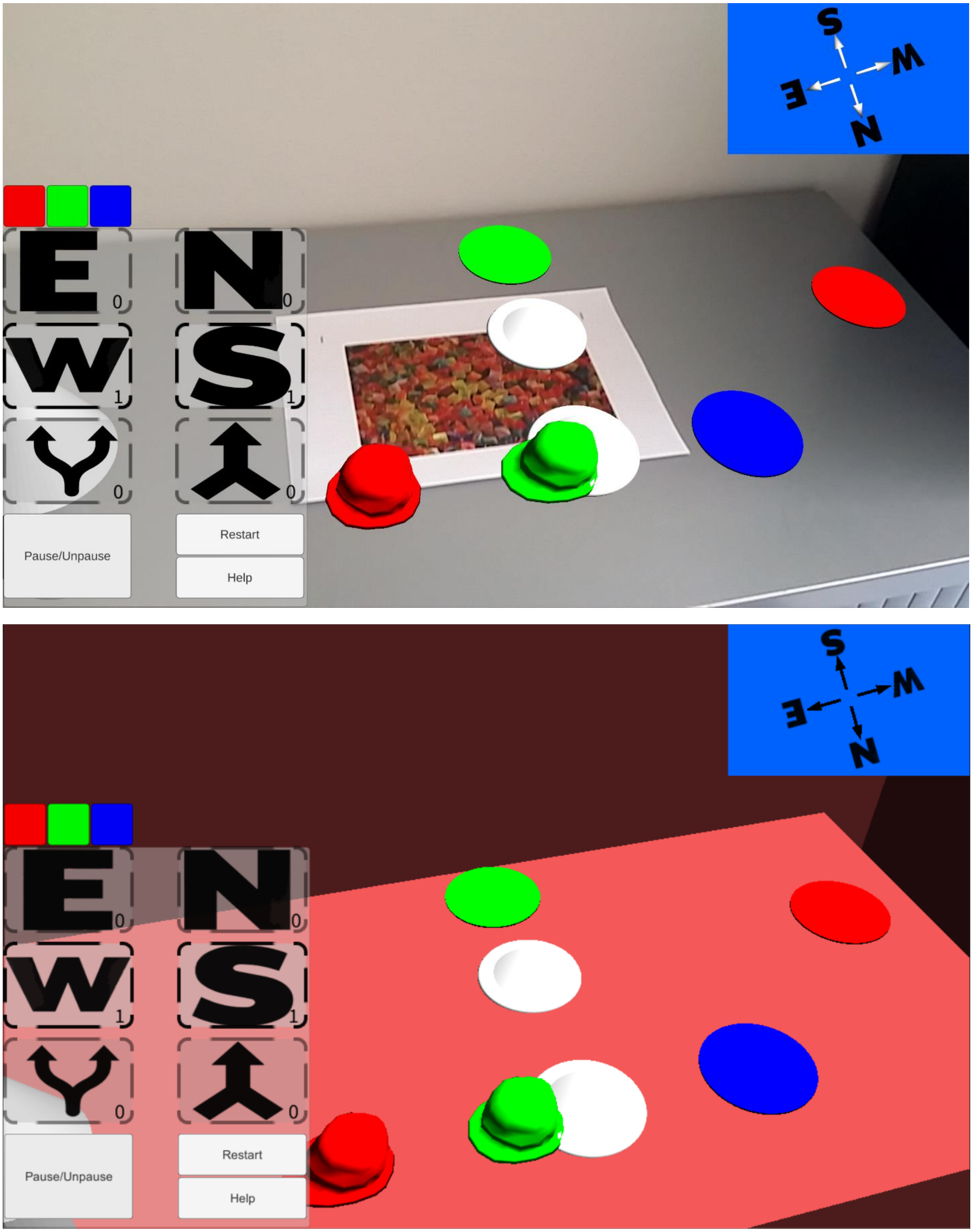


Figure 32: The same level on the augmented reality version (top) and on the pc version (bottom).

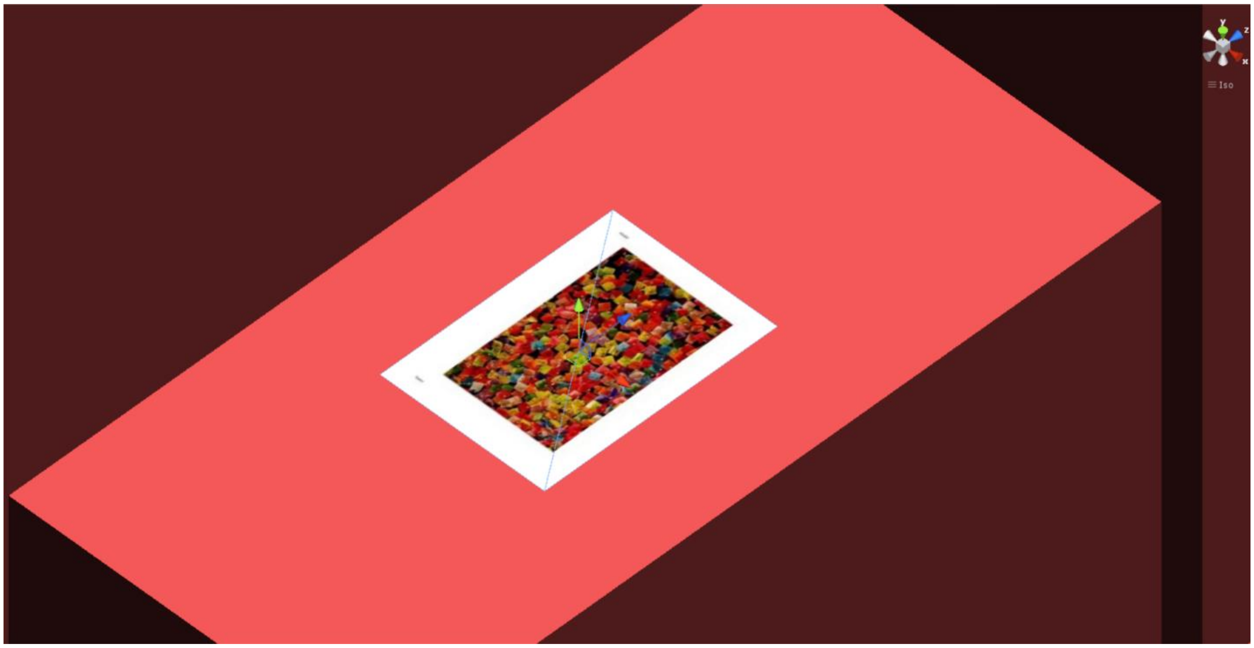


Figure 33: The virtual marker's position in the 3D model in Unity.

Chapter 5

Evaluation

In this part we will discuss the evaluation of the developed game. To evaluate this game we scheduled a user testing period, in which the testers played the game and filled out a questionnaire. In addition to the questionnaire, we analyze their in-game behavior objectively from data gathered during the gaming session. The participants were volunteers with different game experience background, with only condition that they had not played the game before i.e. testers who had played the game during the development phase to help test the mechanics and the game's difficulty were not eligible.

5.1. The goals of the evaluation process

The goal of the user testing is to measure the user's experience from the game. By experience we mean the different feelings produced to the users by the game such as immersion, flow, tension and negative feelings. Furthermore, we wanted to measure the testers' reaction in an augmented reality game that uses the environment as its level. As there are not currently many games with this type of specificity, there are not any guides or formulas of good practices that should be followed when designing this type of games, thus there are previously unexplored aspects which can be interesting to be investigated through the users' feedback. One such aspect is the amount of fatigue the user can have from having to move around with the mobile device to play the game to point at the different markers which represent the levels of the game. The game lasts as long as it takes for the user to complete it with average completion time 24 minutes with 9 minutes deviation, which can be tiresome for a person who has to stand and holding a tablet to play it. Another aspect we wanted to test was how believable the game feels to the player i.e. how "real" do the Slemmings look when they are moving on the furniture. The data gathered from this testing can be useful for similar future projects.

5.2. The user test setup

5.2.1. The questionnaire

For the questionnaire we used the game experience questionnaire (GEQ) from the work of IJsselsteijn et al [46]. In this work they developed a Likert scale questionnaire from 0 to 4 in which each question is associated with one out of the seven components (Competence, Sensory and Imaginative Immersion, Flow, Tension/Annoyance, Challenge, Negative affect, Positive affect). Competence component measures the sense of achievement the player feels by completing the in-game tasks, Imaginative Immersion describes the absorption in the narrative of the game or identification with a character and Challenge measures the amount of challenge the player felt by the game. Tension/Annoyance, Negative and Positive affect are self-explanatory titles and measure the amount of tension, annoyance and the negative and positive feelings the player had from playing the game. The Flow component was first introduced by Csikszentmihalyi [47] which describes it as "holistic sensation that people feel when they act with total involvement". Flow measures the amount of motivation the player feels, the sense of enjoyment he feels by the task itself and the level of involvement he has with the game. Flow can be considered as a form of Immersion as it involves a loss of a sense of context, while flow describes a level of complete involvement. By calculating the average scores of the questions we get the total scores for each component. From the components we decided to not measure Imaginative Immersion as our game does not have a story, any form of narration or a main character with which the player can relate to. The questions along with their associated component can be seen in Appendix C.

5.2.2. The in-game data and background information

Apart from the questions of the GEQ questionnaire in-game data were gathered along with tester background information and game specific questions. The tester background information were questions regarding the participants' background like age, gender, gaming experience while the game specific questions were about their game experience such as ease of use of controls and mechanics. The in-game data were measuring how often the player lost track of the marker, how many retries were needed to complete the game and the total time to complete the game. From the in-game data we wanted to find any correlation between the players' score and the feelings produced by the game. The complete questionnaire can be seen in Appendix C.

5.2.3. The participants

Twenty one subjects (12 Males, 9 Females) aged between 20 and 34 years (M: 25.62 years old, SD: 3.485) participated in the experiment. The subjects participated voluntarily, as it was explained that no compensation will be offered, by filling a participation form posted online.

5.2.4. The material

For the test we used a Samsung Galaxy Tab Pro 8.4" with Quad-core 2.3 GHz processor, 2 GB RAM and 8 MP camera with dimensions 219 x 128.5 x 7.2 mm and weight 331 g. The camera's resolution is important as quality of the captured images influences the performance of Vuforia on recognizing the markers. The screen size is important due to the fact that part of the screen is occluded by the UI elements of the game which contain the game's actions. The 8.4" screen is big enough as it has enough space for the UI elements and for the player to view the Slemmings. Due to the controls of the game the player must hold the device in one hand and use the second hand for playing the game. Therefore the weight and size influence how physically tiresome the game feels after a relatively lengthy session.

5.2.5. The procedure

The participants after signing a consent form played a tutorial of the game. From the tutorial, the players were introduced to the controls and mechanics of the game and the available actions and the goals they must complete to finish the game. The tutorial part consisted of five mini levels which consisted of all the different components and mechanics that will appear during the testing phase and helped the player get used to the game controls and hold the device. Upon completion of the tutorial part, the participant plays the game until he completes all four levels. Finally, the participant must fill out the questionnaire. The whole procedure lasted an average of 40 minutes, depending on the time it takes for the tester to complete the game, as it is the most time consuming part of the procedure. It is important to mention that the participants were allowed to stop taking part in the study at any time if they felt like to without needing to give any reason. All of the participants completed the test. The consent form can be seen in Appendix D.

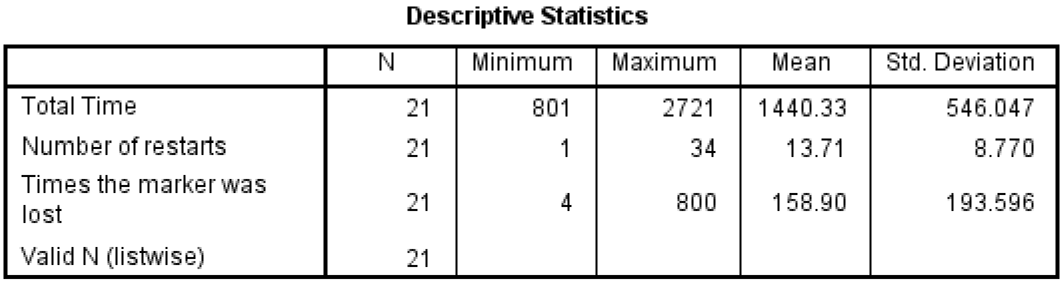
5.3. Results

5.3.1. The Participants' background

Apart from the in-game data and the GEQ questionnaire, we wanted to gather the gaming background information of the participants. From the gaming background we wanted to see whether the game appeals the same on gamers and non-gamers and if their gaming experience influences the answers. From the analysis we see that their gaming experience in a Likert scale between 0 and 4 had a mean 2.33 with standard deviation of 1.623, the mobile gaming experience had 1.9 with standard deviation 1.25 and augmented reality experience which included both using or developing an augmented reality app had 0.38 with .59 SD. From the analysis we can see that our sample consisted of both gamers and also people who never or rarely played games before. Moreover, we asked about their augmented reality experience in order to measure whether people who had some previous experience and understood the technology had different answers than people who had never had any previous contact with augmented reality. From the analysis we see that only the minority had used or developed any augmented reality app before.

5.3.2. Objective Analysis

First we will do the objective analysis of the data we gathered during the game session of the participants. The data we gathered were the players' score which was the time it took for the players to complete the game in seconds, which means the smaller the better, the number of their restarts and the number of times they lost track of the marker. It is important to mention that the participants were unaware of the type of data tracked during the test so to not influence their gaming behavior. An overview can be seen in Figure 34.



Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Total Time	21	801	2721	1440.33	546.047
Number of restarts	21	1	34	13.71	8.770
Times the marker was lost	21	4	800	158.90	193.596
Valid N (listwise)	21				

Figure 34: An overview of the data gathered during the play sessions.

5.3.2.1. Participants' total time

The participant's total time consists of gameplay time, i.e. the time it took to complete the levels, tracking time, i.e. how long it takes for the player to track the marker, and the time it takes to move from one marker to another. As the time it takes to go from one marker to another is insignificant compared to the gameplay time, we assume it is the same for all participants. From the results we see that the total time varied from 801 seconds (13.35 minutes) to 2721 seconds (47.35 minutes) with a mean of 1440 seconds (24 minutes) and standard deviation 546 (9.1 minutes). From the frequency diagram (see Figure 35) we see that most players completed the game in under 1500 seconds (25 minutes). Furthermore, as expected participants who played more video games had better scores than inexperienced players as there was negative Pearson correlation ($r = -0.755$, $p \leq 0.001$, $n = 21$).

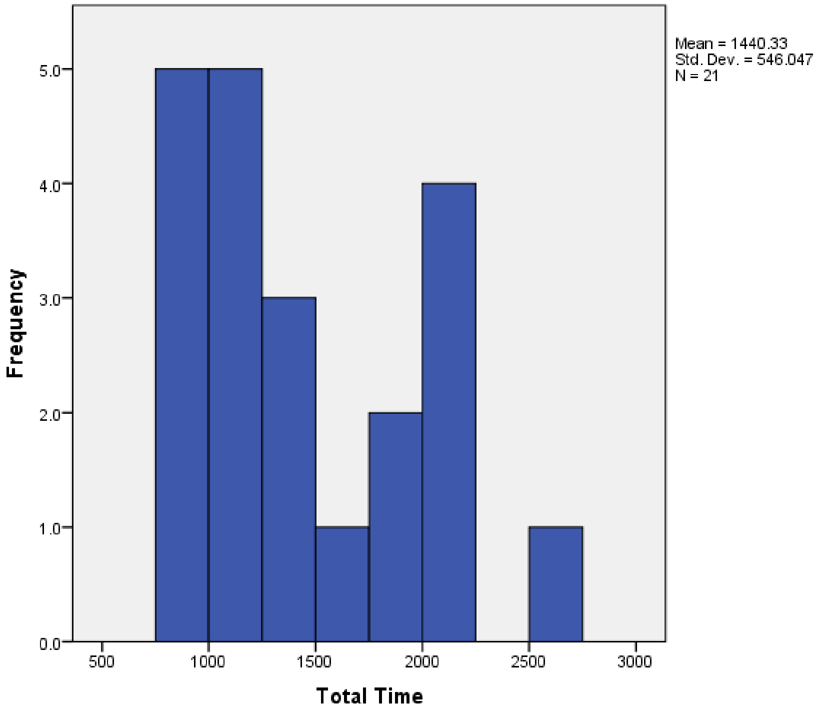


Figure 35: Frequency diagram of the completion Total time of the participants.

5.3.2.2. Number of Restarts

During the game when a player has completed his actions but has not succeeded in completing the level, he must the restart the level in order to try again. The number of total restarts varied from 1 to 34 with mean 13.71 and standard deviation 8.77. As expected there is high correlation between the number of restarts and total time ($r = 0.738$ $p \leq 0.001$ $n = 21$). A lot of players had a trial and error approach when they were trying to solve the puzzles instead of thinking the solution before trying to play level, hence the high number of restarts.

5.3.2.3. Times the marker was lost

The times the game had lost track of the game varied from 4 to 800 with a mean of 158.9 and standard deviation 193.596. From the analysis we could not find the reason which explains why some participants had high number of lost track. There are many factors that might influence this number, such as the way the participant preferred to hold the device, the size of hands as people with longer fingers can reach different parts of the screen easier and even the person’s height might influence the result as taller people tend to hold the device higher and as a result further from the marker, which causes it to lose track. Furthermore, some track losses can be attributed on the auto-focus setting of Vuforia where the camera tries to focus when there is movement in order to capture the marker more easily, which results in small unnoticeable track losses, which cumulatively result in the big values shown in the results. From the frequency diagram we see that that most of the participants had less than 100 track losses.

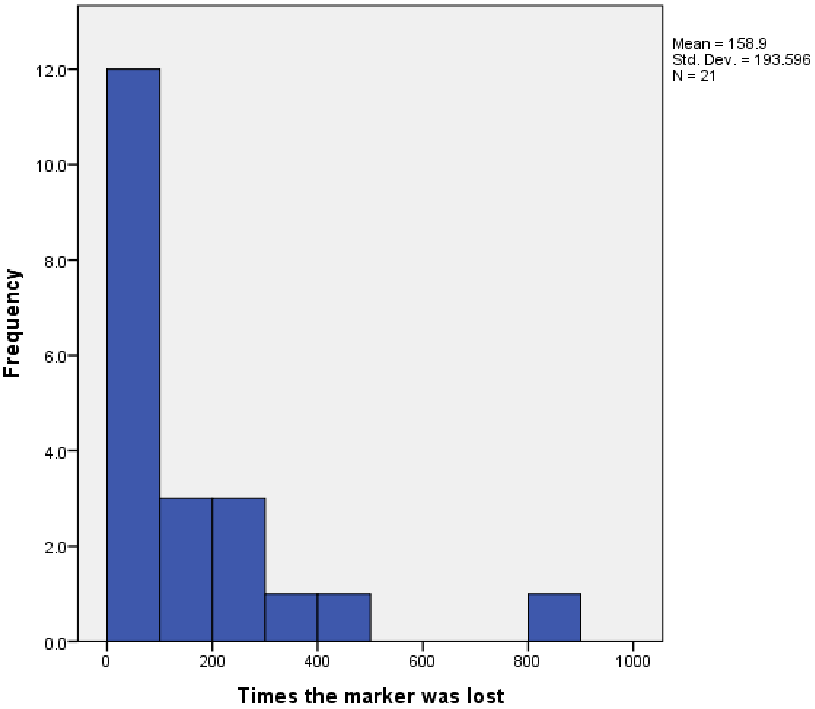


Figure 36: Frequency diagram of the times the marker was lost.

5.3.3. GEQ Component Analysis

As mentioned the GEQ questions were associated with seven components from which we discarded the Imaginative Immersion component due to its inapplicability on RGB Slemmings. The components' score were estimated from the average score of their associated questions, from that score we calculated the mean value of all the participants to calculate the total score for the component. The results can be seen in Figure 37.

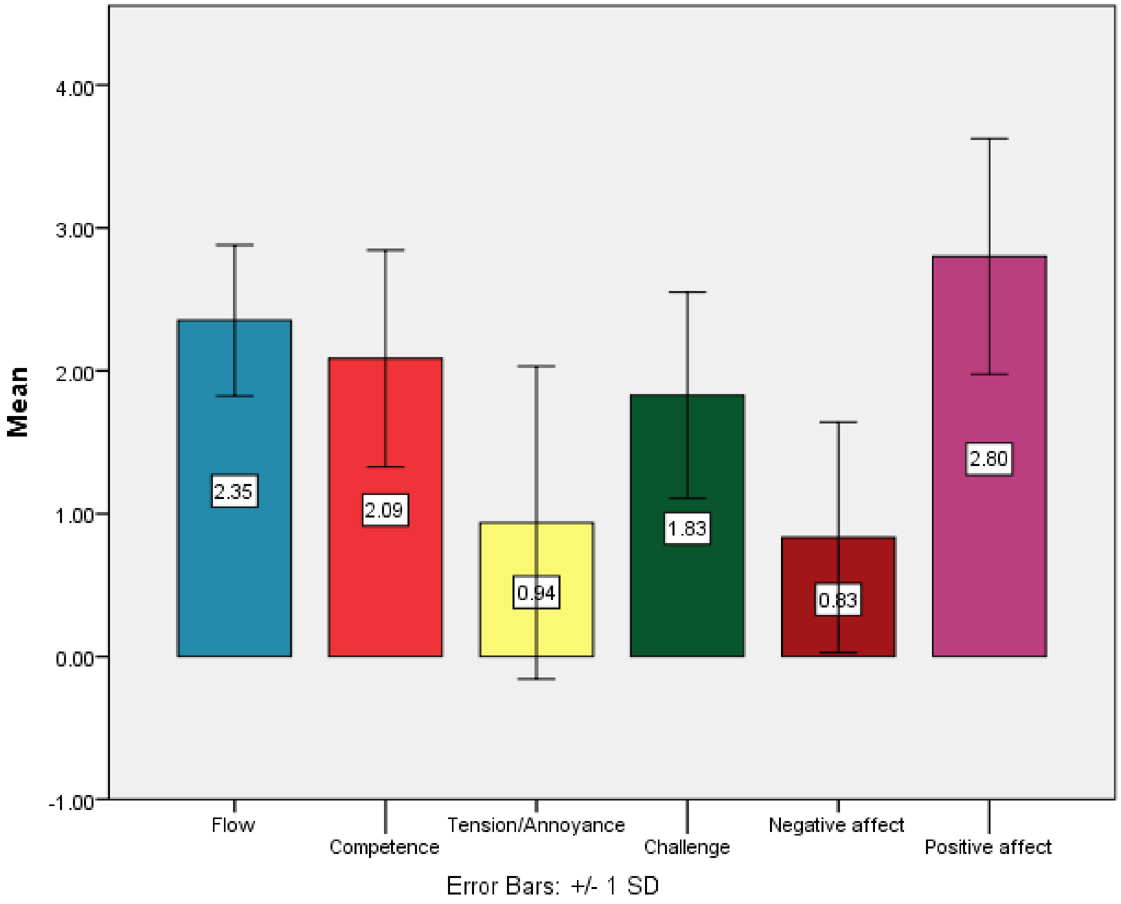


Figure 37: The mean score along with the Standard Deviation errors for each component of the GEQ questionnaire.

The results seem mostly positive as we see that there is relatively high score for Positive affect (M: 2.80, SD: 0.825) and low score on Negative affect (M: 0.83, SD: 0.808) and Tension/Annoyance (M: 0.94, SD: 1.094). Furthermore, above average Flow (M: 2.35, SD: 0.529), were average is 2, indicates that the players found the game motivating enough to continue playing it and that they found enjoyment through the tasks they had to accomplish in the game. The game seemed to challenge the players in a normal amount, as we see from the Challenge component (M: 1.83, SD: 0.722) the game was not too easy nor too difficult for the players. Finally, Competence is score is at 2.09 (SD: 0.758) which seems like it offered a sense of achievement to the players, which can be attributed to the fact that solving puzzles is considered more challenging than playing an action game hence the player feels more satisfied when completing the game.

Furthermore, we found high Pearson correlation ($r = 0.841$, $p \leq 0.001$, $n = 21$) between Negative affect and Tension/Annoyance, which was expected as these components are similar and represent the negative feelings. This means that the Negative affect was greatly influenced by the same factors which brought annoyance and tension to the player. This assumption is proved by the positive correlation between the aforementioned components and the times the marker was lost during gameplay ($r = 0.521$, $p = 0.016$, $n = 21$) with Negative affect and ($r = 0.520$, $p = 0.015$, $n = 21$) with Tension/Annoyance. Due the fact that the marker should be detected by the device in order to show the graphics and play the game, participants who had trouble in detecting the markers and playing the game without losing track expressed much more negative feelings than participants that did not experience such problems.

A moderate correlation between the number of retries and Tension/Annoyance was also noted ($r = 0.492$, $p = 0.024$, $n = 21$), which can be explained by the fact that players get frustrated when restarting multiple times. Furthermore, no fast forward option to make the Slemmings move faster exists, hence there is no way to speed up the level.

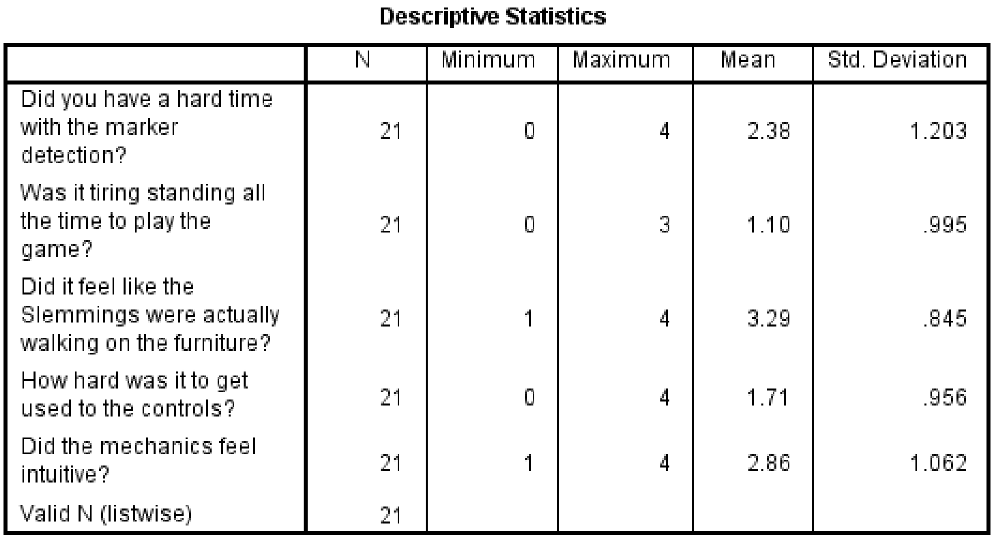
Moreover, we see there is relatively high correlation between Competence and Positive affect ($r = 0.614$, $p = .003$, $n = 21$) which shows that positive feelings are more prevalent when accompanied with sense of achievement on the players. Furthermore, these components have high scores from people who play puzzle games ($r = 0.553$, $p = .009$, $n = 21$) with Positive affect and ($r = 0.563$, $p = .008$, $n = 21$) with Competence. We should note that by puzzle games we did not mean necessarily in video games but also in paper such as Sudoku.

When analyzing the scores, we see that people with high score, i.e. small total time, felt a higher feeling of competence ($r = -0.392$, $p = .079$, $n = 21$), even though they did not know their score compared to the other participants they still felt a sense of accomplishment. The score is the total time in seconds the where lower is better, hence the negative correlation. Scores were also associated with Tension/Annoyance ($r = 0.455$, $p = .038$, $n = 21$), which can be interpreted that tension and annoyance feelings were more prevalent on participants with low scores.

The Flow component even though it had the second highest score (2.35) behind Positive feelings (2.80), it is difficult to attribute its score into one factor. By comparing the Flow with answers from individual questions we see than neither the players' performance nor their previous experience in video games had any correlation with the Flow. This can be justified by the fact that this component is the most complicated one and that there are many factors that influence the players' Flow and Immersion when playing a game. The Flow component was mostly associated by how realistic the Slemmings movement on the furniture seemed ($r = 0.479$, $p = .028$, $n = 21$).

5.3.4. Game Specific Data

Apart from the GEQ component analysis we gathered game specific data for the game in order to measure some game specific aspects of the game and the experience of the participants. From these questions we wanted to test some aspects of the game such as the game mechanics and controls which are not covered by the GEQ and find if and how they influenced the testers' experience. The overview of questions and results can be seen in Figure 38.



Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Did you have a hard time with the marker detection?	21	0	4	2.38	1.203
Was it tiring standing all the time to play the game?	21	0	3	1.10	.995
Did it feel like the Slemmings were actually walking on the furniture?	21	1	4	3.29	.845
How hard was it to get used to the controls?	21	0	4	1.71	.956
Did the mechanics feel intuitive?	21	1	4	2.86	1.062
Valid N (listwise)	21				

Figure 38: The overview of the results

5.3.4.1. Marker Detection Difficulty

On the question regarding the marker's detection difficulty where 0 was not difficult and 4 very difficult, most of the participants were having trouble to detect the marker during the test as they lost track easily of the marker and was difficult for them to capture it back. It should be noted that majority of the participants had not used an augmented reality application before, although the most experienced ones whose answers were either 1 or 2 out of 4 in Likert Scale had similar scores as people with lower scores for this question; M: 2.36, SD: 1.151 for participants who rated 0, M: 2.43, SD: 1.397 for participants who rated 1 or 2. From the Figure 39 we see that most people rated 3 out 4 on the difficulty they had on marker detection and that higher values were more prevalent than low values, which means that marker tracking was indeed difficult for most participants.

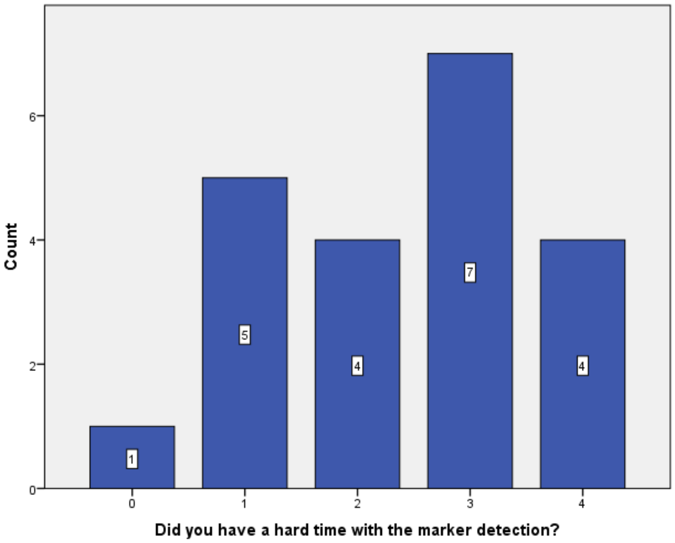
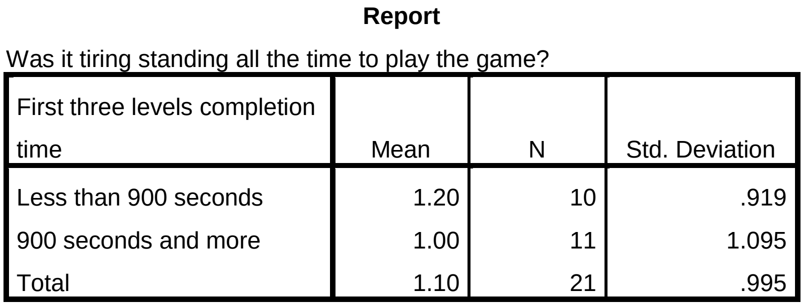


Figure 39: Frequency chart of the question "Did you have a hard time with the marker detection?"

5.3.4.2. Fatigue of the participant

The participants did not have trouble holding the tablet during the test. Although the fourth and last level of the game was played while the participant was sitting, we could not find any correlation between the time it took for the participant to complete the first three levels i.e. the ones he was standing and the answers on this question. To check the results we classified them into two categories on the first we put the participants who had completed the first three levels less than 900 seconds (15 minutes) and on the second we put the rest. From these two categories we calculated the mean value of their scores on the question “Was it tiring standing all the time to play the game?”. The result showed that there is no significant difference between the participants on the first category and the ones on the second, hence we see that there is no relation between time of playing and fatigue for a time period close to the average the participants. The results can be seen in Figure 40.



Report

Was it tiring standing all the time to play the game?

First three levels completion time	Mean	N	Std. Deviation
Less than 900 seconds	1.20	10	.919
900 seconds and more	1.00	11	1.095
Total	1.10	21	.995

Figure 40: Mean value for the two categories on the question "Was it tiring standing all the time to play the game?"

5.3.4.3. Slemming realism

On the question “Did it feel like the Slemmings were actually walking on the furniture?” the data show that most of the participants found the visual result believable enough (M: 3.29, SD: 0.845). The frequency chart shows that most people found the movement very realistic, which shows that the alignment between the 3D model and the actual furniture in real world was successful enough to create an illusion of virtual creatures walking in real world objects. An overview can be seen in Figure 41.

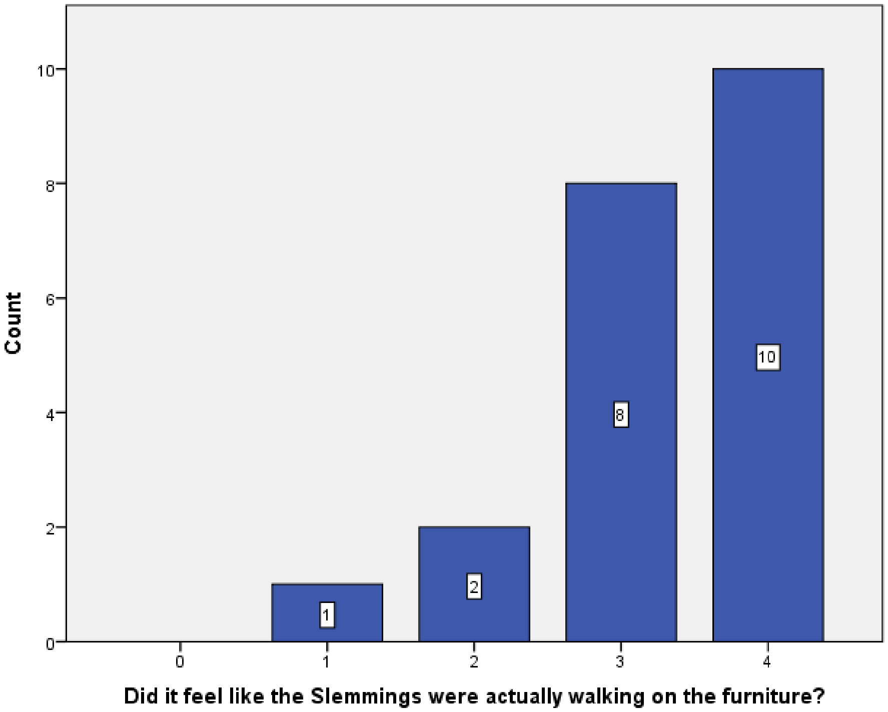


Figure 41: Frequency chart of the question "Did it feel like the Slemmings were actually walking on the furniture?"

5.3.4.4. Controls evaluation

The controls of the game consisted of the UI panel and selecting an action base in the level in which the player can apply the actions. From the results most of the participants did not find the controls easy nor difficult to learn while some of the participants stated on the feedback section that selecting the desired action from the action panel before applying it to action base was more intuitive than the way it was implemented on the game i.e. first selecting the action base then selecting the action. However, it was clear enough for the participants that the player could only interact with the UI panel and the action bases of the level. On a Likert scale where 0 is not hard and 4 is very hard, most of the participants rated 2 (M: 1.71, SD: 0.956), while more participants found the controls not hard than those who found them too hard. An overview can be seen in Figure 42.

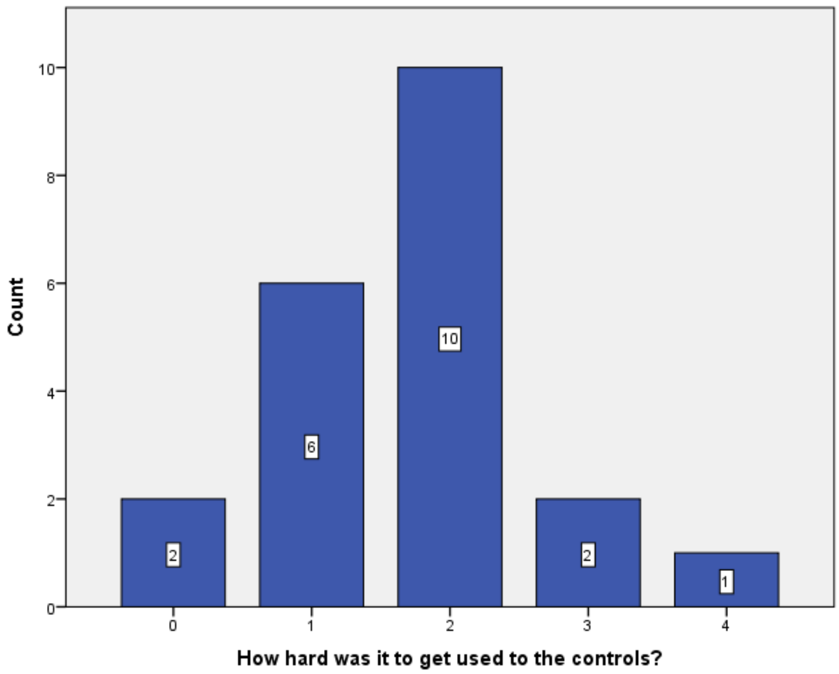


Figure 42: Frequency chart for the question "How hard was to get used to the controls?"

5.3.4.5. Mechanics evaluation

Most of participants found the mechanics intuitive and could understand their goals and the tools they had to achieve them. On a Likert scale between 0 (not intuitive) and 4 (very intuitive), most scores were concentrated on 3 and 4 while there no 0 scores . An overview can be seen in Figure 43.

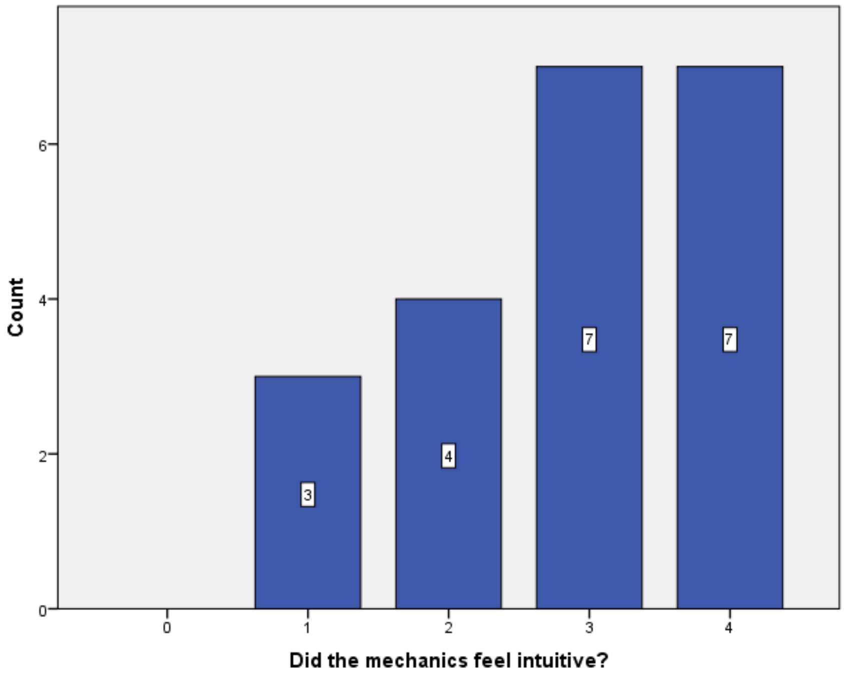


Figure 43: Frequency chart on the question "Did the mechanics feel intuitive?"

5.4. Discussion

As an overview of the data gathered from the participants, we see that their reactions have been mostly positive with high scores on components such as Positive affect and Flow and low scores on Negative affect and Tension/Annoyance. A positive remark is that even though participants had different video game experiences the game appealed the same in both gamers and non-gamers. From the responses we see that the game offered the illusion of creatures walking in the environment which is one of our goals and an important aspect for evaluating this type of technology. Furthermore, the participants did not find it tiresome to move around holding the tablet and pointing to the markers for the duration of our test which shows that handheld devices can be used in similar applications.

On the other hand we cannot discriminate whether the participants' reaction can be attributed to the augmented reality experience or to the game itself. For example one participant stated: "*I liked the puzzles, however I don't think that the augmented reality part added much to the game, except the nuisance of the camera calibration (i.e. the same puzzles would have been as mentally challenging and as entertaining in a normal 3D virtual environment)*" while another participant stated "*Interesting to try, looking forward to see more. Probably would enjoy it more with glasses and just using fingers, not have to use the tablet*". It is safe to say that tastes vary between individuals but this is the case for every type of game, hence we can assume that there were some participants that thought that our approach was interesting and that this type of technology can lead to appealing and realistic apps in the future.

We can conclude from the data and responses, that we had some success in reaching our first research goal, which was about creating an immersive game which uses its environment as part of the level game. The amount of flow and realism the game offered at the players indicate that it succeeded in creating an illusion that virtual and real objects co-exist in the game.

Chapter 6

Conclusions

6.1. Contributions

We developed and evaluated an augmented reality game which uses the real world environment as part of its gameplay. For the development phase, we demonstrated a pipeline which used different types of technologies and explained the process of developing such a game. For each step of the pipeline we discussed the technical details of the used technologies, how they are connected with each other and the occurring problems that showed during the development phase. Furthermore, we described how to use a model of a real room in order to create an augmented reality game which can be played in it.

Finally, our most important contribution was the evaluation part of our game, which was conducted through a user testing and analysis of the responses. The feedback of the user testing showed that the participants found the game immersive and that they found the interaction of the virtual objects with the real ones realistic enough. The conclusions and results of the analysis could be used for future works in this field as there are currently not many documented user experiences for such game, helping future researchers and developers avoiding mistakes done in our work.

6.2. Conclusion

We have reached our first research goal which was about creating an augmented environment in which gave the player the illusion that the virtual environment and the real one are the same. The game offered a form of immersion and entertainment to the participants, as shown from the results which were mostly positive. On the other hand, most of the negative remarks were about the repeated track-loss of the markers, as the players found it sometimes hard to re-track the markers. These pauses until the marker was tracked again made it difficult for the players to immerse into the game. It is encouraging that the game was motivating enough for all the participants to complete it and that it was equally appealing to both gamers and non-gamers.

Furthermore, we did not succeed on our second research goal which was to create a pipeline which automatically creates the 3D model of the room. The results when not as promising as expected and they could not be used for our game. The complexity and at the same time simplicity in texture of indoor environments is an obstacle in creating accurate scans and 3D models. Our pipeline could work in environments with simple shaped furniture and complex coloring, which will facilitate the feature extraction, but this not a realistic use case. The pipeline should contain a more robust scanning technique, invariant to different lightning conditions and types of furniture, which could be succeeded by a purely geometrical SLAM techniques. Moreover, a 3D reconstruction technique, capable of reconstructing different shapes would be also a more suitable for reconstructing point cloud scans with complex shapes. Automatic creation of a 3D environment from scanning is not necessary for creating games such as RGB Slemmings, but can greatly aid the development as it can speed up the process and make it robust to changes in the environment. Also, it is required to play these games in a random gamer's room.

In conclusion although the results of our work cannot be applied in all types of augmented reality applications, the pipeline’s structure and the responses can be useful as a starting point for future applications that use the user's surroundings. More specifically, we saw that most users are not familiar with this type of technology, hence it was important to have clear goals and actions and user friendly type of interaction for them. We faced this problem with the introduction of the tutorial section, which did not only showed how the game was played but also how they should hold the device and that it should point on the marker. Furthermore, we saw amount of immersion they felt was proportional with the amount of realism and interaction the virtual components had with the real ones. Finally, we saw the biggest problem that should be faced was the tracking, as it caused nuisance on the players, hence, it is one of the parts that need improvement.

6.3. Future Works

There is much to be done which can improve our work. A robust pipeline for automated room modeling would speed up the development process as well as improve the quality of the result. This would require a robust scanning technique which can handle different environments and a versatile 3D reconstruction algorithm which can reconstruct many types of shapes. More sophisticated augmented reality hardware and techniques such as marker-less tracking that offer more stable tracking is necessary for more immersive experiences and they can expand the possibilities of such applications. Such hardware can be an augmented reality headset as it offers a more complete augmented reality experience than a mobile device as it is more natural for a person to explore his surroundings by moving his head and eyes and not through a screen. Another improvement would be the use of object recognition instead of marker detection, as the use of markers constrain the players' movement and view as they should be always visible. Furthermore, interacting with the use of hands instead of a touch screen will improve the experience as it will distort the barriers of real and virtual world since the player uses the same tools, his hands, to interact with both worlds. This could be done with finger tracking by the device along with mapping of simple gestures which can be used for interaction with the application.

Moreover, there are many types of applications that can benefit with the use of such technologies. Such applications could be renovation applications in which the user can alter the environment in order to demonstrate future changes in the room, such as different wall colors or furniture, before actually making them. Another field is training simulations, such as firefighter training where the environment could represent a supposedly flaming room, which offer safe form of training. Also the environment could easily change in order to simulate different training scenarios.

Additionally, with an automated room reconstruction, it will be easier for consumers to use the applications in their own environments. An example would be a game such as RGB Slemmings which can automatically create levels based on the input room given by the player. Thus the user will need a Kinect or a similar laser scanner to scan his room which will be then given as input to the game which will then create the levels, solving the constraint of such applications which is that they can be only played in previously scanned rooms.

These were only a few of the possible applications and the future of augmented reality seems very promising with a lot of potential research and applications.

Bibliography

[1] I. E. Sutherland, "A head-mounted three dimensional display," in *Proceeding AFIPS '68 (Fall, part I) Proceedings of the December 9-11, 1968, fall joint computer conference*, New York, 1968.

[2] T. . P. Caudell and D. Mizell , "Augmented reality: An application of heads-up display technology to manual manufacturing processes.," in *Proceedings of the Twenty-Fifth Hawaii International Conference on Systems Sciences*, Kauai, Hawaii, 1992.

[3] Microsoft, "Microsoft HoloLens," [Online]. Available: http://www.microsoft.com/microsoft-hololens/en-us. [Accessed 2015].

[4] "Magic Leap," [Online]. Available: http://www.magicleap.com/#/home. [Accessed 2015].

[5] "Magic Leap Demo," [Online]. Available: https://www.youtube.com/watch?v=kw0-JRa9n94. [Accessed 2015].

[6] Lemmings, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Lemmings_(video_game). [Accessed 2015].

[7] "Kinect," [Online]. Available: https://www.microsoft.com/en-us/kinectforwindows/develop/. [Accessed 2015].

[8] "Table Zombies," [Online]. Available: https://play.google.com/store/apps/details?id=com.srg.tzcards. [Accessed 2015].

[9] "Temple Treasure Hunt," [Online]. Available: http://thoughtshastra.com/temple-treasure/. [Accessed 2015].

[10] "AR Invaders," [Online]. Available: https://play.google.com/store/apps/details?id=com.soulbit7.game.arinvaders. [Accessed 2015].

[11] "PulzAR," [Online]. Available: https://www.playstation.com/en-us/games/pulzar-psvita/. [Accessed 2015].

[12] "Archery," [Online]. Available: http://nintendo.wikia.com/wiki/AR_Games. [Accessed 2015].

[13] "Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=com.srg.tzcards. [Accessed 2015].

[14] "Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=com.thoughtshastra.templetreasure. [Accessed 2015].

[15] "Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=com.soulbit7.game.arinvaders. [Accessed 2015].

[16] "Kotaku," [Online]. Available: http://www.kotaku.com.au/2012/05/im-actually-on-the-bandwagon-for-a-vita-game-called-pulzar-join-me/. [Accessed 2015].

[17] "IGN," [Online]. Available: http://www.ign.com/wikis/ar-games/Nintendo_3DS_Guide_part_2. [Accessed 2015].

[18] J. Aulinas, Y. Petillot, J. Salvi and X. Lladó, "The SLAM problem: a survey," in *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence,* Sant Martí d'Empúries, 2008.

[19] B. Gokturk, H. Yalcin and C. Bamji, "A Time-Of-Flight Depth Sensor – System Description, Issues and Solutions," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04)* , Washington DC, 2004.

[20] M. Labbé and F. Michaud, "Memory Management for Real-Time Appearance-Based Loop Closure Detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems,* San Francisco, 2011.

[21] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding,* vol. 110, no. 3, pp. 346-359, 2006.

[22] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Kauai, Hawaii , 2001.

[23] P. Besl and N. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 14, no. 2, pp. 239-256, 1992.

[24] "RTAB-Map," [Online]. Available: http://introlab.github.io/rtabmap/. [Accessed 2015].

[25] "RTAB-Map," [Online]. Available: https://github.com/introlab/rtabmap/wiki/Tutorials. [Accessed 2015].

[26] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *ISMAR '11 Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Basel, 2011.

[27] T. van Lankveld, M. van Kreveld and R. Veltkamp, "Identifying rectangles in laser range data for urban scene reconstruction," *Computers & Graphics,* vol. 35, no. 3, pp. 719-725, 2011.

[28] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier and B. MacIntyre, "Recent Advances in Augmented Reality," *IEEE Computer Graphics and Applications,* vol. 21, no. 6, pp. 34-47, 2001.

[29] . D. W. F. van Krevelen and R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations," *The International Journal of Virtual Reality,* vol. 9, no. 2, pp. 1-20, 2010.

[30] Z. Huang, P. Hui, C. Peylo and D. Chatzopoulos, "Mobile Augmented Reality Survey: A Bottom-up Approach," in *arXiv preprint*, arXiv, 2013.

[31] G. Simon, A. Fitzgibbon and A. Zisserman, "Markerless Tracking using Planar Structures in the Scene," in *Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, Munich, 2000.

[32] "Vuforia," [Online]. Available: https://www.qualcomm.com/products/vuforia. [Accessed 2015].

[33] "Metaio," [Online]. Available: https://www.metaio.com/. [Accessed 2015].

[34] M. Labbé and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation," *IEEE Transactions on Robotics,* vol. 29, no. 3, pp. 734 - 745, 2013.

[35] M. Labbé and F. Michaud, "Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems* , Chicago, 2014.

[36] M. de Berg, O. Cheong, M. van Kreveld and M. Overmars, Computational Geometry Algorithms and Applications 3rd Edition, Berlin Heidelberg: Springer, 2008.

[37] M. van Kreveld, T. van Lankveld and R. Veltkamp, "On the shape of a set of points and lines in the plane," Department of Information and Computing Sciences Utrecht University, Utrecht, 2011.

[38] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applicatlons to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[39] P. Meer, "Robust Computer Vision," [Online]. Available: http://soe.rutgers.edu/~meer/TEACH/570.html. [Accessed 2015].

[40] T. van Lankveld, M. van Kreveld and R. Veltkamp, "Identifying rectangles in laser range data," Technical Report UU-CS-2011-004, Department of Information and Computing Sciences - Utrecht University, Utrecht, The Netherlands, 2011.

[41] "Point Cloud Library," [Online]. Available: http://pointclouds.org/. [Accessed 2015].

[42] "Vuforia Scanner," [Online]. Available: https://developer.vuforia.com/library/articles/Training/Vuforia-Object-Scanner-Users-Guide. [Accessed 2015].

[43] "Vuforia Developer Portal," [Online]. Available: https://ui-dev2.vuforia.com/resources/dev-guide/vuforia-ar-architecture. [Accessed 2015].

[44] "Star Wars: Pit Droids," [Online]. Available: http://starwars.wikia.com/wiki/Star_Wars:_Pit_Droids. [Accessed 2015].

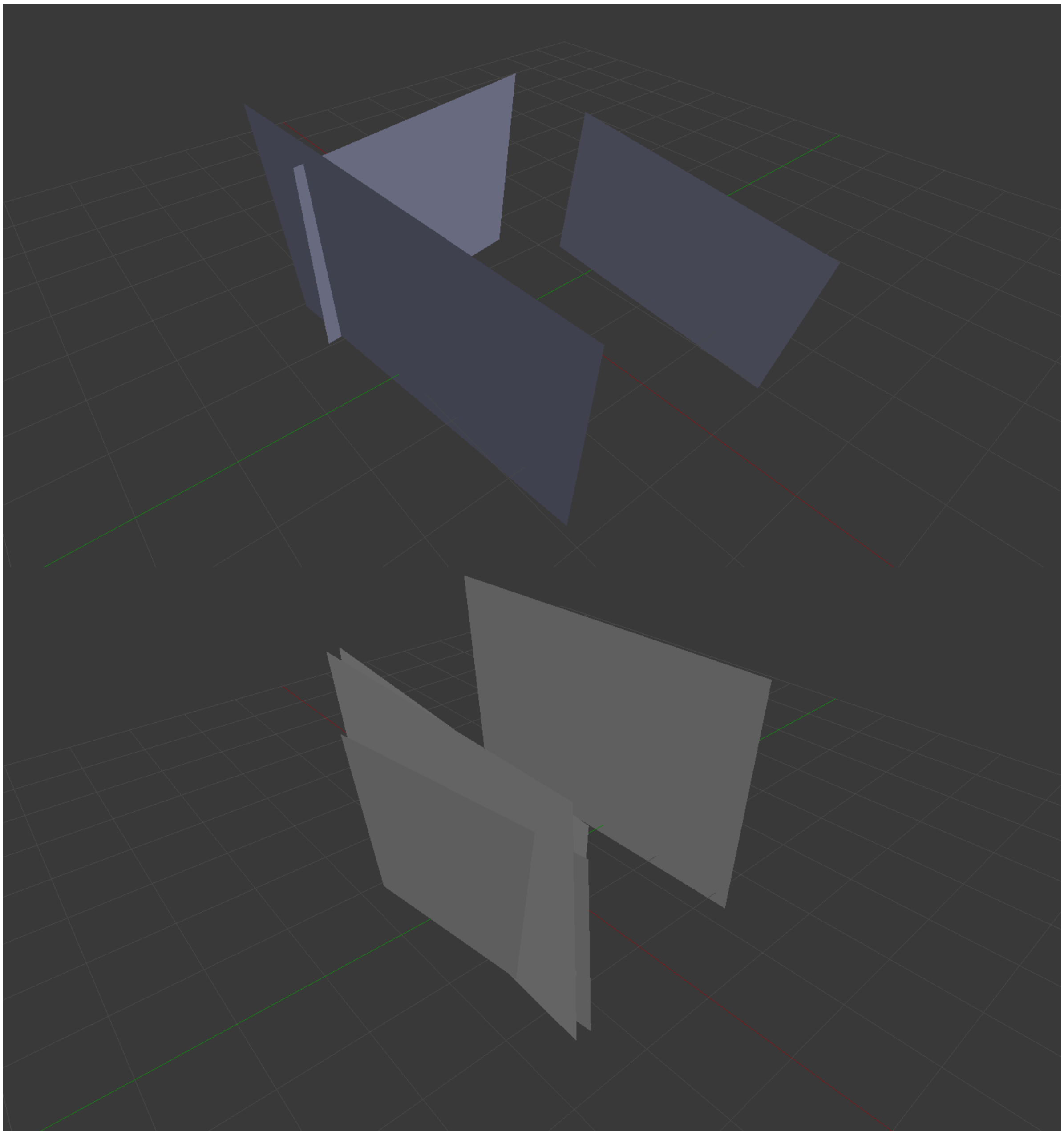
[45] "Unity," Unity, [Online]. Available: https://unity3d.com/. [Accessed 2015].

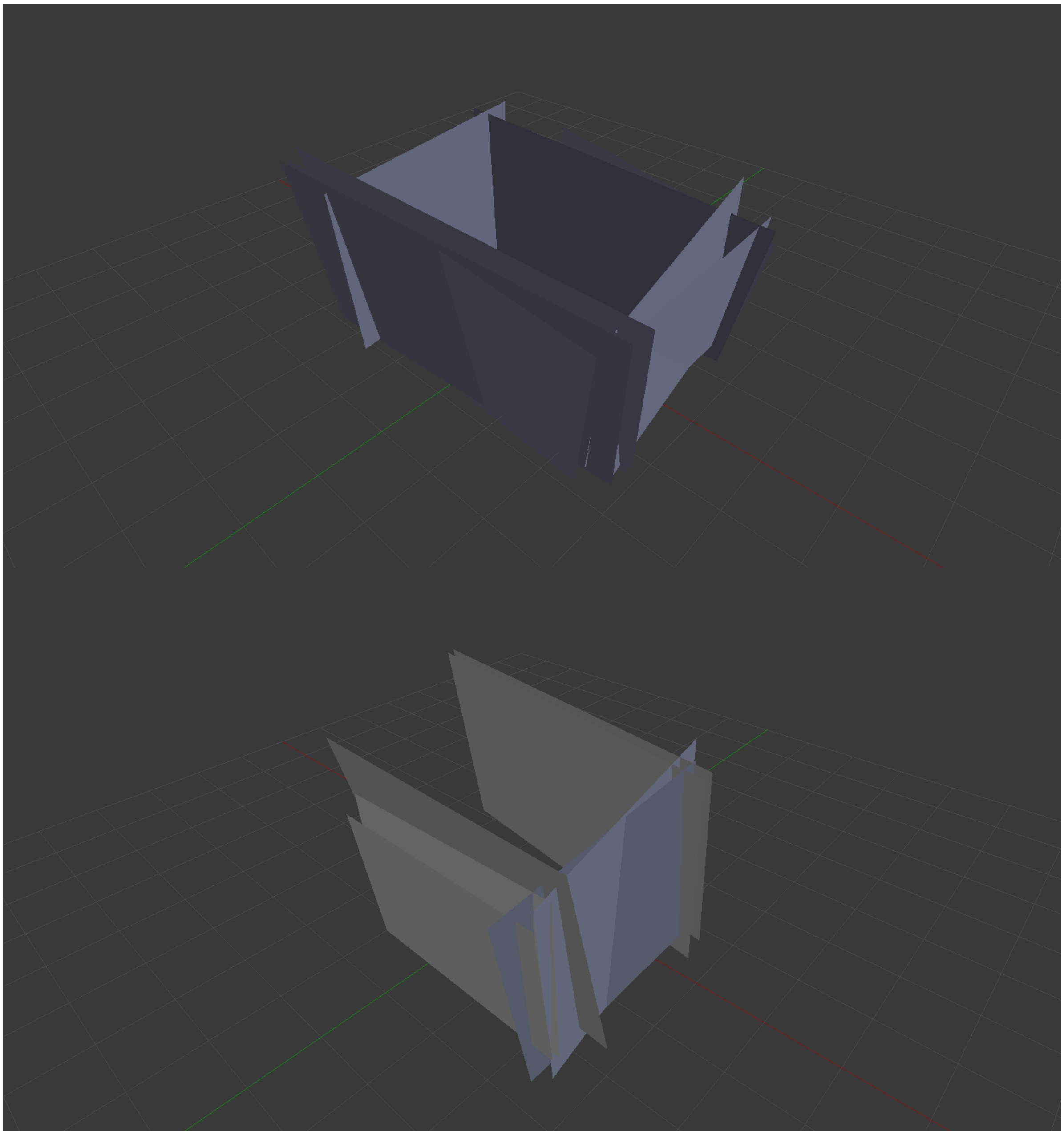
[46] W. IJsselsteijn, Y. de Kort and K. Poels, "The Game Experience Questionnaire: Development of a self-report measure to assess the psychological impact of digital games.," Manuscript in Preparation, (In Preparation), 2013.

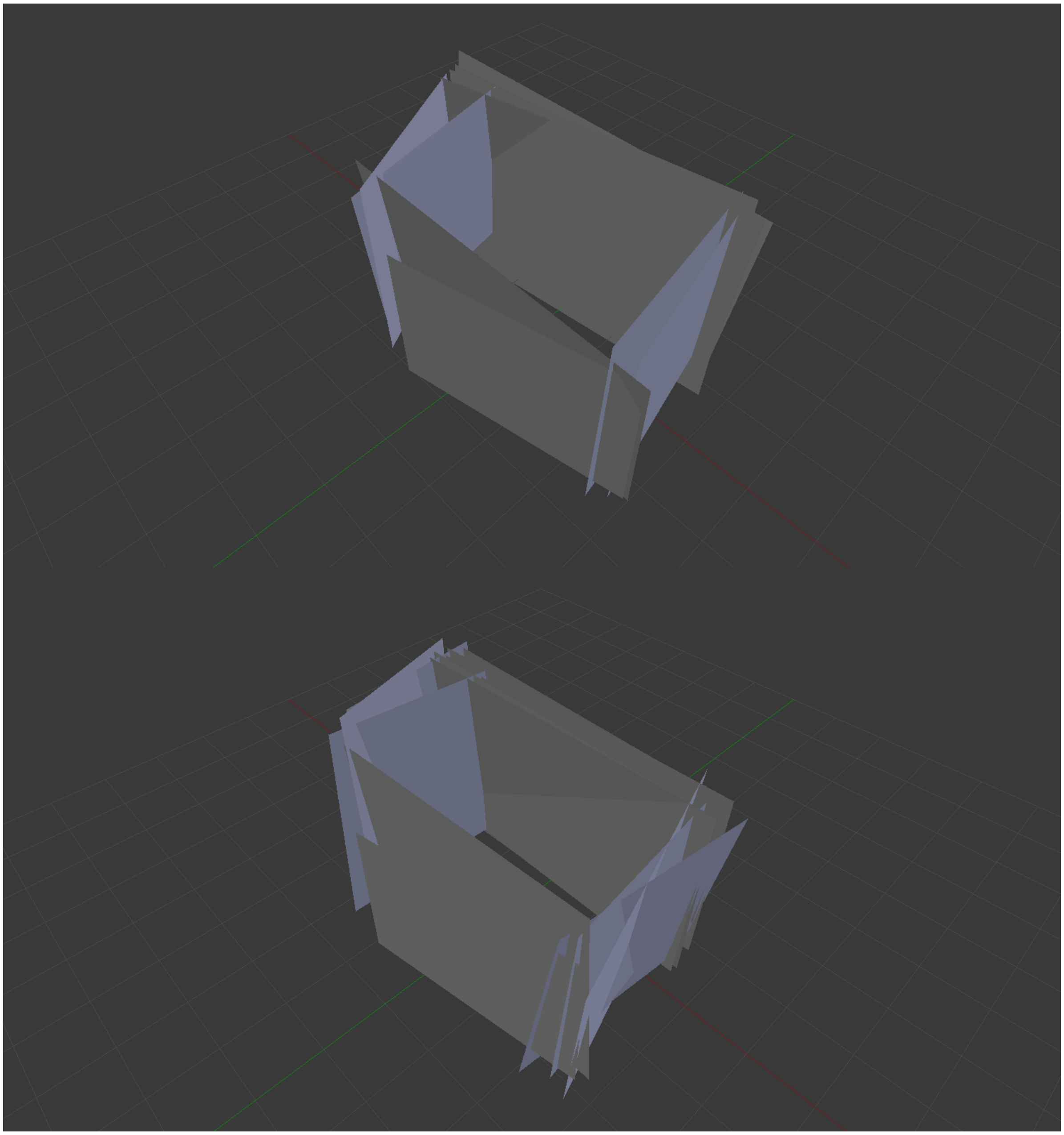
[47] M. Csikszentmihalyi, Beyond Boredom and Anxiety: Experiencing Flow in Work and Play, San Francisco: Jossey-Bass, 1975.

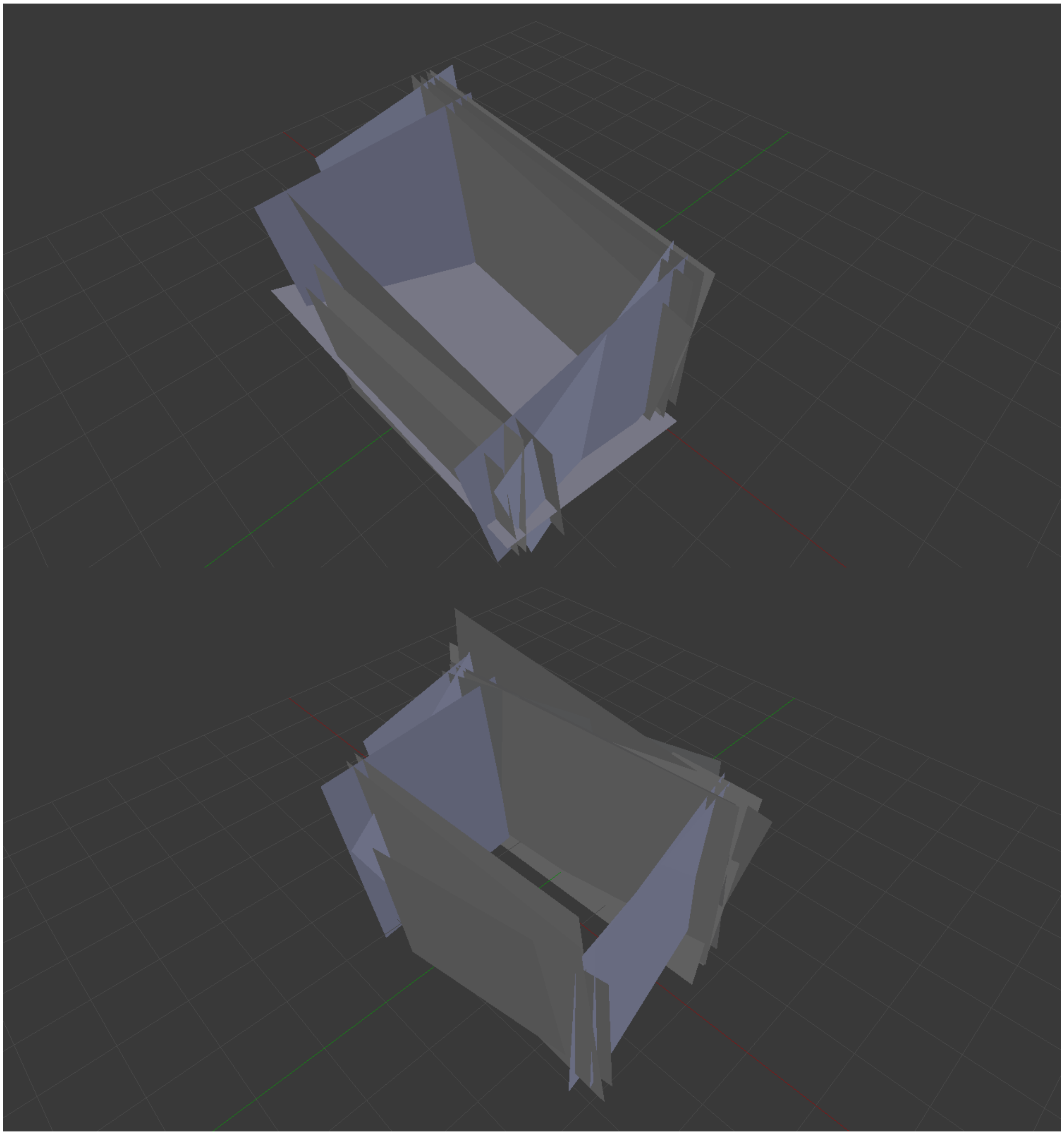
Appendix A

Reconstruction Results









Appendix B

The game levels in the PC version.

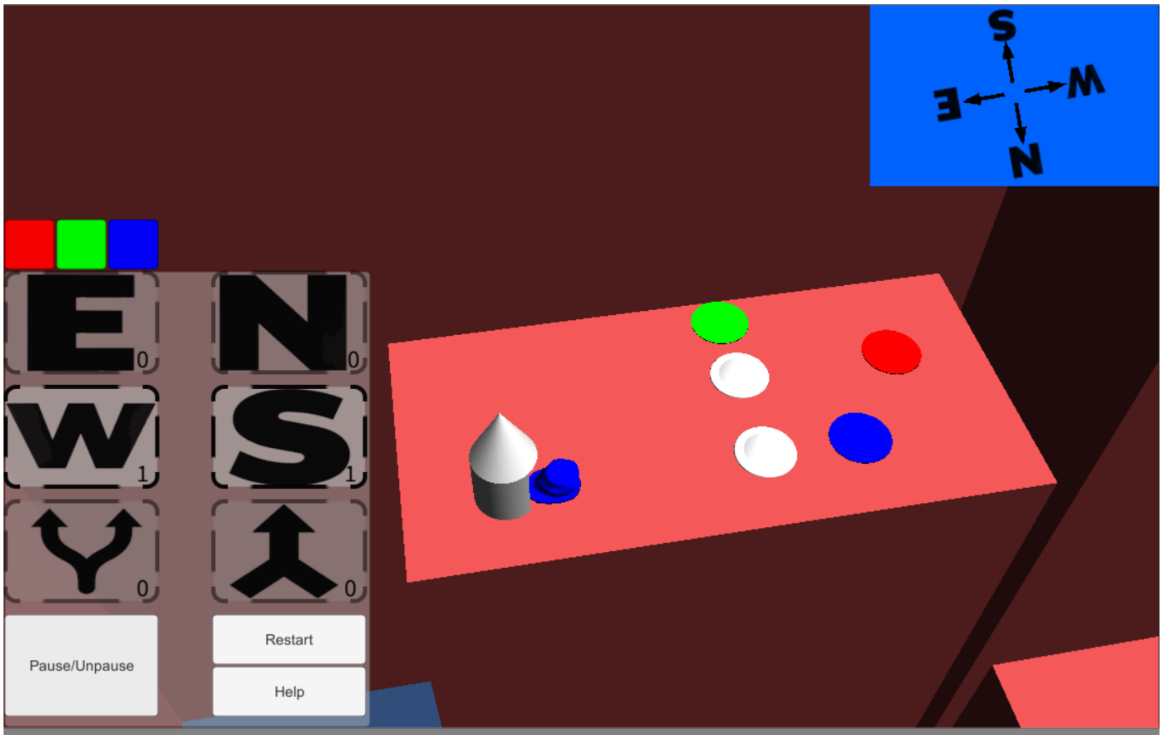


Figure 44: Level 1.

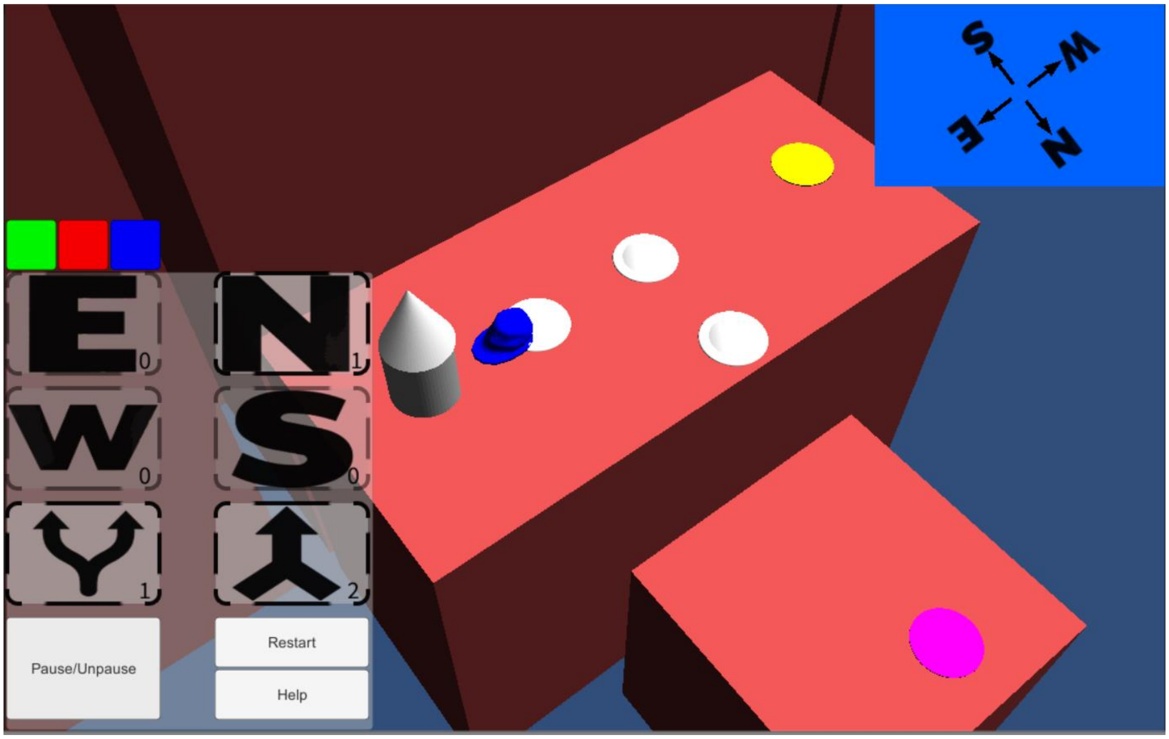


Figure 45: Level 2.

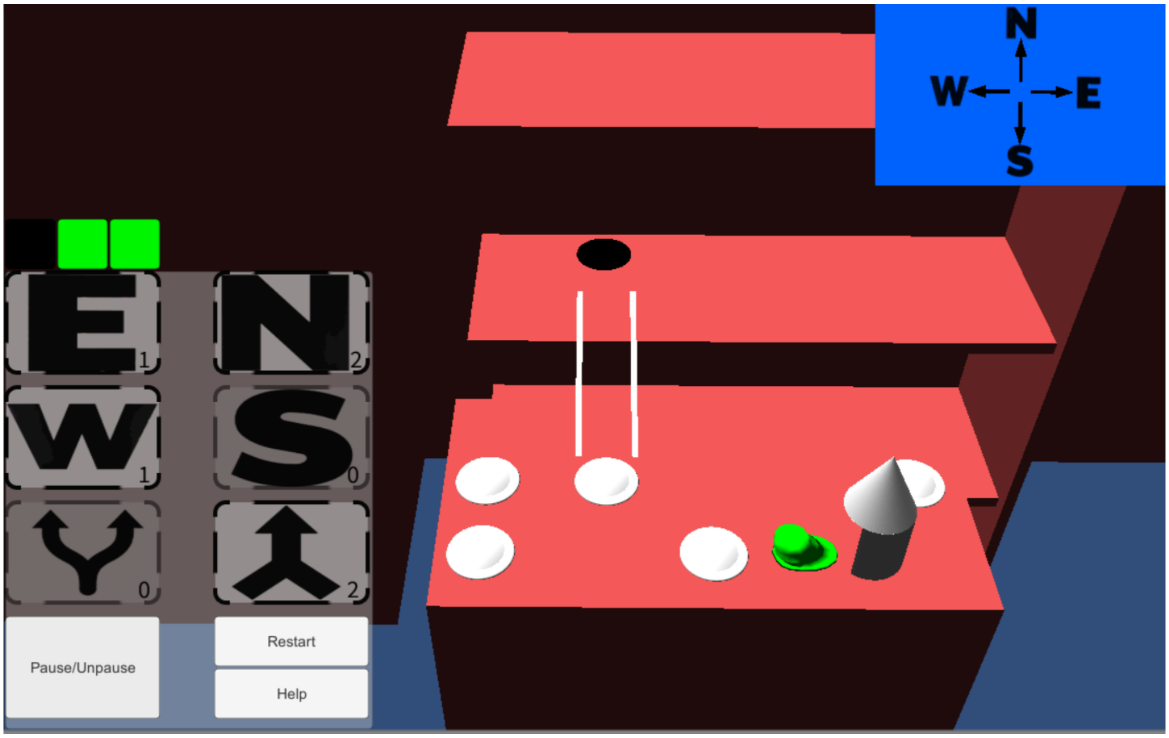


Figure 46: Level 3.

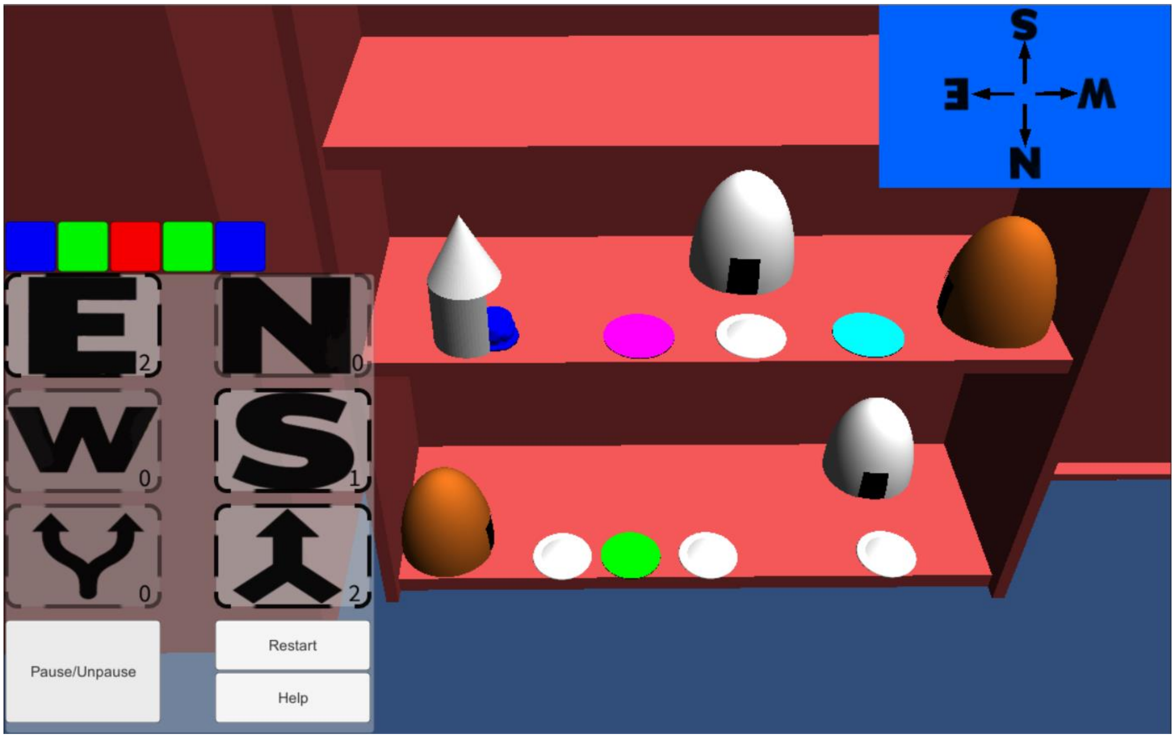
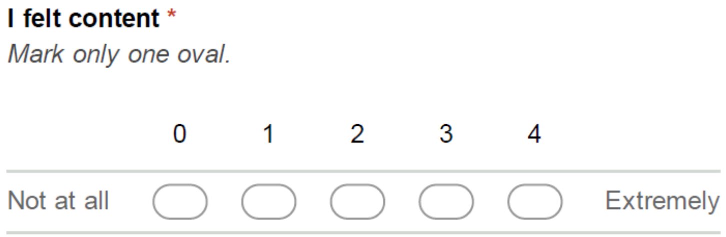


Figure 47: Level 4.

Appendix C

1) GEQ Questionnaire

a) An example of appearance of a question of from the GEQ questionnaire:



I felt content *

Mark only one oval.

	0	1	2	3	4	
Not at all	○	○	○	○	○	Extremely

Figure 48: An example of question of the GEQ questionnaire.

b) The GEQ questions with the associated component

Question	***Component***
I felt content.	Positive affect
I felt skillful.	Competence
I thought it was fun.	Positive affect
I was fully occupied with the game.	Flow
I felt happy.	Positive affect
It gave me a bad mood.	Negative affect
I thought about other things.	Negative affect
I found it tiresome.	Negative affect
I felt competent.	Competence
I thought it was hard.	Challenge
I forgot everything around me.	Flow
I felt good.	Positive affect
I was good at it.	Competence
I felt bored.	Negative affect
I felt successful.	Competence
I enjoyed it.	Positive affect
I was fast at reaching the game's targets.	Competence
I felt annoyed.	Tension/Annoyance
I felt pressured.	Challenge
I felt irritable.	Tension/Annoyance
I lost track of time.	Flow
I felt challenged.	Challenge
I was deeply concentrated in the game.	Flow
I felt frustrated.	Tension/Annoyance
I lost connection with the outside world.	Flow
I felt time pressure.	Challenge
I had to put a lot of effort into it.	Challenge

2) The complete Questionnaire

Q1 I felt content.
Q2 I felt skillful.
Q3 I thought it was fun.
Q4 I was fully occupied with the game.
Q5 I felt happy.
Q6 It gave me a bad mood.
Q7 I thought about other things.
Q8 I found it tiresome.
Q9 I felt competent.
Q10 I thought it was hard
Q11 I forgot everything around me.
Q12 I felt good.
Q13 I was good at it.
Q14 I felt bored.
Q15 I felt successful.
Q16 I enjoyed it.
Q17 I was fast at reaching the game's targets.
Q18 I felt annoyed.
Q19 I felt pressured.
Q20 I felt irritable.
Q21 I lost track of time.
Q22 I felt challenged.
Q23 I was deeply concentrated in the game.
Q24 I felt frustrated.
Q25 I lost connection with the outside world.
Q26 I felt time pressure.
Q27 I had to put a lot of effort into it.
Q28 How hard was it to get used to the controls?
Q29 Did the mechanics feel intuitive?
Q30 Did it feel like the Slemmings were actually walking on the furniture?
Q31 Did you have a hard time with the marker detection?
Q32 Was it tiring standing all the time to play the game?
Q33 Comments.

Tester Background Questions, asked at the beginning.

- Name, Gender
- Age
- How much experience do you have in playing video games?
- How much experience do you have in playing puzzle games? *Not necessarily in video games
- How much experience do you have in playing mobile games?
- How much prior experience did you have in augmented reality games? *Playing or Developing
- Which similar puzzle games have you played before? *Tick all that apply.
    - Lemmings (or any variations)
    - Star Wars: Pit Droids
    - Clones
    - Pingus
    - The Humans
    - I haven't played any similar games
    - Other:

Appendix D

Consent Form

Title Project: RGB Slemmings

Principal Investigator: Alexandros Panayiotou

You are being invited to take part in a user study on augmented reality. You will play several levels of the game “RGB Slemmings” and will be asked about your experience. The study will last for about 35-40 minutes.

Your game actions and answers to the questionnaire will be stored. Your data will be treated anonymously. Upon verification of your data, your name will be replaced by a participant ID. With your consent, the data will be further analyzed. It will not be linked to you.

	Please tick box
1. I have read and understood the instruction for this study and acknowledge that have had the change to ask questions.	☐
2. I understand that I can stop taking part in this study at any time and, if I choose to do so, I do not have to explain why I am stopping.	☐
3. I give permission for the data collected in this experiment to be used for the purposes of this research.	☐
4. I give permission for the data collected in this experiment to be further analyzed and coded. Data will remain anonymous.	☐

Please sign on the line below. By doing so you are agreeing to take part in this study.

X ______________________

Signature

X ______________________

Date

X ______________________

Researcher Signature

X ______________________

Date