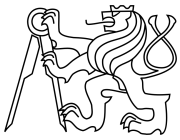
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Augmented Reality Game for Android
Student:	Uršuľa Žákovská
Supervisor:	Ing. Marek Žehra
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The aim of the thesis is to design and implement a game using augmented reality (AR) on a mobile platform.

1. Perform a review of several games using AR on mobile platforms. Describe mainly the way how they use AR.
2. Perform a review of tools for mobile platforms for development of applications with AR features. Choose only those allowing Multiple Target Tracking.
3. Design your own game with AR features. The game must allow later extension to multiplayer mode with more than 2 players.
4. Select implementation platform based on item 2 and implement a prototype of the game specified in item 3. Part of the game can be played in offline single-player mode.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 26, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of Software engineering

Bachelor's thesis

Augmented reality game for Android

Uršuľa Žákovská

Supervisor: Ing. Marek Žehra

17th May 2016

Acknowledgements

I would like to thank my supervisor Ing. Marek Žehra for all the suggestions and support during the development process and writing of the thesis text. Then I would like to thank my family for their care and support during my study on CTU. Also, I want to thank CTU for giving me the opportunity to go on an exchange program where I've gained the inspiration for this thesis. Last but not least, big thank you to all anonymous contributors on the Unity and Vuforia forum sites for sharing their solutions to known problems.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 17th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Žákovská, Uršuľa. *Augmented reality game for Android.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Hlavním cílem této práce je analyzovat *Rozšířenou realitu* (AR) a její implementaci na mobilních platformách. Byl proveden rozbor dvou hlavních přístupů pro mobilní AR i revize současných řešení na trhu. Vzhledem k tomu, že neexistuje jeden nástroj který by byl nejlepším řešením pro všechny případy využití, byla taky provedena analýza nástrojů pro tvorbu mobilní AR. Jako důkaz výsledků z předchozích analýz a příklad jejich praktického využití, design mobilní hry s AR prvky byl navržen a její prototyp realizován s Vuforia frameworkem [1] v herním enginu Unity [2]. Poté byl prototyp úspěšně instalován na mobilním zařízení s operačním systémem Android a otestován uživatelskými testy.

Klíčová slova Rozšířená realita, mobilní hra, Android, Unity, C#, Vuforia UDT

Abstract

The main goal of this thesis is to analyze *Augmented reality* (AR) and its implementation on mobile platforms. Examination of the two main approaches for mobile AR was conducted and a review of current market solutions was performed. Because the single best solution for AR doesn't exist, an examination of frameworks was conducted too. As a proof of the results from the previous analysis and an example of their practical usage, an AR game was designed and its prototype implemented with Vuforia [1] framework in Unity [2] game engine. Then it was successfully installed onto the mobile device with Android OS and evaluated by usability tests.

Keywords Augmented Reality, mobile game, Android, Unity, C#, Vuforia UDT

Contents

List of Figures

List of Tables

Introduction

People surround themselves with more and more technology starting from television, notebooks to smartphones or smartwatches. In the past, we could easily differentiate between real and virtual experience. Interaction in the virtual world had no connection to our current physical world and vice versa. But today almost everyone uses technology that combines both of these worlds. We found a new way to interact with them by creating modified versions of them and merging them into one.

AR (Augmented reality) means extending real world with computer generated elements to get more information or more immersive experience for the user. Most of these elements are meant to enhance our vision, especially in mobile AR. Mobile AR applications are greatly in demand because we take our phones almost everywhere and we rely on them during everyday life. That brings us countless situations where we can use an application with AR.

In fact, AR market grew significantly in the last years and already entered all major industry fields. It is no longer only a tool for academic use or a project for a big team of scientists. Nowadays it is heavily used in advertisement, shopping & retail, arts & games, language translation, education, military, medicine, cinematography, navigation, sport and many more.

There are a lot of companies that benefit from a very simple AR application. For example The Sunshine Aquarium in Tokyo yielded a 152% boost in ticket sales [4] based entirely as a result of the AR mobile application street guide [3], in which little penguins show visitors the way to the aquarium. The IKEA product catalog [5] mobile app that allows you to visualize furniture inside your home is very well known, but there are a lot of other companies that use this strategy. For example Mitsubishi Electric uses the AR based mobile app [6] to let customers visualize air conditioning units in various locations of their homes and maintain them. Not only it is saving money (in this case 2 million dollars) for printing costs because the product models can be stored entirely on the device instead of a traditional paper catalog, but they also recorded a 50 million dollar boost in sales in response to the success of

the app [7].

Figure 0.1: Augmented reality (*Wikipedia*)

On the other hand, numerous companies invested millions of dollars into the development of AR technology and equipment (e.g. Google Glass, Vuzix M100, Epson Moverio BT-2000, Sulon Cortex, Meta SpaceGlasses, Samsung Glass Gear, i2i iPal, Seebright, Sony SmartEyeGlass, LG GG Glass, Microsoft MG Glass, castAR, Atheer One, vrAse [8]). Chart 0.2 made by Envision IP [9] shows the biggest patent holders in AR in 2015. In comparison to today's situation, the chart is missing Apple, which acquired Germany-based Metaio GmbH [10] in 2015, which owns over 170 patents worldwide with 25 issued in the US.

Naturally, all this hype and advance in technology is forcing developers to create software suitable for these new platforms. However for now, the most popular AR mobile platforms remain Android and iOS. Both of them have big base of users and there is abundance of software development tools native or otherwise which can be used for AR development targeting these two platforms. Tractica [11] in 2015 created a chart to demonstrate how rapidly is the AR mobile market growing (picture 0.3). According to Tractica there were 292 million actively used mobile AR apps in 2015 and this number is supposed to grow to 2.2 billion by 2019, which represents approximately 76%

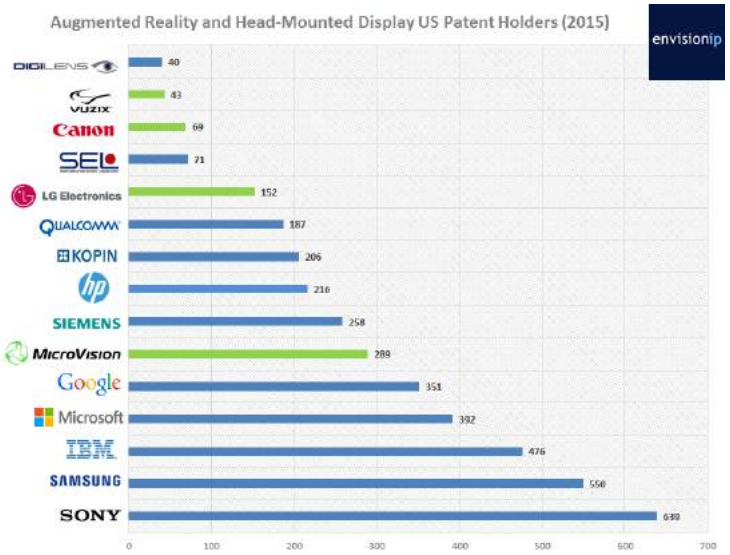


Figure 0.2: AR and HMD US patent holders (*Envision IP 2015*)

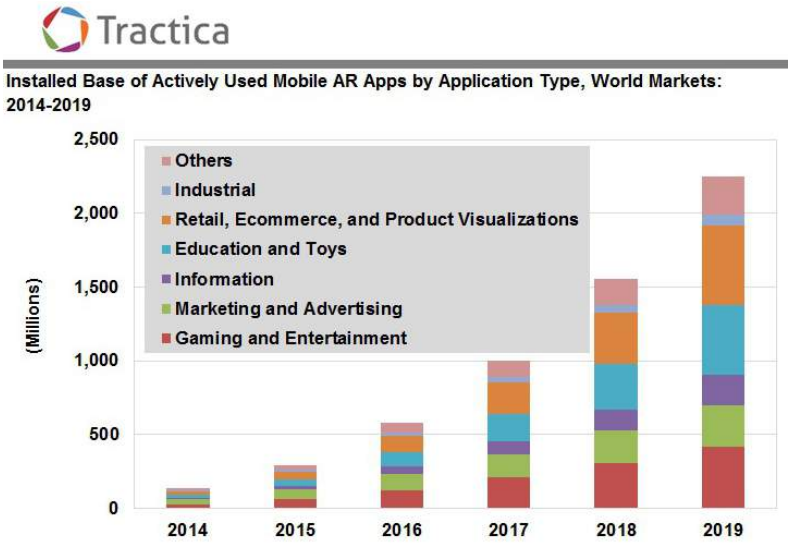


Figure 0.3: Installed Base of Mobile AR Apps (*Tractica LLC. 2015*)

annual growth rate.

Goals and Motivation

Mobile AR is not a recent concept, it was here for many decades, but new innovations introduced in the past few years (mentioned above) in combination with constant improvements in hardware are promising a good future for

mobile AR developers. Also, an important factor is an increasing acceptance by the general public, which means even more areas for AR implementation.

This thesis aims to analyze current situation in the AR technology, mobile apps and development tools. It should explain in more detail what is possible to do in today's mobile applications and what tools are most suited for their development. In the second part of this thesis, a design and implementation of such application is done and the resulting prototype is tested.

Chapter 1

About AR

A lot of people tend to get confused between AR, VR (Virtual reality) and CG (Computer Graphics) effects, especially when it comes to cinematography because there is no shortage of sci-fi and fantasy movies which would feature computer generated content. In this chapter they are defined both, VR and AR, implicitly explaining what CG effects are not. Then a more detailed explanation of mobile AR and what it consists of follows.

1.1 Real, Virtual and Augmented reality

What is the so called reality? Depending on the field, this could became a very tricky question. Luckily we can take a non-philosophic approach and say that we perceive reality through our senses, which are only electrical signals produced by our nerve cells. Which means, that as soon as our brain starts to consider a computer generated objects as part of our world, we shall take it as a reality, but we can't forget that it's still only an illusion.

For this, we categorise computer generated reality as a virtual reality. We come in contact with virtual worlds more often than we think. It's starting to get obvious that very soon we won't do without it in areas like medicine, architecture, education, product design and retailing. Just like in [15], which says that VR *"is a computer-generated environment that provides the user with the illusion of being present in that situation. Virtual reality is produced by providing feedback to our various senses: vision, hearing, movement and sometimes smell. As the user moves or acts, the image seen will change along with appropriate sound and movement."* many people agreed on two main characteristics differentiating VR from other computer generated products like movies or pictures:

- It's in **3D**. Geometric objects in virtual environments have to have some physical properties relative to their alternate world. However, they don't

necessarily resemble our world, they can have abstract non-photorealistic appearance too.

- **Interactive real-time** rendering. We are not only observers, but also participants in the course of this VR world. Especially in *immersive* VR this requires specific human-computer interfaces based e.g. on hand gestures.

(a) Polygon style VR

(b) Open world VR

(c) Indoors VR

Figure 1.1: There is endless number of different worlds in VR (*author*)

This bears a question: What shall we call it if it combines both, real and virtual worlds? Answer to this was published in 1997 by Ronald Azuma in his comprehensive survey on augmented reality [12]. He defines AR as any system that has the following three characteristics:

- registered in 3D
- interactive in real time
- combines real and virtual

Depending on the ratio of real vs virtual content, researchers proposed diversity of theories and models. The most popular being Virtual Continuum [13] created by Milgrim and Kishino in 1994. They categorize Augmented Reality as a version of Mixed Reality, where the other side of the range is called Augmnented Virtuality (picture 1.2).

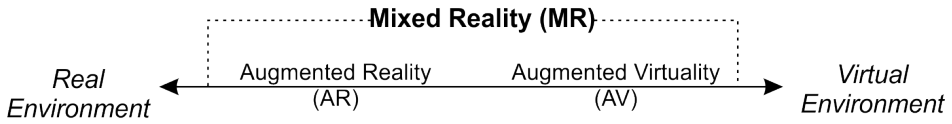


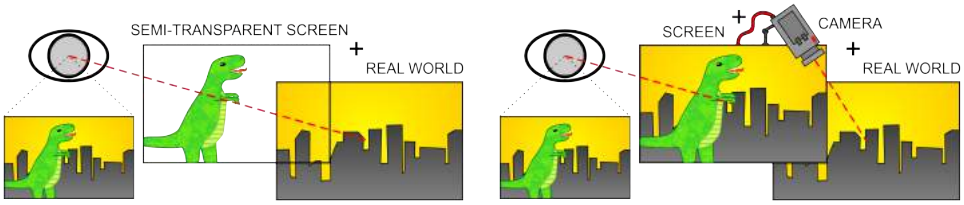
Figure 1.2: Mixed Reality in Virtual Continuum (*author*)

Important fact is, that majority of Mixed reality is targeting visual augmentation. Mostly because it is our most dominant sense (we perceive up to 80% of all impressions by means of our sight), but also because it is the easiest one to enhance. That is why the rest of this thesis is focused solely on enhancing visual perception omitting hearing and other senses.

1.2 Displaying AR

Current technology offers us two commonly used displaying methods.[18] First one is called *optical see-through technology (OST)*. This requires semi-transparent screen. Digital content is projected on the screen, while the whole surrounding area is still visible (picture 1.3a). This moves merging of real and virtual content to observers retina. Also our perception of real world is intact, which lowers the probability of physiological side effects.

Second one is called *video see-through technology (VST)*. Observer is not able to watch real world directly, but only a video of it taken by the camera and displayed on the screen. Which means that merging of the real and virtual content is happening before the data is displayed on the screen (picture 1.3b). This can cause multiple complications including physiological ones for the observer. Possible factors that can contribute to these undesirable side effects, are narrow field of view of the camera, time lag, low update rate, poor resolution and in case of stereo, a spatial stereo mismatch.



(a) Optical see-through (OST) (b) Video see-through (VST)

Figure 1.3: Most popular ways of displaying AR (*author*)

AR on mobile platforms is currently produced via VST and because this thesis is focused on mobile AR, the rest of this thesis will consider only VST

technology, counting on the unwritten rule that all today's smartphones have build-in camera.

1.3 3D Registration and Coordinate Systems

Random overlaying of the computer generated data over the camera feed is not enough. The device needs to know where it is in the world to offer us relevant extra information about our position or objects in our immediate or distant proximity. Ideal information format would include data about all six degrees of freedom (6DOF) [20]: three variables (*x, y, z*) for position and three angles (*yaw, pitch, roll*) for orientation. Many approaches exist for 3D registration and to increase precision and stable results a lot of tracking algorithms can be implemented as well, then it is not only relevant what happens in separate frames, but information is handled on frame-to-frame basis.

Best registration approach depends on the environment and nature of the augmented data. 7 basic approaches (*mechanical, ultrasonic, magnetic, optical, radio, inertial and GPS (global positioning systems)*) exist in general AR, but for the purpose of mobile AR, only 2 approaches are being realized. Mostly because other approaches are not suitable for outdoors or they require special equipment. These two approaches are called Sensor-based (section 1.4) and Computer vision-based (section 1.5). More on them in their respective sections.

Because everything is happening in 3D, it needs to have some rules to keep things in order. This mostly concerns CVB (Computer vision-based) AR. All virtual objects to be displayed on the screen are in the so called scene, in hierarchical relationships to each other. The goal is to correctly position and rotate virtual camera in the scene, so that it represents position and rotation of the mobile device (or its physical camera) in the real world. If it is done correctly, contents of video feed from physical camera are properly aligned with contents of virtual camera renderings of virtual objects and the illusion of coexistence of these two worlds is kept.

Most common way of extracting position and rotation of objects in the scene is to use Euclidean Geometry and ECS (or shortly CS standing for Euclidean Coordinate System). Scene itself has ECS called Global Coordinate System or World Coordinate System (WCS). Virtual Camera itself has its own Coordinate System (CCS) and the same goes for all virtual objects in the scene, each has a separate Local Coordinate System (LCS) with origin typically in the centre of itself. These Coordinate Systems are important e.g. in parenting of an object.

Each object in the scene has certain position, rotation and scale, these characteristics are all represented by values in object's *transformation* matrix. Because of the hierarchical relationships between objects in the scene, transformation matrices don't have values in reference to WCS, but they are

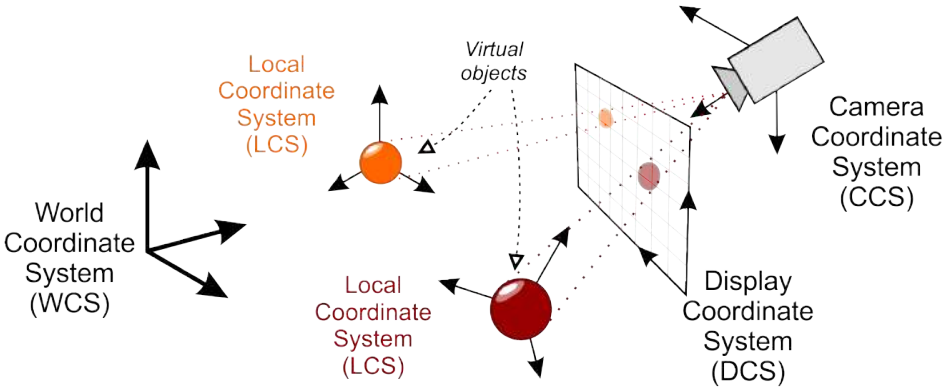


Figure 1.4: Different Coordinate Systems in 3D registration (*author*)

in reference to the LCS of their immediate parent. Which means that if Object A is child of Object B, but object B doesn't have any parent (except for the scene itself), local transformation matrix of the object B is the same as its global transformation matrix. On the other hand, global transformation matrix of Object A is a result of its multiplication with its parent's (Object B's) transformation matrix. With deeply nested objects the process is applied recursively to each parent of a parent.

After repositioning virtual camera in the scene to represent physical camera in real world and calculating global transformation matrix for each object in the scene, it's necessary to project these objects onto the camera frame and then on the screen in Display Coordinate System (DCS). Projection matrix is used for this conversion of 3D coordinates into 2D coordinates.

In the scenario where three-point perspective projection is used to project point P with coordinates (x, y, z) onto point P' (x_p, y_p, 0) on plane ($z = 0$) from Center of Projection (COP), projection matrix should look like this [14]:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ p & q & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ px + qy + rz + 1 \end{bmatrix} \equiv \begin{bmatrix} \frac{x}{px+qy+rz+1} \\ \frac{y}{px+qy+rz+1} \\ 0 \\ 1 \end{bmatrix}$$

Values p, q and r are derived from the distance between individual values of camera coordinates to P coordinates, e.g. in simplified situation where COP is located on a positive part of Z axis at distance d from the origin (picture 1.5), while deriving x_p there will be a substitution of $-\frac{1}{d}$ for r. By considering similar triangles, obtaining x_p is as follows:

$$\frac{x_p}{d} = \frac{x}{d-z} \implies x_p = \frac{xd}{d-z} \implies x_p = \frac{x}{1-\frac{z}{d}} = \frac{x}{1+zr}$$

This is basically projecting Point P onto camera frame and it's using extrinsic parameters to do so, but it's still missing part of the camera calibration

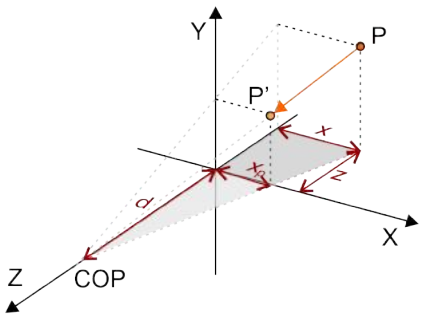


Figure 1.5: Perspective projection of a point (*author*)

process. This part of calibration is concerned with intrinsic parameters of the projection matrix. Coordinates of P' in the camera frame (not CCS) are usually not the same as coordinates of P' on the screen of the display (DCS). Each screen has specific unit range depending on the pixel resolution. If coordinates of P' on the screen are (x_s, y_s) in pixels, (s_x, s_y) is the effective physical size of the pixel and (o_x, o_y) are the coordinates in pixel of the screen center, then matrix for projection of point P' from camera frame to screen frame is as follows [19]:

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{s_x} & 0 & o_x \\ 0 & -\frac{1}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Above relationships are relying on the virtual camera being based on simplified pinhole camera model.

1.4 Sensor-based AR

Nowadays, majority of famous applications which use AR such as *Wikitude World Browser* [16] or *Yelp Monocle* [17] use sensor-based AR *"generally referred to as a GPS plus inertial AR (or, sometimes, outdoor AR system). Sensor-based AR uses the location sensor from a mobile as well as the orientation sensor. Combining both the location and orientation sensors delivers the global position of the user in the physical world."* [18]. Sensor-based system is ideal for browser applications, displaying simple (in a sense of computation) information about objects, buildings or establishments in user's location.

Actually GPS is only one example of common technology used for global tracking in reference to earth coordinate system. It's an American version of global navigation satellite system (GNSS). Other versions are e.g. European Galileo and Russian GLONASS, but GPS is the only one fully supported by majority of mobile devices. Thanks to Assisted GPS (A-GPS) direct visibility with at least four satellites is not necessary anymore. Support of a worldwide

network of servers and base stations ensures signal broadcast in previously unreachable areas like indoor environments or canyons. GPS by itself has an accuracy between 10-15 meters, but with special preparations and equipment its estimation can be accurate within centimeters. [20] Complications come in place when data from GPS needs to be used with AR system that doesn't have its coordinates in reference to earth coordinate system but in ECS instead. In that situation, conversion of latitude and longitude coordinates needs to be done. One of the options is to convert data to an ECEF (Earth-Centered, Earth-Fixed) format and to use an additional coordinate system, the ENU (East-North-Up) coordinate system (picture 1.6). One of the disadvantages of GPS registration is that if the movement of the device is too fast or too small, position reported by the GPS module can cause a lag in the application due to inaccurate or slower update.

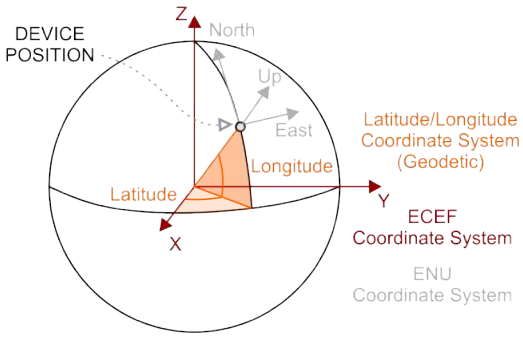


Figure 1.6: Relation of different coordinate systems (*author, based on original from [18]*)

Inertial AR is composed from orientation sensors, which can give us very precise tracking information about rotation and acceleration of the phone. In the current generation of mobile devices three types of orientation sensors are bulid-in [18]:

- **accelerometers** detect acceleration of a mobile device, it is also known as g-force acceleration. Most common model is a multi-axis which registers acceleration in 3 axes: *pitch, roll, tilt*. They were the first inertial sensors to be build into the mobile phones. They are very cheap, but rather inaccurate.
- **magnetometers** can detect the earth's magnetic field. Most of the time they work as compass, measuring the magnetic field in three dimensions. Analogicaly as in the case of a compass, the result gets easily corrupted by magnetic or metallic objects in their close proximity.
- **gyroscopes** are more accurate in the beginning than accelerometers or magnetometers, but their precision decreases with increasing time. They

measure angular velocity using the Coriolis Effect. They are a multi-axis miniature mechanical system (MEMS) which uses vibrating mechanisms.

Different manufacturers produce different quality of these sensors. Lower quality often causes bad side effects of noise, drift or inaccuracy in the measured data and forces application to move or rotate the virtual content without an actual movement of the device. To improve this, a technique called sensor fusion is used, where shortcomings of one sensor are (to certain degree) balanced by results from other sensor. In the picture 1.7 you can see a diagram of Paul Lawitzki's algorithm [21] which is only one of many ways how data from these sensors can be merged.

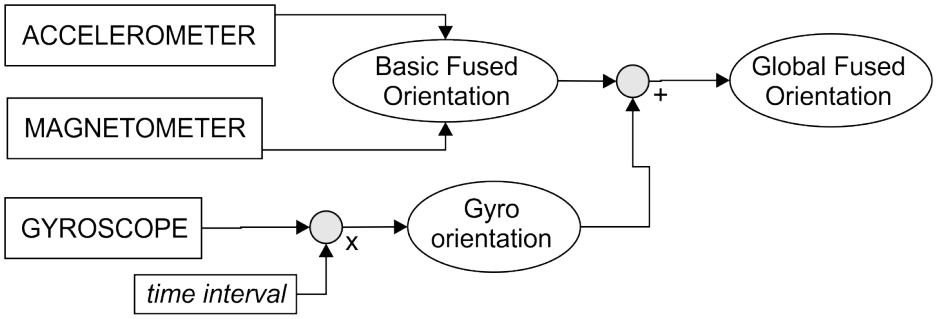


Figure 1.7: Paul Lawitzki's algorithm for sensor fusion (*author, based on the original in [21]*)

1.5 Computer vision-based AR

Idea behind CVB AR is acquiring data and information about surrounding objects from the video stream taken by camera. It employs numerous algorithms like edge, corner or blob detection to recognize objects in the received video frame. Object that algorithms are looking for, is called target and it can be represent by an image, object, group of images or by anything else that is distinguishable with camera. CVB 3D registration is more robust, but it takes a toll on processing. The least computationally expensive targets are 2D image markers with high contrast in color like QR codes and the most costly ones are 3D objects. Nevertheless the tracking pipeline is roughly the same, starting with camera frame grabbing and ending with the final 6DOF pose. Picture 1.8 represents a tracking pipeline for markerless NFT (Natural Feature Tracking) AR (more in subsection 1.5.2).

(A) First a new frame is acquired, then (B) reliable and strong features must be detected in each new frame. Detected features are contained within a small image patch with constant size, e.g. 8x8 pixels. (C) Afterwards,

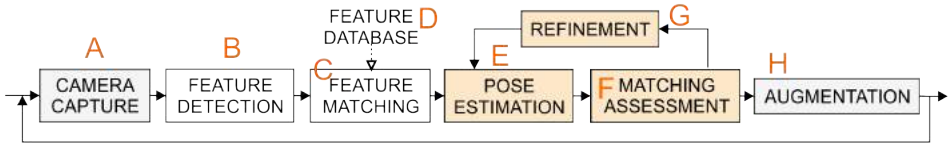


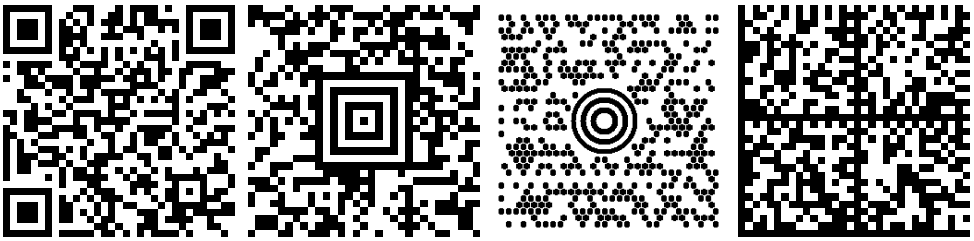
Figure 1.8: Tracking pipeline for Computer vision-based AR (*author*)

those features have to be matched against a previously created feature database holding reference features of the target to be tracked. Common matching methods comparing values of pixels are Sum of Squared Differences (SSD) and Sum of Absolute Differences (SAD). (D) Descriptors must be calculated for all camera features used for matching. (E) If enough correspondences between camera and database features can be found, then the associated 6DOF camera pose may be calculated (or any other 3D registration model). (F) Typically some feature correspondences are faulty and must be removed during the pose determination. (G) The pose can be refined using additional feature correspondences e.g. those found using the just extracted pose. (H) 6DOF pose or other model is forwarded to the underlying system and whole pipeline will restart for the next frame. Information from the previous camera frame, e.g. the previous pose, can be used to improve tracking performance and accuracy in the frame-to-frame tracking. [22] View point variation, illumination, occlusion, scale, deformation, background clutter and intra-class variation are in general biggest problems that CVB algorithms need to face.

1.5.1 Marker based

In the past, it was of utmost importance to use computationally efficient algorithms to maintain a constant frame rate at 30 Hz, which leaves only 33 ms for all the computations each frame. Therefore, first image targets where as simple and distinguishable as possible. Fiducial markers (also called Frame markers) fit the description. They are usually defined in black and white or on a grayscale level. Black and white markers are most feasible for detection with bad lighting conditions. Process of recognizing a marker in captured camera frame image is combination of simpler algorithms like edge or line detection, starting with binarization of the frame image via thresholding, which is taking advantage of high contrast in the marker image. This simplifies the process of tracking pipeline. Template and 2D-Barcode markers are the two types of markers used in this technique. 2D-Barcode marker also known as data marker or ID marker, consists of black and white data cell holding information about object of interest or landmarks. Good example of 2D-Barcode marker is QR (Quick Response) code. [23] QR code is capable of encoding all types of data: alphanumeric characters, Kanji (Japanese), Kana (Japanese), Hiragana (Japanese), symbols, binary, and control codes. Its error correction ability

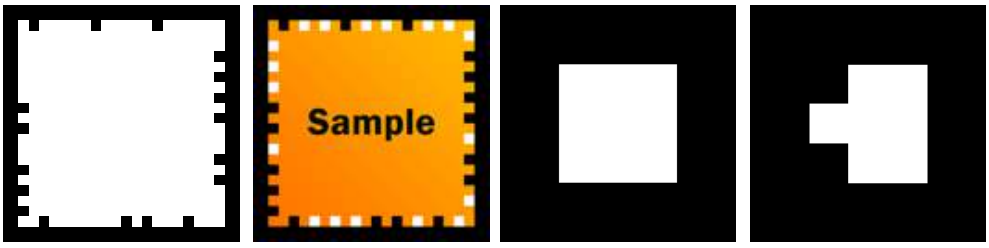
can restore data from a sample that is dirty or damaged on up to 30% of its area. Together with other great features like readability from any direction, this makes QR code a suitable candidate for industrial marking.



(a) QR (Quick Response Code) (b) Aztec Code (public domain) (c) Maxicode (US Postal Service) (d) Datamatrix (Microscan Systems)

Figure 1.9: Various 2D-Barcode markers used in industrial systems to carry data, these four say *"Bachelor thesis: Augmented reality game for Android, 2016"* (*author*)

Frame markers used in prevailing AR SDKs (Software Development Kit) are template markers usually customized to fit special user needs of that particular SDK. Template marker is black and white marker composed of a single image inside a black block border. Insides of the border have specific design to indicate ID of the marker. Some companies choose to make customizable markers like Vuforia with their fixed set of 512 different IDs (picture 1.10a), but some focus on the versatility like ARToolKit [25]. ARToolKit offers set of markers with 3x3 matrix of squares, which yields 64 rotationally unique patterns that are associated with predetermined identifiers (IDs) (picture 1.10a). If 64 is not enough, the user can choose markers with 4x4, 5x5 or 6x6 matrices and respectively 8192, 4194304 or 8589934592 IDs. They also have a combination of marker and markerless technique, when the insides of the black border are completely designed by the user.



(a) Vuforia frame marker, ID 0 (b) ARToolKit frame marker

Figure 1.10: Frame (fiducial) markers can vary depending on the SDK (*Vuforia, ARToolKit*)

1.5.2 Markerless

This method is very powerful. Not only target doesn't have to have limiting characteristics like black and white color or block border, but it also doesn't have to even be known beforehand, because the user can define the target during game-play, although it is helpful if it follows certain rules like no repeating patterns, high contrast, no reflective surface, etc. In comparison to marker-based approach it requires significantly more in terms of processing. If the target is a 2D image, then it is also called Natural feature-based AR (NFT). Each target needs to be first added to the database in form of a set of feature descriptors.

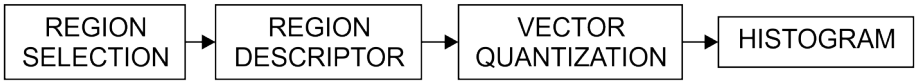


Figure 1.11: Process of image descriptors creation in Bag-of-Words model (*author*)

Best way how to describe this process is to use bag-of-words model (picture 1.11). First "interest regions" (regions with highest potential occurrence of unique features) are detected, then they are described by feature descriptors, descriptors are quantized into "visual words" and each target image is represented as a histogram of visual words.[26]

Most popular interest region detectors are [27]

- Harris detector
- Difference of Gaussians (DoG)
- Laplacian detector
- Scale Saliency
- Maximum stable extremal regions

Here is an explanation of Harris detector to demonstrate the work-flow of interest region detectors. This algorithm is extracting corners from the target image. Corners are useful, because they are rotationally invariant and they have an explicit center point (picture 1.12).

First it calculates change of intensity E(u,v) (equation 1.1) for the shift I(x,y) with window function of w(x,y), which has Gaussian layout for significance of point value (equation 1.2). By deriving E(u,v) into a form with matrix M (by using Taylor's expansion and ignoring higher order items), it can calculate a pair of eigenvalues (λ_1, λ_2) for M. Then it calculates corner response R (equation 1.3) and based on fixed threshold it determines whether

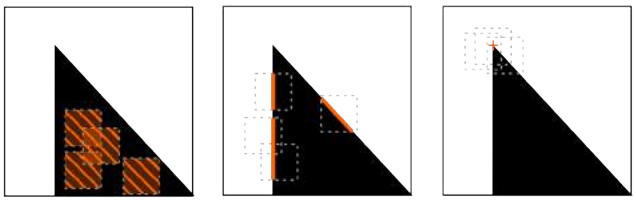


Figure 1.12: Regions of interest for flat, edge and corner area of an image (*author*)

it is a corner, or not. Repeating this for each point of the marker image, it finds points with large R and picks local maxima from among them.

$$\begin{aligned} E(u,v) &= \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \\ &= \sum_{x,y} w(x,y)[I_x u + I_y v + O(u^2,v^2)]^2 \\ &\cong [u,v]M \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned} \tag{1.1}$$

$$w(x,y) = exp(-\frac{(x^2+y^2)}{2\sigma^2}) \tag{1.2}$$

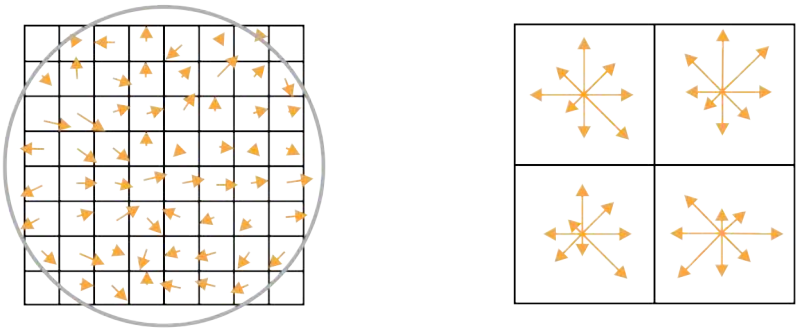
$$R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2, \qquad 0.04 \leq k \leq 0.06 \tag{1.3}$$

After selecting interest regions (interest points in case of Harris detector), they need to be described by one of the descriptor formats. Most common descriptors are [28]

- SIFT (Scale Invariant Feature Transform)
- Steerable filters
- Spin images
- PCA-SIFT
- Gradient Location and Orientation Histogram(GLOH)
- Shape context
- Geometric blur

For example the SIFT descriptor algorithm computes the gradient magnitude and orientation at each image sample point in a region around the keypoint location selected by the interest region detector in the previous step.

Then they are weighted by a Gaussian window (indicated by the overlaid circle in picture 1.13a). After that a orientation histograms are created (in the picture 1.13b only 2x2 descriptor array is computed from an 8x8 set of samples). The most complex complete SIFT descriptor is created from 16x16 samples and has 8 orientations for each 4x4 histogram array cell which means 128 dimensions. [26]



(a) Image gradients

(b) 2x2 keypoint descriptor

Figure 1.13: SIFT: (a) part of the image with magnitude and orientation of its gradients (b) keypoint descriptor created from neighbouring gradients (*author, based on the original in [26]*)

When application is running, matching of current descriptors extracted from camera frame against all descriptors in the database by brute force is not an option. Since for each frame usually 50-100 features have to be matched against 5000 features in the database SIFT implementation uses a k-d Tree together with a Best-Bin-First strategy to minimize the number of comparisons. Matched features determine target's position and rotation in the camera frame (picture 1.14) and 3D registration begins.

1.6 Comparison

Obviously each method for 3D registration has its pros and cons. There isn't an ultimate solution and that is why developers have to first analyse which approach is the most suitable for their issue's domain. Because each approach is lacking in some sense (image capturing capability, sensor information accuracy, data availability on networks, energy consumption, computation costs, sensitivity to lightning conditions, etc.), it is very common to combine 2 or 3 of them to achieve the desired results. In table 1.1 is a comparison of some of these techniques for mobile platforms. [29] *Range* column represents size of the region that can be tracked within. *Setup* column has amount of time for instrumentation and calibration. *Precision* column shows granularity of a single output position. *Time* column represents duration for which useful

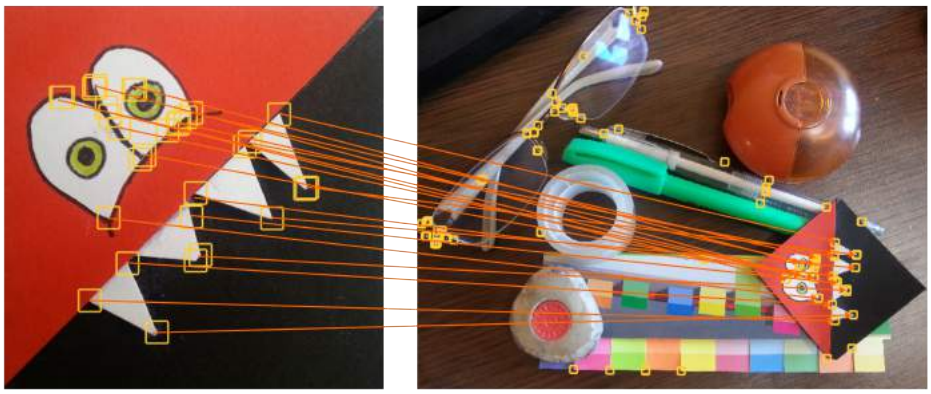


Figure 1.14: Corresponding features from image target in the databese to target in the camera frame (used Harris corner detector for interest region detector) (*author*)

tracking data is returned (before it drifts too much) and *Environment* column shows where the tracker can be used (indoors or outdoors).

Table 1.1: Comparison of common tracking technologies.

Technology	Range(m)	Setup(h)	Precision(mm)	Time(s)	Env
Marker-based	10	0	10	∞	in/out
Markerless	50	0-1	10	∞	in/out
GPS	∞	0	5000	∞	out
WiFi	100	10	1000	∞	in/out
Accelerometer	1000	0	100	100	in/out
Inertial	1	0	1	10	in/out

Chapter 2

Mobile AR applications

Before anyone begins to implement a new technology into their designs, they need to make sure that it has its use in there. So the question would be *what can AR bring into mobile applications?* Or even better question would be *what can **only** AR bring into mobile applications?* In section 2.1 are current successful applications that have a very clear answer to that question. There are intentionally no AR games, first of all because I personally don't consider the ones on the market to be on the same level as the applications mentioned there and secondly, because analysis on AR games is in the section 2.2. That section is primarily focused on the question *will user understand it?* Because everyone can have good idea, but that is rarely enough. Especially when there is a new technology that is not very well known. Most people wouldn't know how to control the application, which means designers need to test and anticipate their action or rely on established design patterns to make appropriate adjustments in the user interface (UI).

2.1 Few use cases of AR in mobile apps

There are certain prevailing types among existing successful AR applications. The following examples are trying to illustrate the most dominant trends. In addition to these trends there is a continual interest in using AR in educational applications, but they are usually more content oriented than concept oriented.

2.1.1 Augmented reality browser

Wouldn't it be great if product could advertise itself? Imagine you are in the DVD shop and there are 3 different movies that you are trying to choose from, but you can't decide. Wouldn't it be just sweet if you could quickly check out their trailer or read some critique? Of course you can search for them on the Internet yourself, but that's just tedious and majority of consumers wouldn't bother doing that. But what if you could just activate an app on your phone,

aim the camera on the from page of the cover and get all the content related to the movie? That certainly doesn't sound like a big hassle, especially if the same app worked not only for movie covers, but for other things too, like grocery products, magazines, brochures, logos etc. Applications like this already exist. Right now two most popular augmented reality browsers are *Wikitude* [16] (on Android and iOS) and *Layar* [30] (on Android and iOS). They specialize in AR presentation of campaigns, projects, promotions, games and personal content. They can provide consumers with videos, animations, 3D models, Facebook and "Buy Now" buttons. Usually their strongest suit is computer vision-based AR, but they often integrate sensor-based AR as a secondary source of information or they are all-together a mixture of AR browser with crowd-sourced location guide.

2.1.2 Crowd-sourced location guide

Built on social media exposure and the need of marketing for small local businesses, a lot of applications can provide user with lists of restaurants, shops, sigh-seeing spots and services with their according ratings & reviews just based on user's location (sensor-based AR). Naturally the biggest appreciation comes from people who just moved in and are not familiar with their new environment or tourists, who are deliberately searching for fresh ideas to try. It is not a surprise that credibility of the information displayed by these apps is quite questionable, because of all the bad effects social media can have, but it can also often prove itself more useful than any other certified guide book, which unlike these apps can become outdated. *Yelp* [17] (on Android and iOS) and *Field Trip* [31] (on Android and iOS) are just two examples of crowd-sourced location guides.

2.1.3 Face recognition

Since the rise of social media a lot of people started to *upgrade* their profile pictures with additional computer generated content like emoticons or so-called stickers. It went even further and now its completely possible to make a video calls wearing a virtual hat on your head or a moustache on your face. *Mybrana* [32] (on Android and iOS) has such features and many more for real-time picture editing, which can be then instantly shared by their social network. It's probably not necessary to mention that it is a computer vision-based AR recognizing face patterns.

2.1.4 Text recognition

Anyone who ever travelled in a foreign country, where they use an unfamiliar language, knows how frustrating it can be to translate signs and information tables, especially when they use different writing system or alphabet set. That

is why, one of the best apps with computer vision-based text recognition is *Google translate* [33] (on Android and iOS), which enhances user with a translation of that text.

2.1.5 Outdoor guides

Applications showing additional information about surrounding environment or sky using sensor-based AR (sometimes supported by computer vision-based AR). 3 great representatives of this type of AR applications are *Theodolite* [34] (on iOS), *Star Walk* [35] (on Android and iOS) and *Sun Seeker* [36] (on Android and iOS). *Theodolite* is a multi-functional viewfinder. It includes compass, two-axis inclinometer, rangefinder, GPS, maps, tracker and many more, which makes it a great app for any outdoor adventurer. *Star Walk* is a fully packed encyclopedia about celestial bodies and their constellations. It serves faithfully to any astronomy junkie or stargazing romanticist. Last but not least *Sun Seeker*, is an app showing information about sun and daylight sky. It has data about anything from solar paths, its hour intervals, its equinox, winter and summer solstice paths to rise and set time. It is an ideal tool for photographers, gardeners or architects.

2.1.6 Traffic navigation and alerts

Most people would separately buy a GPS navigation and permanently mounted it in their vehicle, but nice thing about using a navigation app from phone is that it can also incorporate other functions of the phone, e.g. camera. *iOnRoad Augmented Driving* [37] (on Android and iOS) monitors objects in front of the driver in real-time (computer vision-based AR), calculates the driver's current speed (sensor-based AR) and as the vehicle approaches, alerts (with an audio-visual warning pop-ups) about a possible collision, allowing the driver to brake in time.

2.2 Mobile AR games

Because at least part of the mobile AR games is from real world, the other part, augmented part, is based on Reality-Based Interactions (RBI), so that it can mix as seamlessly as possible. Publication [38] deals with design patterns for RBI in mobile AR games. These design patterns leverage different embodied skills (naïve physics, body awareness and skills, environmental awareness and skills, social awareness and skills)[39]. Embodied skills are part of embodied cognition, which is defined in [38] as *"Embodied cognition posits that our understanding and interpretation of the world around us is rooted in our, often unconscious, experience of our bodies. Through bodily interactions with the physical world, we develop "image schemas" that encode the structure and relationships learned in these encounters. These schemas are then adapted*

and applied to future experiences through the cognitive process of "metaphorical mapping."" In this case, substitution for physical world would most likely be mobile games without AR or board games in real life. Nevertheless in [38] they derive 9 very comprehensive pre-patterns (9PP) for mobile AR games (table 2.1), which should anyone designing an AR mobile game follow or at least keep in mind.

Table 2.1: 9 Pre-design patterns for AR mobile games [38]

Title	Meaning	Embodied skills
Device metaphors	Using metaphor to suggest available player actions	body a&s, naïve physics
Control mapping	Intuitive mapping between physical and digital objects	body a&s, naïve physics
Seamful design	Making sense of and integrating the technological seams through game design	body a&s
World Consistency	Whether the laws and rules in physical world hold in digital world	naïve physics, environmental a&s
Landmarks	Reinforcing the connection between digital-physical space through landmarks	environmental a&s
Personal presence	The way that a player is represented in the game decides how much they feel like living in the digital game world	naïve physics, environmental a&s
Living creatures	Game characters that are responsive to physical, social events that mimic behaviours of living beings	body a&s, social a&s
Body constraints	Movement of one's body position constrains another player's action	body a&s, social a&s
Hidden information	The information that can be hidden and partially revealed can foster emergent social play	body a&s, social a&s

The following 10 games demonstrate both, good and bad, application of these 9 pre-patterns. They are roughly ordered in decreasing ratio of computer vision-based AR vs sensor-based AR and they all have some unique features either in the way they use AR or in their game-play and that is the reason why they were chosen.

2.2.1 AR Basketball

Available on iOS [40]. Simple basketball game (picture 2.6a), where player can throw a ball using swipe gestures. It requires player to print the image target for computer vision-based AR beforehand. As said before, it is a simple

game, so once the player learns the best combination of settings it is almost impossible to miss, but thanks to the well polished visual and sound effects it's still a good game for distraction. As far as 9PP go, World consistency, Landmarks and Personal presence are very strongly integrated into the game-play, because it seams as if the ball was always thrown from just below the device (metaphor to throwing a ball in real life) and as it follows gravity rules, it is aimed at a hoop, the only (except for the ball) computer generated object in the game.

2.2.2 AR Defender 2

Available on iOS [41]. This game takes an old tower defence concept to a new level. Game is playable in a pure VR but player can toggle AR whenever he desires. That sometimes proves to be essential for the game-play, especially when it is impossible for the player to use the image target because of bad lighting conditions or a lack of image target. This game has two modes, one for single player and the other one is for multiplayer via WiFi connection up to 4 players. Control mapping from 9PP is integrated through ray-casting e.g. when building and placing a new object in the scene. Landmark is obviously the main tower that player is trying to protect and a combination of Personal presence and Living creatures are 6 hero avatars, from which can player choose one to play with. Also Seamful design is very well managed through constant written feedback that informs player about what is happening and what should be done if the image target is lost or found.

Figure 2.1: AR Defender 2 (*[41]*)

2.2.3 Tilt Augmented Reality

Available on Android [42] and game requires an Android Wear smart-watch as a target. This game needs to be installed on both, smartphone with Android and smartwatch with Android. During the game-play player has to tilt his wrist (with the watch on) back and forth and up and down to roll the sphere in the scene to pick-up boxes without falling. It is a unique version of Control mapping and it is implicitly relying on the range of Body constraints and World consistency's gravitation.


Figure 2.2: Tilt Augmented Reality (*[42]*)

2.2.4 Crayola Color Alive

Available on Android and iOS [43]. This is not a traditional game by itself. First, children have to color in pictures in their physical coloring books, then they can use this app to bring their drawings into life with AR by aiming their device's camera at the colored page. Computer vision-based algorithms recognize which image design (black and white picture) was used in that particular picture and then it extracts color and texture of individual regions to use it as a texture for 3D models (picture 2.6b). These 3D models usually have some animations, special effects or mini games added to them. This customization of 3D model is one of the strongest representations of enhancing a relationship with AR Living creatures (9PP).

2.2.5 AR Soccer

Available on iOS [44]. Once again simple concept but very well executed. Player gets to choose between easy, medium or hard mode and then the game can start (picture 2.6c). Device metaphors (from 9PP) deliver the experience of kicking the ball with your foot just like in physical world except for the gravitation, which is shifted to lead to the bottom of the screen instead of the center of the Earth. Computer vision-based algorithms use edge detection to differentiate the foot from the background that is why, it is recommended to play it over a solid colored background, preferably white, otherwise it can feel sort of glitched and unresponsive. In this case Personal presence is fulfilled to the highest level because player is actually playing with and seeing his own foot.

2.2.6 A.R. Warriors

Available on Android and iOS [45]. This is a computer vision-based AR using a user defined image target which means that player doesn't need to print anything, he just takes a picture of the background when he is prompted to do so by the game and the Computer generated content takes that target as a center of the scene. For a more stable tracking this game uses also sensor-based AR namely inertial sensors. Unfortunately that is the only interaction with AR that user can get from this app because everything else is controlled through UI buttons. Pleasant feature is that each Living creature (9PP) in the game has a unique set of attacks, damage animations and special effects.

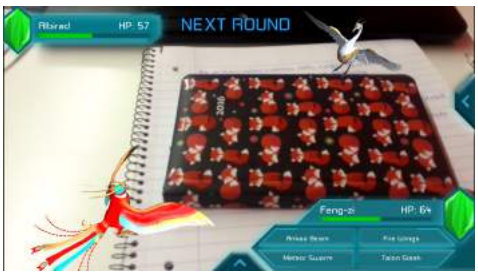


Figure 2.3: AR Warriors (*author*)

2.2.7 Paparazzi

Available on Android [46]. The goal of the game is to earn money by taking paparazzi pictures of the main game character. As player gets closer and snaps more pictures of this superstar, the character will become more and more angry until finally, he "jumps" on the players phone to break the camera (picture 2.6d). Then the player needs to try to shake him off, otherwise the camera gets "damaged". This game is a specimen of Seamful design (9PP). The transition between computer vision-based AR (NFT of one dollar bill target or printed image target) and sensor-based AR (accelerometer) is great plus player really experiences a feeling of being attacked by the character.

2.2.8 AR Invaders

Available on Android and iOS [47]. This game is using sensor-based AR (inertial sensors), although it pretends to use the computer vision-based AR too but camera stream is just used as a background image for the game. It is a simple FPS (First Person Shooter) with an alien forces theme. It has two settings, one is 180° view, the other one is 360° view and it is also possible to play it in multiplayer mode. The Device metaphors (9PP) are very well represented with 2D screen overlay of HUD (head-up display) with crosshair in the middle and the buttons for shooting on the sides.

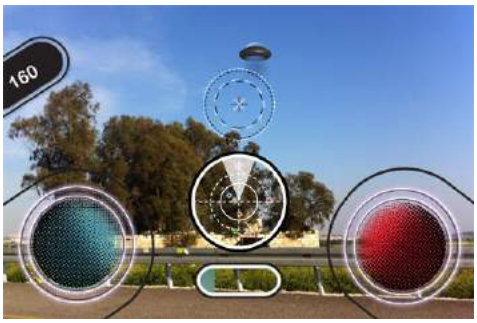


Figure 2.4: AR Invaders (*[47]*)

2.2.9 Ingress

Available on Android and iOS [48]. Ingress is a massively multiplayer online sensor-based AR (GPS) game with players all around the world (picture 2.6e). Playing this game is a good way to discover new places, hot spots and to find inspiration to explore. The game has a complex science fiction story with a continuous open narrative. Each player is an "agent" belonging to one of the two factions "The Enlightened" or "The Resistance" that the player base is divided into. These factions compete against each other by protecting or destroying game objects (portals) and information. Each agent also has a profile page which contains a wide variety of information, including the agent's name, current level, earned badges, completed missions and a long list of stats. The most attractive part of the game is probably the ever-growing community of players and events that are regularly done all around the world. Looking at 9PP the most strikingly represented one is Landmarks followed by Personal presence.

2.2.10 Pokémon GO beta

Figure 2.5: Pokémon GO (*[49]*)

In development [49], only closed Beta testing available in some countries, expected release for Android and iOS in 2016. Long anticipated mobile AR game from the same authors as the successful Ingress game (about 30% of the portals from Ingress have been turned into Pokestops). This is expected to be a big hit when it comes out thanks to the Pokémon franchise and hopefully it will increase general awareness of mobile AR applications. For now it seems that it will use only sensor-based AR (mainly GPS) and the camera stream will function only as a background if the player decide to toggle AR on (it's possible to stay visually only in VR). From 9PP it soundly integrates Landmarks and Living creatures interactions.


(a) AR Basketball

(b) Crayola Color Alive

(c) AR Soccer

(d) Paparazzi

(e) Ingress

Figure 2.6: mobile AR games (*[40]*, *[43]*, *[44]*, *[46]*, *[48]*)

Chapter 3

Mobile AR development

There isn't an ultimate tool for AR applications. Depending on the context of the application developer has to consider multiple approaches. In case of simple content browsing functionality, an end-to-end branded app solutions like Aurasma [50], Blippar [51], Layar [52], Wikitude [53] and many more are the most convenient choice. For big number of AR content entries an AR content management systems are usually necessary. Augment [54], Blippbuilder [55], Layar Creator [56], Webcam Social Shopper [57] and others are web based content platforms that offer managing and building geolocations, NFT targets or 3D models for AR experiences.

However creating an AR game most of the time requires more functions and bigger freedom for modifications. In the following section (3.1) there is a selection of 6 tools that can all be used for both, Android and iOS platform development. As mentioned before depending on the criteria for AR app this set of tools could be different, this particular one is selected based on the likeliness of being used for mobile AR game development, on the number of available tutorials and community support and on the versatility of the features they offer, in no particular order.

3.1 Tools for mobile AR

3.1.1 ARToolKit

ARToolKit [25] is an open source library (under dual-license: GPL, commercial) for creation of AR applications. It was designed by Dr. Hirokazu Kato in C language, ported to many different languages and platforms like Android, Flash or Silverlight and is very widely used in augmented reality related projects.

3.1.2 Vuforia

Vuforia [1] is an AR software development kit and it enables AR developer to create interactive AR applications supported by Qualcomm AR platform. Vuforia can run on Android, iOS and Unity 3D platforms. Its computer vision system has a lot of features including target tracking (marker, image, object, text), virtual buttons, Smart Terrain^{TM} and Extended Tracking. Target database is stored either locally on the device or in the Cloud.

3.1.3 Mataio

Metaio [10] is an AR software development kit. It was acquired by Apple in 2015 which stopped taking new customers and so far there are only assumptions about Apple's intentions with Metaio's future but because of its spectrum of features it earned its place in this selection. Metaio SDK supports among others 2D and 3D target tracking, face detection, gesture detection, SLAM and location tracking. It can be used to develop AR apps e.g. for Android, iOS, Windows PC, Google Glass or using Unity. Target database is either local or in the cloud.

3.1.4 D'Fusion

D' Fusion [58] is a patent technology of Total Immersion to design and deploy AR applications. It supports 2D and 3D target tracking, finger pointing and face detection. It is suitable for deployment on various platforms including Android and iOS. It is considered as one of the best SDK for fast recognition on live video streams.

3.1.5 ARLab

ARLab [59] currently offers two AR SDKs for Android and iOS AR development. With AR Browser SDK user can add and remove POIs (Point of interest for GPS based AR) independently from the scene in real time, interact with them and perform actions on them. Image Matching SDK supports recognition and matching of any image targets without any connection to the Internet for more than 1000 images loaded in pools of 50-60 images from local resources or remote URLs. ARLab is also preparing to launch Image Tracking SDK, Object Tracking SDK and Virtual Button SDK.

3.1.6 Catchoom

Catchoom's CraftAR [60] offers native development of AR in SDKs for Android and iOS platforms or plugins for Unity and Cordova. CraftAR supports storing of image targets on both, cloud for very large collections and on-device for hundreds of images (with no Internet connection required).

3.2 Detailed comparison

Each approach in AR requires specialized AR software. It can include toolkits, SDKs, browsers for capturing and rendering content, etc. This detailed comparison is focused on computer vision-based AR because the difference in optimization and effectiveness of algorithms in this approach is often what makes developers choose one over another. Sensor based AR applications on the other hand are usually developed with libraries accessing device's sensors in native SDKs or it can be easily integrated with third-party tools.

Availability of few basic computer vision-based features in SDKs from 5 different companies is in the table 3.1 (all these SDKs have Free and Commercial options except for D'Fusion which has only Commercial SDK version). If the feature isn't marked with "✓" it means it wasn't in the Free version or listed in the offered features of Commercial version.

Table 3.1: Comparison of computer vision-based features in AR SDKs.

	Vuforia	Metaio	D'Fusion	ARLab	Catchoom
Marker based	✓	✓	✓	✓	✓
NFT	✓	✓	✓	only recognition and matching	✓
3D Object tracking	✓with optimization for cylinder and box	✓	✓		
Face tracking		✓	✓	✓	
Visual Search	✓up to 100 locally and cloud	✓up to 100 locally and cloud	✓up to 500 locally	✓1000+ in pools of 50-60	✓up to 100 locally and cloud
Content API	✓with Vuforia Cloud	✓OpenGL support, in-house 3Drenderer			✓

In 2014 a very comprehensive study [61] was done with AR frameworks (ARToolKit, Vuforia, Metaio, D'Fusion, ARLab, Catchoom) on Android platform displaying strong and weak points of each framework's markerless detection. Similar research can be found for iOS platform in Master's Thesis [62] from 2012.

In [61] they were extensively testing markerless detection with different environmental criteria (e.g. light intensity, visible target area, distance) and

target criteria (e.g. contrast ratio, size, aspect ratio) and reviewed available performance optimization and additional usability (e.g. face tracking, text detection, multiple target tracking, extended tracking). Then they proposed use case scenarios for mobile AR applications and deduced recommendations for the most suitable framework based on the measured data in previous tests (table 3.2).

Table 3.2: Recommended frameworks for different use case scenarios

Scenario	Framework
Interior Design App indoors, various viewpoints and distances, occluding objects, using more than one target for better placement, extended tracking	**Metaio**, Vuforia
Magazine App indoors, image grayscale, contrast, size, visibility, images printed on different materials, text detection	**Metaio**, Catchoom
Bus Shelter App outdoors, sudden light changes, image distance, deterioration, visibility, dynamic background, fast moves	**Vuforia**, Catchoom
Supermarket Promotions App outdoors, multiple target tracking, 2D or 3D target, damaget target, need of flash	**Vuforia**, Metaio
Tourist Translator App outdoors, text and background contrast, text distance, flickering, deterioration, text on different materials	**Metaio**, Vuforia
mCommerce App outdoors, face recognition, stable tracking for fast moves and changing in light intensity, switching to front camera	**D'Fusion**, Metaio

Chapter 4

Design my own game

For the purpose of this thesis I decided to design and implement a game incorporating computer vision-based AR. I mostly wanted to demonstrate how real world 3D objects and computer generated 3D objects can coexist in one small-scale scene and sensor-based AR is not really suitable for that. In addition CVB AR unlike GPS doesn't require any connection which is desirable because part of this game should be playable offline.

4.1 Recommended design concepts for CVB AR applications

When it comes to CVB mobile AR or CVB handheld AR in general, there are certain things which if considered carefully can facilitate UI interactions. Some of them are results from the analysis (chapter 1.5) and some of them are from mobile design concepts.

- **Support one-hand interaction** as much as possible
- Consider the **natural viewing angle**
- Make sure **at least one tracking surface** is in view or implement counter-measures
- **Do not tire players out physically** unless that is the goal
- **Do not encourage fast actions**

Of course there is even more suggested design concepts in mobile UI, but one that is strikingly different for AR UI is that users without any experience with AR will most likely hold their devices in their right hand, (usually) the dominant hand, to make sure of stable position of the device with good angle and operate with left hand, which is the opposite from normal scenarios (hold in the left hand and operate with right hand). This is especially important

for orientation of the screen because you want to prevent users from covering their cameras with fingers.

4.2 Game system requirements

Even without story and context of the graphical content there are Functional and Non-Functional requirements that arise for this mobile CVB AR game with two modes: offline single player and online multiplayer mode. There are also requirements covering expected feature of any more complex game (e.g. data persistence, multilingualism). Requirements caused by story and context of graphical content are in this case simply categorized in Functional requirements under *System for creation and deleting of game objects*, *Gameplay mechanics* and *Minigame gameplay mechanics*.

4.2.1 Functional requirements

Functional requirements define specific behavior or functions not in any particular order. They are marked *compulsory* or *optional*, depending on whether they are required by the thesis assignment and also they are marked *implemented* or *not implemented*, depending on whether they were implemented in the prototype or not. If they are not implemented, prototype's system design is prepared for their future implementation without any radical changes.

RF1 **compulsory, implemented**
3D Registration and overlaying of computer generated content (3D objects)

RF2 **optional, implemented**
Active interaction with 3D Objects

RF3 **compulsory, implemented**
Gameplay location independent (both indoors and outdoors)

RF4 **compulsory, implemented**
Gameplay without WiFi connection

RF5 **compulsory, not implemented**
Gameplay with WiFi connection

RF6 **compulsory, implemented**
AI for offline mode (without WiFi)

RF7 **compulsory, not implemented**
Networking and match making server for multiplayer mode (with WiFi)

RF8 **optional, implemented**
Data persistence for continuous playing

RF9 **optional, implemented**
Dynamic language selection

RF10 **optional, implemented**
Audio (music and sound effects) on/off

RF11 **compulsory, implemented**
System for creation and deleting of game objects

RF12 **compulsory, implemented**
Gameplay mechanics

RF13 **optional, one implemented**
Minigame gameplay mechanics

RF14 **optional, not implemented**
Connection to social networks

4.2.2 Non-Functional requirements

Non-Functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviors. They address e.g. usability, performance, supportability, etc. In this case most of them are specified by Vuforia, which I've chosen as an AR framework based on analysis in section 3.2 (more on that in section 5.1).

RNF1 Portable to all Android devices (mobile, tablet) and their screen sizes

RNF2 Support for Android 4.0.3 *Ice Cream Sandwich* and higher

RNF3 OpenGL ES 2.0 on Android (with addition of Metal (iOS 8+) for possible futute deployment on iOS)

RNF4 Back camera and good visibility and lighting conditions

RNF5 Support FBX format for 3D models with animations

RNF6 WiFi connection and data saving if connection fails

RNF7 Sustain at least standard minimum frame rate for smooth animations

RNF8 Response time kept at minimum

RNF9 Sufficient place in real world to play the game (enough space, stability, etc.)

RNF10 Enough internal memory space for installation (roughly 60MB)

4.3 Description of the game world

Now follows a plain description of the game world and its background story disregarding everything from technology used in the implementation to game mechanics and concepts used to make this game playable.

4.3.1 Background story

I find the best explanation to be by potential promotional text because that is also what people would read when considering installation of this game.

"Each mobile device is vulnerable to the world and its elements. Not everyone realizes it, but this also applies to the digital world and when no one is paying attention, small malicious creatures composed only from "ones" and "zeros" attack at their full power...

Yes, I'm talking about viruses and other scoundrels. And yes, there are applications that claim to be able to handle them, but none of them is as capable as your AVUnit could be. Each member of the AVUnit is skilled at something else and that is what makes them so effective. Not only that, but their teamwork is excellent too because they often train against other AVUnits.

Get your own AVUnit today and you will never have to worry about tomorrow again!"

It's obvious that the goal of this game is to have as good AVUnit as possible and to destroy viruses on your device. Player himself is gradually creating his own team (or unit) of Antivirus warriors (that is why AVUnit). These members of AVUnit are just like viruses (or really, anything else on the device) just programs with particular code that requires some memory space (MEM). At the beginning, player has only one member in his AVUnit and that is AVUnit Captain. Caiptain is the only member of the unit that player can't get rid of and every new member is created by copying Captain's code (which again requires some MEM). Just like code in real life, code copied from the Captain can be edited and that is how player acquires different types of AVUnit member.

Figure 4.1: Images of some nasty viruses from the game prototype (*author*)

As stated before, the main purpose of AVUnit is to get rid of viruse which is done in Minigames where AVUnit members clean memory, cpu or gpu and by doing that, they release new resources of MEM, CPU and GPU respectively. With more resources player can create more AVUnit members or equipment for them. This equipment is especially helpful during battle simulations with

other player's AVUnits when AVUnit trains its teamwork. During these battle simulations, all equipment (programs) have to use some CPU and GPU resources to be able to run, which means that the player is limited by his CPU and GPU resources. At the end of the simulation these resources are release and can be used again for the next simulation.

Battle simulations naturally can't take place in the device because that could compromise the hardware (all the explosions and so on). That is why they take place in real world (Augmented reality) away from sensitive hardware and this also gives a chance for members from different AVUnits to meet.

The rules for battle simulations are as follows:

1. Number of AVUnits that can participate in a battle simulation is limited to 2, 3 and 6 (picture 4.2)

2. Each AVUnit has to be represented by exactly n of its members (n is set beforehand)

3. Each member can use only 4 pieces of equipment during one simulation and occupy only one triangle (spot) on the battle platform

4. Members don't have to relieve their identity (and they don't) until they are hit with some weapon by other AVUnit's member

5. Members can pretend (and they do) that empty spots on their part of the battle platform are in fact occupied until they are hit with some weapon by other AVUnit's member

6. AVUnit is declared defeated immediately after all n of its members are defeated

In the basic simulation there is only platform (object) and participating members from all AVUnits. But being it a simulation, no one can say what number and types of obstacles can appear to test their skills. Of course after something like that, every surviving member of an AVUnit gains some well-deserved experience from this simulation and that helps him or her improve their basic skills (defence, offence or error resistance). So how do you defeat an AVUnit member? Easy, you have to cause more errors, in his or her program, than they can handle (indicated by error resistance).

4.3.2 Characters and objects in the game

All in all there are only AVUnit members, equipment, viruses and battle platform in this game world, but they have many variations. More important ones are AVUnit members and equipment, so they are described in bigger detail.

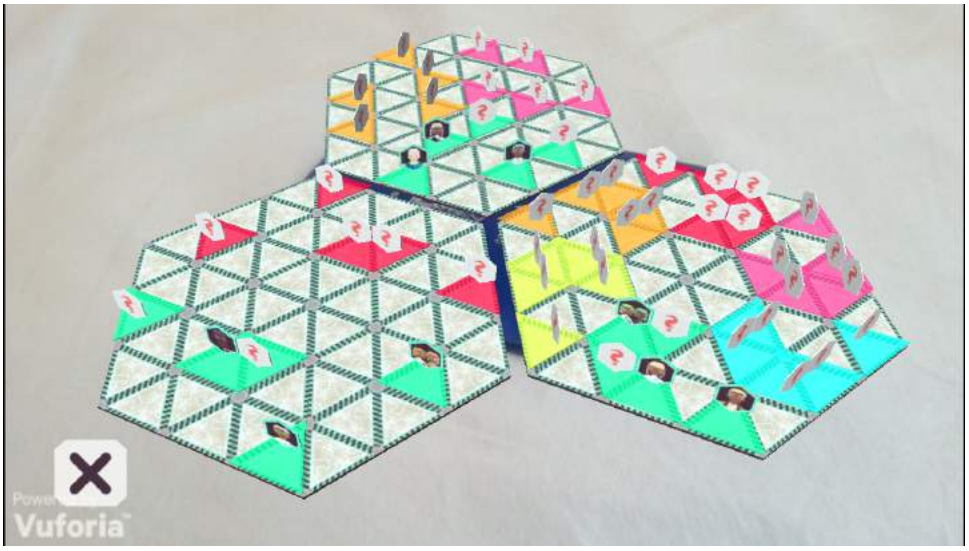


Figure 4.2: Platform with 2, 3 or 6 AVUnits (bottom left, upper middle, bottom right)(*author*)

4.3.2.1 AVUnit members

There is Captain (type) and 5 different types of members (picture 4.3), Base, Cleaner, Scanner, Patch and Shield. Each member can level up after gaining enough experience and gain new abilities specific for his, her or their type, except for Base.

Figure 4.3: 5 different types of members (from left to right: Base, Cleaner, Scanner, Patch and Shield) from various color sets (*author*)

Base is special, this type can still level up and improve its basic skills (offence, defence, resistance), but it can't gain new skills. On the other hand, this type can use any equipment that is not meant only for Captain.

Captain (picture 4.4) is the only one that can use any equipment, but there can be only one member of this type and every time her code is copied, it is represented by Base code. This code if edited can become a code of Cleaner, Scanner, Patch or Shield. **Cleaner** is the most offensive type, **Scanner** represent a true gadget type with a lot of knowledge, **Shield** is the most defensive type and **Patch** type is a mysterious duo of small cheeky girls.

During the battle simulation, all members are represented by hexagons, floating above the battle platform. Player's own hexagons have an image of member's head on them, if there is any member residing in that spot.

4.3.2.2 Equipment

Weapons and other equipment can be created from MEM resources. It needs CPU, GPU resources and a compatible AVUnit member to be used in the battle. There are two main types of equipment: defence and offence (picture 4.4). Defence type of equipment boosts AVUnit member's error resistance while he or she attacks with offence equipment. Then again, there are two types of offence equipment. One is aimed in a straight line in front of the member and the other one is fired under 45 degrees angle resulting in arc trajectory.

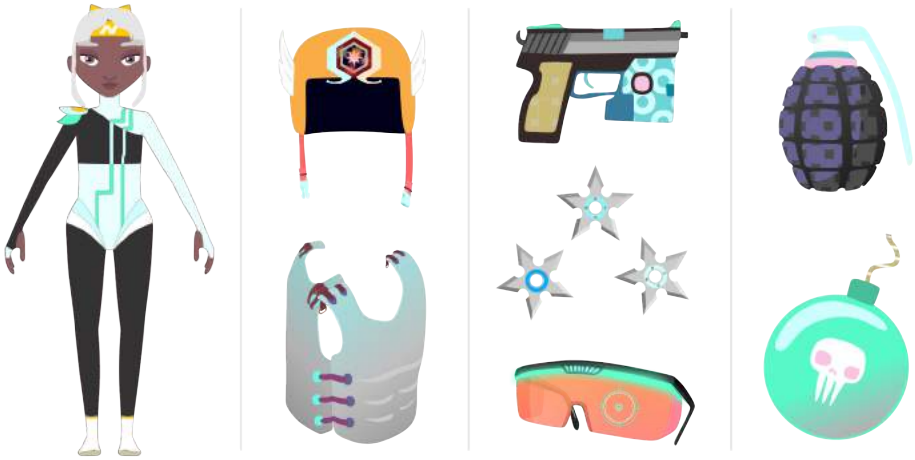


Figure 4.4: From left to right: Captain member (can use any equipment), defence equipment (helmet, armor), offence equipment with straight trajectory (gun, shurikens, laser glasses) and offence equipment with arc trajectory (grenade, bomb) (*author*)

4.4 Use cases and their scenarios

Use cases are made up of a set of possible sequences of interactions between systems and user. They are related to a particular goal and they should cover in their scenarios all Functional requirements. In diagram 4.7 it is possible to see the flow of the UI screens related to the following scenarios.

UC1 Browsing collections (AVUnit members, *acquired equipment, available equipment*)

1. tap mini icon with image of the category that is to be browsed
2. tap the *browse* (or *create*) button in the description board or tap the main image to enter the particular category
3. if category is not empty, browse by taping on the arrows or main image

UC2 Creating new equipment from the *available equipment*

1. tap mini icon with image of the *available equipment* category
2. tap the button in the description board or tap the main image to enter the category
3. if category is not empty, browse by taping on the arrows or main image until the desired equipment is found
4. tap the *create* button in the description board
5. question asking whether it really should be created will pop up if there is enough resources, if there is not enough resources (MEM) the statement of lacking resources will pop up
6. after its creation, this equipment will be automatically added into *acquired equipment* category

UC3 Deleting *acquired equipment* or AVUnit member

1. tap mini icon with image of the category from which the object is to be deleted
2. tap the button in the description board or tap the main image to enter the category
3. if category is not empty, browse by taping on the arrows or main image until the desired object is found
4. tap the *delete* button in the description board
5. question asking whether it really should be deleted will pop up
6. in case of *acquired equipment*, right after its deleted, it will be added into *available equipment* category

UC4 Creating new AVUnit member

1. tap mini icon with image of creating a new AVUnit member
2. tap the button in the description board or tap the main image to enter
3. if there is no code copied yet, tap *copy* button, if there is enough resources (MEM) a question whether it should be created or not will pop up, if there is not enough resources a statement of it will pop up
4. now this copy can be hired as a new Base type member or it can be deleted to get back resources (MEM) or the code can be edited to make a new type of the AVUnit member. To edit the code tap *edit code* button and enter correct word deciphered from the cipher that is displayed, if its correct the code will change and with it also the type of the member. To add it into the AVUnit tap *hire* button and enter the name for this member, then it will be added into the AVUnit members category. To delete it tap *delete* button and a question asking whether it really should be deleted will pop up. After deleting or hiring the copy the spot will be free for another copy of Captain's code.

UC5 Cleaning MEM, CPU or GPU from viruses and earning MEM, CPU and GPU resources

1. in the main menu tap the *resources* button
2. select member that should be doing the cleaning
3. tap the image of the resources you want to clean/earn (MEM, CPU, GPU)
4. play minigame, after the minigame, depending on your score the according resources will be added to resources counter in the left upper corner in the menu

UC6 Enter battle simulation with AVUnit

1. in the main menu tap the *battle* button
2. select 3 different members for this battle and tap arrow to continue
3. for each member select their equipment and tap arrow to continue
4. tap *offline* or *online* button depending on which mode you want to play (prototype has only offline mode), if it is online mode you will also have to select how many AVUnits you want to play with (2,3 or 6 including yours)
5. select position for each member or leave the preset one, then tap arrow to continue

6. if online mode was selected you will have to wait to get matched by matching server or manually select your group match

UC7 3D registration after entering battle simulation

1. read instruction that are displayed after entering battle simulation and tap *ok* button
2. aim camera under 90 degrees angle at a surface that is suitable to be an image target for CVB AR (flat, high contrast, no reflection, no repeating patterns, no symmetry, good lighting conditions) and after the "x" mark is swapped for "✓" tap the center image

UC8 Shooting in the battle simulation

1. tap on the hexagon with head image of the member you want to attack with
2. pick the weapon
3. set rotation and power (if available)
4. fire

UC9 Change language

1. in the main menu tap the small icon in the bottom left corner with cogwheel image on it
2. pick desired (and offered) language and tap *set* button

UC10 Turn sounds on and off

1. in the main menu tap the small icon in the bottom left corner with sound image on it to toggle sounds on and off

4.5 Entities in the game mechanics

Normally design should stay coherent throughout the whole system, but in this case it is useless to keep certain data during the battle stage of the game, which are on the other hand essential while browsing the menu. And again, if battle stage wasn't so computationally demanding it would be completely plausible to keep that data, but in this case I decided to divide the entities into two groups. One is used in the menu stage (or scene) of the game and the other one in the battle stage of the game. Therefore diagram 4.5 represents basic entities and their relationships for menu stage of the game and diagram 4.6 in battle stage of the game. It is completely ignoring minigame entities (e.g. card, virus) and relationships because they are not important for this thesis and also all graphical entities that are used to display these entities (e.g. images, texts, labels).

Entities and relationships in the prototype implementation are little bit simplified because not including the networking would leave some redundant entities. That is why in the prototype instead of having separate Local Player and AIPlayer, they are both included in the BattleController's logic to keep the unnecessary referencing as low as possible.

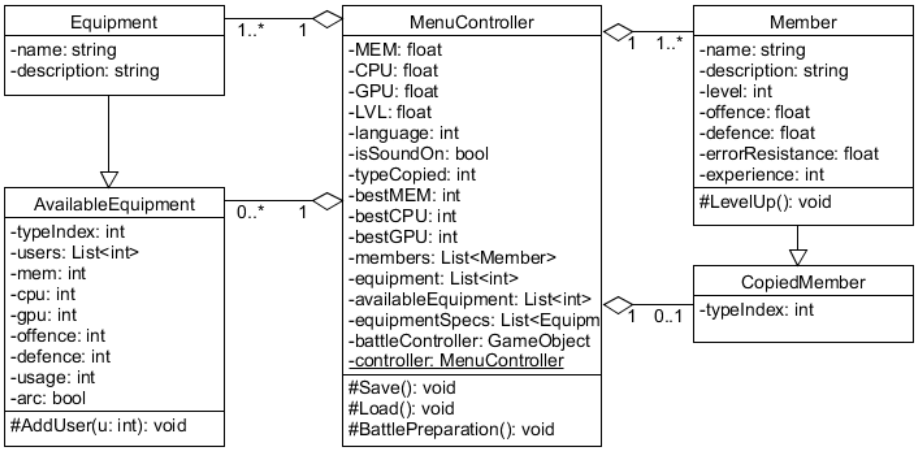


Figure 4.5: Entities in the menu stage of the game (in implementation it is Menu scene) (*author*)

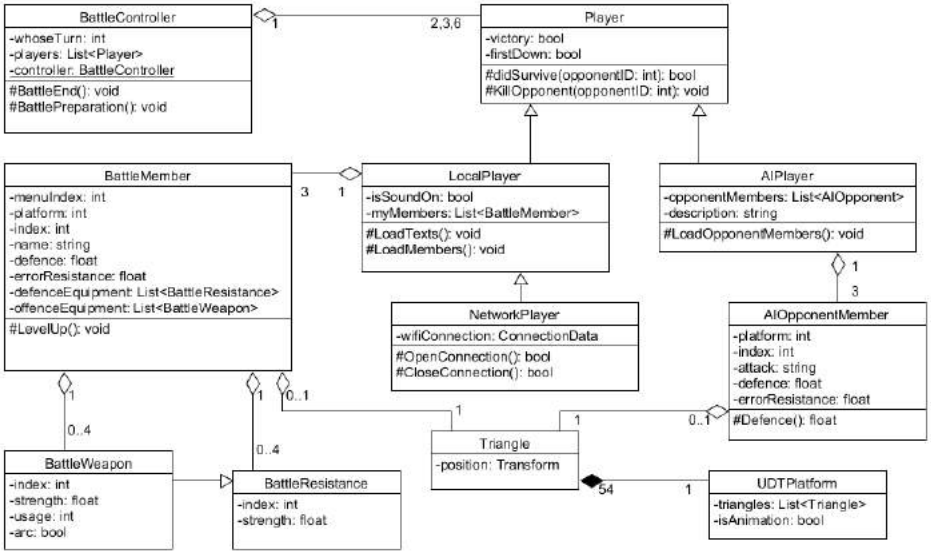


Figure 4.6: Entities in the battle stage of the game (in implementation it is Battle scene) (*author*)

4.6 Scenes and UI screen flow

A diagram 4.7 is a representation of UI screens and looking from the implementation angle, which scene they belong to. All actions in the Use case section 4.4 are just different sequences of steps following arrows in this diagram.

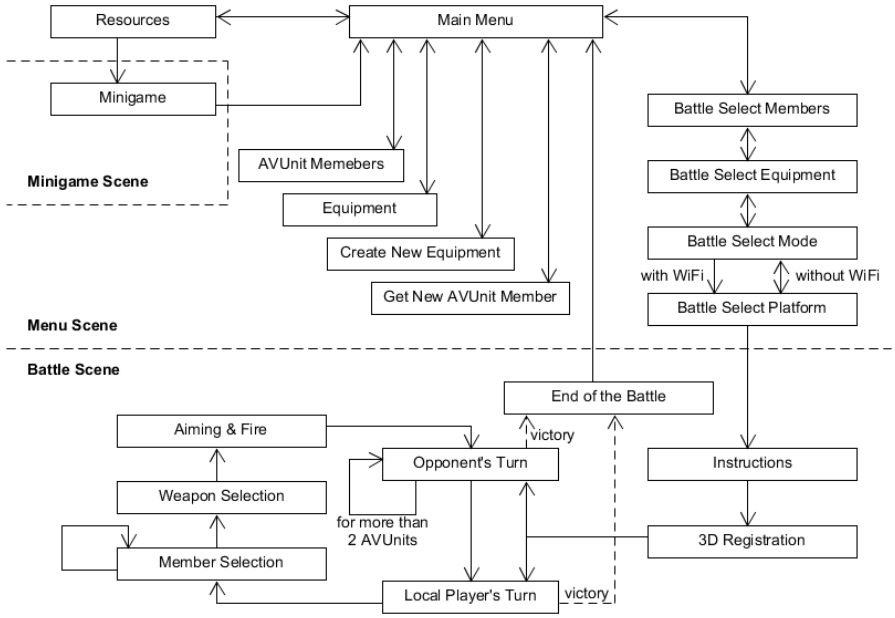


Figure 4.7: Scenes and UI screen flow (*author*)

4.7 My game design and 9PP

Now in the table 4.1 it is possible to see how well this design follows or contradicts each of the Pre-design patterns from section 2.2. It is also a high time to sum up what AR brings into this game that wouldn't be possible to achieve without AR or with high difficulty.

First of all, it is easier for the player to observe physics mechanics because he or she can rotate the target image or camera around the target image to view the scene from different angle. Thanks to that, it is also possible to aim with great precision because player can view the scene from the top of the platform and if there is an obstacle, he or she can view the scene from the side to see how high the obstacle is. Player can also inspect the graphics and objects from a close distance, which is definitely a new experience if the author put a lot of details that are not visible from normal distance on them.

Table 4.1: My game design and 9 Pre-design patterns as a criteria of evaluation

Title	Its representation in the design
Device metaphors	Not particularly emphasized but some object metaphors are used in aiming e.g. arrow to indicate the direction
Control mapping	Raycasting for selection of a member on the platform to shoot with
Seamful design	Registration instructions and assurance of the image target quality before its acquisition
World Consistency	Physics system simulating real world physics laws
Landmarks	Battle platform as the main landmark with set of subordinate landmarks of hexagons representing potential AVUnit members
Personal presence	Player is detached and his or her presence is placed onto avatar (AVUnit members)
Living creatures	AVUnit members (creating, equipping, leveling up, battling with)
Body constraints	Emphasis on no constraints
Hidden information	Temporarily hidden position and identity of the AVUnit members during the battle simulation

Chapter 5

Implementation

As mentioned multiple times before, implementation will be focusing only on the offline part of the game, nevertheless, tools and technology should be selected based on the whole proposed design so even the networking should be taken into consideration while choosing the right set of tools.

5.1 Selected tools and technology

The idea of this game is that it should be heavy on graphics and use 3D models with animations, not even mentioning physics system. Applications with these requirements are quite difficult to make in the native development tools and are often made in the specialized game engines. Of course with native development tools you can achieve higher efficiency, but game engines usually have a lot of optimized algorithms and design patterns that would take years to implement and there is really no point in reinventing the wheel. Some of the most famous 3D game engines nowadays are Unreal, Unity, CryEngine, JMonkey, Panda3D, Blender Game Engine and many more [63]. I have chosen Unity based on the services it offers and its compatibility with AR frameworks analyzed in section 3.2, which (with exception of ARLab) are all fully compatible with Unity. Unity is a game engine that deploys to a lot of platforms, but to build the APK format for Android, Unity needs Android SDK installed on the computer. More on Unity (Personal Edition, version 5.3) and how it works in section 5.2.

Also based on the evaluated use case scenarios in the table 3.2 in section 3.2, I've decided to use Vuforia because the only higher scoring SDK is Metaio, but since it was discontinued for now, it loses in the offered services and potential future updates. Vuforia on the other hand, offers number of new features in every release and besides Unity version, it also supports native Android and iOS development. More on Vuforia (Unity plugin, version 5.5.9) in section 5.3.

There is a lot of 3D modeling software on the market, most of them highly specialized for certain professions or industries like AutoCAD, Houdini, ZBrush, Mudbox or SketchUp [64], but when it comes to simple universal 3D

modelling software the race is won by Maya, 3ds Max, Cinema 4D or Blender [65] from which only Blender isn't under commercial license (it is under GNU GPLv2+). Therefore I've chosen to use Blender (version 2.68) for creation of all 3D objects, their animations and any 2D content rendered from 3D content.

As far as the 2D content goes I've been using Corel DRAW Graphics Suite X5 for years now, so it was an easy choice although based on the complexity of the 2D content (which is low) in this game, basically any 2D vector editor with layer function would suffice.

Audio editing was done in the AudaCity (version 2.1.0), which is a free, open source, cross-platform software for recording and editing sounds.

5.2 Unity

Important for this prototype is that Unity has its own physics system and networking system that consists of game objects called GameObjects that implement specific interfaces and has attached special codes called scripts to them. These scripts, as soon as they are attached to the GameObject, become *components* of this GameObject. In general the behavior of GameObjects in project's scene is controlled by the *components* that are attached to them. Although Unity's built-in *components* can be very versatile, most of the time it is necessary to go beyond what they can provide and write custom scripts aka *components*. Scripts (*components*) allow developers to trigger game events, modify *component's* properties over time or respond to user input. Unity supports two programming languages natively:

- C# (the one I have used)
- UnityScript, a language designed specifically for use with Unity and modelled after JavaScript

In addition to these, many other .NET languages can be used with Unity if they can compile a compatible DLLs. To edit the scripts Unity offers its MonoDevelop editor or with Unity 5 they offer Visual Studio as a first option and MonoDevelop as a second one. A script is only a template and until it is attached to a GameObject as its *component*, it isn't executed at all.

By default any class in Unity is a child of MonoBehaviour class, which is a base class for all new Unity scripts. In addition every GameObject has a position, rotation and scale in space (whether 3D or 2D), and this is represented by the Transform *component*. To take advantage of the physics system a RigidBody (or RigidBody2D for 2D) *component* needs to be attached to the GameObject. For Networking a NetworkIdentity *component* has to be attached to the networked objects and NetworkBehaviour used in networked scripts.

In Unity scripting, there are a number of event functions that get executed in a predetermined order as a script executes. In diagram 5.1 the execution order is summarised and some ordering and repetition of event functions during a script's lifetime are described with a little bit more detail. The most commonly used are Awake, OnEnable, Start, OnCollisionXXX, OnMouseXXX, Update, OnDrawGizmos and OnGUI.

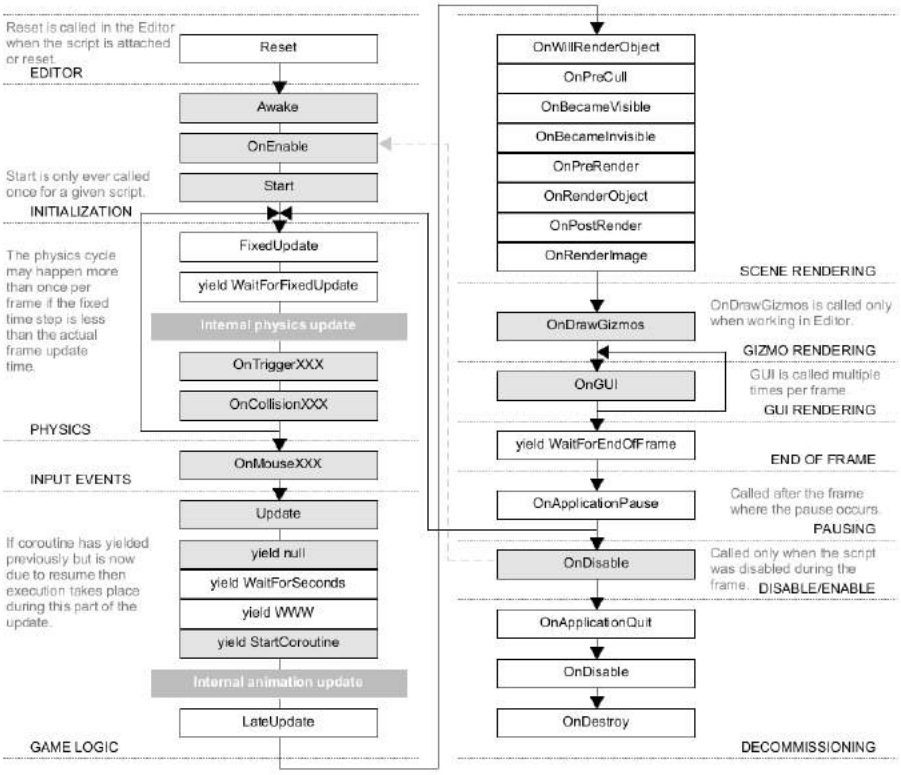


Figure 5.1: Execution Order of Event Functions in Unity (*author, based on original from Unity Documentation [66]*)

Because called function, runs until completion before returning, this effectively means that any action taking place in a function must happen within a single frame update. If something needs to take place over several frames, there are two main ways how to do it. First adding code to Update fuction that executes the process on a frame-by-frame basis (like fading an image of a member in AVUnit members category in code 5.1) and second often more convenient is to use coroutines (like flipping an image in AVUnit members category, code 5.2). A coroutine is like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame.

Code 5.1: Using Update function for process that takes place over more than just one frame

```
public float minimum = 0.0f;
public float maximum = 1f;
public float fadeDuration = 1.0f;
private float startTime;
...
void Update () {
...
if(isFadeOff){  //image is disappearing
   float t = (Time.time - startTime) / fadeDuration;
   image.color = new Color(1f,1f,1f,Mathf.SmoothStep(maximum,
   minimum, t));
   if(image.color.a <= minimum){
       isFadeOff = !isFadeOff; //disappeared and starts to appear
       ...
```

Code 5.2: Using coroutines for process that takes place over more than just one frame

```
public class GUIMemberImageSwap : MonoBehaviour {
   public SpriteFlipper flipper;
   private float startTime;
   ...
   public void next () {  //flip to the next image
   ...
   if(...
       flipper.FlipImage(); //stop coroutine and start new one
       startTime = Time.time;
       isFlipping = true;
       ...
}

// ***** end of the class and start of another one *****

public class SpriteFlipper : MonoBehaviour {
   ...
 public void FlipImage(){
   StopCoroutine(Flip ());
   StartCoroutine(Flip ());
 }
 IEnumerator Flip (){ //coroutine
   float time = 0f;
   while(time <= 1f){
     float scale = scaleCurve.Evaluate(time);
     time += (Time.deltaTime / duration);

     Vector3 localScale = transform.localScale;
     localScale.x = scale;
     transform.localScale = localScale;
     yield return new WaitForFixedUpdate(); //place where
   coroutine pauses and returns control to the Unity
   }
```

```
33   }
34 }
```

5.2.1 UI system

Unity has its own UI system which if used right can correctly readjust, display and operate on various screen ratios and sizes. Interactive UI objects are linked to Events system. Input API (Application Programming Interface) uses raycasting (converting screen coordinates to scene coordinates by casting a straight line from pointer into the scene) to determine what the pointer (e.g. finger on the device or mouse on the screen) is over. There are 3 provided Raycasters in Unity that exist by default:

- Graphic Raycaster - Used for UI elements
- Physics 2D Raycaster - Used for 2D physics elements
- Physics Raycaster - Used for 3D physics elements

During the battle stage of the game player can select one of his members by tapping on the screen where the augmented 3D object of his member is rendered. That is realized in code by implementation of the Physics Raycaster. In code 5.3 it is tested through Input API whether player tapped somewhere on the screen and then the ray is queried against all GameObjects with Collider component on them (the only ignored objects are objects on the IgnoreRaycast layer).

Code 5.3: Physics Raycasting

```
1  ...
2  RaycastHit rayHit;
3  Ray ray;
4  ...
5  void Update(){
6  ...
7  if(isPickingMember){
8     if(Input.touchCount > 0){ //player touched screen
9         Touch touch = Input.GetTouch(0);
10        if(touch.phase == TouchPhase.Began){
11            ray = Camera.main.ScreenPointToRay(touch.position);
12            if (Physics.Raycast(ray, out rayHit)){ //returns info
    in rayHit about collisions of ray with other GameObjects
13                if(rayHit.collider.gameObject...
14                ...
```

5.2.2 Data persistence

Unity has excellent serialization which can serialize a whole class or List data structure if its items are serializable. The game prototype loads and saves

data as one instance of a PlayerData class that doesn't inherit from MonoBehaviour (classes that doesn't inherit from MonoBehaviour act as structures and can have their own customized constructor which is not advised to use with MonoBehaviour) into a "playerInfo.dat" file (code 5.4).

Code 5.4: Serialization and deserialization

```
1  [Serializable]
2  class PlayerData{
3   public float LVL;
4   public float MEM;
5   ...
6   public List<CMember> members;
7   public List<int> equipment;
8  }
9
10 // ***** end of the class and start of another one *****
11
12 public class GameController : MonoBehaviour {
13 ...
14 public void Save(){
15  BinaryFormatter bf = new BinaryFormatter();
16  FileStream file = File.Create(Application.persistentDataPath + "
      /playerInfo.dat");
17  PlayerData data = new PlayerData();
18  data.LVL = LVL;
19  ...
20  data.equipment = equipment;
21  bf.Serialize(file, data);
22  file.Close();
23 }
24 public void Load(){
25  if(File.Exists(Application.persistentDataPath + "/playerInfo.dat
      ")){
26    BinaryFormatter bf = new BinaryFormatter();
27    FileStream file = File.Open(Application.persistentDataPath + "
      /playerInfo.dat", FileMode.Open);
28    PlayerData data = (PlayerData)bf.Deserialize(file);
29    file.Close();
30
31    LVL = data.LVL;
32    ...
33    equipment = data.equipment;
34  }else{
35    LVL = 0;
36    ...
37  }
38 }
```

5.2.3 Multilingualism

All the info data in the prototype that is separable from the code is stored into XML files. There is a special class that parses these files into lists (index

indicating the language) of dictionaries (or lists of lists of dictionaries). For example, first you select language then you select type of member and then its attribute (english/0 → Base/1 → "description"). This makes it easier to add another language or member or equipment without the necessity of changing the code and it can be done by anyone. Code 5.5 shows parsing of the file with normal UI text.

Code 5.5: Parsing language data from XML files

```
public TextAsset dictionary;
public List<Dictionary<string, string>> languages = new List<
   Dictionary<string, string>>();
...
void Reader(){
    Dictionary<string, string> obj;
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.LoadXml(dictionary.text);
    XmlNodeList languageList = xmlDoc.GetElementsByTagName("
   language");
    foreach(XmlNode languageValue in languageList){
        XmlNodeList languageContent = languageValue.ChildNodes;
        obj = new Dictionary<string, string>();
        foreach(XmlNode value in languageContent){
            switch(value.Name){
                case "name": //descriptors for all text strings
                case "loading":
                ...
                default:
                    Debug.Log(value.Name+" is unused"); break;
      }
    }
    languages.Add(obj);
  }
}
```

5.2.4 Physics system

Shooting in the battle stage of the game prototype is done using Unity's physics system which means attaching (Box/Sphere/Mesh/...) collider *component* onto every GameObject that should be included in the collision detection and attaching RigidBody *component* on the projectile GameObject (because it is the one that is moving and should be influenced by gravity). Actual firing of the projectile is the last part of sequence of commands securing right position and rotation of the projectile (code in 5.6). Detecting a collision with another object ensures OnCollisionEnter function which is called immediately after projectile hits anything with a collider.

Code 5.6: Firing a projectile

```
public Rigidbody projectile;
public Rigidbody projectile2;
```

```
3  public Transform shotPos;
4
5  public void SetShotPos(){
6     shotPos.rotation = Quaternion.identity; //(0,0,0) WCS rotation
7     shotPos.position = transform.position + new Vector3(0f, 0.104f
      , 0);
8  }
9  public void ShootArc () { //shooting in an arc trajectory
10    shotPos.position = transform.position + new Vector3(0f, 0.104f
      , 0.0192f);
11    shotPos.Rotate(-45,0,0);
12    Rigidbody shot = Instantiate(projectile,shotPos.position,
      shotPos.rotation) as Rigidbody;
13    shot.AddForce(shotPos.forward * BattleGameController.
      controller.powerVal);
14 }
```

5.3 Vuforia

Unity Version of Vuforia is in form of a Unity package that has to be imported into the project as any other package. After its import, a new folder will appear in the file system inside Assets folder with all necessary basic scripts and AR GameObjects that are shipped by Vuforia.

Vuforia scripts inside Unity are coded in C#. First important difference in a scene with Vuforia GameObjects in comparison to classic Unity scene is camera. Vuforia uses a special ARCamera which can be found in the imported package and has to be used instead of normal camera GameObject. This will allow Vuforia to stream live camera feed in the background of the scene and also use it for 3D registration of the CVB AR target. ARCamera has multiple modes and by default it doesn't focus, but in the prototype it is set to have auto-focus (code 5.7).

5.3.1 User Defined Target

A new feature in Vuforia called UDT (User Defined Target) relieves player from downloading and printing the target image. Player creates the target image on the fly instead. One camera frame gets selected as a target image from which then NFT descriptors are extracted and saved into the database. I have chosen this type of target to be used in the game prototype. I also allowed extended tracking because only one target will be created and tracked for each battle and it is likely to be a static target. Extended tracking utilizes features of the environment to improve tracking performance and sustain tracking even when the target is no longer in view.

UDT requires a GameObject UDTBuilder in the scene that takes care of the building process while the AR target's GameObject in the scene has to have a component that actually decides when the image target is created

and with it also this target GameObject gets instantiated into the augmented scene. Code 5.7 shows this component (in this case called SimpleUDTHandler) which inherits from both MonoBeviour and IUserDefinedTargetEventHandler. In the Start function it is possible to see the camera autofocus command line. In the Update function only medium and high quality image targets are allowed to be captured and when Player taps the image with "✓" on the screen the BuildNewTarget function will be called and the UDT creation process is finished.

Code 5.7: UDTHandler component for the image target's GameObject

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine.UI;
5  using Vuforia;
6
7  public class SimpleUDTHandler : MonoBehaviour,
       IUserDefinedTargetEventHandler {
8      private UserDefinedTargetBuildingBehaviour
       mTargetBuildingBehaviour;
9      private ObjectTracker mObjectTracker;
10     private DataSet mBuiltDataSet;
11     private bool mUdtInitialized = false;
12     private ImageTargetBuilder.FrameQuality mFrameQuality =
       ImageTargetBuilder.FrameQuality.FRAME_QUALITY_NONE;
13     public ImageTargetBehaviour ImageTargetTemplate;
14     public UnityEngine.UI.Image target;
15     public Button button;
16     public Sprite X;
17     public Sprite OK;
18     public GameObject arms;
19
20     void Start() {
21         mTargetBuildingBehaviour = GetComponent<
       UserDefinedTargetBuildingBehaviour>();
22         if(mTargetBuildingBehaviour) {
23             mTargetBuildingBehaviour.RegisterEventHandler(this);
24         }
25         CameraDevice.Instance.SetFocusMode(CameraDevice.FocusMode.
       FOCUS_MODE_CONTINUOUSAUTO); //set camera in autofocus mode
26     }
27     public void OnInitialized() {
28         // look up the ImageTracker once and store a reference
29         mObjectTracker = TrackerManager.Instance.GetTracker<
       ObjectTracker>();
30         f (mObjectTracker != null) {
31             // create a new dataset
32             mBuiltDataSet = mObjectTracker.CreateDataSet();
33             mObjectTracker.ActivateDataSet(mBuiltDataSet);
34             // remember that the component has been initialized
35             mUdtInitialized = true;
36         }
```

```
37      }
38      public void OnFrameQualityChanged(ImageTargetBuilder.
    FrameQuality frameQuality) {
39          mFrameQuality = frameQuality;
40      }
41      public void OnNewTrackableSource(TrackableSource
    trackableSource) {
42          // deactivates the dataset first
43          mObjectTracker.DeactivateDataSet(mBuiltDataSet);
44          // destroy the oldest target if the dataset is full
45          if (mBuiltDataSet.HasReachedTrackableLimit()) {
46              IEnumerable<Trackable> trackables = mBuiltDataSet.
    GetTrackables();
47              Trackable oldest = null;
48              foreach (Trackable trackable in trackables)
49                  if (oldest == null || trackable.ID < oldest.ID)
50                      oldest = trackable;
51              if (oldest != null) {
52                  mBuiltDataSet.Destroy(oldest, true);
53              }
54          }
55          // get predefined trackable (template) and instantiate it
56          ImageTargetBehaviour imageTargetCopy = (
    ImageTargetBehaviour)Instantiate(ImageTargetTemplate);
57          // add the trackable to the data set and activate it
58          mBuiltDataSet.CreateTrackable(trackableSource,
    imageTargetCopy.gameObject);
59          // re-activate the dataset
60          mObjectTracker.ActivateDataSet(mBuiltDataSet);
61      }
62      void Update() {
63          if(mUdtInitialized && (mFrameQuality == ImageTargetBuilder
    .FrameQuality.FRAME_QUALITY_HIGH ||
64        mFrameQuality == ImageTargetBuilder.FrameQuality.
    FRAME_QUALITY_MEDIUM)){
65        target.sprite = OK;
66        button.gameObject.SetActive(true);
67      }else{
68          target.sprite = X;
69          button.gameObject.SetActive(false);
70      }
71    }
72    public void BuildNewTarget() { //build target with its object
73          string newTargetName = "MyUserDefinedTarget";
74          mTargetBuildingBehaviour.BuildNewTarget(newTargetName,
      ImageTargetTemplate.GetSize().x);
75          arms.GetComponent<ARSetPlatform>().startTime = Time.time;
76    }
77  }
```

Chapter 6

Usability Testing

After initial test (on Samsung Galaxy S3 (GT-i9300 with Android 4.3 and 8 Mpx camera) and 12 testing subjects with various mobile user experience), it became obvious that this game shouldn't and actually can't target the whole spectrum of users from beginners to expert users of mobile technology. If the game would be divided into 2 parts, the Menu and the Battle part, then difference in the performance of individual testing subjects in the Menu part was nearly unnoticeable and the minigame was equally enjoyed (classic game of pair matching) although no one was able to get to the last round. However Augmented reality in the Battle part of the game caused less experienced users (mostly older citizens) to panic and it often resulted into poor image target quality (not taken under 90 degrees angle or with bad lighting etc) or them constantly covering the camera lens with fingers or their hand. Still the test wasn't completely useless because it uncovered some shortcomings in the UI of the Menu part and significant bug in the Battle part:

- unsuitable image icon for "Go back" action
- unintuitive action assigned to the main menu image
- mini icons design in the main menu wasn't explicit enough
- selection of the members and equipment for battle wasn't automatic enough
- the set of members and their equipment doesn't get updated after second loading of the Battle scene in one session of the game

I deducted that the main problem with the Battle part of the game was lack of feedback that would indicate for the less experienced users what is happening and I added it into the list of changes.

For the second testing after fixing the problems in the prototype from initial test, subjects that were chosen had medium or high mobile user experience to

primarily test the Battle part of the game. Half of them got list of actions that they were supposed to perform (use cases from section 4.4) and half of them were left to do whatever they want in the game. Naturally the first half performed better because the second half of testing subjects didn't know at first what is even possible to do.

In the first round they got a game with no setup where they had only Captain type member and one gun, which means their actions were fairly limited and completely excluded Battle part of the game. They all stated that informative pop-up messages telling them that they don't have enough MEM, CPU, GPU or members, step-by-step navigated them into what they should actually do, but they also said that it would be better if in the first run of the game there would be a set of messages about what to do to get started.

In the second round they got a game with initial setup of 8 AVUnit members, 10 pieces of equipment, 140 MEM, 120 CPU and 130 GPU resources. Only 3 people out of 10 went to look at their members and equipment and the rest of them went straight to the Battle part of the game where 9 of them successfully defeated their AI opponent and one of them spend most of the time testing the physics system, image tracking and recovery and in general trying to break it until he got killed by his opponent. There were no visible problems in the battle's gameplay, but without an actual self-improving opponent (which would be present if the networking was in the game) the battle against a fairly predictable AI will most likely loose its charm after some time.

Conclusion

In this thesis I've made a detailed research on mobile AR. First, I've analysed and compared each approach that can be taken while making a mobile AR application, then I have searched for current solutions on the market and reviewed necessary tools for their development. With knowledge gained from this research I've designed an AR game and implemented its prototype. The result of this thesis is a working AR game for Android that fulfills the requirements (allowing player to interact with computer generated content in AR world).

During implementation despite various initial tests with both Unity and Vuforia separately, it proved to be quite tricky to combine them in one project. For example, some design patterns used in minigame (pair matching) scene or animations with events attached to them, couldn't be used in the AR scene and I had to experiment with alternative solutions.

This game prototype can be used as a core part for a game with networking which would bring uniqueness into the gameplay and also increase competitiveness in players. Other future extensions could be more complex 3D models with animations for AR part of the game, but that is rather limited by device specifications and could prevent some users from being able to play this game.

This game wasn't only about coding, but also required creation of 2D and 3D content which definitely forced me to learn and improve several of my technical skills.

Personal evaluation

After spending a lot of time on various forum sites that deal with issues caused by implementation of AR onto mobile platforms, I believe that unless the application is flawlessly executed with seamless UI design, it will miss the understanding of its audience. There's real potential for this kind of AR software, but the trick has always been to develop an interface that will make sense in such a rich sensual environment. General knowledge and experience with AR is still very low and most people expect results like the ones in sci-fi movies or

performance as good as in VR. Having to develop a simple mobile AR game myself made me appreciate the time and effort that developers have to put into their development of AR graphics and algorithms.

Bibliography

[1] *Vuforia* [online]. PTC Inc. 2016, [cit. 2016-05-14]. Available at: http://www.vuforia.com/

[2] *Unity* [online]. Unity Technologies 2016, [cit. 2016-05-14]. Available at: https://unity3d.com/

[3] *Penguin NAVI* [online]. Sunshine Aquarium 2013, [cit. 2016-05-14]. Available at: https://www.youtube.com/watch?v=IK4-zPD_25U

[4] OSER A. *Augmented Reality in Marketing: 5 Compelling Statistics*. pulse [online]. 2014, [cit. 2016-04-23]. Available at: https://www.linkedin.com/pulse/20141209155630-28910946-augmented-reality-in-marketing-5-compelling-statistics

[5] *IKEA 2016 Catalogue* [online]. IKEA 2016, [cit. 2016-05-14]. Available at: https://www.youtube.com/watch?v=xC6t2eEPkPc

[6] *MeView Augmented Reality Maintenace Enterprise Application* [online]. Mitsubishi Electric 2013, [cit. 2016-05-14]. Available at: https://www.youtube.com/watch?v=Edi02M1nS8g

[7] *Mitsubishi Electric, Cooling & Heating* [online]. [cit. 2016-05-14]. Available at: http://www.metaio.com/fileadmin/upload/documents/pdf/case-study/A4-metaio_use_case-mitsubishi_meVIEW-AR.pdf

[8] David Murphy *Augmented Reality Glasses: What You Can Buy Now (or Soon)* [online]. tomsguide [cit. 2016-05-14]. Available at: http://www.tomsguide.com/us/best-ar-glasses,review-2804.html

[9] *Envision IP* [online]. Envision IP [cit. 2016-05-14]. Available at: http://www.envisionip.com/

[10] *Metaio GmbH* [online]. Apple Inc. 2016, [cit. 2016-05-14]. Available at: https://www.metaio.com/

[11] *Augmented Reality for Mobile Devices* [online]. Tractica LLC. 2015, [cit. 2016-05-14]. Available at: https://www.tractica.com/newsroom/press-releases/installed-base-of-mobile-augmented-reality-apps-to-reach-2-2-billion-by-2019/

[12] AZUMA, Ronald. *A Survey of Augmented Reality*. Presence [online]. 1997, 6(4), 355-385 [cit. 2016-04-14]. DOI: 10.1162/pres.1997.6.4.355. ISSN 10547460. Available at: http://web.b.ebscohost.com

[13] MILGRAM P., KISHINO F. *A taxonomy of mixed reality visual displays*. [online] IEICE TRANSACTIONS on Information and Systems, vol. 77, no. 12, 1994, pp. 1321-1329. [cit. 2016-05-14]. Available at: http://cs.gmu.edu/~zduric/cs499/Readings/r76JBo-Milgram_IEICE_1994.pdf

[14] SOURIN, Alexei. *COMPUTER GRAPHICS From a Small Formula to Cyberworlds*. 3. edition. Singapore: Pearson Education South Asia Pte Ltd, 2012. ISBN 978-981-06-9234-6.

[15] BCS GLOSSARY WORKING PARTY. *VIRTUAL REALITY*. BCS Glossary of Computing and ICT. 2013. ISBN 1780171501.

[16] *Wikitude GmbH* [online]. version 8.2.6 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.wikitude

[17] *Yelp* [online]. Yelp Inc. [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.yelp.android

[18] GRUBERT, Jens. *Augmented Reality for Android Application Development*. [online] 2013. ISBN 1782168559. [cit. 2016-05-14]. Available at: http://site.ebrary.com/

[19] KEMAO, Qian. *Imaging Geometry*. NTU Singapore, CCE4003/CZ4003 Computer Vision, 2014.

[20] VAN KREVELEN D., POELMAN R. *A Survey of Augmented Reality Technologies*. Applications and Limitations [online]. 2010, [cit. 2016-04-15]. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.454.8190&rep=rep1&type=pdf

[21] LAWITZKI, Paul. *Android Sensor Fusion Tutorial* [online]. CodeProject, 2014 [cit. 2016-05-14]. Available at: http://www.codeproject.com/Articles/729759/Android-Sensor-Fusion-Tutorial

[22] SEAH, Hock Soon. *Tracking for AR*. NTU Singapore, CE/CZ4001 Virtual and Augmented Reality, 2014.

[23] KHAN, Akif. *Rebirth of Augmented Reality - Enhancing Reality via Smartphones*. Bahria University Journal of Information & Communication Technology [online]. 2015, vol. 8, no. 1, s. 110-121. ISSN 19994974. [cit. 2016-05-14]. Available at: http://search.proquest.com.ezproxy.techlib.cz/docview/1695973798?pq-origsite=summon

[24] FIALA, M. *ARTag, a Fiducial Marker System using Digital Techniques*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) [online], vol. 2/(2005), pp. 590-596 vol. 2. [cit. 2016-04-14]. DOI: 10.1109/CVPR.2005.74. ISSN 1063-6919. Available at: http://ieeexplore.ieee.org

[25] *ARToolKit* [online]. ARToolKit [cit. 2016-05-14]. Available at: http://artoolkit.org/

[26] XU, Dong. *Object Recognition - Part II*. NTU Singapore, CCE4003/CZ4003 Computer Vision, 2014.

[27] MIKOLAJCZYK K., et al. *A Comparison of Affine Region Detectors*. International Journal of Computer Vision. [online] 2005, vol. 65, no. 1, pp. 43-72. [cit. 2016-04-14]. DOI: 10.1007/s11263-005-3848-x. ISSN 1573-1405. Available at: http://link.springer.com

[28] MIKOLAJCZYK K., SCHMID C. *A performance evaluation of local descriptors*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005 [online], vol. 27, no. 10, pp. 1615-1630. [cit. 2016-04-14]. DOI: 10.1109/TPAMI.2005.188. ISSN 0162-8828. Available at: http://ieeexplore.ieee.org

[29] CARMIGNIANI J., et al. *Augmented reality technologies, systems and applications*. Multimedia Tools and Applications. [online] 2011, vol. 51, no. 1, pp. 341-377. [cit. 2016-04-14]. DOI: 10.1007/s11042-010-0660-6. ISSN 1573-7721. Available at: http://link.springer.com

[30] *Layar* [online]. Layar. version 8.5.0 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.layar&hl=sk

[31] *Field Trip* [online]. Niantic, Inc. version 2.0.9 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.nianticproject.scout&hl=sk

[32] *Mybrana, Fun Videos with AR FX* [online]. Mybrana Network. version 1.52.0 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.mybrana.mybrana3d&hl=sk

[33] *Google Translate* [online]. Google Inc. [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=sk

[34] *Theodolite* [online]. Hunter Research and Technology, LLC. version 5.0 [cit. 2016-05-14]. Available at: https://itunes.apple.com/us/app/theodolite/id339393884?mt=8

[35] *Star Walk - Astronomy Guide* [online]. Vito Technology. version 1.0.10.21 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.vitotechnology.StarWalk

[36] *Sun Seeker* [online]. ozPDA. version 4.6.2 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.ajnaware.sunseeker&hl=sk

[37] *iOnRoad Augmented Driving Pro* [online]. iOnRoad. version 2.0.1p [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.picitup.iOnRoad.pro

[38] XU Y., et al. *Pre-patterns for designing embodied interactions in handheld augmented reality games*. IEEE International Symposium on Mixed and Augmented Reality - Arts, Media, and Humanities. [online] 2011, pp. 19. [cit. 2016-04-14]. DOI: 10.1109/ISMAR-AMH.2011.6093652. ISSN 2381-8360. Available at: http://ieeexplore.ieee.org

[39] JACOB R.J.K., et al. *Reality-Based Interaction: A Framework for Post-WIMP Interfaces*. Proc. ACM CHI 2008 Human Factors in Computing Systems Conference. [online] pp. 201-210, ACM Press (2008). [cit. 2016-04-14]. Available at: http://www.cs.tufts.edu/~jacob/papers/chi08.pdf

[40] *ARBasketball - Augmented Reality Basketball Game* [online]. Augmented Pixels Inc. version 2.0.0 [cit. 2016-05-14]. Available at: https://itunes.apple.com/us/app/arbasketball-augmented-reality/id393333529?mt=8

[41] *AR Defender 2* [online]. Bulkypix. version 1.3 [cit. 2016-05-14]. Available at: https://itunes.apple.com/us/app/ar-defender-2/id559729773?mt=8

[42] *Tilt - Augmented Reality* [online]. 13 App Design. version 1.2.0 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.thirteen.tilt&hl=sk

[43] *Crayola Color Alive* [online]. DAQRI. version 1.8.0 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.DAQRI.crayola.coloralive&hl=sk

[44] *ARSoccer - Augmented Reality Soccer Game* [online]. Laan Labs. version 0.8 [cit. 2016-05-14]. Available at: https://itunes.apple.com/us/app/arsoccer-augmented-reality/id381035151?mt=8

[45] *A.R. Warriors* [online]. aCrm Net S.r.l.. version 1.0 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.acrmnet.arwarriors&hl=sk

[46] *Paparazzi - Augmented Reality* [online]. Pixel Punch. version 1.3.1 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.pixelpunch.paparazzi

[47] *AR Invaders* [online]. Soulbit7. version 1.0.1 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.soulbit7.game.arinvaders&hl=sk

[48] *Ingress* [online]. Niantic, Inc. version 1.99.1 [cit. 2016-05-14]. Available at: https://play.google.com/store/apps/details?id=com.nianticproject.ingress&hl=sk

[49] *Pokémon GO* [online]. Niantic, Inc. beta version [cit. 2016-05-14]. Available at: http://nianticlabs.com/blog/

[50] *Aurasma* [online]. Hewlett-Packard Development Company. L.P. [cit. 2016-05-14]. Available at: https://www.aurasma.com/

[51] *Blippar* [online]. Blippar [cit. 2016-05-14]. Available at: https://blippar.com/en/

[52] *Layar* [online]. Blippar [cit. 2016-05-14]. Available at: https://www.layar.com/

[53] *Wikitude* [online]. Wikitude GmbH [cit. 2016-05-14]. Available at: http://www.wikitude.com/

[54] *Augment* [online]. Augment [cit. 2016-05-14]. Available at: http://www.augment.com/

[55] *Blippbuilder* [online]. BlippBuilder [cit. 2016-05-14]. Available at: https://blippar.com/en/solutions/self-service-solutions/

[56] *Layar Creator* [online]. Blippar [cit. 2016-05-14]. Available at: https://www.layar.com/accounts/login/?next=/creator/

[57] *Webcam Social Shopper* [online]. Zugara Inc. [cit. 2016-05-14]. Available at: http://webcamsocialshopper.com/

[58] *D'Fusion Studio* [online]. Total Immersion. [cit. 2016-05-14]. Available at: http://www.t-immersion.com/products/dfusion-suite/dfusion-studio

[59] *ARLab* [online]. Augmented Reality Lab S.L. [cit. 2016-05-14]. Available at: http://www.arlab.com/

[60] *Catchoom CraftAR* [online]. Catchoom [cit. 2016-05-14]. Available at: http://catchoom.com/product/craftar/augmented-reality-and-image-recognition/

[61] MARNEANU I., EBNER M., ROESSLER T.*Evaluation of Augmented Reality Frameworks for Android Development*. International Journal of Interactive Mobile Technologies (iJIM). [online] 2014, vol. 8, no. 4, pp. 37-44. [cit. 2016-04-15]. DOI: 10.3991/ijim.v8i4.3974. ISSN 18657923. Available at: http://online-journals.org/index.php/i-jim/article/view/3974

[62] ROCKENSCHAUB Dominik. *Entwicklung und anwendung einer systematischen vorgehensweise zur analyse marker basierter augmented reality frameworks für mobile endgeräte.*. Master's thesis, Johannes Kepler Universität Linz, Linz, Austria, 2012.

[63] SLANT *What are the best 3D game engines?* [online]. Slant [cit. 2016-05-14]. Available at: http://www.slant.co/topics/1495/~3d-game-engines

[64] MATTERHACKER *Finding the Right 3D Modeling Software For You* [online]. MatterHacker posted 26-8-2015 [cit. 2016-05-14]. Available at: https://www.matterhackers.com/articles/finding-the-right-3d-modeling-software-for-you

[65] *Blender* [online]. Blender Foundation [cit. 2016-05-14]. Available at: https://www.blender.org/

[66] *Execution Order of Event Functions* [online]. Unity Documentation [cit. 2016-05-14]. Available at: http://docs.unity3d.com/Manual/ExecutionOrder.html

Appendix A

Acronyms

6DOF Six Degrees Of Freedom

9PP 9 Pre-Patterns

A-GPS Assisted Global Positioning System

API Application Programming Interface

APK Android Application Package

AR Augmented Reality

CCS Camera Coordinate System

CG Computer Graphics

COP Center Of Projection

CPU Central Processing Unit resources

CTU Czech Technical University

CVB Computer vision-based

DCS Display Coordinate System

DDL Data Definition Language

DoG Difference of Gaussians

ECEF Earth-Centered, Earth-Fixed format

ECS or CS Euclidean Coordinate System

ENU East-North-Up format

GNSS Global Navigation Satellite System

GPS Global Positioning Systems

GPU Graphics Processing Unit resources

LCS Local Coordinate System

MEM Memory resources

MEMS Multi-axis Miniature Mechanical System

NFT Natural Feature Tracking

OS Operating System

OST Optical See-Through technology

POI Point Of Interest

QR code Quick Response code

RBI Reality-Based Interactions

SAD Sum Of Absolute Difference

SDK Software Development Kit

SIFT Scale Invariant Feature Transform

SSD Sum Of Squared Difference

UI User Interface

URL Uniform Resource Locator

VR Virtual Reality

VST Video See-Through technology

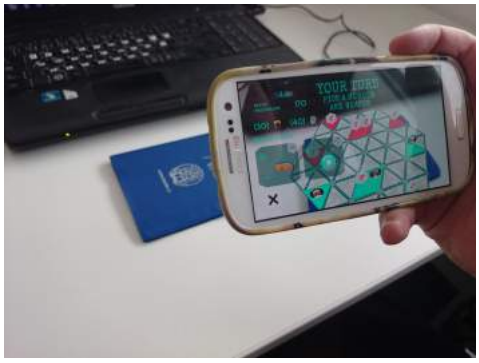
WCS World Coordinate System

XML eXtensible Markup Language

Appendix B

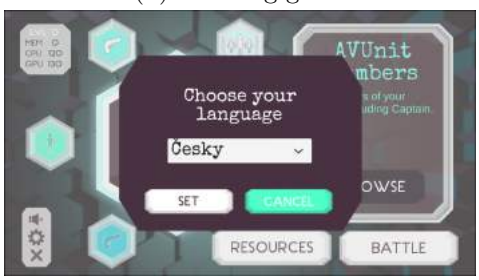
Screenshots and photos

(a) 3D registration



(b) Running game

(c) Main menu



(d) Change language

(e) Browsing AVUnit members category

(f) Deleting AVUnit member

(a) Browsing existing equipment category

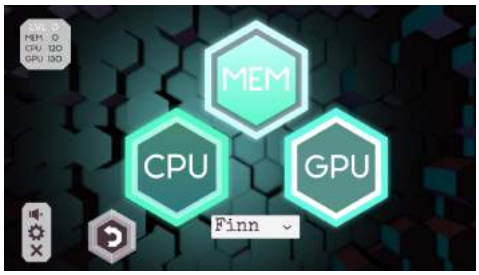
(b) Deleting existing equipment

(c) Coping Captain's code

(d) Copied and edited code

(e) Browsing available equipment category

(f) Creating equipment



(g) Minigame menu

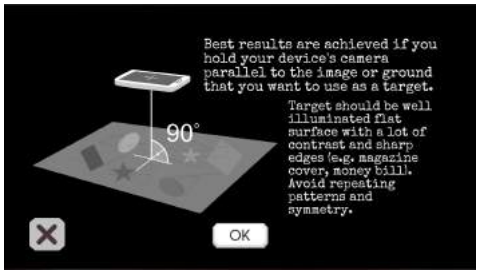
(h) Minigame: Pairs matching

(a) Pairs matching: Out of moves

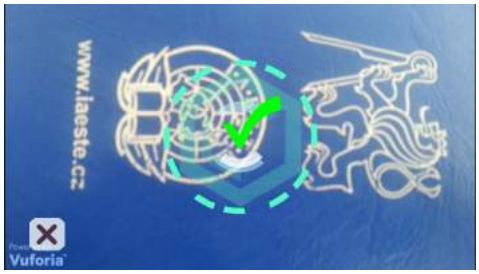
(b) Selecting members for battle

(c) Selecting equipment for battle

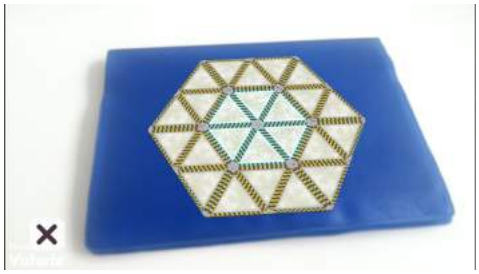
(d) Selecting platform for battle



(e) Instructions for 3D registration

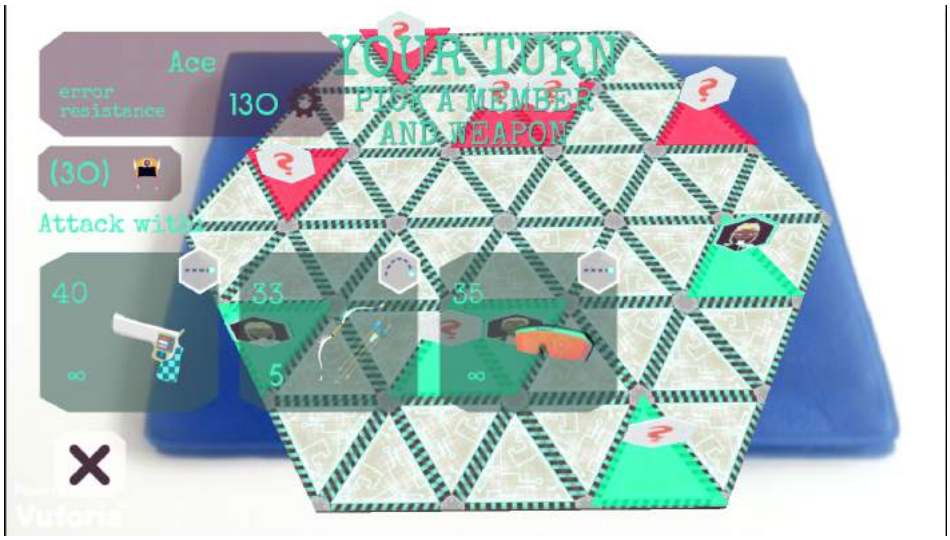


(f) 3D registration

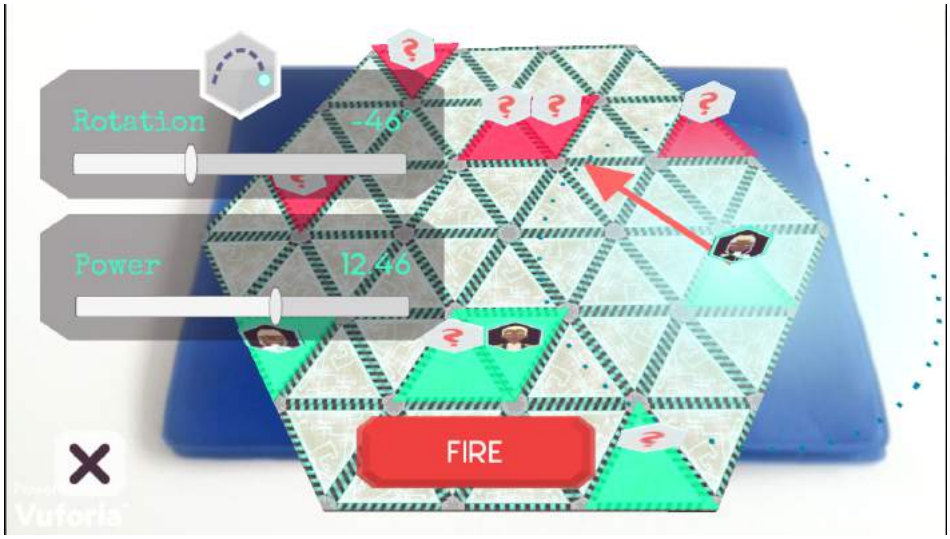


(g) Initial animation

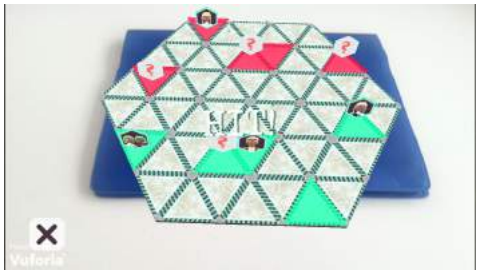
(h) Start of player's turn



(a) Selected member



(b) Selected weapon



(c) Reply about player's action

(d) Reply about opponent's action

Appendix C

Contents of enclosed CD

- `README.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `apk` ..... directory with application in apk format
  - `AVUnit` ..... directory with Unity project
    - `Assets` ..... directory with materials used in the project
      - `Scripts` ..... directory with scripts used in the project
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format