BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUGMENTED REALITY FOR DESK BOARD GAME

ROZŠÍŘENÁ REALITA PRO DESKOVOU KARETNÍ HRU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JIŘÍ RICHTER**
AUTOR PRÁCE

SUPERVISOR **Ing. VÍTĚZSLAV BERAN, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Master's Thesis Specification/18692/2017/xricht19

Brno University of Technology - Faculty of Information Technology

Department of Computer Graphics and Multimedia — Academic year 2017/2018

Master's Thesis Specification

For: **Richter Jiří, Bc.**
Branch of study: Computer Graphics and Multimedia
Title: **Augmented Reality for Desk Board Game**
Category: User Interfaces

Instructions for project work:

1. Get acquainted with the principles of creating graphical user interfaces and with the testing of user experience. Get acquainted with the methods of image and depth data processing.
2. Select a suitable card game and design a system that will recognize the game situation using an image or a depth data processing and that will project the relevant information back to the scene.
3. Create the annotated data set for the system training and testing.
4. Select suitable tools and implement the designed system.
5. Perform the user tests and evaluate the user experience.
6. Create a poster and a short video presenting the key results of your work.

Basic references:

- R. Unger and C. Chandler. *A Project Guide to UX Design: For User Experience Designers in the Field or in the Making* (2nd ed.). New Riders Publishing, 2012.
- Gary R. Bradski, Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*, ISBN 10: 0-596-51613-4, September 2008.
- D. Schmalstieg and T. Höllerer. *Augmented Reality: Principles and Practice*. Addison-Wesley Professional, ISBN-10: 0321883578, 2016.

Requirements for the semestral defense:
Items 1, 2, 3 and partly item 4.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Beran Vítězslav, Ing., Ph.D.**, DCGM FIT BUT
Beginning of work: November 1, 2017
Date of delivery: May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

Jan Černocký
Associate Professor and Head of Department

Abstract

The aim of this work is to create a system, which will enhance the experience from playing the board and especially the card games, by projecting a relevant multimedia information in a game area. At first, the benefits of card and board games are described. Next, the principles of Augmented reality and their usability during design of such system are presented. The designed system is divided into three modules. First of them is module, which analyze the state of game in the real-world — provide the information about player activity and the types of game objects which are presented in a game area. Second of them is module, which control whole system, provide user interface for its calibration and implements game state space, for which the system is intended. Third module provides output of the system — it creates a multimedia content, which is projected in a game area, and it is relevant to current game state. Finally, the designed system is implemented for card game Bang. Its ability to read the real-world game state is measured and user experience is tested.

Abstrakt

Cílem této práce je vytvořit systém, který bude vylepšovat zážitek z hraní deskových a především karetních her, promítáním relevantních multimediálních informací do herního prostoru. V první části jsou popsány přínosy deskových a karetních her. Dále jsou představeny principi rozšířené reality a jejich použití při návrhu takového systému. Navržený systém je rozdělen na tři moduly. Prvním z nich je modul, který analyzuje stav hry na stole v reálném světe — poskytuje informace o aktivitě hráčů a jaké typy herních objektů se vyskytují na stole. Druhým je modul, který se stará o řízení celého systému, poskytuje uživatelské rozhraní pro kalibraci systému a implementuje stavový prostor hry, pro kterou je systém určen. Třetím je modul, který poskytuje výstup systému — vytváří multimediální obsah, který je po-té promítnut na stůl, a je relevantní k aktuálnímu stavu hry. Na závěr je navržený systém implementován pro karetní hru Bang, otestována jeho schopnost udržovat krok se stavem hry v reálném světě a provedeno testování na uživatelých.

Keywords

augmented reality, card games, board games, user interface, user experience, projection on table, calibration, card detection, card classification, card game Bang

Klíčová slova

rozšířená realita, karetní hry, deskové hry, uživatelské rozhraní, uživatelská zkušenost projekce na stůl, kalibrace, detekce karet, klasifikace karet, karetní hra Bang

Reference

RICHTER, Jiří. *Augmented Reality for Desk Board Game.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vítězslav Beran, Ph.D.

Rozšířený abstrakt

Deskové a karetní hry se znovu stávají velmi populárními. Lidé je vyhledávají pro jejich sociální faktor, který přínášejí a protože je baví. Dalším dnes populárním tématem je rozšířená realita, tedy kombinace reálné a virtuální informace. Cílem této práce je zkombinovat principy virtuální reality a deskových a karetních her s cílem zvýšit zážitek, který hry přinášejí. Nejprve se práce věnuje přínosům deskový a karetních her a principům virtuální reality. Dále je navrhnut systém, který má tuto kombinaci umožnit. Nakonec je navrhnutý systém implementován pro karetní hru Bang.

Pro kombinaci reálné a virtuální informace byl navržen „Experience Improvement System“ systém. Tento byl rozdělen na tři základní moduly, které odpovídají základním funkcím, které jsou nutné pro běh systému. Jaké jsou tyto tři základní funkce?

První z nich je funkce systému sledovat stav reálného světa. Protože systém kombinuje dva druhy informací — reálnou a virtuální — je nutné, aby tyto informace spolu korespondovali. K tomu je nutné znát stavy obou světů — reálného i virtuálního. Byly zanalyzovány informace ve vztahu k deskovým a karetním hrám, které musí být systém schopen sledovat a navrženy metody pro jejich detekci. První informací je „Kdo je aktivní hráč?“. Pro získání této informace byla použita metoda *Background subtraction — Odečtení pozadí*, která umožňuje zachytit pohybující se objekty v obraze. Jako je ruka aktivního hráče. Druhou informací je „Jaké herní objekty jsou v prostoru hry a na jakých pozicích jsou tyto umístěny?“ Protože herní prostor může obsahovat více stejných herních objektů, byl prostor rozdělen na předem definované části, s informací, které objekty se v takových částech mohou vyskytovat a jakému hráči patří. Toto rozdělení není nutné, pokud lze pouze podle typu objektu poznat všechny relevantní informace. Dále byly navrženy dvě metody pro detekci a klasifikaci objektů. První z nich je *Template matching* nad gradienty obrazů. Druhou metodou je detekce a popis zajímavých bodů v obraze pomocí metody *SURF* a metodou pro hledání nejbližších sousedů je vyhodnocena podobnost.

Další funkcí systému je zrcadlit stav z reálného světa. Důvod k zrcadlení je ten, že ne vždy je možné zjistit stav reálného světa jen z aktuální informace ale je potřeba mít informaci i o sekvenci stavů, které k aktuálnímu stavu vedli. Tento modul tedy obsahuje stavový prostor hry z reálného světa. Dále tento modul řídí celý systém — žádá informace o stavu první modul, popsaný výše a poskytuje informace třetímu modulu, který je popsaný níže. Modul také poskytuje uživatelské rozhraní pro nastavení a kalibraci systému.

Poslední funkcí systému je poskytovat multimediální obsah uživatelům. Tento modul překládá kontrolní sekvenci od řídícího modulu a vytváří výstupní informaci — obraz, který je pomocí projektoru přenášen do herního prostoru. Efekty, které tento modul vytváří jsou z důvodu zachování jednoduchosti omezeny na statické obrázky.

Navíc je součástí systému také kalibrační proces s asistencí uživatele. Uživatel definuje herní prostor pomocí čtyř markerů pro rozšířenou realitu a spustí kalibrační proces systému. Výsledkem kalibrace systému je schopnost systému brát relevantní informace jen z definovaného herního prostoru a používat stejný souřadný systém pro detekční i výstupní část. To je nutné, aby výstup systému byl správně umístěn vzhledem k reálným objektům.

Navržený systém byl implementován pro karetní hru Bang. Protože se jedná o karetní hru, která má volně definovaný herní prostor, což často vede k přehlédnutí důležité karty, bylo prvním vylepšením pevné definování herního prostoru. Byly definovány hráčské pozice v herním prostoru — kvůli detekci aktivního hráče. Dále byly definovány jednotlivé pozice, kam je možné položit karty patřící konkrétnímu hráči. A pro podporu přidání efektů, byla definována oblast, kam se budou tyto efekty umisťovat, aby byly vždy viditelné. Protože

hra Bang obsahuje komplexní stavový prostor, byla implementována pouze podmnožina stavů, pro demonstraci správnosti návrhu.

Navržený systém byl otestován. Nejdříve byla vyhodnocena schopnost systému určit korektně třídu karty z množiny možných karet. Byly otestovány obě navrhované metody z následující úspěšností — *Template matching* 49%, *SURF* a hledání nejbližších sousedů 63,5%. Nedostatečné výsledky klasifikace karet jsou způsobený nedostatkem detailů v obrazech získaných z reálného prostředí a přílišnou podobností karet mezi sebou. Dále bylo provedeno uživatelské testování, ve kterém proběhlo testování detekce aktivity uživatelů a testování uživatelské zkušenosti pomocí předdefinovaných scénářů.

Návrh systému se podařilo ověřit vytvořením reálného systému pro karetní hru Bang. Úspěšnost systému v klasifikaci karet by se dala výrazně vylepšit označením karet pomocí lépe detekovatelných znaků. Další práce na implementaci systému by se týkala optimalizace rychlosti systému a rozšíření systému o další efekty, včetně animací.

Augmented Reality for Desk Board Game

Declaration

Hereby I declare that this masters's thesis was prepared as an original author's work under the supervision of Ing. Vítězslav Beran, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Jiří Richter
23. May 2018

Acknowledgements

I am thankful to Ing.Vítězslav Beran, Ph.D. for mentoring, patience and valuable advice during time I was writing my thesis.

Contents

Chapter 1

Introduction

Playing card and board games has always been the popular way of spending time with friends, family and even complete strangers. As the social interaction and even the interaction with real objects is the core of experience and enjoyment obtained from such games during playing, none of it should be taken from players. To enhance the experience and especially the enjoyment, new element is presented – digital information combined with information from the real-world.

The goal of this thesis is to propose, how to use advanced technologies for natural interaction between human and computer. The requested result of utilization of these method is the enhancing of user experience during their play of the card and board games. The enhancing is achieved by addition of multimedia data into the real world. Thesis contain overview of usable method to achieve the presented goal, principles of Augmented reality are described and the devices to transfer virtual information into the real-world are presented.

To achieve a goal to enhance the real-world with additional data created in computer, new system is designed. At first, the requirements and key issues of such system are presented. Next, the methods to solved the key issues are proposed and system is divided into its key parts. The key parts and communication between them is designed. For each key parts, a methods, a procedures and technical specifications to fulfill its task are proposed.

Finally, the designed system is implemented for card game BANG. The realization for every key part of designed system is described. When designed system is implemented testing is performed. The testing is composed from system parts reliability test and user testing.

Chapter 2

Card games and advanced reality

In this chapter, why people are playing card and board games and senses their are using during play-through are discussed. Next, in what environment are the games played and what are key parts of these environments is described. What is Mixed reality, and how useful it can be for the games, follows. Finally, the technology and algorithm useful for combination of card and board games and mixed reality are introduced.

2.1 Games and their perception

There are various benefits of card and board games. Board games have been studied in cognitive psychology research[3, Chapter 1], which is a study about mental processes, such as perception, thinking, learning and memory[1]. Many studies were performed to evaluate the effects the games have on these mental processes. The games may improve perception, creativity or even logical thinking and almost all of them are using a social interaction. The benefits of the game are highly dependent on a logic and a system of the game. But the main point of any game is for players to have fun, enjoy what they are doing, and to let the players self develop without knowing that their are studding or perform any kind of training.[2]

Therefore, the objective of most games is to make an experience and an enjoyment players receives from the game as memorable as possible, otherwise the game would not be played. This is achieved by acting on players senses. During the design of every game, the key part are discussed – randomness, theme, game components, number of player, etc. [4] All of these are important, but in this thesis, the focus will be placed on another one – how a game interacts with players.

Players perception

The player can percept information through all of his senses. It is hard task to order the senses according to their importance during play, because again, it depends on the type of game. But in general, the sight is significantly used during all daily task and playing of the game is not an exception. When interaction between the players occur, the hearing is often taking a role too. Another widely used sense is touch, which taking its part when an interaction with the game objects occur. Using of the other senses — smell and taste

[1] https://www.merriam-webster.com/medical/cognitive%20psychology
[2] http://www.healthfitnessrevolution.com/top-10-health-benefits-board-games/

Figure 2.1: Examples of the game areas.

— depends on the type of the game being played, but in general their use is rare. If more senses are used, the more options are available to stimulate the player with.

The player senses, by which game areas are perceived, were discussed above, but let's discuss ways a player and a game communicate with a simple example. The game objects – such as cards, figures, cube, etc – are used to communicate. The player uses these game objects to progress in a game, which mean that with game object the player is communicating with game and progress the game into new state. But these same objects are also used to communicate between the players them selfs. Apart from the game objects, the players are using speech and body language between them.

Game environment

Game environment can be separated in the area where the players reside – player area – and the area, where the game components are – game area. The player area is usually variant and unstable but the game area is often predefined, specified in game rules. The examples of game areas are shown on figure 2.1. As these can differ for each game only subset will be consider for our case – the game area must be limited and it is possible to cover it with camera. The game objects must be placed inside such game area and must be detectable in camera image of this area. The game objects may vary in the size, shape, color, etc. The examples of existing game objects are shown on figure 2.2

Game user interface

As the game already contain user interface defined by game designer, this should be respected. The game has its design, and if any extension for the game is created, the concepts of *unity and variety* [14, p. 187] should be used. The new extension should contain the elements which to some degree correspond to the design of the base game. The players should have the feeling it is still the same game, not something created separately. On the other side, if the extension is not bringing anything new, there is no point in creating it in the first place. It should have some degree of variety. As most of players are familiar with base game, the news should not be dominant over the base. They should complement each other. The game extension can enhance the game in several ways. It can extend the game logic, add new game objects or the extension can contain only visual changes.

Figure 2.2: Examples of the game objects.

2.2 Mixed reality

In this chapter the concept of Mixed Reality is presented and a parameters for technical devices needed to use it are described. Finally, the example of such devices are shown.

Mixed Reality concept

The goal is to enhance the real-world and its interaction with people, especially with their vision and hearing. The digital and the real-world is combined and presented to user. This leads to the need to have methods to offer such combined information to users. This is a base for the principles of the **Mixed reality**. Mixed reality holds the promise of creating direct, automatic, and actionable links between the physical world and electronic information. It provides a simple and immediate user interface to an electronically enhanced physical world [11, p. 2].

As discussed above, combination of real and virtual world creates mixed reality. As seen on figure 2.3, the levels of contribution from real and virtual world can differ – this is called *virtuality continuum*. When the physical world provides more information than virtual world – the *Augmented reality* is created. On the contrary, when the virtual world provides more information than physical world, the *Augmented virtuality* is created [11, Chapter 1] [7].

The link between the physical and virtual world needs to be established. To achieve this, the states of both world must be always known and one has to correspond to other. Usually the virtual world state is keeping the peace with physical world, so the state of real-world needs to be provided to virtual one.

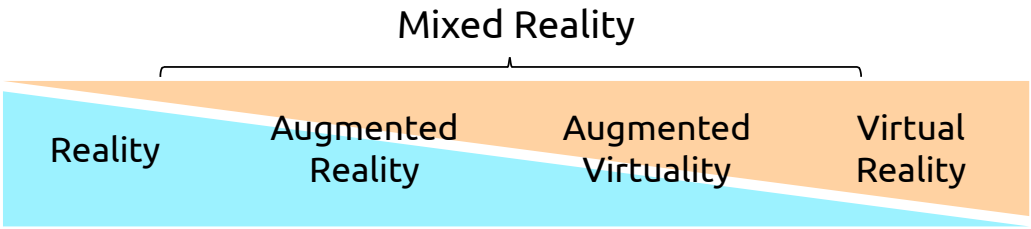


Figure 2.3: Virtuality continuum, all possible combinations of the real and virtual worlds[11, p. 29].

Mixed Reality components

The complete AR system requires at least three components: a tracking component, a registration component and a visualization component. A fourth component can be used — a spatial model. It stores information about real and virtual world. The real-world model is required to serve as a reference for the tracking component, which must determine the user's location in the real world. The virtual-world model consists of the content used for the augmentation. Both parts of the spatial model must be registered in the same coordinate system [11, p. 5].

Tracking component

A way to read the physical world state – it is information – and send it to virtual world is needed. A camera, an optical tracking component, is suitable for this task[11, p. 105]. But is a camera enough to read the real world state? Certainly not. The state of physical world is too complex to cover it with simple image. Some help can bring using more images – video. This provides time context to image. To use more sophisticated cameras, like infrared camera is also possible [11, p. 106-108]. Different sensors – touch screen, thermometer, etc. – can also be used. The use of more sensors to obtain additional information is called Cooperative Sensor Fusion [11, p. 118-120]. With more sensors the better description of the real-world state can be obtained but with the costs of more computation needs. The data from sensors needs to be combined. There can also be an issue with timing as the state is keep changing. For this reason, it's better to use only units of complex sensors like cameras with additional simple sensors like for example thermometers. Luckily, for the goal to enhance the user experience during game play, there is no need to read whole state of physical world, but only some predefined parts of it. For example – *Is human interacting with table and objects on it?*

Registration component

A registration component is the next mandatory component. That is a component, which maps the virtual object to the real-world coordinate system.

The inseparable process to perform registration is calibration. Calibration is the process of comparing measurements made with two different devices, a reference device and a device to be calibrated. The reference device can be replaced with a known reference value or, for geometric measurement, with a known coordinate system. The objective is to determine parameters for using the device to be calibrated to deliver measurements on known scale [11, p. 87].

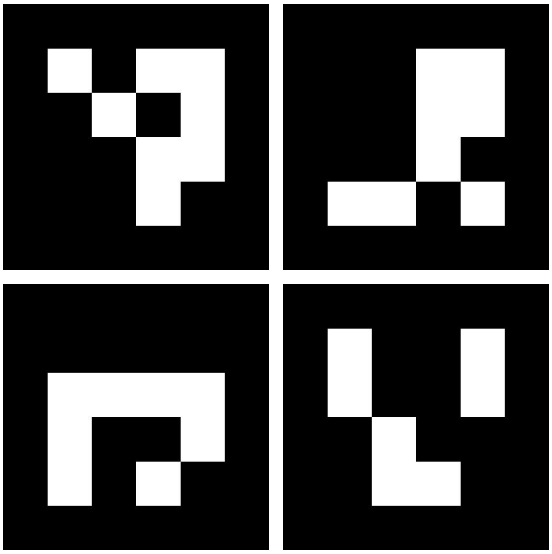


Figure 2.4: AR makers from arUco library.

As the registration is complex task and create fast and universal approach with the real-world objects is more resourceful demanding. The quality of data – images from camera – must be higher and more computational resources is required [11, p. 112], the workarounds were created. The most ones used these days are **Augmented Reality markers (AR markers)** [11, p. 109]. These are placed in real-world and creates the clues for registration. Their advantage is that it is easy and fast to find them in image using methods for image recognition. Examples of markers, from OpenCV library arUco, are shown on figure 2.4

Visualization component

When the additional electronic information is created we need a visualization component to hand over such information back to real world. This must be able to affect some human sense, and the requirements are specific to the sense if affects.

As discussed in section 2.1 of human perception the most used senses are sight, hearing and touch. Because human received most of the information about surrounding using sight, there exists a glass like devices these days. A principles of these devices differs and these are based on two different methods of reality augmentation – Optical see-through (OST) and Video see-through (VST) [11, p. 40-41]. The functionality principles of these devices are shown on figure 2.5. The existing examples of these devices, the Microsoft Hololens and HTC Vive, is shown on figure 2.6.

An OST devices rely on optical element that is partially transmissive and partially reflective to achieve the combination of virtual and real [11, p. 40]. A VST devices achieve the combination of virtual and real electronically. A digital image of the real-world is captured through a video camera and transferred to the graphics processor which combine it with the computer generate images [11, p. 41].

Although a glass like OST devices works well these days there are still limits in a way it can be used – mostly technological. These days the most crucial limitation for such devices is Field of View (FOV) [5]. As demonstrated on figure 2.7 the currently existing devices covers only small partition of human field of view. This breaking apart the combination of

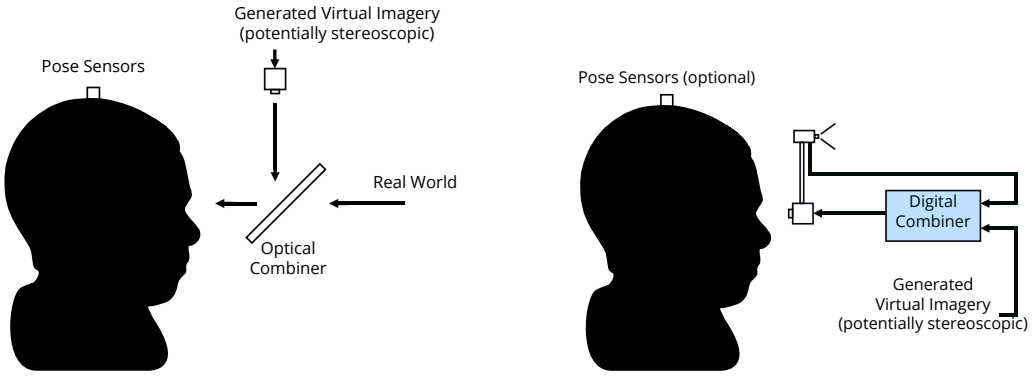


Figure 2.5: Principles of OST(left) and VST(right) device.

Figure 2.6: Existing OST(left) and VST(right) device.

Figure 2.7: Field of view limitation of current OST devices.

real and virtual world. Another crucial thing for OST displays is to balance the brightness and contrast. When there is too much light, the virtual information will be too dimmed to be seen. It may seem that it is reasonable to use VST devices, but these also have limitations. For example, VST are not suitable for dangerous work because when the error occurs and the display turn off the user cannot see anything. A latency can also cause accidents. The next limitation, common for OST and VST devices, is the distortion and aberrations. Every concrete display will involve optical elements, such as lenses. These optical elements are not perfect and these can introduce distortion, such as fish-eye effect, if the wide field of view is desired. [11, Chapter 2]

A glass like device is taking a main role in many discusses these days as the displays of the future [6]. But they still need time to develop technologically. It is uncomfortable to have them put on for several hours as they weight around 500g [3] and the head of user usually start to sweat. To overcome the need to have display attached to the head, next category of displays for Mixed reality is presented – Projected Displays.

Projected Displays using the projector as the device to transfer virtual information to the real-world. With the use of projector, the *Spatial Augmented Reality* [11, p. 78] can be created, without any explicit displays. With this approach, the projection is directly reflected from surfaces of real objects, altering their appearance. The projection cannot change the shape of the object, but adds surface details, texture, shadows and shading, and even the impression of dynamic behavior, if animated content is projected [11, p. 78]. Unfortunately, even these devices have limits. Shading from the users moving in environment or the limited focus distance of projector – this can be solved by using more projector with different focus distance. With the information of user position, the *View-Dependent Spatial Augmented Reality* can be made. This allow the projected objects appear anywhere in space, not just on object surface. Unfortunately, the projects objects can be properly seen only by the user, which position is tracked [11, p. 79].

[3] https://en.wikipedia.org/wiki/Comparison_of_virtual_reality_headsets

Figure 2.8: Handheld AR device SmartPhone with popular AR game *Pokemon GO*.

Another device popular these days for augmented reality usage is smart phone. It belongs to category of **Handheld displays** [11, p. 69]. A smart phone houses both the actual display and a tracking component camera rigidly mounted in a casing – the transformation from display to camera can be pre-calibrated. It also has built in accelerometer and other sensors, which can be used for pose estimation and tracking. Usage of smart phone for augmented reality is shown on figure 2.8

2.3 Change detection

This section describe a technique which enables to detect the changes in video or in the sequence of consequential images. This technique is called **background subtraction**, sometimes term foreground detection is associated with the same technique.

In order to perform background subtraction, we must first "learn" a model of the background. Once learned, this *background model* is compared against the current image and then the known background parts are subtracted away. The object left after subtraction are presumably new foreground objects [2, p. 265-266].

How the **background and foreground** is defined? If camera is pointed to a parking lot and a car come and park, then this car is a new foreground object. But it should not stay foreground forever. Another example, when the background object is moved, it will show as foreground in two places: the place it was moved to and the hole left after the object. Or if the lightning of a scene will change, should all object be foreground? To be able to cover all these examples, the higher-level scene model, in which the multiple levels between the background and foreground states are defined. Also, the timing-based method of slowly relegating unmoving foreground objects to background objects. When the global change in scene occur, the training of a new model is needed.

If you have image of background alone, like image of the room without visitors, image of the road without cars or image of the game area without players, etc., it is an easy job. Just subtract the new image from background and the foreground is what is left. It becomes more complicated when the foreground is casting shadow, which is moving together with it. Example of the background subtraction is on figure 2.9.[9]

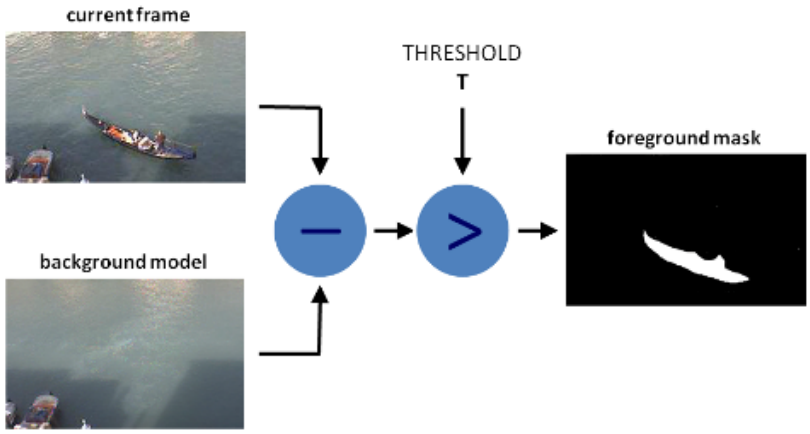


Figure 2.9: Example of performed background subtraction of known background model.
Source: https://docs.opencv.org/3.4/Background_Subtraction_Tutorial_Scheme.png

2.4 Object classification

The system, which have to reacts on the real information at scene, needs to have information about the current situation in the real world. One of the solution that is used these days is camera shooting and detection of interest points or the object matching – searching for object in camera image. One of the base requirement for the system, which is described in chapter 3 is to be able to detect the game object in scene and classify it to predefined classes – types of game objects.

In this section, the two methods to detect and classify objects are described. Each method uses different approach, first of them is using template matching – the template is compared to camera image and the score based on evaluation function is calculated. Second of them is using image matching – interest points are located and described in template and in an camera image. The score is calculated by comparing the the descriptors of interest points.

Template matching

The first method, called Template matching, is often used, if an image contains some previously defined object or, in particular, if a pre-defined sub-image is contained within a test image. The sub-image is called a template and should be an ideal representation of the sought pattern. The matching technique involve the translation of the template over the image and the evaluation method is calculating the similarity in every position. Result is a score which represents it. Score is used to determine if the template is contained in an image and its location in image is also obtained as result of matching [15, p. 119].

Based on evaluation method used, we are looking for a maximum or a minimum score. For example, similarity function, based on cross-correlation is calculated using equation 2.1. The result of performed template matching is shown on figure 2.10. For this method, the maximum value (white) is the most possible position of template in image. The result of equation 2.1 is a score which represent the similarity among other values, but to receive score between $0 - 1$, where value 1 is perfect match and 0 is no similarity the normalized evaluation function has to be used. Such function for cross-correlation evaluation method is in equation 2.2 [10].

Figure 2.10: Input image (top left), template (top right) and results (bottom) of template matching.

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x', y+y')) \tag{2.1}$$

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x', y+y'))}{\sqrt[2]{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}} \tag{2.2}$$

As the evaluation method is working with the values of pixels in template and in an image, the problem with high dependency on brightness with usage of raw images from camera, may appear. This can be solved by preprocessing the image and template into brightness non-dependent form, like gradient description of an image. To transform image into his gradient form the Sobel kernel can be used. It is applied on image using convolution. The Sobel kernel of size 3x3 is shown on figure 2.12.

The another feature of template matching methods is that they are not invariant to rotation and scale [15, p. 121]. To overcome the scale problem, template is moved over an image multiple times. At first, the image is in his full size but with every template matching pass, the image is scaled to smaller sizes. To overcome the rotation variance, attempt to rotate input image, to match the template rotation, is made. This can be performed using *Hough transform* to find the lines and their direction in an image. The expectation is made, that the most lines in image are horizontal or vertical – as the tables and walls in the real-world are. From lines direction the rotation angle is found and image is rotated.

Figure 2.11: Input image (top left), template (top right) and results (bottom) of template matching performed on gradient of images.

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 2.12: Sobel kernel 3x3 to detect gradient in X-axis (left) and Y-axis right.

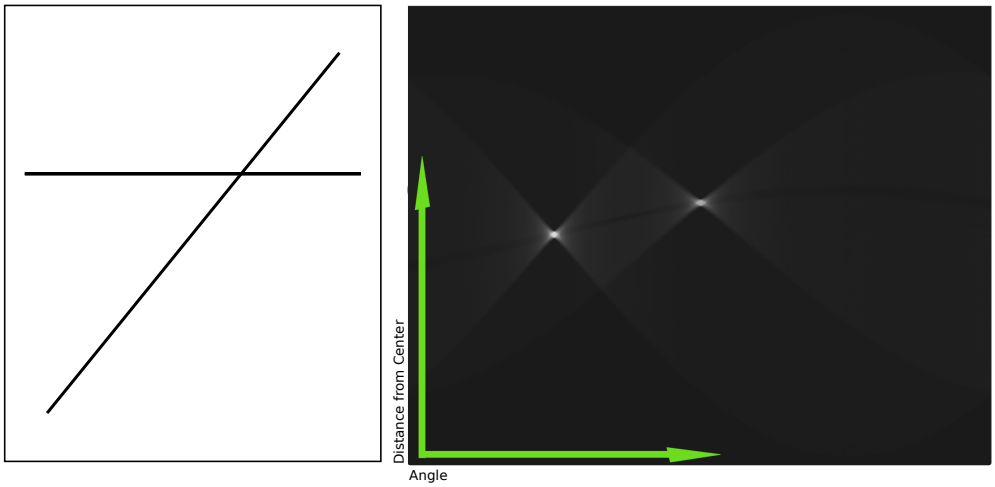


Figure 2.13: Input image (left) and result of Hough transform (right).

Hough transform

Hough transform is method to detect shape which can be represent in mathematically form like equation. The shape is transformed from image to parametric space. It is often used to find lines, circles and ellipses. For example, the lines can be described mathematically in slope intercept form using equation 2.3, where m is a slope and c is Y-intercept point. Or in a parametric form using equation 2.4, where ρ is the length of a normal to the line from origin and θ is the angle this normal makes with the X-axis [15, p. 130].

$$y = mx + c \tag{2.3}$$

$$\rho = x \cdot cos\theta + y \cdot sin\theta \tag{2.4}$$

Input to Hough transform is usually the output of edge detector applied on an image. Coordinates of every pixel, where the edge was detected is transformed to parametric space. All pixels, which lies on same line in input image are projected into same coordinates in parametric space. Because during edge detection an inaccuracy can occur, as a consequence, pixels of one line are projected into small area, around the point, which represents a line. Finally, the maxims are found in a result image of Hough transform. Each maximum represents one line in an input image. Example of Hough transform line detection is shown on figure 2.13. The two white point on right image represent the two lines on the left.

Image matching

Another method to find objects in image or compare image similarity is based on detecting and describing interesting points in a template and in an image, which are then compared. The one representative of method for detecting and describing interesting points, called SURF – Speeded-Up Robust Features, is described below. Comparing of interest points descriptors can be performed using nearest neighbors algorithms.

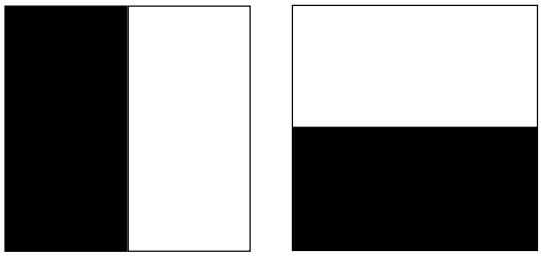


Figure 2.14: Haar-wavelet.

SURF

At first, „interest points“ are selected at distinctive locations in an image, such as corners, blobs, and T-junctions. Next, the neighborhood of every interesting point is represented by a feature vector. Finally, the descriptor vectors are matched between different images. Matching is often based on a distance between the vectors. The high advantage of SURF detection and description of interest points is that it is scale and rotation invariant [1, Chapter 1].

The base of SURF detector is the Hessian matrix, for both the location and the scale. The blob-like structures at location where the determinant is maximum are detected. The Hessian matrix is calculated as follows. Given a point $x = (x, y)$ in an image I, the Hessian matrix $H(x, \rho)$ in x at scale ρ is defined in equation 2.5, where $L_{xx}(x, \rho)$ is the convolution of the Gaussian second order derivative with the image I in point x, and similarity for $L_{xy}(x, \rho)$ and $L_{yy}(x, \rho)$ [1, Chapter 3].

$$H(x, \rho) = \begin{bmatrix} L_{xx}(x, \rho) & L_{xy}(x, \rho) \\ L_{xy}(x, \rho) & L_{yy}(x, \rho) \end{bmatrix} \tag{2.5}$$

When the „interest points“ are detected, the way to describe them is needed. Around every interest point a circular region is chosen, then from such region, the square region is chosen aligned to selected orientation. This way the SURF invariant to rotation is achieved. The rotation of interest point is calculated using responses in x and y direction on Haar-wavelet, shown on figure 2.14. All the responses must be normalized to the scale, to obtain relevant information. Once the responses are calculated and weighted, these are represented as vectors [1, Chapter 4].

Nearest neighbors search

The nearest neighbors search is a problem of finding the point closest to the given point in high-dimensional data. This problem is trivial in 2-dimension space, where the linear approach can be used. We can simply calculate the Euclidean distance between two points – p and q – using equation 2.6. But with more dimension the search begin to be computational expensive operation, especially when there exist many points to check.

$$d(p, q) = \sqrt[2]{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{2.6}$$

As searching for the closest matches to high-dimensional vectors is computational expensive, the approximate version was designed and library performing this operation was created. The library is called *FLANN – Fast Library for Approximate Nearest Neighbors.*

Library is working with Euclidean distance vector space, which is appropriate for most problems in computer vision. Library contains several algorithm for nearest neighbor and automatically choosing the best one, with optimal parameters, according to the data set provided, to minimize the search cost of future queries [8, Chapter 1].

Chapter 3

Experience Improvement System

In this chapter "the design of a system with a goal to improve the players experience and enjoyment" from playing the game is presented. The key system functions are described and the modules, which such system needs are designed. The system will respect principles of augmented reality, where virtual information is combined with the real-world one.

The goal of a system is to enhance the user experience from playing the board and card games. How the experience can be enhanced? Multimedia content is very popular these days, its on the Internet, in a television and in a smart phones. People are used to gets a lot of multimedia content every day. Its hard for board and card games to be interesting from multimedia point of view with a limited cost and without using modern technologies. The enhancing of multimedia content for games can also leads to enhancing of the user experience. This leads us to first requirement for system – it must be able to generate and combine multimedia content, and transfer it into the real-world. But, would be the random or predefined sequences of multimedia content enough to enhance user experience? Certainly not, if the multimedia content is not relevant to the game and the real-world state, the effect will be opposite. This leads to another requirement for a system – it must be able to present relevant multimedia content. And it must be relevant to the actions which are currently ongoing in the real-world. More precisely, the requirement is to be able to observe the real-world and detect certain events in it.

3.1 System characteristics

As discussed in chapter 2.2, the state of the real-world (game) is needed. This state is analyzed and according to it, the virtual information is created and presented to users. To be more specific for our case, a limited size game area exists, this area is observed from above with a camera and virtual information is presented to users using a projector suspended above, just like camera. The design of the system is based on this resolution and demonstrated on figure 3.1.

Because the purpose of The Experience Improvement system is to be supplement for the game, the players should not be limited by it. They play the game as they are used to and the system is providing them with extended information about the game state. As we need to successfully communicate with the players, the system need to react as quickly as the game is played. The reaction from system should be delivered to players together with reaction from the game – ideally, when the state of a game changes, the system should react

Figure 3.1: Experience Improvement system environment.

the same time when other players see that the state has changed. **The reaction time of the system** is taking high priority, otherwise the system will not be usable.

As mentioned above, the current state of a game is important information which is mandatory. But what exactly defines the game state? For some games, there may not be simple question, but for some the answer can be pretty simple – like in Ludo, only a few positions on game boards need to be checked. Because, it is challenging task to track interaction between players – people in general, for their chaotic behavior, at least from the system point of view – system will focus on following only the game objects (cards, figures, game boards etc.), which have fixed and easier to describe parameters. To be able to follow the game objects, system needs to know what these are and it needs to have the ability to detect them. For the system design, only such games are considered, where the game state is defined by the game objects in the game area. Examples of game objects can be seen on figure 2.2. The players are using these real game objects during the play. To find out the state of a game, the analysis of game objects on the table is required. To successfully distinguish the game object from the rest objects, the system must learn what is a game object and where it can appear. When a new object appears in the game area, the system must change its inner state to correspond to the state of the game in the real-world. This change of a system state may cause the creation of new information which is in some way provided to players.

The characteristics of the system have been discussed above and it leads us to the design of three modules, that are mandatory for it. The module to observe game objects – **Observer**. The module which represent all available states that can occur in the game and also hold the current game state **Game Control and Logic**. And lastly, the module which will create the multimedia information and provides it to players – **Visualizer**. The design of the system and connection between three modules is shown on figure 3.2.

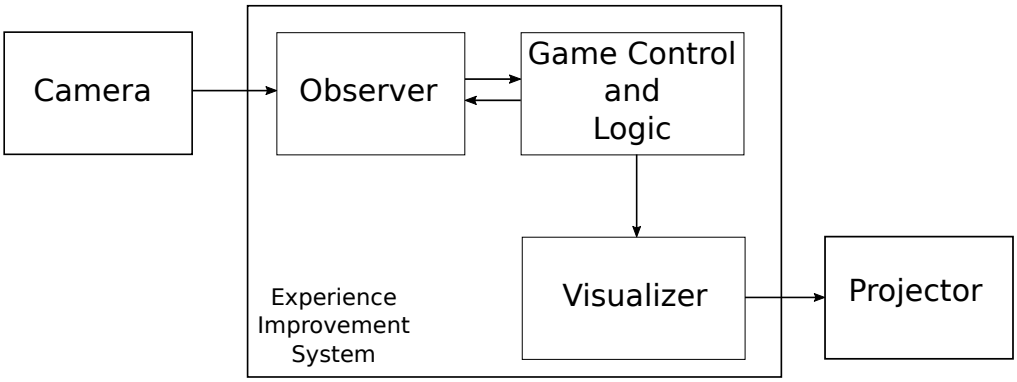


Figure 3.2: Design of Experience Improvement System.

3.2 Observer module

Let's look more detailed on **the input of the system**. The camera observe the game and surrounding area – the example of the game areas are on figure 2.1. It is mandatory for camera to see the whole game area. The removing of the surrounding area was performed by calibration, so only game area is in an image. Before closer look to the game area, let's discuss the problems which come with using only one camera for input.

As demonstrated on figure 3.3, the shadow can occur during the capturing of the image. Especially when the players interact with the game area. Because there is not simple solution for such disturbance of input it will be consider during design of a system and on top of that, it will be used to obtain extra information — who is changing the game area. The analysis of the game area based on objects was not started yet, and the important information about which players are interacting with the game space was already found – there might be more than one player in general. This disturbance can also be used for simple communication between players and designed system – the user simply creates shadow on a specific place in unused regions of the game area, on which the system can react in a preset way.

A simplification was already made in section 2.1 that the game area has a rectangular shape, and can be covered by camera. Now, design a module which will analyze an image of the game area and detect and classify game objects in it is needed. Additionally, the module will also provides the detection of player, which is interacting with a game area described above. There are several differences that may occur between two consecutive images:

- A new game object has appeared in the game area.
- An old game object has been removed.
- The player is interacting with the game area.

It may seem that there exists a fourth differences that can occur. An object has been moved to different location, but that means that some object has been removed from its place and some object has appeared in a new place. So, this option is already covered by the options described above.

The module of observer is implemented as closed separate system, which is providing information described above, which are the key functions of this module. The process of Observer module is shown on figure 3.4.

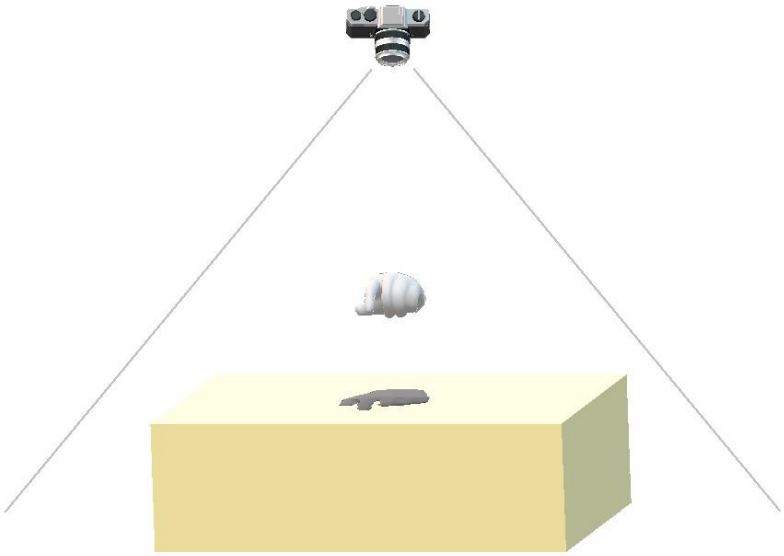


Figure 3.3: Shadowing created by players hand during capturing of the image.

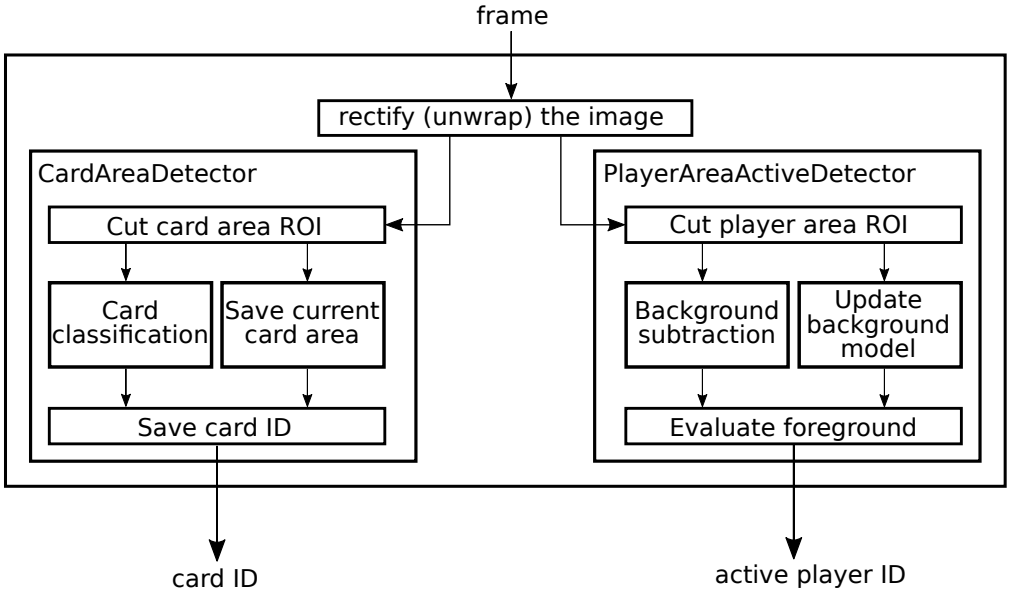


Figure 3.4: Diagram of Observer's inner parts and actions it is performing.

Active player detection

The first information we need to determine is — who is changing the state of the game area. The one who is changing it is considered as an active player. Because the game area is observed from above, if the player interferes with it he will cover a part of it with his body parts — most probably with his hand. The module must be able to recognize such covering and the algorithms for background subtraction is used. The background subtraction method is described in section 2.3. To be able to use this method, we expect that the game area is static in some time window. When we use a camera, which takes a twenty frames every second and the usual reaction time of human, which is in average approximately 283 ms [1], we have at least five frames to learn what parts of the game area are static. This is the worst possibility when the players react immediately without much thinking. More probably, this will take much more time — tens of frames — to think about a best move and reacts. We already know that someone is changing the game area. To determine which player is causing it, a part of the game area is assigned to individual players as seen on figure 3.5. The player which cause the change of the game area is then determined according to the part of the game area in which the change occur.

In digram 3.4 the *Evaluate foreground* action can be seen, this apply evaluation function on image of player area, from which if the player is active is recognized. If more players are active at the same time, the intensity of every active player is returned. The currently active player is then determine by the highest intensity.

As was already mentioned, a player activity is recognized by intensity score – if it is bigger than the upper threshold, player is active. Before the area of the player, which was active, can be considered active again. The intensity score has to decrease below lower threshold, otherwise it is in state *was active*. The background model is updated every time the new image is captured, this way if a new object appears, it is slowly propagated to a background. The detector is set as *was active* from the beginning to let the subtractor learn an initial background model first. The evaluation is calculated using equation 3.1. The evaluation needs to have the thresholds set to appropriate values. The upper thresholds should be set, so the system is able to recognize minimal activity of player which should be recognized as activity, but not to low to cause false alarms. The lower threshold should be set approximately to half of the upper threshold.

$$eval_{area} = \frac{\sum^{areaSize} pixel_{value}}{areaSize} \tag{3.1}$$

Game object detection and classification

The second information we need to determine is – if any game object appear or disappear. At first, detection of the game objects is needed. The placement of the objects is determined by the game rules and for most games, the same type of game object can appear in game area more times. To solve this issue with multiple same objects, the placement of the game objects is predefined during whole game by configuration file described in next section. Next, when the position of game object is determined as on figure 3.6 we need to find out what type of game objects are in this predefined areas. These areas are cut off from an captured image of whole game area and compared using matching methods presented in section 2.4 with representative of every class. It is required to create template for every type of game object, which can appear during the game play. These templates are used

[1] https://www.humanbenchmark.com/tests/reactiontime/statistics,15.1.2018

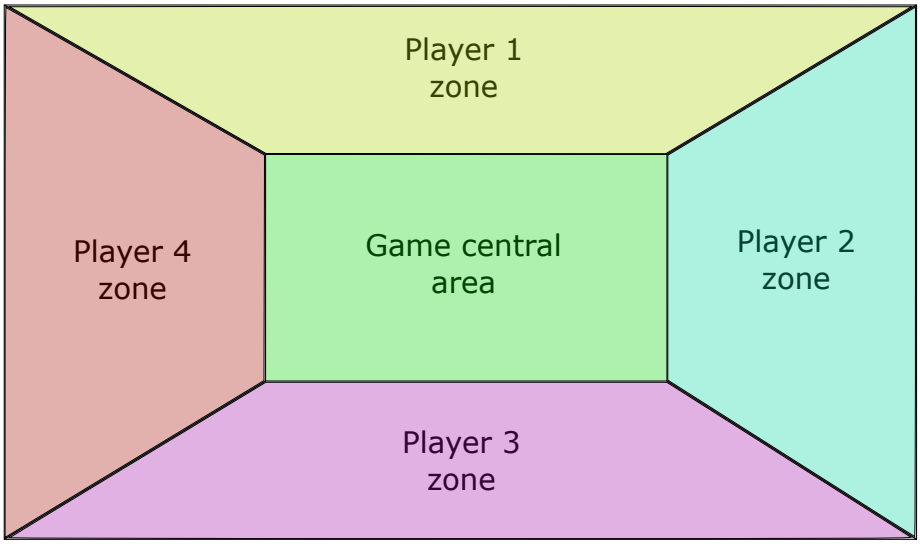


Figure 3.5: Example of the game area separation for individual player with central common area.

for matching method to classify an object. The type of game objects with its position is returned to the Game control and Logic module.

3.3 Game Control and Logic

Game control and Logic module is managing the application run. It holds all possible game states which are supported by the system. The module has two main tasks:

1. Manage the application processes.
2. Mirror game state from the real-world.

The current state of the game is probably the most important information for our system and it is not always possible to determine it exactly from current game objects in game area. Often, the information about several previous states may be needed, to determine it. The system should implement a state space of game or at least some closed subset – according to the game, for which the Experience Improvement System is created, the state space can be extensive. The ability to keep records about previous states in this subset is required. As it can be useful to keep the records about states and the way one state is change into another – for example, for game analytics – the goal of this system is different – to improve the user experience from the game.

Design a different interaction between players and a system is appropriate. There is an option to access the system through already existing components like computer, which will host the system. But that leads to the need, to have the computer presented in a game area or near to it. Also, it means that some player needs to leave the game interface and use the computer's one. As this would be disturbing, to incorporate a control interface of the system into an interface which it provides for game play is appropriate – observer module of system is already able to recognize the changes in game area, which can be also used to control system.

Figure 3.6: The game area of the Royal Goods game with the game objects marked with red rectangle and game areas marked by blue, green and yellow rectangle.

For every action the player performs, the response from a system that that action was done should be received. Otherwise cases, when the user is confused may appear. We need to design a system in a way, that every interaction which affects the system has a feedback.

Managing the application

At first, Observer module described in section 3.2 is contacted to obtain new information – Is any player active? Has any game object changed? The newly obtained information are processed, depending on the types of information acquired the game state is updated – this is described more detailed below in Game state mirroring section.

Next, the drawing update instruction, in response to state change and newly acquired information, is generated and inserted into the drawing update instruction pool. This pool is common for Game Logic and Control module and Visualizer module defined in section 3.4. The update instruction must hold all information required by Visualizer to be able to successfully generate appropriate effect. The information needed in update instruction depends on the game, the system is created for, and on the effects its supports. But in general at least these information should be contained:

- player – ID of active player, which cause a state change
- game object position – ID of position of game objects, which cause a state change
- state – new state of a game

Game states mirroring

The new information are acquired regularly and applied on current game state, starting with the default one. All possible game states and transitions between them are stored in the form of finite-state automaton. The example of automaton is shown on figure 3.7.

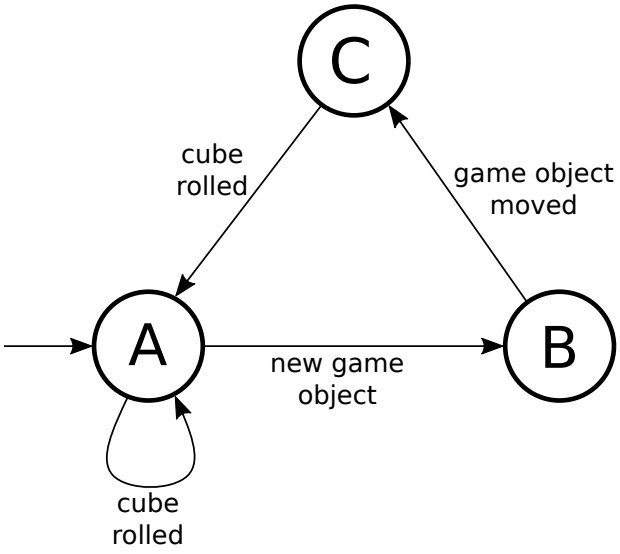


Figure 3.7: Game state automaton example.

The module have to mirror the real-world state. Every time the state of the game in the real-world is changes, **the same change** must occurs in the module according to data obtained from an Observer. As this module is highly depended on a game for which the Experience Improvement System is implemented, specifics are left for a creator. Especially the state machine design, which implements a state space of a game. The design should respect the specifications discussed above.

Configuration loading

Additionally, this module also performing a configuration load from file created by user. The xml file format can be chosen and it contain at least these following information – number of players, possible positions of game objects and their association to players, common area for all players, size of game objects in millimeters. The configuration file needs to be loaded before actual game can start. The positions should be entered in relative coordinates, which means that whole game area in configuration file has square shape with size 0-100 in both axes.

3.4 Visualizer

The last module of Experience Improvement System is Visualizer. It is important part of the system, as it has responsibility of providing output to users. *The module is creating multimedia information and control the process of transferring them into the real-wold.*

The objective of this module could also be described as translation – the update instruction is translated into the visual information. As the visual information has to be consistent, the module is using predefined pairs, each pair is composed of a state contained in update instruction and multimedia content.

The loop of the module is composed of these steps:

1. Update all running effect.

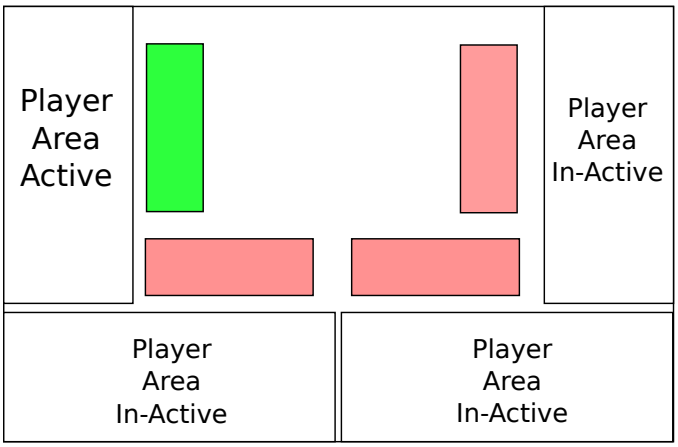


Figure 3.8: Example of system response to active player.

2. Is any new update instruction in pool? If not, jump to step 5.
3. Take next instruction from update instruction pool and process it.
4. If not all instructions were processed, go back to step 3.
5. Send rendered image to output device – projector.

The visual information has to be specifically designed for a game, for which the Experience Improvement System is implemented. This applies on the pairs of a state from update information and multimedia content too. The one exception can be considered – as for the overwhelming number of games, the active player recognition is needed, a system should visually show the currently active player. This will give the users the immediate reaction that a system is keeping the pace with them. This leads to a recommendation to always have in some form a pair composed of state *player active* – and appropriate multimedia content. The example, how such multimedia content can look is in figure 3.8.

The features of visual information should follow the principles presented in chapter 2.2. If the game chosen for system is suitable for it, a multimedia content can be composed of an image and a sound and more. As described in section 2.1, this can enhance the amount of information which is successfully processed by an user.

Additionally, if a game does not limit the game objects position, the Visualizer should provide this information. As was described in section 3.2, the every game object has to have limited area where it can occur. But how the user recognize where to put these game objects, if the game rules does not limit it? This is where the Visualizer is involved. It can provide the necessary visual information for users to know, where in a game area, a game object should be laid down.

3.5 Calibration

Before the components of system are described, the process of calibration for such system is described. The calibration is needed for process of registration described in section 2.2. Calibration process is composed from several parts.

At first, the calibration to determine the game area position in image and size is performed. Next, the calibration of projector used for **Visualization module** is performed.

The calibration process expects that camera calibration of its intrinsic parameters was already performed.

Game Area calibration

Before the first calibration step, the whole game area must be built. The image of complete game area is shown on figure 3.1 and it consist of 3 parts:

- Camera
- Table
- Projector

There are three goals of this calibration:

1. Define the size of game area defined by user, using AR markers.
2. Find relation between one pixel in image and millimeter in the real-world.
3. Find transformation which ensure, that the camera is perpendicular to the game area.

Size of game areas can differ according to environment in which it is played and the system has to be able to adapt to these different sizes – the table calibration is performed. The way to mark the table is required and the calibration system has to be able to find markers in an image from camera. AR markers are meeting these conditions and detection of them in image can be easily performed using vast number of open-source libraries.

Next, the pixel to real-size ratio is needed. This information is required, because user is providing the real-world size of game object in configuration file in millimeters. To be able to find relation between one pixel in image and millimeter in the real-world the real size of printed marker is required. During the calibration process the size of a side in pixels is measured from an image and using equation 3.2 the number of pixels to one millimeter is calculated.

$$mmInPixel_{camera} = \frac{realWoldSize}{pixelSize} \tag{3.2}$$

The core process of calibration is to detect the AR marker in every corner of a table. Output of such detection are the positions of markers corners in clock wise order, starting with top left corner. Because of system limitation, the game area marked must be in rectangular shape. Table calibration is composed of three steps and requires the user assistance:

1. User marks the table using four AR markers – each in one corner.
2. System detects the AR markers and expects these are the corners of a rectangle shaped table. Example of defined game area using AR markers on table can be seen on figure 3.9.
3. Compute perspective projection from 4 corresponding points. Corresponding points are shown on figure 3.9 with blue color.

When the calibration is successfully finished, all future images taken by camera must be transformed using projection matrix obtained in step 3. The result of calibration is shown on figure 3.10.

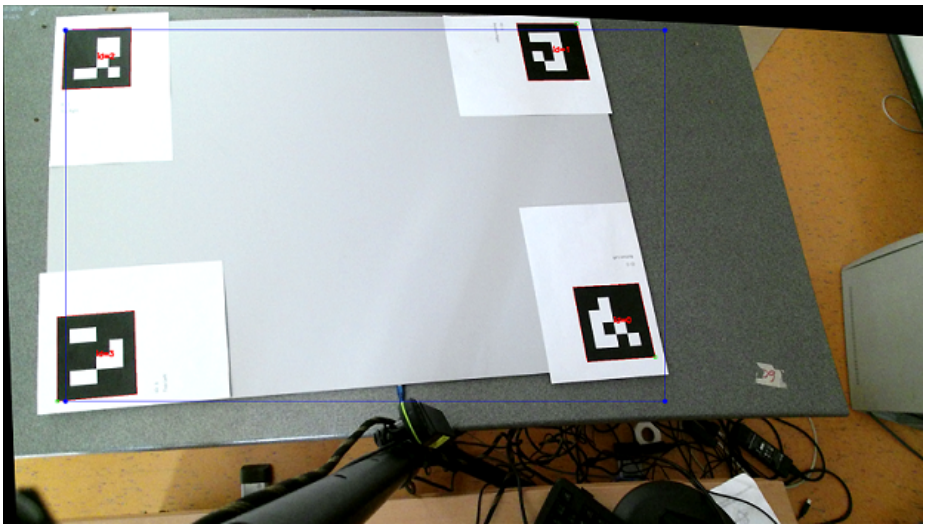


Figure 3.9: Game area marked using augmented reality markers from arUco library and corresponding points marked by green and blue color.

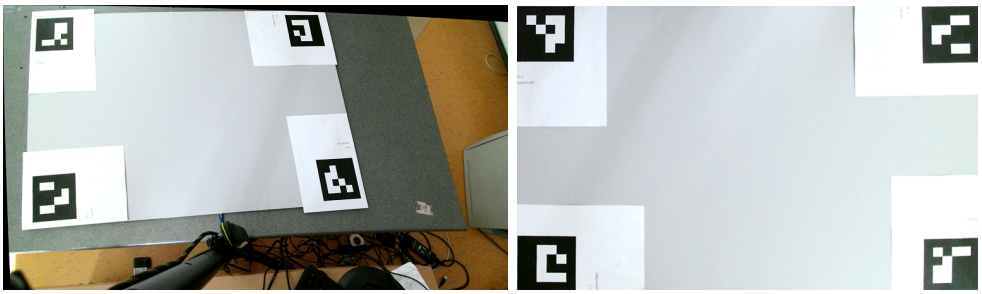


Figure 3.10: Camera view (left) and table extracted after table calibration (right).

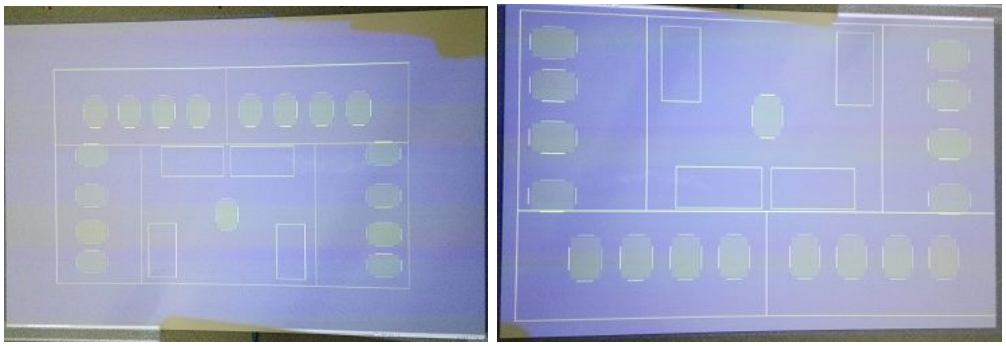


Figure 3.11: Different size of projected object, when projector is $1m$ (left) and $1.6m$ (right) from projection screen.

Projector calibration

The goal of last part of calibration is to determine the size of objects projected on table and the position of a game area table in projector space. The size is highly dependent on a distance of a projector from a table, as demonstrated on figure 3.11. The calibration is started by projecting the chessboard on a table in a game area defined by user during table calibration described in previous section. Then the system detects the chessboard corners and calculate the size of side of one square of projected chessboard ($squareSize_{pixel}$). Using the value millimeter in pixels ($mmInPixel_{camera}$) – determined by game area calibration – the real size of projected chessboard square ($squareSize_{real}$) is calculated using equation 3.3.

$$squareSize_{real} = \frac{mmInPixel_{camera}}{squareSize_{pixel}} \tag{3.3}$$

The real-world size of one square of projected chessboard has been found. Next step is to find the pixel size of one square in Visualization part of application, to be able to project object of specific size, defined in configuration file. The pixel width of whole chessboard ($chessWidth_{pixel}$) is known. From this and the ratio of one square size to whole chessboard size – which is easily manually pre-calculated, as demonstrated on figure 3.12 and equation 3.4, the millimeter in projector image pixels ($mmInPixel_{projector}$) is calculated using equation 3.5.

$$squareToChess_{ratio} = \frac{chessboardWidth_{image}}{squareWidth_{image}} \tag{3.4}$$

$$mmInPixel_{projector} = \frac{squareSize_{real}}{chessboardWidth_{pixel} \cdot squareToChess_{ratio}} \tag{3.5}$$

Finally, the projector does not need to be hanging precisely in the middle of game area, because of that the projected image is shifted according to the red marked chessboard corner on figure 3.12. The distances x and y are used to calculate the corresponding point between image from camera, where the position of red chessboard corner is already known from previous steps, and image generated in system, where the top left corner of whole chessboard is known.

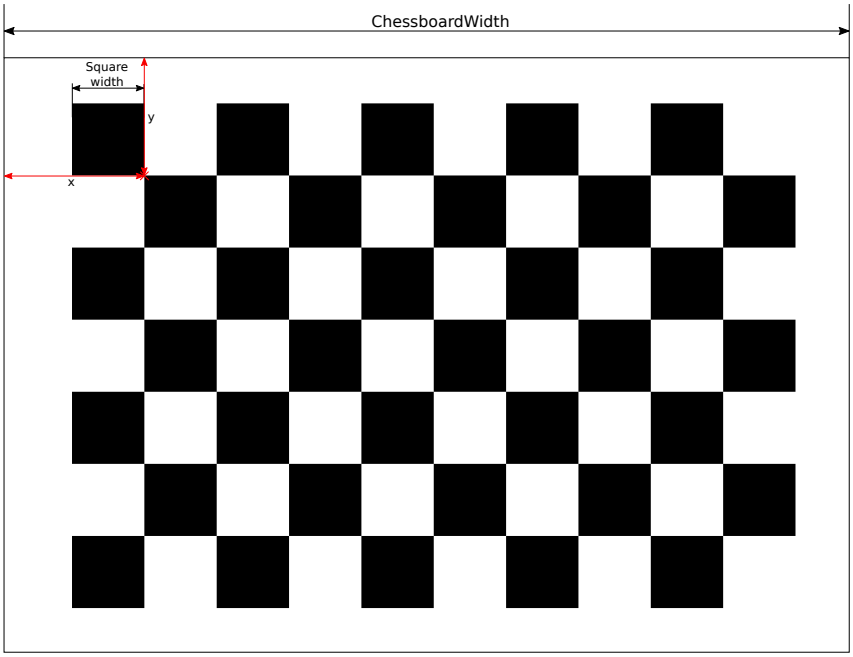


Figure 3.12: Chessboard with marked distances required for projector calibration.

Chapter 4

AR BANG

In this chapter, the implementation of the system from chapter 3 is described. At first, specific card board game is chosen and its principles are described. Next, the states which will be enhanced using augmented reality are chosen. Finally, the composition of system modules including the function used in them are described.

4.1 Bang

For system implementation, the popular social card game Bang has been chosen. The plot of the game is a theme from the Wild West. It can be classify as strategic war game, where exists up to four roles – *Sheriff, Deputy Sheriffs, Bandits and Renegades.* Depending on the number of players the number of Bandits, Deputy Sheriffs and Renegades is chosen. There is always only one Sheriff and he is publicly known. Other players roles are hidden until a player dies or until the game is finished. The game end either when Sheriff dies – Bandits are the winners – or when all Bandits and Renegades dies – Sheriff and Deputy Sheriffs are the winners – or when Renegade kill Sheriff in final duel when all other characters are already dead, in that case Renegade is the winner [16].

During the game, the players are unloading cards to attack other players or to enhance their abilities — like the range of their weapon or the distance from other players. When the player is under an attack he can unload the card to defend himself or to lose a life. After the last life is lost, player dies. The example of the game area of the game BANG is in figure 4.1. There exist three types of cards in a game and the way how they are played is different.

- **Gold cards** – these are unloaded from the players hand and they have immediate effect.
- **Green cards** – these must be placed before the player in the game area and all others can see them. They need to charge and cannot be played until the player finish his move – these can be played next player turn or during other players turn, depending on type of card. When they are charged, the player can play them same way as gold cards.
- **Blue cards** – as Green cards, this must be placed before the player in the game area. These will give player a permanent boost.

Figure 4.1: Example of the game area from the BANG game.

Figure 4.2: Card from BANG game for attacking.

Figure 4.3: Card from BANG game for defending.

4.2 Augmented reality for Bang

The main objective of the system is to enhance the level of enjoyment the players receive from it and add structure to the position of game objects on the table. At first, the enhance of a game area is described. Next, which states of a game will be enhanced is decided. Finally, the ways these states are enhanced are described.

The game area of a BANG is table composed of central area, where the playing deck is, and player area for every player, where he has its boosts. To add the virtual information, new area is created – effects area, as shown on figure 4.4. Every time the player is interacting with his player area or central area, the border color of his player area is changed to mark the activity.

Main part of the BANG is to kill the other players. There exist a few attacking cards in the game to fulfill this goal. A few examples of these types of cards is in figure 4.2. These cards can be divided into two categories – the ones which attack one player and the ones which attack all players. When the attack on one player is performed, the attacking card – BANG – is played and multimedia content – image of gun – is projected in effect area. In response, the player under attack is defending him self. They have to unload the card with semantic meaning miss. An example of such cards are in figure 4.3. Or they have to take a life if they are unable to defend. If the defending of player was successful, the image of smile is projected into his effect area. Example of images for effects area are shown in figure 4.5.

4.3 Implementation tools

For implementation the following programming languages, libraries and tools were chosen.

C++

C++ is a general-purpose programming language providing a direct and efficient model of hardware combined with facilities for defining lightweight abstractions [12, p. 9]. C++ source code is transferable on variety of systems, where it is able to run after compilation, like *Windows, Unix, Mac OS X*. This language was chosen because it allow programmer to use object oriented approach and the OpenCV library support. As the C++ is very

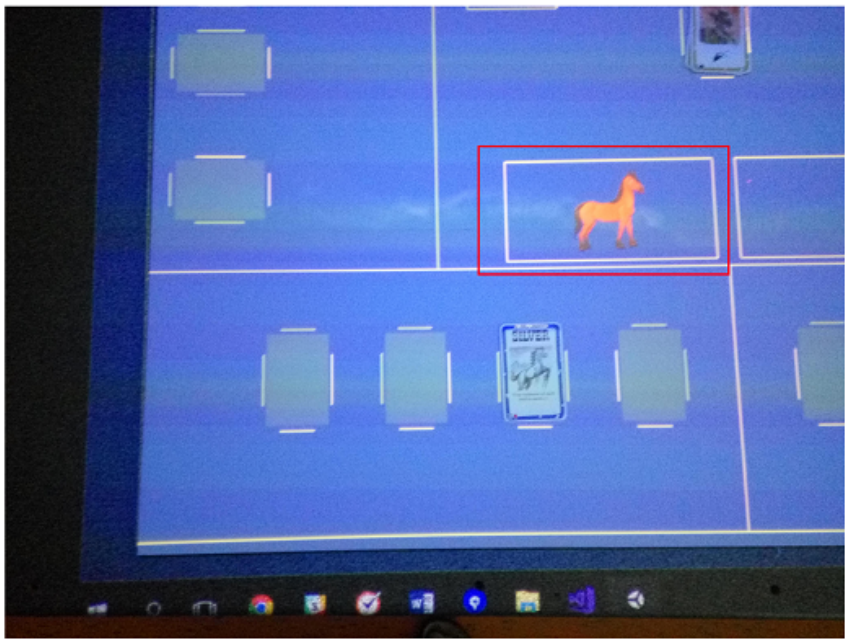


Figure 4.4: Example of effect area for player (marked red).

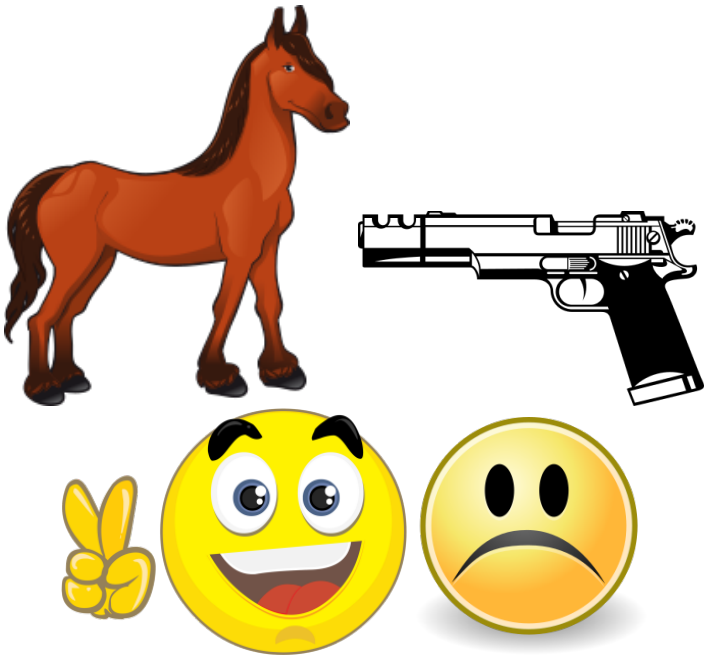


Figure 4.5: Images, which are shown in game after change of state occur.

popular these days for its versatility and control (which the programmer has, especially with memory management and ability to write low-level code close to hardware if needed), the more information can be easily found in extensive literature, like [12].

OpenCV

OpenCV is an open source computer vision library. This library is written in C and C++ and runs under *Linux, Windows and Mac OS X*. It's goal is to provide a simple-to-use computer vision infrastructure that helps build fairly sophisticated vision application quickly [2, p. 1]. The OpenCV library was chosen, because it contain vast number of computer vision algorithm and allow us to easily switch from one algorithm to another if some has poor result on our case.

Unity

Unity is game engine which comes with Unity editor to easily develop application and games. It also provides elements for user interface as buttons, labels, etc. The Unity currently describe it self as: "More than an engine, Unity offers everything you need to build beautiful and engaging content, boost your productivity, and connect with your audience. Tool and resources include continuous engine updates, multi-platform support, and documentation, forums and tutorial.[1]"

The Unity has been chosen, because it provides vast possibilities for graphics content creation, animations and UI developing. This is useful especially for Visualizer module described in section 4.4, but it also allows support for game control, because it provides environment for continuous game loop – fps (frames per second) limit setting, UI service call – and it also provides xml documents parser for easier configuration loading. The Unity comes with good documentation and vast number of tutorials. All of this and more can be found in on-line literature [13].

4.4 ARBang Modules

The goal is to implement complex system designed in chapter 3. The implementation is following the design and is divided into three parts – Observer, Game Control and Logic and Visualizer. The relation and connection between these part is shown on figure 4.6. Because every part has different requirements, the tools chosen for each part are different too. These are:

- C++ with OpenCV library
- Unity

Observer

In this section the implementation of Observer module and its tasks are described. Module is implemented in C++ programming language with OpenCV library and provide the computer vision for ARBang application. Module is build as Dynamic linking library (dll) and imported to *Game control and Logic module*. As the module is additional library, it

[1] https://unity3d.com/unity

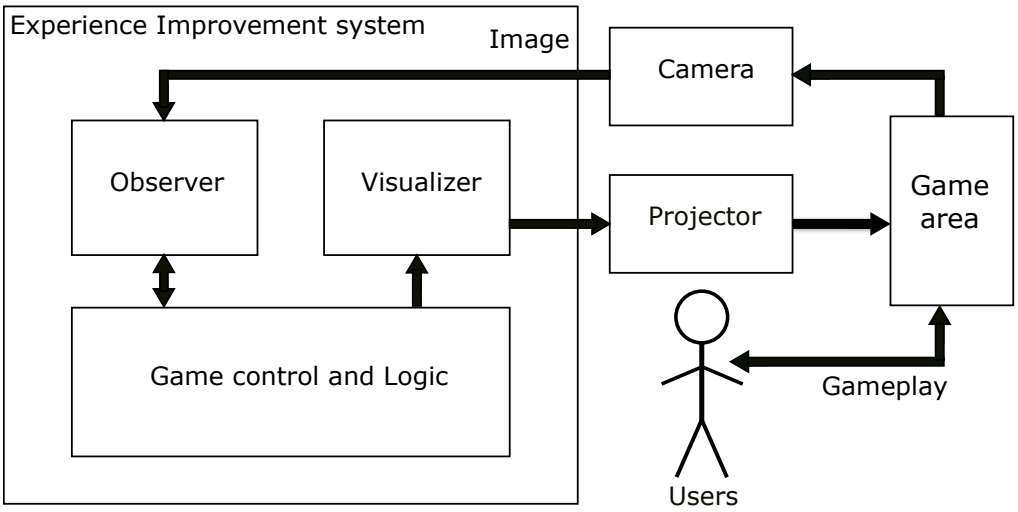


Figure 4.6: Diagram of Experience Improvement system implementation.

can be replaced with different implementation if needed. The created library was named **Image Detection Access Point (IDAP).**

Module tasks

The main task, the Observer module is performing, is processing the current image of Bang game area and provide answers to two important questions:

1. Who is active player – who is interacting with Bang game area?
2. Has any card appear/disappear from game area?

In addition to this main task, the Observer module also provides the support for calibration described in section 3.5. Lets discuss in more detailed way how the answer to the two base question above is found.

Active player detection

For every player, the area parameters (size, position) are loaded from configuration file provide by user, the example of configuration file is attached as appendix C. It is used to create $PlayerAreaActiveDetector$ class instances. Apart from the area, the player ID from configuration is associated with every detector. $PlayerAreaActiveDetector$ class implements only one function, which is $isAreaActive(image)$. This function returns the intensity if the players activity is detected, otherwise zero is returned. To subtract background from an image, the OpenCV methods are used. At first, the background subtractor is created by $createBackgroundSubtractorMOG2()$ function, with following parameters – history set to 5 previous frames and shadow detection is off. Every time the new image from camera is taken, using method $apply(image, mask)$ of subtractor, the background is removed from input image and foreground is saved to mask. The foreground is evaluated using equation 3.1. The evaluation is compared to upper threshold and if it is higher, the area is considered as active. For the evaluation function the optimal threshold were experimentally determined and set as follows:

- Upper threshold: 15
- Lower threshold: 7

The thresholds are defined in file *PlayerAreaChangeDetector.h*.

Card detection and classification

For every card area loaded from configuration file – position and card size is needed, the instance of *CardAreaDetection* class is created. During the creation, the region of interested is calculated from the captured image size, position and size of game object. Because the calibration of system may cause inaccuracies between the position of card defined in configuration and its real position, the region of interest is enlarged by 20% on every side to enlarge probability the whole card is in it.

Before the actual classification can be performed, the templates of all possible cards are loaded when the system is started. All images from folder *Assets/ARBang/cardData* are loaded and used as templates, one for each out of twenty eight classes. To connect the template loaded with the card of Bang, the name is used. The names of all supported Bang card with their templates are enclosed in appendix B.

Additionally, during the templates loading, the gradient of mean card is calculated. This mean card gradient is used to determine if any card is presented in a position that is being checked. The template matching is performed on gradient of current image, on this position, with the gradient of mean card as template. The normed cross-correlation evaluation method was chosen for template matching for card detections. The equation used for evaluation is defined in section 2.4. This approach has been chosen to speed up the processing, if no card is presented on the position that is being checked. In such cases only one template matching has to be performed, instead of matching with template from every class. Two thresholds had to be experimentally found – first of them is a minimal value the template matching function has to return in order for card to be detected and second of them is a minimal mean value the gradient of the current image must have to even perform the matching for card detection. If the mean value is not high enough, the card type *none* is returned. The second threshold had to be added, because the template matching method returned high score for empty places. This was caused by the fact, that the gradients of cards contains large parts of empty area, which was successfully matched with gradient image of empty card position. The two thresholds are defined in *CardAreaDetection.h* file and were set as follows:

- Mean value minimum threshold: 20
- Minimum template matching score to detect a card: 0.7

The process of card detection described above is performed by function *isCardChanged* of *CardAreaDetection* class and precedes the card classification process, in case the card is successfully detected. If no card is detected, the fast return with card type *none* is used. The function needs current frame from camera, templates of all cards and gradient of mean card to perform detection and classification. Two methods were tested for card classification, the first one is Template Matching with Hough transformation and second one is nearest neighbors with *SURF* descriptors. The result of these methods are presented in section 4.6. As the second method – SURF with nearest neighbor search – achieved better results, only this method of classification is supported in a final solution. After the classification is performed, a new founded card type is returned.

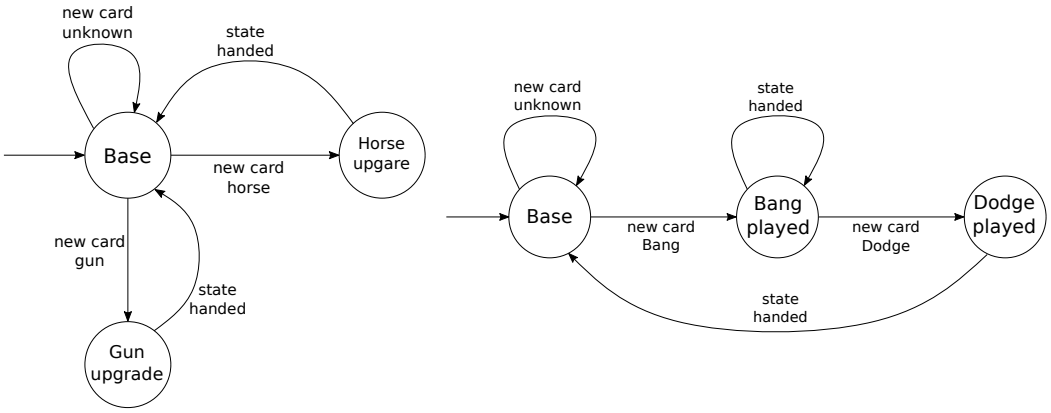


Figure 4.7: Player state automaton (left) and Bang state automaton (right).

For testing, both matching methods were implemented using OpenCV library. For Template matching, function *matchTemplate* – which require input image, template and match method – is provided by the library. The classification using nearest neighbor is performed by *FlannBasedMatcher*. The OpenCV does not contain the SURF implementation but it is provided by OpenCV contributor library. The limitation of SURF is that the method is patented, that is why it is not presented in base OpenCV as OpenCV is open source, but it can be used without charge for academic purposes.

Game control and Logic

In this section the implementation of ARBang application control module is described. The module controls the Observer, Visualizer and hold the state of the game. Module is implemented in Unity using c# programming language.

Module tasks

Module is entry point of application and provide user interface for calibration. When the application is started, the first screen is presented to user – game lobby. The button *Start* is disabled until the calibration is successfully performed using button *Configurate*, the complete process of calibration was described in section 4.5. The module also implements subset of the state space of Bang game. It is separated into two types of state automatons – one type for every player and second type for whole game. The automatons are shown on figure 4.7.

When the game is started by pressing button *Start*, the configuration file is loaded. Next, the game loop is launched. The whole loop is written in algorithm 4.1. The loop consist of three parts – contact observer module, update game state and create update informations for Visualizer. The loop is defined in *GameControl.cs* file in function *Update*(). Whole loop process is written as algorithm 4.1 below.

Contact observer

At first, the next image from camera is captured using function *PrepareNextFrameCaller*(). All function using Image Detection Access Point API are defined in *CameraControl.cs* file.

Algorithm 4.1 Game control main loop.

```
1: cardsPosToCheck[] = {type = none, typeN[], checked = 0};
2: activePlayer ← −1;
3: mostActiveIntensity ← 0;
4: while true do
5:     Observer.PrepareNextFrame();
6:     for all players do
7:         intensity ← player.IsActive();
8:         if (intesity > mostActiveIntensity) then
9:             activePlayer ← player;
10:        end if
11:    end for
12:    if commonArea.IsActive() then
13:        cardsPosToCheck.Add(commonArea.Cards);
14:    end if
15:    if activePlayer ≠ −1 then
16:        cardsPosToCheck.Add(activePlayer.Cards);
17:    end if
18:    Observer.PrepareNextFrame();
19:    for all cardsPosToCheck do
20:        cardType ← Observer.IsCardChanged(cardPos);
21:        if (cardType ≠ cardPos.type) then
22:            if (cardPos.typeN.Contain(cardType)) then
23:                cardPos.typeN[cardType] ← cardPos.typeN[cardType] + 1;
24:            else
25:                cardPos.typeN.Insert(cardType);
26:            end if
27:        end if
28:        cardPos.checked ← cardPos.checked + 1;
29:    end for
30:    cardPos.typeN[cardType] ← cardPos.typeN[cardType] + 1;
31:    if cardPos.checked < CHECK_LIMIT then
32:        for all cardPos.typeN do
33:            if (typeN[cardType] > CONFIRM_NEEDED) then
34:                CreateUpdateInfo(cardPos, cardType);
35:                cardsPosToCheck.Remove(cardPos);
36:            end if
37:        end for
38:    else
39:        cardsPosToCheck.Remove(cardPos);
40:    end if
41: end while
```

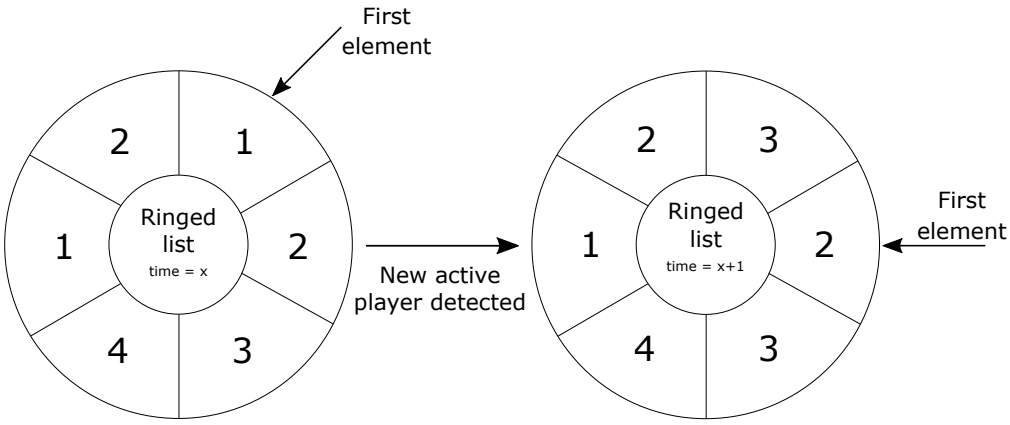


Figure 4.8: Demonstration of *RingedList* activity.

Active player calls: For every player defined in configuration, $IsPlayerActiveByID(id)$ function is called. This is using IDAP[2] API call $IsPlayerActiveByIDCaller(id)$. The function returns the evaluation of player area if it is considered as active by Observer module. The evaluation is necessary because game control use it to determine active player if it seems that more active player occurred – there can always be only one at the same time. Player is chosen active, if he has the highest evaluation on his active area – he was a most active.

Because some states need the information about the previous active player, the currently active player is added to *RingedList* object, which store last six active player, which are available for states to use if required. *RingedList* is defined in *RingedList.cs* file and it is implemented as circular buffer and its function is demonstrate on figure 4.8.

Card changed calls: When the activity of player is detected, all possible card positions of this player are marked as change could occurred. These areas are checked for next several frames, if any card appear/disappear from a game area. It is checked five times to overcome the problem that the card is still not properly placed on table and it is blurred in image, which cause the incorrect detection. If, at least two out of five frames classify the card to same class, the new card has been identified, the system update the player state, game state and creates the update instruction for Visualizer. If a card which has changed is in common area, the update instruction contain the state of a whole game Bang. Otherwise, a card is associated to one player and only that player state is contained in update instruction. To detect if the card is associated to one player, each card position has player ID specified in configuration file. If it is associated to special player with ID zero, card position is in common area. The check itself is performed using function $HasGameObjectChanged(id_{area}, id_{card})$ where id_{area} is identification of card area from configuration file and id_{card} is send by reference in which the result is returned.

Drawing update instruction generation: When new card appear in a specific place and its type is confirmed, two drawing update sequences are generated. First of them is update instruction to create green bounding box around newly detected card at the specific place, to let the player known the card was successfully processed by system. Second of

[2]Image Detection Access Point library

them is instruction with effect caused by the newly detected card. If a card of class *Silver, Appaloosa or Mustang* appear, the state *Horse upgrade* is inserted into update instruction. If a card of class *Volcanic, Schofield, Remmington, Carabine or Winchester* appear, the state *Gun upgrade* is inserted into update instruction. These were two examples of update instruction, more states and cards which causes them are specified in state automaton on figure 4.7. All information a drawing update instruction of AR Bang contain are:

- Active player ID – player which played new card
- Previously active player ID – player which played previous card
- Active player intensity – auxiliary value for instruction creation
- Card ID – position where new card appeared
- State – the state new card caused
- Card type – type of card at card ID position
- Appear – true, if new card appeared; false, if card disappeared
- Effect – if some effect is not defined by game state, the effect can be set directly
- Priority – effect priority, reserved for future usage

Visualizer

The Visualizer module is taking care of ARBang application output. The output is multimedia content – image – provided to user by projector in a game area. It is important part of application because this is what player receive from Experience Improvement System. All previous modules worked with information obtained from game area or with internal representation of game. Because the module need to generate graphic output, the Unity described in section 4.3, was chosen as tool for implementation. It gives us the complex support for graphics object and animation creation. Visualizer is implemented in $DrawEffectControl.cs$ script.

The Visualizer is started when the game is started from lobby and his *Update* function is called every frame. The function is checking if any drawing update instruction has been inserted into drawing update instruction pool. If it was – which mean that update for drawing is ready – all instructions in pool are processed and pool is emptied. First informations which are loaded from drawing information pool, after the game is started, are the definition of a game area. This result of these instructions is shown on figure 4.9. Procedure of update information processing is composed of several steps. At first, the effect according to the state contained in update instruction is chosen. All supported effects and the state which causes them are written in table 4.1. Next, place where it should be projected is determined, also from update instruction. Then, new effect on a specific place is drawn and projected into a game area.

The Visualizer also showing currently active player, which is determined by the activity founded by active player detector. This is not changing the state of player and the effect it should caused – mark border – is included directly in drawing update instruction. Which means, that the drawing control script is not deciding effect based on state.

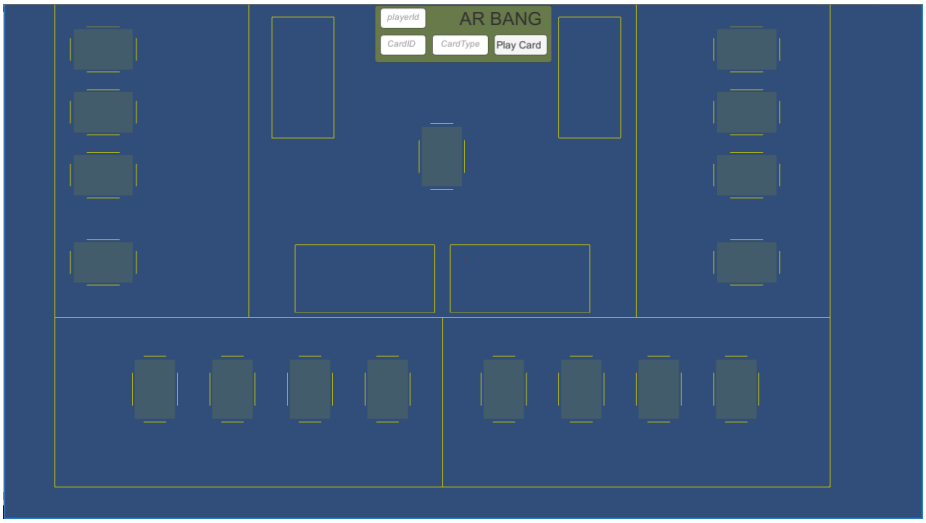


Figure 4.9: Game area of ARBang when the game is started.

played card position	state	effect
player	NEW_CARD_GUN_UPGRADE	Image of gun before player.
player	NEW_CARD_HORSE_UPGRADE	Image of horse before player.
common	BANG_PLAYED	The image of gun before the active player.
common	DODGE_PLAYED	The image of smiley before the active player.

Table 4.1: States and the effects they are causing.

4.5 System calibration

To combine the real-world and virtual information, the calibration has to be performed. The system is using camera, table and projector. These need to know values dependent on mutual position to cooperate with each other. At first, the intrinsic camera parameters are found, next the position of table in camera image is calculated. Finally, the projector position according to table is calibrated.

Camera calibration

Calibration of intrinsic camera parameters is performed using OpenCV library. To perform the calibration the chessboard shown on figure 4.10 is needed. The size of chessboard may differ, but the pattern must be invariant to rotation. The process of calibration is composed from 4 parts:

- At first, take images containing chessboard at different distance and at different angles in all axes of chessboard to camera. The more samples are provided the better the result of calibration can be, although there are other factors, which can influence the results like auto-focus or zoom.
- After the images are taken, the chessboard corners are detected inside of them.

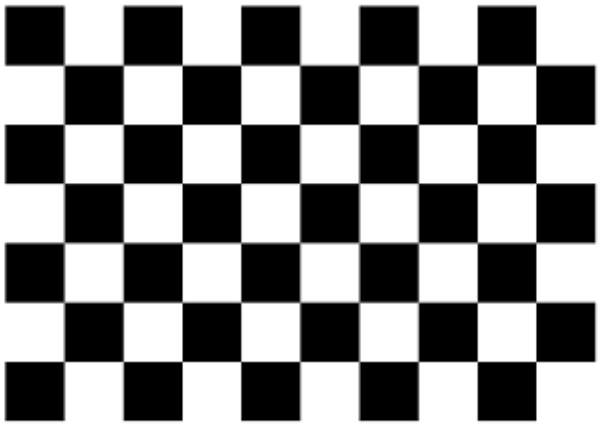


Figure 4.10: Chessboard 9x6 used for camera intrinsic calibration.

- Using the real size of side of the chessboard square and detected corners, the intrinsic camera parameters are found – distance coefficients and 3x3 camera matrix.
- Finally, using *undistort* [3] function with intrinsic camera parameters, the taken image from camera is corrected.

Camera calibration is separate part of calibration and it is related to camera used for Observer module. When the calibration is successfully performed, it is saved to permanent storage and there is no need to perform this calibration again, unless the camera will change.

To perform the camera calibration, the UI was created. This is accessible by choosing *Configurate>Calibrate Camera* in game lobby. The UI provide four buttons with described functionality:

- *Start Capturing* – Image from camera is captured and saved every 2 seconds, the user should use the time between to change the position of chessboard before camera, the captured images are shown in right part of UI.
- *Calibrate* – Pressing the button perform the calibration, if at least 25 images was captured. Only the images where chessboard is detectable are used.
- *Load* – If camera calibration was already performed, this can be used to load it.
- *Back* – Return to previous menu.

Game Area calibration

This section is describing the implementation of game area calibration from section 3.5. To perform game area calibration table has to be marked. This is achieved using OpenCV library *arUco*. The library is not part of OpenCV but it is introduced in OpenCV contributors modules. The arUco library allows to generate the AR markers and detect them in an image. The image from camera is provided to detection function, which return the IDs of markers founded in image and sequence of coordinates of four corners of every marker, starting with top left corner. Only the top left corner is used to define the game area.

[3] https://docs.opencv.org/3.1.0/da/d54/group__imgproc__transform.html

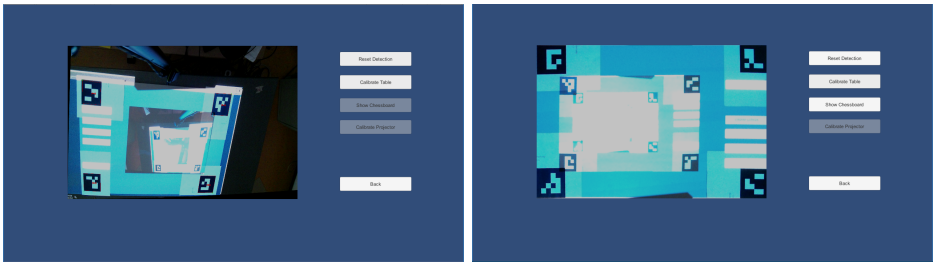


Figure 4.11: GUI for table calibration, state after markers were detected (left) and table calibration performed (right).

UI for game area calibration was created in application and it is accessible by choosing *Configuration>Table calibration* in game lobby. The table calibration GUI is shown on figure 4.11. Player begin game area calibration by placing the AR makers in every corner of a game area – table. Next, the button *Detect Markers* is pressed. Now, the detected markers are shown in GUI. The next steps of calibration are inaccessible, until all four markers are successfully detected. Next step of calibration is game area delimiting. This is performed by pressing *Calibrate Table* button. The result of calibration is then shown in GUI, where instead of whole image from camera, only the game area is seen. The UI contains another two buttons, which perform the projector calibration described below.

Projector calibration

The projector calibration is performed using same UI scene as game area calibration, shown on figure 4.11. After game area is calibrated, the button *Show Chessboard* is enabled. Pressing the button project the chessboard in game area. Next, by pressing button *Calibrate Projector*, the image of game area is captured. The chessboard corners are found and the projector start calibration. After the calibration is successfully finished, the game area should be show in UI, instead of chessboard.

4.6 Testing

To measure the quality of designed and implemented system, the testing was performed. Testing was divided into two separate parts. At first, ability to recognize game objects – cards – on table was tested. Next, the implemented system was presented to the group of test subject to perform several predefined task and evaluate system.

Card recognition

The important task of system is to detect and classify cards on table, which holds the information of a current game state. Two methods for object classification were presented in section 2.4. First of them is Template matching on gradient images. Second of them is „interest points“ detection and description using SURF with nearest neighbor search.

Because the Bang contains vast number of different cards, the random representative sample was chosen for card classification testing. More precisely, twenty eight card types was chosen, each represents one card class. All classes with their templates are enclosed in appendix B. The another set needed for testing of card classifier are real data obtained in a real environment. At first, components of designed system were assembled and the

Figure 4.12: Template of class *BangA* and same class image captured in a real environment.

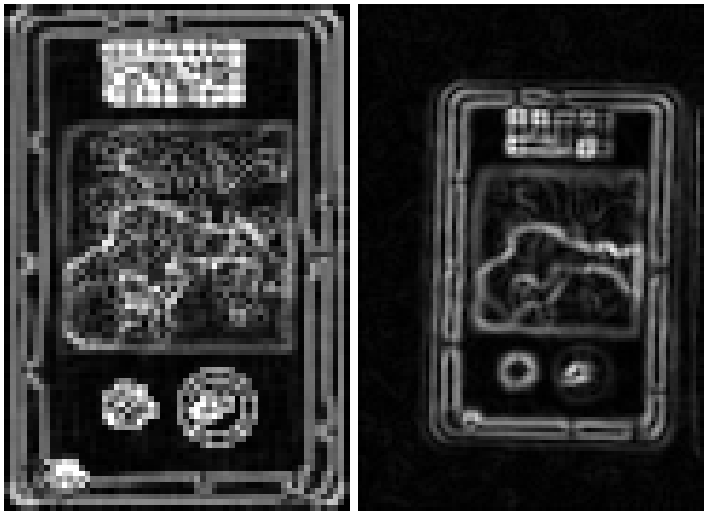


Figure 4.13: Gradient images of class *BangA* template (left) and detected card (right).

calibration described in section 3.5 was performed. Next, for twenty card classes, twenty images of cards were captured with different light intensity and rotation. The example of class template *BangA*, together with image captured in a real conditions are shown on figure 4.12. The results of card recognition tests were written in table 4.2

From the results of card classification testing can be seen that non of the tested method is sufficient for reliable classification. Template matching achieves success rate of 49% and the SURF with nearest neighbor search achieves success rate of 63.5%. The poor results of methods are caused by bad quality and insufficient number of details of captured images. An insufficient number of details is key problem in classification with SURF method, there is not exist enough distinctive points, which the method could use and which would distinguish cards properly. The key problem with template matching method is that significant part of cards is similar – yellow bounding box, name at the top and image in the middle. These similar parts caused high score for template matching for all classes and poor card details did not give enough score to distinguish them.

Class set	Template matching		SURF and NN	
	Correct	Success rate	Correct	Success rate
BangA	6/20	0.3	17/20	0.85
BangB	15/20	0.75	18/20	0.9
Bier	0/20	0	2/20	0.1
DodgeA	5/20	0.25	12/20	0.6
DodgeAM	4/20	0.2	19/20	0.95
DodgeB	4/20	0.2	20/20	1
DodgeBM	4/20	0.2	17/20	0.85
Dynamite	0/20	0	14/20	0.7
GatlingGun	8/20	0.4	19/20	0.95
Remmington	20/20	1	9/20	0.45
Carabine	12/20	0.6	0/20	0
Schofield	9/20	0.45	12/20	0.6
Volcanic	8/20	0.4	2/20	0.1
Winchester	16/20	0.8	9/20	0.45
Hideout	16/20	0.8	13/20	0.65
Appaloosa	13/20	0.65	17/20	0.85
Mustang	17/20	0.85	7/20	0.35
Silver	14/20	0.7	10/20	0.5
Indians	19/20	0.95	18/20	0.9
Prison	6/20	0.3	19/20	0.95
Success rate	49%		63.5%	

Table 4.2: Result of card classification testing.

User testing

The user testing was composed of three separate parts. At first, test subject ability to start the system with previous instruction and one demonstration was tested. Next, the subjects sat down around table and start playing card to test system response to player activity. Finally, the subject played predefined scenarios to test how the system with effect affects them. Because the system is unable to classify card with enough precision, the card were manually added to system by operator. After the tests were performed, test subject answer to several questions, their answers are written in table 4.3. The tests were performed on five subject, which were in an age between $20 - 30$.

Two test subjects were able to set up system them self without any assistance needed after one demonstration. Another three tests subjects need assistance during set up processes, because it is composed from too many steps and they would need a manual with procedure. Eighty percent of subjects evaluate the set up as manageable to do it alone.

Next, the system ability to detect active player was tested. This composed of ten rounds of simple Bang game, where the system only marked the active player. The subjects were instructed to behave like in a normal game but do not interfere with table when it is not

their turn. Ten rounds with four players were play, which leads to forty player activity, which system should recognized. Out of those forty, a system was able to correctly detect active player at 92.5% cases. Out of ten rounds played, a system was able to correctly follow players sequence in eight of them. Two rounds when error occurred was caused by insufficient activity in player area to be detectable.

Finally, the test subjects played predefined scenario composed of these steps:

1. Play card *Silver*, to perform horse upgrade.
2. Play card *Carabine*, to perform gun upgrade.
3. Play card *BangA*, to perform attack.
4. Play card *DodgeB*, to defend from attack.

The card types played were manually added to system if not recognized, to overcome the classification inaccuracy.

Discussion was performed with test subjects after the test, to find out their opinions on system. The most discussed think was the quality of effects, the animations were requested. The next often discussed issue was a slow system response. When the card was successfully recognized by system without operator assistance, it was not fast enough. This was caused by a lot of processing needed to perform classification.

Question	yes [%]	rather yes [%]	rather no [%]	no [%]
Do you play board games?	100	0	0	0
Do you know BANG?	100	0	0	0
Could you set up system your self?	60	20	20	0
Is projected game area well laid out?	80	0	0	20
Are cards predefined spaces annoying?	0	0	20	80
Are effect a pleasant diversion?	100	0	0	0
Would you use such system?	60	20	20	0
Is system too complicated?	20	0	40	40

Table 4.3: Answers of tests subjects after their perform test sequence.

4.7 Future work

The future work would consist of extend the number of states of Bang the system is able to recognize and extend the effects these states causes. This would be straightforward work as only the Bang state machine needs extension and for each state, the new effects needs to be defined.

Next, the card classification system need enhancement. The one option, how to enhance the card classification is to mark the card with distinctive signs. These could replace the image in the center, which is not required for players to recognize card. And the additional information these images are carry would be supplemented by system effects. These could be small versions of augmented reality markers or similar predefined patterns, which would not suffer from insufficient details. Another option is to replace the card classification system

with different one. This new classification system may be convolution neural network, which has a potential to provide better results, but more tests need to be performed to prove its benefits.

Chapter 5

Conclusion

The benefits of the social games and their effect on the human senses were described. The most used sense during the play are the sight, touch and hearing — not necessary in this order, as it highly depend on the game played. Overall, the social interaction and improvement of cognitive functions – such as perception, memory and creativity – were presented as key benefits. Next, the requirements for system, which improves user experience during a game play were presented. These were taken into an account and "Experience Improvement System" was designed.

The designed system was divided into three key parts. *Observer*, which is providing the information about the real world state. *Game control and Logic*, which controls the system and holds the state space of game, for which the system is created. And *Visualizer*, which creates multimedia output projected back into the real world.

The card game Bang was chosen for implementation of the designed system. At first, the game states which will be enhanced by multimedia effects – images – were chosen. Except for the game states, the game environment was also enhanced by defining the area where the game objects can appear. The tests were performed with implemented system. Its ability to classify Bang card was measured, using two introduced methods – *Template matching* on gradients of images resulted in success rate of 49% and *Image matching* using SURF and nearest neighbor search resulted in success rate of 63.5%. Finally, user testing was performed, where the benefits of system were questioned – these resulted in following conclusion: The system suffers from slow response rate and poor graphics of multimedia output. If these can be solved, 80% of respondents would like to use such system.

Bibliography

[1] Bay, H.; Tuytelaars, T.; Gool, L. V.: *SURF: Speeded Up Robust Features*. 2006. [Online; cit 28.04.2018].
Retrieved from: https://lirias.kuleuven.be/bitstream/123456789/71383/1/Bay_Tuytelaars_VanGool-surf-eccv06.pdf

[2] Bradski, G. R.; Kaehler, A.: *Learning OpenCV*. O'Reilly. first edition. 2008. ISBN 978-059-6516-130.

[3] Gobet, F.; de Voogt, A.; Retschitzki, J.: *Moves in Mind: The Psychology of Board Games*. 01 2004. ISBN 1-84169-336-7.

[4] Karjalainen, A.: *What are the building blocks of a good board game?* [Online; cit 19.03.2018].
Retrieved from: hthttps://boardgamegeek.com/blog/1952/what-are-the-building-blocks-of-a-good-board-game

[5] Kreylos, O.: *HoloLens and Field of View in Augmented Reality*. August 2015. [Online; cit 09.03.2018].
Retrieved from: http://doc-ok.org/?p=1274

[6] Mayne, M.: *See the future: VR headsets and hardware trends*. February 2018. [Online; cit 27.04.2018].
Retrieved from: https://www.ibc.org/tech-advances/see-the-future-vr-headsets-and-hardware-trends/2696.article

[7] Milgram, P.; Kishino, F.: A Taxonomy of Mixed Reality Visual Displays. vol. vol. E77-D, no. 12. 12 1994: pp. 1321–1329.

[8] Muja, M.; Lowe, D. G.: *Fast approximate nearest neighbors with automatic algorithm configuration*. [Online; cit 03.05.2018].
Retrieved from:
http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_visapp09.pdf

[9] OpenCV.org: *How to Use Background Subtraction Methods*. [Online; navštíveno 19.12.2017].
Retrieved from:
https://docs.opencv.org/3.2.0/d1/dc5/tutorial_background_subtraction.html

[10] OpenCV.org: *Template Matching*. [Online; navštíveno 30.04.2017].
Retrieved from:
https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

[11] Schmalstieg, D.; Höllerer, T.: *Augmented reality: principles and practice.* Addison-Wesley. 2016. ISBN 03-218-8357-8.

[12] Stroustrup, B.: *The C++ Programming language.* Pearson Education, Inc.. fourth edition. 2013. ISBN 978-0-321-56384-2.

[13] Technologies, U.: *Learn Unity.* [Online; navštíveno 29.04.2016]. Retrieved from: https://unity3d.com/learn

[14] Unger, R.; Chandler, C.: *A Project Guide to UX Design.* New Riders. 2009. ISBN 978-0-321-81538-5.

[15] Vernon, D.: *Machine Vision: Automated Visual Inspection and Robot Vision.* Prentice Hall International. 1991. ISBN 0-13-543398-3.

[16] Řábek, S.: *BANG - Pravidla a FAQs.* [Online; navštíveno 15.01.2018]. Retrieved from: http://www.bang.cz/cs/pravidla-a-faq.html

Appendix A

Obsah přiloženého paměťového média

```
/
├── thesis.pdf ..... Text of the thesis.
├── tex/ ..... LaTeX source files.
├── src/ ..... Application source files.
│   ├── ImageDetectionAccessPoint/ ..... Source code for Observer module.
│   └── ARBang/ ..... Unity source codes of ARBang application.
├── poster.pdf ..... Poster of the thesis.
├── video.mp4 ..... Video of the thesis.
├── video_long.mp4 ..... Longer version of the video.
└── README.txt ..... Compile instruction for IDAP and ARBang.
```

Appendix B

Card classes

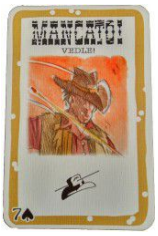
class	template	class	template	class	template
BangA		BangB		DodgeA	
DodgeAM		DodgeB		DodgeBM	
Barrel		Schivata		Bible	
IronPlate		Sombrero		Cappello	

Table B.1: Card classes 1-12 with templates representing each class.

class	template	class	template	class	template
Bier		Indians		GatlingGun	
Howitzer		Saloon		Silver	
Mustang		Appaloosa		Hideout	
Dynamite		Prison		Volcanic	
Schofield		Remmington		Carabine	
Winchester					

Table B.2: Card classes 13-28 with templates representing each class.

Appendix C

ARBang configuration file

```xml
<?xml version="1.0" encoding="UTF-8" standalone = "yes"?>
<ARBang>
    <ARBangTableSettings id="0">
        <CentralArea start_x="35" start_y="0" width="30" height="65" />
        <Players numberOf="4">
            <Player id="0">
                <!-- Player with id 0 is settings for whole table and contain cetral area with packs of cards -->
                <ActiveArea start_x="35" start_y="0" width="30" height="65" />
            </Player>
            <Player id="1">
                <ActiveArea start_x="0" start_y="0" width="25" height="65" />
                <EffectsArea start_x="28" start_y="3" width="8" height="25" />
            </Player>
            <Player id="2">
                <ActiveArea start_x="0" start_y="65" width="50" height="35" />
                <EffectsArea start_x="31" start_y="50" width="18" height="14" />
            </Player>
            <Player id="3">
                <ActiveArea start_x="50" start_y="65" width="50" height="35" />
                <EffectsArea start_x="51" start_y="50" width="18" height="14" />
            </Player>
            <Player id="4">
                <ActiveArea start_x="75" start_y="0" width="25" height="65" />
                <EffectsArea start_x="65" start_y="3" width="8" height="25" />
            </Player>
        </Players>
        <CardsPosition>
            <!-- common -->
            <Card id="0" player_id="0" size_id="0" left_top_corner_x="47" left_top_corner_y="25" turn_ninety="false" />
            <!-- player 1 -->
            <Card id="1" player_id="1" size_id="0" left_top_corner_x="2" left_top_corner_y="5" turn_ninety="true" />
            <Card id="2" player_id="1" size_id="0" left_top_corner_x="2" left_top_corner_y="18" turn_ninety="true" />
            <Card id="3" player_id="1" size_id="0" left_top_corner_x="2" left_top_corner_y="31" turn_ninety="true" />
            <Card id="4" player_id="1" size_id="0" left_top_corner_x="2" left_top_corner_y="49" turn_ninety="true" />
            <!-- player 2 -->
            <Card id="5" player_id="2" size_id="0" left_top_corner_x="10" left_top_corner_y="73" turn_ninety="false" />
            <Card id="6" player_id="2" size_id="0" left_top_corner_x="20" left_top_corner_y="73" turn_ninety="false" />
            <Card id="7" player_id="2" size_id="0" left_top_corner_x="30" left_top_corner_y="73" turn_ninety="false" />
            <Card id="8" player_id="2" size_id="0" left_top_corner_x="40" left_top_corner_y="73" turn_ninety="false" />
            <!-- player 3 -->
            <Card id="9" player_id="3" size_id="0" left_top_corner_x="55" left_top_corner_y="73" turn_ninety="false" />
            <Card id="10" player_id="3" size_id="0" left_top_corner_x="65" left_top_corner_y="73" turn_ninety="false" />
            <Card id="11" player_id="3" size_id="0" left_top_corner_x="75" left_top_corner_y="73" turn_ninety="false" />
            <Card id="12" player_id="3" size_id="0" left_top_corner_x="85" left_top_corner_y="73" turn_ninety="false" />
            <!-- player 4 -->
            <Card id="13" player_id="4" size_id="0" left_top_corner_x="85" left_top_corner_y="5" turn_ninety="true" />
            <Card id="14" player_id="4" size_id="0" left_top_corner_x="85" left_top_corner_y="18" turn_ninety="true" />
            <Card id="15" player_id="4" size_id="0" left_top_corner_x="85" left_top_corner_y="31" turn_ninety="true" />
            <Card id="16" player_id="4" size_id="0" left_top_corner_x="85" left_top_corner_y="49" turn_ninety="true" />
        </CardsPosition>
    </ARBangTableSettings>
    <ARBangGameSettings>
        <CardSize id="0">
            <height>95</height>
            <width>65</width>
        </CardSize>
    </ARBangGameSettings>
</ARBang>
```

Figure C.1: Example of configuration file for ARBang system.