



Faculty of Electrical Engineering, Mathematics and Computer Science

Tracking the Progress of an Escape Room to Support the Game Host

Computer Science Bachelor's Thesis

Ege de Bruin, Robin Hurkmans,
Jasper Kroes & Lisette Veldkamp

Thesis committee:
Dr. Ir. W.P. Brinkman, TU Delft Supervisor
J.W. Manenschijn, Popup-Escape
Ir. O.W. Visser, TU Delft Bachelor Project Coordinator

To obtain the degree of
Bachelor of Science in Computer Science and Engineering
at the Delft University of Technology

Delft, Thursday 5th July, 2018

Executive Summary

Escape rooms are multi-player games that contain several puzzles that need to be solved in order to open locked chests and discover new clues, which eventually enables the players to escape the room. While the players are inside the escape room, the game host observes the group through live cameras. When players tend to fail to make it out of the escape room in time, the host needs to give them hints to keep them on track.

Popup-escape is a company that designs escape rooms. They have asked us to develop an application that supports the game host in the process of observing escape rooms. Hence, we developed an application that displays live video streams and shows valuable information about the progress of the game. The game host can configure the escape room in the application before players enter the escape room. This configuration sets up how the escape room is structured. The game host indicates the number of chests (key points in the game) that need to be unlocked and the time it should take players to open it. The application then processes the incoming video streams and detects chests that have been opened, as well as the level of current activity. The progress is measured against time. When the progress made is falling short compared to the preconfigured time limits, the host gets a warning, alerting him that the players in the escape room need a hint in order to be able to finish the game in time.

Preface

This report describes the bachelor end project we conducted in a timespan of ten weeks in order to obtain a Bachelor of Science degree. The project was commissioned by Popup-escape with the goal to provide support in the process of interaction between host and players in escape rooms.

We would like to thank Popup-escape for welcoming us into their office and for their input and enthusiasm towards the software project. Furthermore, we would also like to thank our coach, Willem-Paul Brinkman, for guiding us through the process.

*K.E. de Bruin
R. Hurkmans
J. Kroes
L.S. Veldkamp
Delft, June 2018*

Contents

Contents	iv
1 Introduction	1
1.1 Problem description	2
1.2 Problem analysis	3
2 Research	5
2.1 Detection and tracking	5
2.2 Generic escape rooms	8
2.3 Measuring progress	8
3 Design	10
3.1 Overview	10
3.1.1 Front-end	11
3.1.2 Back-end	14
3.2 Design principles	16
4 Implementation	17
4.1 Front-end	17
4.1.1 Video player	17
4.1.2 Log	18
4.1.3 Chest detection pane	18
4.1.4 Configuration	20
4.1.5 Progress bar	21
4.1.6 Status	22
4.2 Back-end	23
4.2.1 Person detection	23
4.2.2 Chest detection	23
4.2.3 Chest tracking	24
4.2.4 API calls	25
5 Evaluation	27
5.1 Testing	27
5.1.1 Manual front-end tests	27
5.1.2 Final user tests	28
5.1.3 Demonstrations with client of Popup-escape	29
5.2 Code Quality	29

5.3	System Requirements	30
6	Conclusion & Discussion	31
6.1	Conclusion	31
6.2	Discussion	32
6.2.1	Activity detection	32
6.2.2	Chest detection	32
6.2.3	Other improvements	32
6.3	Ethical Implications	33
6.4	Recommendations	33
A	Original Project Description	35
B	Research Report	36
B.1	Escape rooms	37
B.2	Recognizing objects in video	38
B.2.1	Object detection	38
B.2.2	Object tracking	40
B.2.3	Feature extraction and person descriptor generation	42
B.2.4	Object (re-)identification	43
B.3	Sound	43
B.4	Recognizing Behaviour and Emotions	44
B.4.1	Behaviour	44
B.4.2	Emotions	45
B.5	Progress	46
B.6	Conclusion	47
C	Configuration Manual	49
D	Test Scenarios	50
D.1	Menu tests	50
D.2	Detection tests with streams	53
D.3	Progress tests	54
E	User Test Interview	57
F	Software Improvement Group	58
F.1	First SIG evaluation	58
F.2	Second SIG evaluation	59
G	Info Sheet	60
	Bibliography	61

Chapter 1

Introduction

Escape rooms are a relatively new form of entertainment. Nicholson [1] defines escape rooms as *live-action team-based games where players discover clues, solve puzzles, and accomplish tasks in one or more rooms in order to accomplish a specific goal (usually escaping from the room) in a limited amount of time*. Essential for escape rooms and game hosts is that people that play the game have a good time. The game host is responsible for accomplishing this.

Popup-escape is a company that designs and builds escape rooms. When an escape room is played, a game host needs to keep an eye on the players to ensure that they will have a good experience. In order to achieve this, the host can give hints to teams that are stuck in a phase or behind schedule. Popup-escape would like to support their game hosts. As every escape room should be a positive experience for the players, it is important that the game host has a correct overview of the current situation in the escape room. This is why Popup-escape would like the game hosts to be supported with their tasks of watching players and giving hints, so that any team can complete the escape room with as much help as they need. This ensures that no players leave the escape room feeling disappointed or angry because they were not able to make much progress.

In this document the process of building a software application to support the game host is described. In this chapter, first the problem at hand is defined. This is necessary to understand what potential solutions could fit the problem. Next, the defined problem is analysed to figure out the necessary requirements of a fitting solution. These functional requirements demanded research to find out how one could best implement them. The highlights of this research are described in chapter 2. In the succeeding chapter the design of the application is discussed. The design is based on the researched requirements of the solution for the problem and on several design principles. How this design of the product is eventually implemented is discussed in chapter 4. During the development process, the product has often been tested. The results of the tests and the quality of the code are indicators for the success of the product. These evaluation factors are talked about in chapter 5. Subsequently, in the last chapter, conclusions are drawn from the findings of the project, and the success of the project and its difficulties are discussed. Ethical implications are also brought to attention. And lastly, some recommendations are given for future work that could be valuable additions to the application.

1.1 Problem description

The client has indicated the desire to have an application that supports the role of the game host in the process of observing and interacting with players. The challenge of the project is therefore to support (and partly automate) the game host in the process of interacting with players that are playing an escape room. The game host observes players through cameras and interacts with players by giving hints based on the state of the progress of the room and the time that has passed. Currently, the host can only check whether players are on schedule (and if they are likely to finish the escape room on time) by paying close attention to the live camera streams and checking the progress of the players. If the players are getting behind schedule, the host should give the players a hint, so they can catch up. The progress indicates how far players are in the process of completing the escape room. The observation process of the host can be partly automated.

The goal of this project is *to provide the game host with an application that uses camera footage to observe the escape room and track the progress players are making in the game.* This will help the game host to have a clear overview of the current state of the escape room. Moreover, this allows the host to make judgement calls about whether the players are in need of a hint or not.

The resources available for this project are several cameras, objects that are used in escape rooms and gathered data. The data consists of camera footage of multiple escape room games that can be used for testing purposes. The cameras allow us to work with live streams and to record more videos while the escape room objects enable us to create actual escape rooms. The camera footage can be used to detect people and objects, as well as to track these same people and objects in the room. Furthermore, people's movements or the status of objects (e.g. a chest inside an escape room can be open or closed) can be useful for keeping track of the progress of an escape room. The client has indicated that he would like the application to work on a generic escape room, which means that it should be able to deal with most escape rooms.

More data (beside the camera footage) could be extracted from sound recordings. However, currently sound is not recorded in the escape rooms. The game host is also a form of input that can be used to gather data: observations made by the game host are a valuable input source for the application.

The final software application should be able to keep track of players and relevant objects in a generic escape room, and give useful information about the progress of the game (extracted from the given input) through an intuitive interface. Furthermore, the application should be functional and well-tested. The final product aims to provide the game host with a better sense of the progress of the game than he would have without the application. This enables the game host to better decide upon whether he should give hints or not. The application will process gathered data in order to estimate the progress of the game.

1.2 Problem analysis

Based on the problem description, the problem is further analysed in this section. As stated before, the desired product should be a support tool for the game host. The tool should observe the escape room through camera footage and track the progress of the game with the extracted information. To ensure that we are in fact developing the product that the client wants, we have set up requirements for the product.

Since it is important that the requirements are prioritized for the development process, the requirements are analysed with the MoSCoW method. This method groups requirements by their priority level. The minimum viable product is described by the *Must-Have* requirements. *Should-Have* requirements are desired but not necessary for the product. *Could-Have* requirements are extra requirements that can improve the product but are only implemented if time allows it. And lastly, *Won't-Have* requirements are requirements that will not be implemented during this project but that could be interesting for future projects.

After discussing all potential requirements of the product with the client, the requirements have been prioritized. The client indicated that the end product must at least be able to display live camera streams of escape rooms so that the game host can watch the players. Furthermore, the product must detect moving people and open chests in the room, as these two factors are indicators of the progress of the game. For example, when people are standing still for a considerable amount of time, it is likely that they are not making any progress in the process of completing the escape room. Likewise, when a chest is opened in an escape room, the players continue to a new stage of the game. Additionally, the product must display useful information about the game to the host, so that the game host can easily decide whether players need a hint or not. This useful information can for example be the current progress of the game compared to the time players have to escape. As a matter of course, all written code should be well-tested and of high quality. These requirements are necessary for the minimum viable product.

Other wishes that the client has, are the tracking of people and chests in the escape room, as well as the ability to detect other relevant objects in escape rooms beside chests. Tracking people can help in the process to discover where people are in the room when there are occlusions that block the detection of people. Tracking the location of chests can help the application to not detect the same opened chest as a second opened chest for example. Input from the host should also be asked whenever it is helpful to measure the progress of the game. Furthermore, the application should work on generic escape rooms, so that the client can use it for (almost) all constructed escape rooms.

There are also some requirements to which the client gives a very low priority. These requirements could be implemented in the final product should there be enough time. A complete overview of all requirements can be found in table 1.1.

Table 1.1: The requirements of the product

Priority	Type	Requirement
Must-Have	Functional	The application must be able to load in and show a live camera footage stream in real-time.
		The application must be able to detect people on camera footage.
		The application must be able to detect chests on camera footage.
		The application must be able to detect whether a chest is open or not.
		The application must be able to detect whether a person is moving or not.
		The application must have a Graphical User Interface that displays useful information for the game host.
	Non-functional	There must be at least 80 percent line coverage of the back-end code.
		The code of the application must score at least 4/5 on the SIG maintainability check.
Should-Have	Functional	The application should keep track of the location of people.
		The application should keep track of the location of chests.
		The application should be able to detect other relevant objects.
		The application should work with interaction from the game host.
		The application should work on generic rooms.
Could-Have	Functional	The application could keep track of the location of other relevant objects.
		The application could map the layout of the room.
		The application could support the use of multiple rooms.
		The application could track certain keywords said by players in game.
		The application could track the behaviour of people.
		The application could label the players by tagging their name.
		The application could deal with persons who leave the field of vision.
The application could track knowledge of people about the game.		
Won't-Have		The application will not track emotions of people.

Chapter 2

Research

The problem analysis brought forward the wishes of the client. To find out how to implement the desired requirements for the product, we needed to dive into the literature, as some of the requirements are not that straightforward to implement. The client indicated that the application must detect and track people and chests in the room, and that the application should work on generic escape rooms. Additionally, the progress of the game should be measured. This measurement will guide the game host through the observation process. This chapter only describes the highlights of the research that have been used in the final product. The full research report can be found in appendix B.

2.1 Detection and tracking

As stated before, the application must be able to detect and track people and chests in the room. Consequently, several techniques on detection and tracking have been researched. The techniques that we will use during the development process are explained in this section.

In object recognition different phases can be distinguished. These phases are shown in figure B.1 found in an article by Bedagkar-Gala and Shah [2]. This article is about the detection of people. However, any object can be recognized in the same way. To be able to recognize an object in a video, the object must first be detected in the video. Since we are working with videos, the tracking of object movements is also important. Furthermore, a descriptor that describes an object by extracting relevant information from the video is needed. Lastly, the previously constructed descriptor can be used to (re-)identify an object.

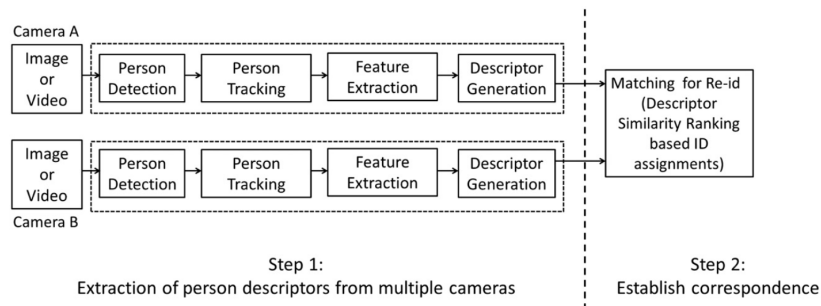


Figure 2.1: Identifying a person on multiple cameras [2]

Object detection

There are many different techniques that can be used for the purpose of detecting objects. In this section an overview of these techniques is given.

Yilmaz et al. [3] describes common object detection methods. These techniques are divided in four different categories: point detectors, background subtraction, segmentation, and supervised classifiers. In this project we will make use of background subtraction.

Background subtraction models the background of a scene. At each incoming frame deviations from this background model are detected. Considerable changes in an image frame compared to previous frames indicate a moving object. The pixels that indicate this change are further processed and an algorithm is used to classify the pixel region as an object [3]. An example of this method is shown in figure B.2. The pixels of the beforehand instantiated background model are subtracted from the pixels of a frame. If the difference in pixel values exceeds a certain threshold, an object is detected.

Background subtraction is used in most state-of-the-art tracking methods for fixed cameras. The reason for this is that it is insensitive to illumination changes, noise and periodic motion of the background region. This means that it can accurately detect objects under different circumstances. Also, the methods are computationally efficient and can therefore detect objects quickly, which is beneficial for real-time tracking. Background subtraction can be easily implemented with the OpenCV library [4].

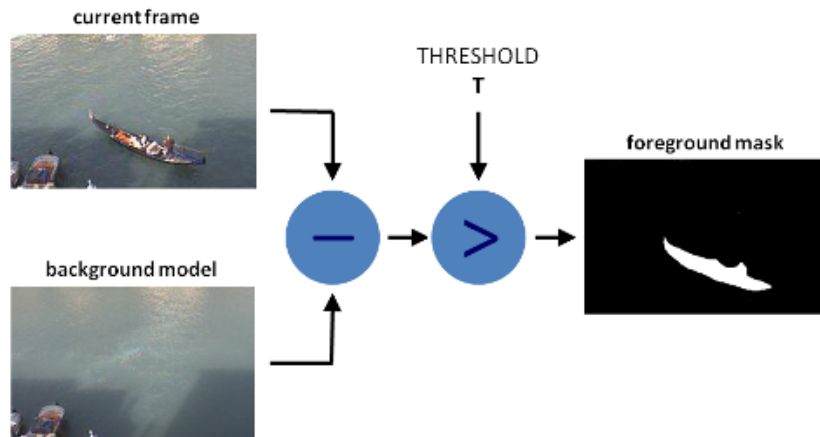


Figure 2.2: Background subtraction example [4]

Object tracking

The task of object tracking in general can be described as finding out where an object moves to and what path it follows throughout subsequent frames of a video. Every tracking algorithm needs an object detection mechanism either in every frame or when the object appears in the video for the first time. Some common tracking methods are described in Yilmaz et al. [3], where they are divided into three different categories: *point tracking*, *kernel tracking* and *silhouette tracking*. The silhouette tracking technique will be adopted in the implementation of the final product.

Silhouette tracking estimates an object region in each frame by using previous frames. Information about the region may be considered as the appearance density and shape model. By using object models, silhouettes are tracked by either shape matching as shown in figure B.4(c) or contour evolution as shown in figure B.4(d). The only techniques that are suitable to track multiple objects and handle full occlusion belong to the subcategory contour evolution. Within this category only the techniques *particle filtering* [5] and *gradient descent* [6] are relevant. It is not known whether these techniques can be used in real-time. The first technique tracks a maximum of only two objects and is therefore not relevant for this project. The second technique works well with background subtraction (for object detection). Furthermore, this technique focuses on detailed analysis of the shape deformation during intense action (such as sports). Therefore, this technique is not completely suitable for this project as such motions are not expected and may make the application unnecessarily computationally inefficient.

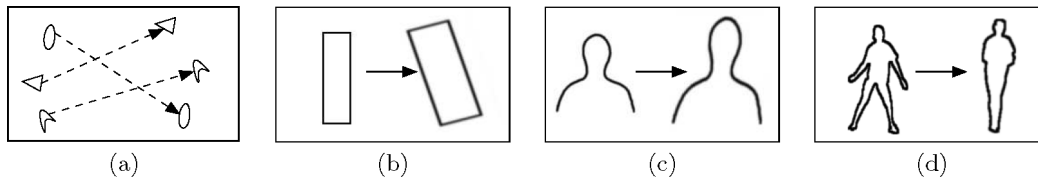


Figure 2.3: Tracking approaches: point tracking (a), kernel tracking (b), silhouette tracking by shape matching (c), silhouette tracking by contour evolution (d) [3]

When tracking objects one should wonder if it would be a good idea to use multiple cameras as opposed to only one. Using only one camera means a limited field of vision, limited depth determination and a higher chance of occluded objects. However, it may be easier to implement detection and tracking algorithms using one camera than multiple cameras.

Using multiple cameras can increase the field of vision where objects can be found. The difficulty here is that a good feature extraction for objects is needed, so that the same object can be recognized on multiple cameras.

Kumar et al. [7] propose a way of using multiple cameras in real-time. When using multiple cameras instead of one, it increases the chance of detecting objects. This is because of the afore mentioned reason that more cameras increase the total field of vision. It also means that objects that get occluded on one camera might appear on a different camera, making it possible to find them.

For this project at least two cameras are available for use. The cameras can be placed opposite of each other in the room, so that the field of vision is maximized. The cameras produce video feeds that are saved in a Real-Time Streaming Protocol (RTSP), which can easily be processed with Java as the programming language. This, together with our personal preferences, was the reason to use Java for this project.

Feature extraction and descriptor generation

Feature extraction and descriptor generation are combined in this section due to the fact that all extracted features will be put into an object, and this object is the descriptor of a specific item. There are several methods that can be used to extract features from objects: SIFT, salience maps, texture extraction, and colour extraction. In the implementation of the product we will make use of colour extraction.

There are multiple techniques which describe colour in an image. A couple of those techniques are described in a paper by Manjunath et al. [8]. One promising descriptor is the *dominant colour* descriptor, which returns the salient colour distribution in the image. Another descriptor defined is the *scalable colour descriptor* (SCD). Simply put, the SCD defines a HSV colour histogram of the image and compresses it using the Haar transform. The third technique is called the *colour structure histogram* and aims at identifying localized colour distributions by using an 8x8-structuring element. The resulting histogram is constructed in the HMMD colour space.

2.2 Generic escape rooms

All escape rooms require players to solve puzzles and accomplish tasks in one or more rooms in order to escape the room in a limited amount of time. There are however different paths of puzzles that players must follow for escape rooms. There are three main approaches for path creation: *linear path*, *open path*, and *multi-linear path* [9]. In a linear path, puzzles must be solved in a specific order: the solution of one puzzle gives access to the next puzzle, and so on. This approach allows for a fairly simple escape room. Differently, an open path contains puzzles that can be solved in any order. Usually there is a final puzzle that cannot be solved until all the other puzzles have been completed. While an open path allows for all team members to be able to always do something to help the progress, in a linear path however one puzzle may be solved by a single person while the rest cannot do anything. Lastly, a multi-linear path is a combination of the linear and open path approach. It involves a series of linear paths that can be done in parallel.

In every escape room one will find chests to be present. These chests can be opened by a key that needs to be found or a code that needs to be discovered. The chests that need to be opened are key points in the escape room and indicate the beginning of new phases of the game. It is therefore that chests are an important part of a generic escape room.

For this project, the application will focus on escape rooms that have linear paths only. This allows for a good basis that is easy to work with and it can be expanded on in future projects. This generic escape makes use of chests that need to be unlocked during the game.

2.3 Measuring progress

The application should be able to measure the state of progression in the game. Since there was no relevant literature concerning this topic, we put our heads together and discussed the matter with the client to come up with solutions.

One indication for the progress of the game is how many of the total amount of chests have been opened. When a chest is opened, its contents lead to the next puzzle to solve. Thus, for every chest opened, the players make progress in completing the game.

To know how many of the total amount of chests have been opened, tracking algorithms will be used. To make the detection of chests easier, chests can be given a certain colour or pattern. Another option to gather information about the status of chests or other objects (such as objects part of a puzzle), is to put trackers or sensors on objects. These trackers or

sensors can then pass along signals to the application.

One more important aspect of progress in escape rooms is time. The time that has passed since the beginning of the game can be a good indicator of the phase of the game in which the players should be. When players are solving puzzle *A* and 10 minutes have passed, while puzzle *A* could normally be solved within 5 minutes, the group takes too long to solve puzzle *A*. This should then alert the game host to give a hint. If puzzle *A* is normally solved in 20 minutes, and only 10 minutes have passed, no hints need to be given. There is always a path of puzzles that leads to the opening of a chest, and chests will be tracked in the application. Therefore, the progress of an escape room and whether players are behind schedule or not, can be measured by comparing the time players spend on the required process to open a chest, and the time it usually takes to open this same chest. The game host knows how much time players should spend on every chest in the escape room.

The time that people are standing still instead of moving around can also be an indicator of the progress of puzzles. When people are not moving for a long time, it probably means that they are stuck at a certain point. However, standing still does not necessarily mean that the players are stuck in the game, as people might also stand still while solving puzzles. Therefore, all these analyses of deriving the progress of the game are assumptions and must be confirmed by the game host.

Chapter 3

Design

The requirements that are set in the problem analysis form the basis for the design of the product. This chapter gives an overview of how the application is set up and which decisions are made to meet the requirements. Both the back-end and front-end side of the code, as well as their functionalities, are discussed. Furthermore, this chapter also focuses on several quality attributes that are important for the product.

3.1 Overview

As earlier mentioned in the problem description, the product that is asked for by the client should offer support to the game host. Popup-escape uses cameras in their escape rooms that make use of Real Time Streaming Protocol (RTSP). This is a network protocol for the use of real time streaming. Several escape room games at Popup-escape have been recorded with these cameras and are made available to us to work with by Popup-escape.

Because the product needs to support the game host in observing the escape room, a minimal requirement is that the product must be able to display RTSP streams coming from the provided cameras in an informative Graphical User Interface (GUI). Moreover, the product must show important information, extracted from the camera footage, about the escape room. This information should give a good representation of the progress in the escape room and one way to do that is by detecting opened chests in the room. Many escape rooms make use of chests which players need to open to make progress and the amount of opened chests therefore gives a representation of the progress. When no chests are opened after a long time of playing, it indicates that the people in the room are not progressing well. To open a chest, sometimes certain sections need to be solved to get the key or code for this chest. These sections can be any sort of puzzle and it can give a representation on how close the people in the escape room are to open a chest. It is difficult to track the status (finished or not finished) of the puzzle, because it can be any sort of puzzle. However, it is possible to include a sensor inside the puzzle to send a signal when the section is completed. A requirement for the product therefore is to deal with these signals. To do that, the product needs to accept Application Programming Interface (API) calls so that these signals can be received. Chests could also have sensors which send a signal when the chest is opened and these signals need to be processed by the application as well.

Another important aspect of the game are the people who are playing it. Hence, information could be extracted from person detection. From research that was done by inspecting an

escape room being played, it became clear that the activity of players is important information for the game host. When people are not moving in an escape room, they are usually not progressing fast. That is why measuring the activity of people is an important requirement.

For the chests, as for the sections between chests and the activity, the time is an important aspect for the game host. The amount of time it takes to open a chest for example, helps the game host to know if players take too long to open a chest or complete a section. That is why time will be an important aspect for the application.

The design of this product is constructed in such a way that all of the requirements can be met. A global overview of the design is depicted in 3.1. The design consists of two main components: the front-end (the user interface) and the back-end of the application. While the front-end deals with user input, the back-end deals with the input from RTSP streams and API calls. Frames are requested by the front-end and retrieved from the back-end. For the information from the back-end two queues are created. One queue handles information sent from the back-end to the front-end. The second queue handles detected chest frames. The structure of the design allows the application to make use of multiple forms of input. All this input is processed and displayed in a responsive and informative GUI for the host.

The front-end of the application is linked to the back-end of the application. In the next two sections the design and the functionalities of the back-end and front-end are explained in more detail.

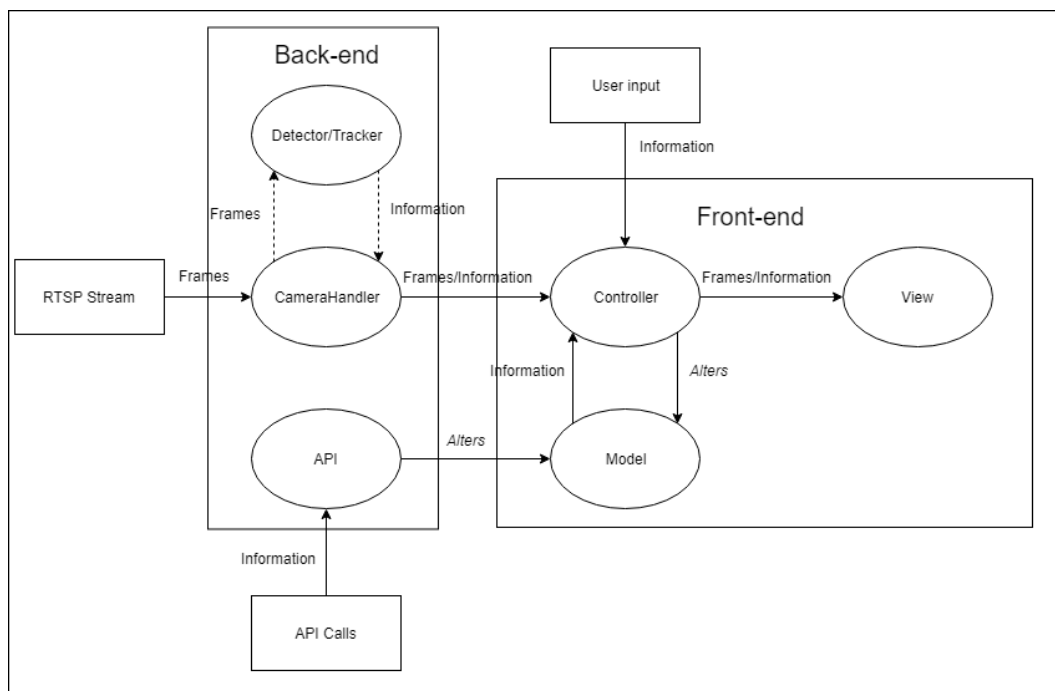


Figure 3.1: Global overview of the application.

3.1.1 Front-end

One of the functional must-have requirements that the product needs to have according to the MoSCoW analysis in section 1.2, is the requirement of having a GUI. This GUI needs to show live streams sent by the RTSP cameras and display useful information of the played

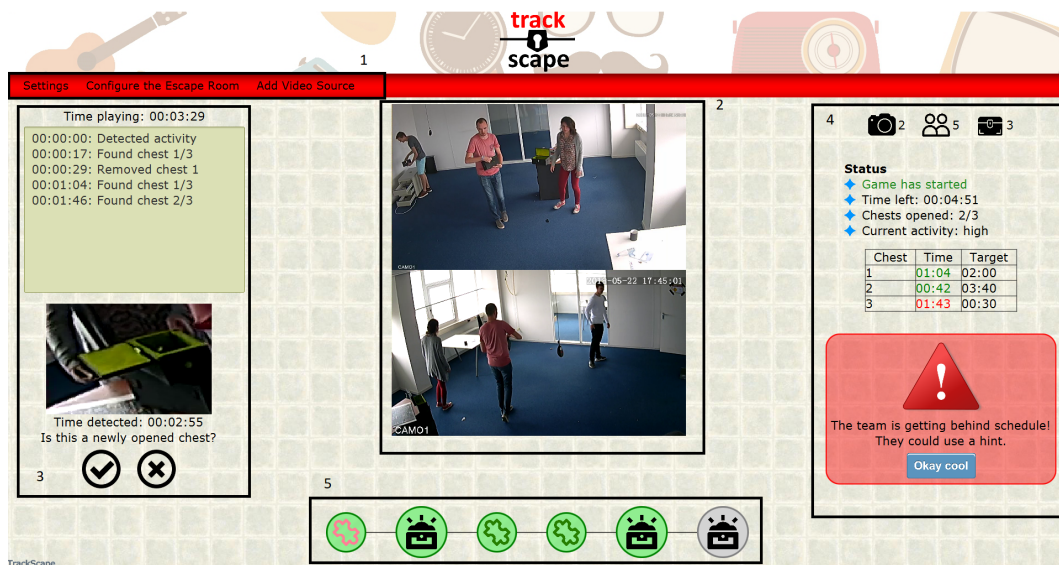


Figure 3.2: Overview of the components of the application

escape room to the game host. The GUI can be divided into the following components: the menu, video(s), the timer and logger, the game status, and the progress bar. Each of these components is further explained.

Menu

In the menu, as shown by number 1 in figure 3.2, three drop-down menu tabs are present in which the game host can exit and reset the application, configure the escape room in various ways and add (additional) media in the form of videos or streams.

The first menu tab allows the user to either reset or exit the application.

In the second menu tab the escape room can be configured by loading a predefined JSON configuration file from the local PC hard drive. It can also be configured by using a standard configuration, which the game host can set beforehand for easy access. This is useful when the same escape room is played often. Furthermore, the game host can configure the escape room inside the application without creating a JSON file beforehand. For this option, a window appears in which the game host can fill in the amount of players, amount of chests and the total duration of the game. Then for each chest the amount of sections are asked for, as well as the time when the first warning should be given (warning time) and the time at which the chest really needs to be opened (target time).

The last menu tab is used when the game host wants to add media such as a stream or a video file of the escape room to the application. A JSON file already contains the path to the video or link to stream, so when a user decides to include a stream in the JSON file while configuring the escape room beforehand, this option is not necessarily needed. However, when the user manually configures an escape room inside the application, adding a stream or video to the configuration is needed before being able to start.

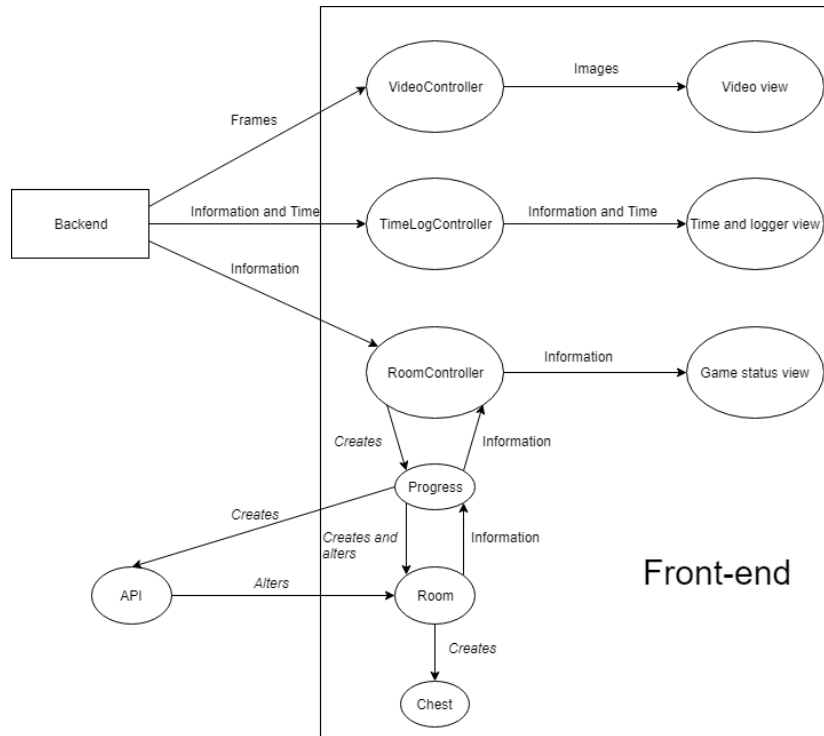


Figure 3.3: Overview of the front-end.

Video(s)

To display RTSP streams from cameras inside an escape room, a pane needs to be created where the camera footage can be shown. In the middle of the GUI one or more videos can be shown, as seen by number 2 in figure 3.2. These can be pre-recorded videos, but the most interesting videos are of course the live streams. In this way the game host can always see what is happening in the escape room. The *VideoController* (in figure 3.3) receives the frames from the back-end and processes them to show to the game-host. The possibility to show pre-recorded video files is also present, which can be used for testing purposes.

Timer and logger

The GUI needs to show important information to the game host, for example the time played by the people in the escape room and when chests are opened. At the left side of the screen, as shown by number 3 in figure 3.2, a timer is displayed which shows the total time that has passed since the first activity was detected. Below the timer, a text box is drawn in which textual logs including timestamps can be printed. These logs contain important information about the game, like when the escape room started and when chests are opened.

Below the information log an image of the detected chests including their time stamps will be shown. The game host is then asked to either accept or deny the detection of the chest. Only when the user accepts, an appropriate log is printed in the logger. Otherwise there is no information shown in the log.

The information shown is controlled by the *TimeLogController*, as seen in figure 3.3. This controller reads from the information queue where information from the back-end is placed.

Also information can directly be shown by the controller, as explained above. The images of the detected chests are also passed through the information queue from the back-end and shown sequentially.

Game status

To give a clear overview about the status of the game a game status pane is created. This pane is located at the right side of the video pane, as seen by number 4 in figure 3.2 and contains information about the chests, activity and people in the room.

At first the amount of cameras, people and chests in the escape room are shown. Below this, the status of the game is shown. It is shown whether the game is started or ended, how much time is left, how many chests are opened and what the current activity is in the escape room.

Underneath, a table is drawn where for each chest it is shown what the target time is. Left from the target time, the time of how long the players are busy with the chest is shown. The timer of the first chest starts when the first activity is detected and the other timers start when the previous chest is finished. By the timers, the game host can see whether the players are getting behind schedule. If this is the case, a warning pops up showing this message. This message can also be shown earlier when a warning time earlier than the target time is set in the configuration.

The Game Status is controlled by the *RoomController* in figure 3.3. The *RoomController* receives information from the model, where information of the room and every chest are stored. It also alters information of the model when for example a chest is set to be opened. Information about the activity is received from the *CameraHandler*.

At the bottom part of the application, a progress bar is drawn showing all the chests and its corresponding sections that need to be completed for each chest. This is shown by number 5 in figure 3.2. Like the game status, the progress bar is controlled by the *RoomController*. The controller gets information about the progress in the room from the progress object and the bar is filled accordingly. The progress bar can also be clicked on, for example to undo progress or set a chest to done when it is not detected. The *RoomController* then updates the progress object with the new status of the game.

3.1.2 Back-end

The back-end is responsible for processing and detecting frames in camera footage. It detects the activity and objects in an escape room and sends this information to the front-end.

Firstly, the activity in an escape room is important information for the game-host to have. It is helpful to know if people are moving or not, to know how they are progressing in the game. The activity of people is done by detecting people in a room and tracking their movement pattern. While it was decided that it is a must have to detect people in a room, the most information we wanted to perceive with this is the activity in an escape room, and this can be done in different ways. Another way of detecting activity on camera is to search for differences between frames of the camera footage. Under the condition that a camera is not moving, the only differences between frames can be caused by people moving around, or a person moving an object. That is why the difference between frames is a useful method to determine the person's activity in a room. The *CameraHandler* in the software sends frames to an activity detector which uses Background Subtraction to search for differences on camera

footage. Every activity value is stored and to determine if activity is low, the activity value of a certain frame is compared to previous activity values.



Figure 3.4: Background subtraction. The more white pixels there are the more activity there is on the frame. Therefore for the activity, the amount of white pixels is counted.

A downside of detecting activity this way is that big objects, or objects close to the camera, show higher activity than objects far from a camera. This is because objects close to the camera take up more pixels on the frame, causing more pixels to be changed. Using multiple cameras and taking the average activity is a way we use to converge the activity value to the true activity in the room.

The amount of chests opened in an escape room is also helpful as it gives a representation about the progress. Only an opened chest needs to be recognized, because the amount of closed chests is not that important or can be easily deduced by knowing the total amount of chests and amount of opened chests. The inside of a chest can therefore have something recognizable to detect. That is why the inside of the chest is painted in a striking colour (in our case we painted the inside of the chests fluorescent yellow as seen in figure 3.5).

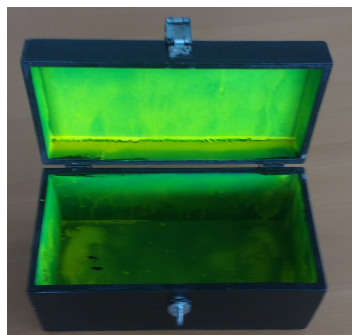


Figure 3.5: One of the chests used for detection

The detection of the painted opened chests is done by searching for the pre-described colour in the frames. These detected chests are sent to the front-end for confirmation. The detection is done every frame, so the problem arises that chests are recognized multiple times. To deal with this, the contours of the colour are tracked. Overlapping detected chests in subsequent frames are discarded. It is also possible to have objects in the background which have the same colour as the inside of a chest. That is why a background subtractor is used to cut out the background so it can not be detected. A third problem is that movable objects also could have the same colour as the chest. The tracker should ensure that this object is only detected once, so the host only needs to tell the application once if the detected object is a newly found chest. The tracker and background subtractor process the frames independently from each other, as seen in the sequence diagram of figure 3.6.

In a later stage of the project another approach of detecting opened chests was given. Chests or other objects (such as a door through which the participants can escape), can have sensors attached to them which send a signal when they are opened. For our application an API is needed to be able to deal with external signals. A server is therefore created which can handle requests. An API call can be made to set the next chest to opened or to set the next section to opened. The API handler then sends this information to the front-end which alters the progress of the escape-room.

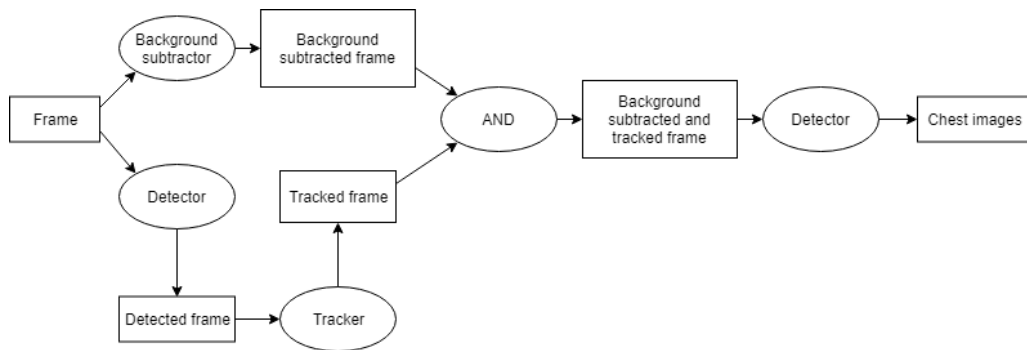


Figure 3.6: Sequence of detecting a chest.

3.2 Design principles

During the development of the product we stuck to all of the SOLID principles for object-oriented computer programming. These five principles make software designs more comprehensible, flexible and maintainable.

The first principle, the *single responsibility principle*, says that a class should only have a single responsibility. In other words, a class should have only one job. The second, *open-closed principle*, indicates that objects should be open for extension, yet closed for modification. This means that a class should be easily extendible without having to modify the class itself. The *Liskov substitution principle* states that objects should be replaceable with instances of their subtypes. The *interface segregation principle* means that many client-specific interfaces are better than one general-purpose interface. Finally, the *dependency inversion principle* implies that high level entities should depend on abstractions (and not on concretions),

Together these principles form the basis of high quality software, which is what we are working towards for the final product: understandable and maintainable software.

Chapter 4

Implementation

This chapter shows for each component in the product a description of how it is implemented. We start off by talking about the items in the front-end. Afterwards, the implementation of the back-end will be explained.

4.1 Front-end

The front-end of our application is responsible for user interaction. In this part we will therefore explain what happens when an element is interacted with. We will go more into detail on what happens when interaction has been made.

4.1.1 Video player

Loading in a video

When the menu item *Add Video Source*, shown in figure 4.1, is pressed, a stream or video file can be loaded into the application. Whenever someone chooses to load in a stream (or video file), a new camera is added to the *CameraHandler* class with that stream as video source. The OpenCV class *VideoCapture* can make a connection with both files and RTSP streams by simply giving a string representation of the destination of the video source.



Figure 4.1: The menu pane as shown in the GUI

Start

Whenever the start button, which is the button in figure 4.2, is pressed, the *CameraHandler* class starts to ask for new frames for each of their instantiated cameras. When no cameras are instantiated, the application shows a pop-up which says that there needs to be at least 1 instantiated camera before the start button should be pressed. After this button gets pressed and everything is instantiated correctly, the videos will start playing.

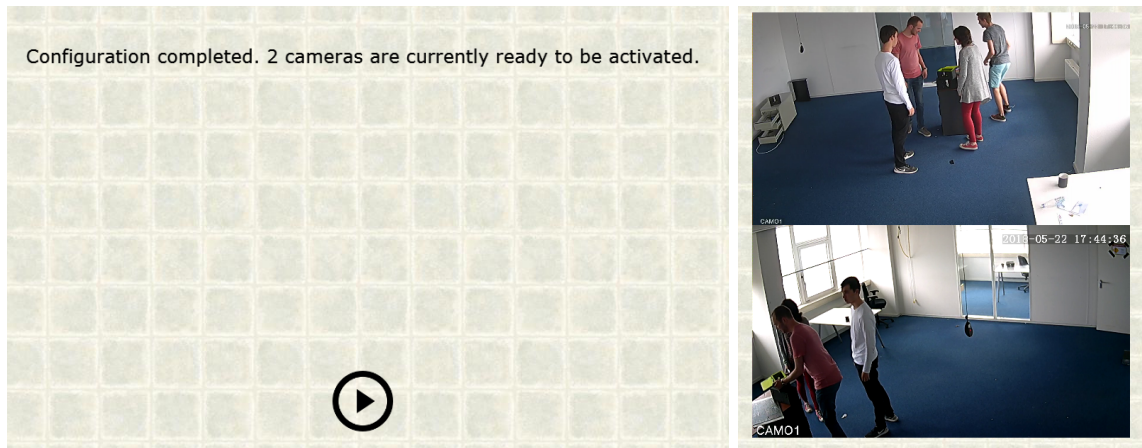


Figure 4.2: The video player before (left) and after (right) the start button is pressed

4.1.2 Log

In the log pane (figure 4.3) various events are projected. When an event, for example a chest gets found, gets triggered, this is shown in the log pane. The log message is constructed by getting the time from the timer above the pane and a event specific message. Events that are part of the progress are logged with an extra progress indicating part. The 1/3 in figure 4.3 is an example of this. This progress part is computed by looking how many chests are opened. Together with the total amount of chests the progress gets shown as: [chests opened]/[total amount of chest].

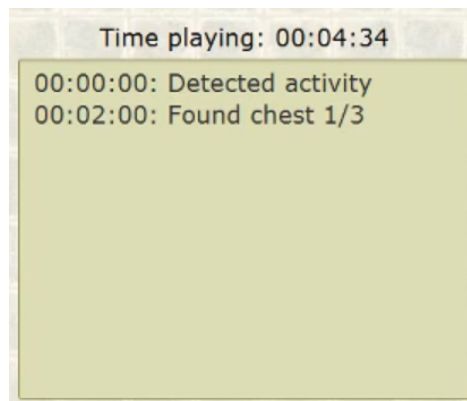


Figure 4.3: The log pane as shown in the GUI

4.1.3 Chest detection pane

The chest detection pane shows a picture of a potential newly opened chest. All potential chests are received from a queue, which enables us to show only one potential chest at a time. In addition to receiving the image of a potential opened chest from this queue, the time since the start of the game (this is the time as shown above the log pane) is also obtained.

To verify if the potential chest is a newly opened chest, we ask the game host for input via the two buttons located below the image. In our code we can set a boolean flag, which

can be found in the class *TimeLogController*, in order to save images whenever one of those buttons is pressed. Whenever approve is pressed images will be saved in `./files/chests/correct` and if disapprove is pressed the images will be saved in `./files/chests/incorrect`. The boolean flag can only be set from within the source code in the class *TimeLogController*. We have added this functionality after the client requested it, so that the gathered images can be used to improve the detection of chests (possibly via machine learning) in the future.

Approve

The first of those two buttons is the approve button. By pressing this button the game host notifies the application that the chest shown in the image is indeed a newly opened chest. After pressing the button we loop through the list of chests and call the `approveChestFoundByHost()` method on the next chest. The next chest is the only chest that has as status `To_Be_Opened`. Whenever a chest has the status `To_Be_Opened` and `approveChestFoundByHost()` gets called on it, the status gets changed to `Opened` the next time `update()` is called.

Now that the chest is opened the log will get a new message printed in it, saying that the next chest is opened. When a configuration of the escape room has been loaded or filled in, the progress bar and the status pane will get updated as well to show that the chest has been found.

When every GUI element that needed to be updated has been updated, the time of the found chest is saved. All new potential chests which are found between the time the approved chest was found and the five seconds later are discarded. This time out is added for the reason that in escape room generally chest are not opened shortly after each other. The timer ensures that the chance of detecting chests within a very short time is significantly decreased.

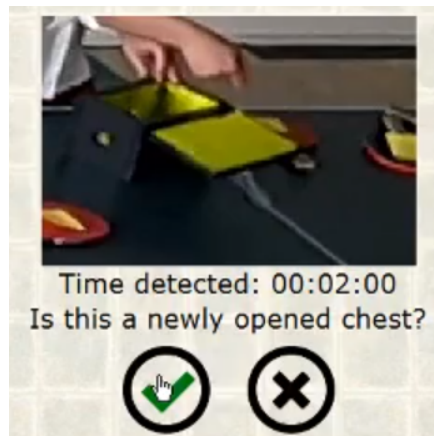


Figure 4.4: An opened chest found by detection

Disapprove

The second button is the disapprove button. Whenever this button is pressed the current possible chest is removed from the queue. As opposed to pressing the approve button, the disapprove button does not start a timer nor does it adjust anything in the back-end.

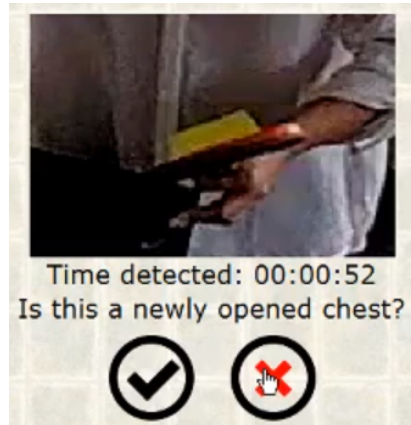


Figure 4.5: A post-it found by detection

4.1.4 Configuration

In our application it is possible to configure the escape room via a JSON file or manually in the application. By configuring the escape room, one can set parameters which are used to keep track of the progress. It is possible to set the amount of people, the time until the escape room is finished, the port that listens to API calls, the cameras, and the chests.

For cameras and chests some additional parameters are asked. Cameras should have a link or file given to them, in order to instantiate the camera to read frames from that link or file. Chests, as being our main approach to keep track of progress, need the amount of sections that need to be completed until the chest is opened, the time when players get behind schedule (the warning time), and the time when the puzzle needs to be completed (the target time). The amount of sections refers to the amount puzzles, including opening the chest, that need to be solved in order to advance in the escape room.

Information retrieved from either the JSON file or the manual configuration is used to construct a progress object. A progress object constructs a room and can retrieve information about the progress from that room once the escape room has started. The room object itself holds all the information retrieved after configuring the escape room.

File

When using the JSON file to configure the escape room, you can fill in all of the aforementioned parameters. A guide on how to write this file correctly can be found in appendix C. When this JSON file is loaded in to our application via the menu, we parse it using a simple JSON parser. Information retrieved by this parser is then used to construct a progress object in the model as described before.

Manual

When using the manual configuration option in the menu, you can fill in a form asking for most options that are also present in the configuration file. In figure 4.6 it is shown what the manual configuration looks like. The only thing that cannot be instantiated here is the port for the API calls, which will become the default port 8080.

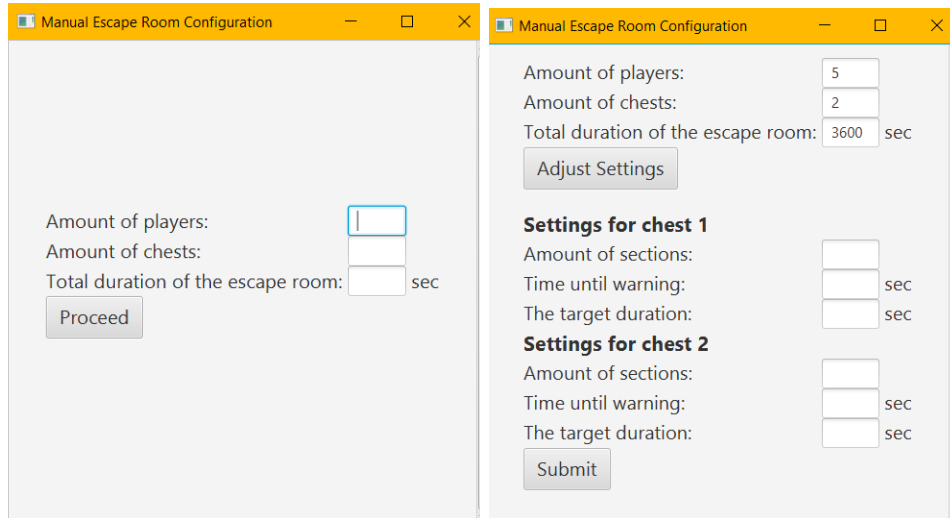


Figure 4.6: Manual configuration

4.1.5 Progress bar

The progress bar, figure 4.7, shows icons of chests and puzzle pieces. The chest icons represent a chests that need to be opened during the game. The puzzle piece icons before each chest represent the puzzles that need to be completed in order to be able to open the chest. As the name might indicate, the progress bar visualizes the progress in the escape room. This progress is saved in a progress object which is constantly updated according to the real progress. When progress is made, the items that have been completed in the escape room are coloured green in the progress bar. When all icons are coloured green, all chests are opened. The point in the game where all chest have been found is not necessarily the end of the escape room, therefore only a message that all chests have been opened is shown in the status pane.

Furthermore, the progress bar has been made interactive for the game host to manually update the progress to the current state of the game. This might be needed when a chest has not been detected or when the approve button has accidentally been pressed. The puzzles can only be completed by making an API call or manually pressing them in the progress bar. When a chest icon is pressed, the progress object gets updated. This leads to all chests being set to represent the current progress. When a chest icon is green the chest status is set to Opened. When a chest icon is grey and the previous chest icon is green, the chest status is updated to `To_Be_Opened`. All other grey chests get the status `Waiting_For_Section_To_Start`. For each green puzzle piece, the subsection of the chest it belongs to (the chest icon on the right) is set as completed.



Figure 4.7: The progress bar as found in the GUI

4.1.6 Status

The status pane in the GUI, figure 4.8, shows all kinds of information to the host. This information is all extracted from the back-end. At the top the amount of cameras, people and chest can be found. These values correspond with the configured parameters.

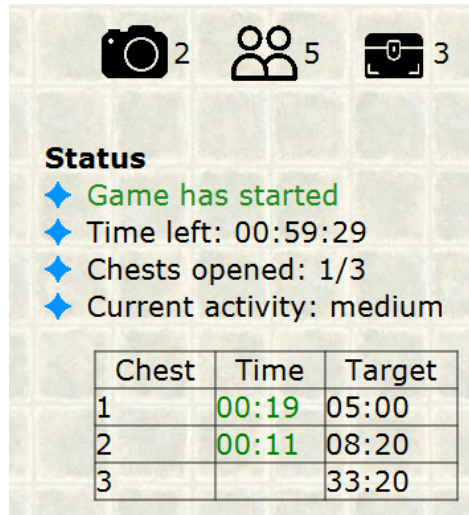


Figure 4.8: The status pane as found in the GUI

Below this information four items are placed. The top two have to do with time. *Game has started* can also display *Game will start soon*, when there has not been any activity, and *Game has ended*, when there is no time left. The next item, chests opened, shows the amount of chests that have as status Opened. Current activity displays the amount of activity. A detailed description of how the activity is calculated can be found in 4.2.1. Before any activity has been found we say that there is zero activity. While there is zero activity, we state that the game has not yet been started and therefore the timer does not run. This way the game host can already start the stream while the people have not yet entered the room, without having the game being started.

Below this a table is shown. This table gathers the target time from each chest and compares it with the time spent on this section (found in the time column) to decide if a warning should be shown. Once the time spent on the section exceeds the warning time of the chest, a warning, figure 4.9, is shown below the status pane. Whenever "Okay cool" is pressed the warning will get snoozed. This means that the warning will reappear if the snooze timer has drooped to zero and the chest has still not been found. Whenever the time spent on this section exceeds the target time, the warning will reappear (even when the snooze timer did not yet reach zero) and the time will turn red.

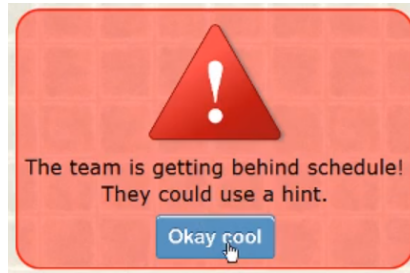


Figure 4.9: The warning pane as found in the GUI

4.2 Back-end

The back-end of our application deals mainly with processing frames, but also with handling API calls. In this section we will show how the processing of frames (detection/tracking) is done and how API calls are handled.

4.2.1 Person detection

At first we wanted to detect each person in a frame individually, so that we would be able to let the game host label each person with its name. This way the game host would be able to give personalized hints to the group of players. We tried to look for people in the frames by using a predefined person detector in OpenCV. This detector was however not working too well, therefore we decided to discard the idea of this detector.

While detecting people individually did not work, detecting them as a group did. In an escape room all objects are stationary, except for objects that are being moved by people. This means that all movement in the room is caused by people. The detection of a group of people can thus be seen as the detection of activity in an escape room. We implemented the detection of activity by using background subtraction. In background subtraction stationary objects will become background after a short while. This way changing pixels will be set to changed and not changing pixels will become background. Whenever pixels are set to be changed, we can say that some sort of activity is detected. By splitting the previous activities into three parts, we are able to distinguish between a low, medium and high activity level. Each of those activity levels has roughly the same amount of frames. These activity levels are shown in the status pane of the GUI, and can help the game host decide if people need a hint.

4.2.2 Chest detection

Opened chests are detected by searching for the colour in which the inside of the chests are painted. Frames of the camera footage are sent to a detector which has multiple steps in detecting the colour. First there are two methods used which analyse the frames differently. The first one uses the background subtraction from OpenCV to search for changes in the frames. This background subtraction is a neural network which trains the background continuously with every frame causing stationary objects to be placed into the background. The background will then be subtracted from the frame. This ensures only newly seen opened chests will be detected, because when the chest is not moving it will become part of the background.

Then another method from OpenCV, `Core.InRange()`, accepts a range of colours (represented as an upper and lower bound scalar) and returns a frame with binary pixels to show which pixels fall in the colour range. For this check we first convert our frame from BGR (blue green red) values to HSV (hue saturation value) values. By doing so we can ignore changes in illumination (the value from HSV). In our environment the chests are painted yellow-green on the inside, therefore we set the range to yellow. OpenCV, however, makes use of hues between 0 and 180 instead of 0 and 360. Our resulting scalars to use for the `inRange()` method are (17, 120, 80) for the lower bound and (35, 255, 205) for the upper bound. When using other colours the first parameter in these scalars can be adjusted by picking the right colour from figure 4.10 and dividing the degree by 2.

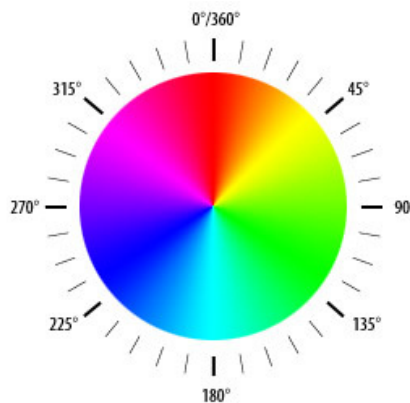


Figure 4.10: Degrees of HSV hue values

These two results are then combined and only the pixels which are true in both results are used for detecting the chests. The combination ensures that only new pixels with the painted colour of the chests are detected.

OpenCV provides functions for all these computations, which provide adjustable parameters for better results. The background subtraction parameters can be set to decide how fast unchanged pixels will be set to the background. Because we only want to detect newly opened chests and do not want to detect these chests multiple times, this is set to a low value.

The implementation of these functions results in a frame with the contours of the detected chests. OpenCV finds this contours and draws rectangles around them. These rectangles are parts of the frame which represent the detected chests. Before the parts of frames are used, the rectangle needs to exceed a predefined size. Otherwise very small contours, for example only one pixel, can be detected as a chest.

The resulting frames are returned to the *CameraHandler*. The *CameraHandler* sends the frames to a queue of frames with the founded time as timestamps. In the front-end this queue is read frame by frame and shown to the user for confirmation. In figure 4.11 it is shown that a chest is detected.

4.2.3 Chest tracking

In order to reduce the amount yellow-greenish object found in subsequent frames we decided to make use of tracking. Whenever we know that an object in the current frame is the same object as an object present in the previous frame, we discard that object from the potentially

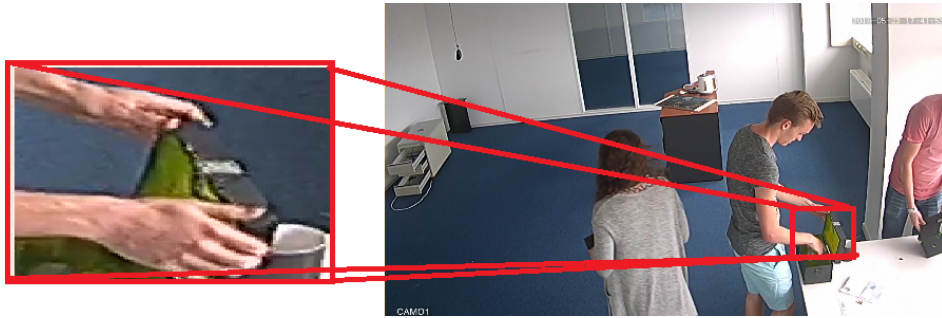


Figure 4.11: Detected chest (left) and its position in a frame shortly after the detection (right)

new chest list. The tracking is done by comparing the previous frame to the current one. In 4.12 you can see an example of our tracking method. In the current frame it is shown that object 1 has overlap with the red box around object 2, therefore we say object 1 equals object 2 and object 2 is not added to the potentially new chest list. Object 3, however, does not have overlap with any object in the previous frame, and is therefore added to the potentially new chest list. We chose to extend the area under which we search for overlapping objects, the red box, because objects can sometimes move entirely from the position in the previous frame but not too far. We do not track objects every frame, because tracking every frame would make our application too slow to use in real time.



Figure 4.12: Tracking chests

White areas represent an object. Grey areas represent an object in the previous frame. Red boxes represent the search area. Green numbers represent the object ID.

4.2.4 API calls

The API is responsible for handling HTTP requests. In the application Jetty is used for this machine to machine communication. The API module in the applications provide endpoints to which external requests can connect. Now there are two available requests to be made. One to request that the next chest will be set opened and one to set the next section, or puzzle, to be completed. The calls that can be made are **section?completed=true** and **chest?detected=true** preceded by **IPAddress:portid/** where **IPAddress** is the ip adress and **portid** is the port id. The default portid is 8080, but this can be set to another port in the configuration file.

The requests are simple HTTP protocols and are easily extended to more endpoints in our application. In further development for instance the API protocols can be extended by setting a certain chest to be opened instead of only the next one. The current design however limits the chests to be opened in a certain order, when chests can be opened in a different order the game-host needs to differentiate the chests in another matter.

Chapter 5

Evaluation

After the research phase and the three development sprints, the product was finalized. This chapter discusses the success of the developed product by looking at how the product was tested and at the quality of the code.

5.1 Testing

In the process of creating the application, all newly written code was thoroughly tested. Before any branch could be merged to the master branch, new features needed to be automatically tested. Eventually, a line coverage of 54.5% has been achieved. The code coverage of all back-end code is 97.5%. The code of front-end elements have been tested manually. These tests are performed by playing the role of the end user. For this, video footage from actual escape room games was used. This video footage was made available to us by Popup-escape. With these videos, it was possible to run the application on video frames from an actual game. Hence, it enabled us to gain insights in how well the application performed.

However, to be able to really assess the product, real user feedback was necessary. Therefore multiple user tests have been done during the development process. At multiple points in the development process demonstrations were shown to the client. After these sessions, the client gave feedback and shared his thoughts about the development process. This was very helpful and allowed us to focus on the priorities of the client. Finally, some manual test scenarios were performed to cover all the front-end functionalities. Also, a final user test was done with a real escape room host and with players from another project team. Results from the demonstrations, the manual tests and the user tests are given in the next sections.

5.1.1 Manual front-end tests

The components of the front-end are not tested by automatic tests as done for the back-end. In order to test all the components of the front-end properly, manual tests were performed. These tests, also called scenarios, consist of running and using different visual functionalities in the GUI of the application. The scenarios represent end user behaviour. A scenario test succeeds when the behaviour of the application is as expected. Each of the scenarios is carried out according to a certain format, as shown below.

Scenario *<no.>*: *<title>*

Date tested: *<the date>*

Description: *<the description of what will be tested>*

Execution: *<no.>*: *<the description of the step>* or *<the description of the execution>*

Expectation: *<explanation of what is expected and to what extent this is realised>*

Results: *<the results after executing the scenario>*

The different scenarios for which the application is tested can be found appendix D. All manual tests had the expected results.

5.1.2 Final user tests

At the end of the last sprint, two user tests were performed. One is performed by us and the other is done by other students. These users are not part of the development team and are therefore not biased. During the test two persons play the escape room and two others host the game using the application.

For the test we have designed and constructed a simple escape room is designed by ourselves and makes use of three chests of which the insides are painted yellow-green. During the game these chests need to be opened in order to proceed in the game. The yellow-green insides are supposed to be detected by the system to assist the game host in hosting the game more efficiently.

The first time, the game is only played by two of the test subjects, but hosted by two persons out of the development team. The reason for this is that the host needs to have knowledge of the escape room in order to properly host the game. So after the first game is played, the players become the hosts for the next game. Both of these games are documented and the unbiased test subjects hosting the game are interviewed afterwards.

Test hosted by development team

During this test, two people of the development team hosted the game played by two people of the unbiased team. The players managed to complete the game and escape the room in time. However, only one of the three chests were detected and shown in the application. Another test was not detected at all and the third chest was detected because of the yellow ball pit balls inside of it. Also, the yellow memos we used in the escape room were detected by the application. This means that the colour yellow is detected properly. The reason that not all chests were detected by the application can be explained by the fact that the yellow parts of the chests were out of sight.

Test hosted by unbiased team

During this test, the people who played the escape room in the previous test hosted the game. The other two people of their team played the escape room. During the game, again only one of the three chests were detected due to the same reasons as the previous test. Also, a lot of times the yellow memos were detected by the application. Furthermore, the hosts accidentally pressed the stop button, so the stream was closed. Due to the fact this button is not really needed for the application, we deleted this button after the test.

However, in this test the focus was more on the way the hosts used the application. In order to find their experience during hosting, we conducted an interview afterwards which can be found in appendix E.

In summary, the overall experience of the unbiased hosts was positive, as the application was easy to use and understand. It was easy to keep track of the progress of the game, because of the logger, timers and warnings. However, they found creating a configuration file complex and prefer configuring the escape room inside the application, but would prefer to fill in the total duration of the escape room in minutes instead of seconds. The client preferred to fill in seconds instead of minutes so we left it to seconds. Also, the activity level did not have added value for the hosts and they would have liked to enlarge the streams. This feedback can be implemented in future versions of the application so we take this as recommendations.

5.1.3 Demonstrations with client of Popup-escape

During the project, we had weekly meetings with the client. During these meetings we showed our progress of the development of the product. Then, feedback was given which we tried to implement in the upcoming week(s).

Furthermore, we also performed two demonstrations during our project in which we let the client test our application after which we received feedback. The first demonstration took place in the fifth week and was primarily focussed on the performance of the detection of the yellow-green-green chests. This did not work as expected, because chests were detected multiple times. Thus, we improved the detection algorithms in the weeks after. Moreover, on request of the client, we implemented the functionality of API calls which can be sent to the application.

The second demonstration took place in the eighth week in which the client tested the GUI of the application. We created an escape room which was played by two people of the development team. The client played as game host during the game. The feedback we received was positive, however, the application crashed during the test. Afterwards, we did some investigations to find out what caused the failure to happen. Possible causes could be the memory usage of the PC or that the connection with the cameras was lost. So it was necessary to find out how to catch these errors and make the application run flawless.

Some adjustments needed to be made which we implemented after the demonstrations. For example, we adjusted the usage of the start/stop buttons for the stream, we made sure the application could be reset and stopped at all times, and we made the display of the application scrollable. Furthermore, we saved the images of the detected chests to be used for potential machine learning, we made some gradations in the warnings and we changed the timers of the chests to show the time of the individual chest instead of the accumulation.

5.2 Code Quality

The created software has been evaluated by the Software Improvement Group (SIG) during the project. SIG measures the quality of software. The developed code received 4/5 stars on their maintainability model. The highest score of 5 stars was not obtained because of the factors *Module Coupling* and *Unit Size*. This means that the degree of interdependence between software modules in the code is too high, and that several implemented methods are bigger in size than necessary. Since the received feedback from SIG, the code has improved. Dependencies of software modules are better divided, so that the degree of interdependence

between different modules is contained. Additionally, all methods have been reduced to under 20 lines.

Furthermore, to ensure comprehensible code, the Checkstyle tool was used during the project. This tool helps programmers to write consistent and documented code.

5.3 System Requirements

To be able to run the application, a computer running on Windows is required, as the application is only tested on the Windows operating system. It works at least on a computer with minimal memory of 6 GB RAM. The recommended memory is 8 GB RAM. This is only needed for playing two video streams in the application. For more than two videos, the application does not function properly on a computer with the minimal system requirements, so a faster computer is needed.

Chapter 6

Conclusion & Discussion

6.1 Conclusion

The goal of this project was to create an application that assists the game host of an escape room in the process of hosting a game and interacting with players by giving hints. The application consists of a GUI in which the user can configure the escape room and load live video feeds of the activity inside the escape room. When a chest is opened by the players, the game host is asked to confirm the opened chest after which the progress of the game is updated accordingly. The progress is shown in a log, a progress bar and the current status of the game. Together with timers of the complete game, timers of the individual chests, and the activity level inside the escape room, the game host can keep a clear overview on the progress of the game.

According to the implemented features, the basic requirements of the client have been met. However, the client wanted the application to be able to detect people on camera footage. This is not done by recognizing people's contours in the live video feeds, but by tracking the activity level of the movements inside the escape room. Moreover, the detection of chests is not always reliable due to for instance lighting and occlusions. As a consequence, the game host is asked to confirm whether a detected object is a chest.

Additional requirements of the client, like keeping track of the location of people, detecting other relevant objects or supporting multiple rooms have not yet been implemented. However, other requirements that the client requested during the development process have been implemented, such as the ability to make API calls to the application, or the ability to store images of detected chests. Other less essential functionalities like keeping track of the location of chests and allowing the game host to interact with the application have been implemented. The functionalities that have not been implemented could be implemented in future versions of the application.

Furthermore, the boundary of line coverage of the back-end has been met. At least 80% line coverage was demanded, while the total line coverage of the back-end was 97.5%. The front-end was not completely automatically tested. Instead, manual tests and user tests have been performed and documented in order to cover the performance of the functionalities of the front-end.

6.2 Discussion

We have created a valuable product for the client, but nothing is perfect. There are features which can be improved and there are features which were preferred to be in the application but were not. These possible improvements will be discussed in this section.

6.2.1 Activity detection

The detection of activity is done by comparing subsequent frames to each other. While it is true that most activity in a room is done by persons, there could be objects which move without people interacting with it. The assumption is also made that the camera is static and will not move. However, it is possible that due to vibrations, the camera can vibrate causing the application to detect activity which is not present. A bigger problem is when there are sudden light changes in the room. The light could be turned on or off, which leads to big changes between frames with every pixel being different. This will have a sudden very high activity.

These problems may not happen frequently and the first will not have a high impact, but the problem that far away persons have a lower activity than close by persons is one which will happen quite frequently. Objects close to the camera take up much of the frame and there will be much activity detected with slight movements. Using multiple cameras is a way how we deal with this issue, but still it is not waterproof. Two persons standing close to two cameras will still give a higher activity than when the two persons standing in between the two cameras.

6.2.2 Chest detection

Opened chests are detected by painting the inside of a chest a certain colour and searching for that colour on the camera footage. Tracking the colour when detected is done to not detect objects in that colour multiple times, but especially small objects are often multiple times detected as chests, because small objects are difficult to track. Also, opened chests could leave and re-enter the field of vision causing it to be detected multiple times. Taking into consideration however which colour the chest is painted, the game host could ensure that that specific colour is not present on another object in the escape room. The game host does however have no influence on the clothing the people playing the game wear. While large coloured objects are tracked well, so for example shirts in the same colour of the chest will not be detected as a chest often, when only a small part of the clothing is in the same colour, it can be detected often. We have chosen for a salient colour to minimize this problem.

6.2.3 Other improvements

There are features which we wanted to be present in the application, but did not implement. The detection of persons was thought to be an important feature of the application. If the right libraries were used and maybe if a different programming language was used, because many libraries for person detection is done in different programming languages, mostly in python, it would have been a possible feature. However, during the project it was noted that the use of detecting persons is not of high importance in supporting the game host in interacting with players, in comparison with its complexity. For that reason it was decided to leave person detection out of our application.

6.3 Ethical Implications

Testing of the product is done by using pre-recorded escape rooms, provided by Popup-escape, and by using self-created footage. This causes that escape rooms with players with a darker skin colour (all recorded people are Caucasian) could generate unwanted results. We, however, do not think that it will actually give rise to any problems as we only identify players as a group via activity.

Furthermore, since the images of all chests detected by the application can be stored, it is possible that the images of people that are mistakenly detected as a chest, are stored as well.

It was chosen to create a product where the need of attributes for the players is not present. This way the players do not need to have some device put on them neither do they have to change their clothes. The product is created with the intention of creating a more pleasant experience for players, moreover by supporting the game host in making decisions about giving hints.

6.4 Recommendations

The product is created for escape room games which take place in a single room and have a linear path of chests to be opened. However, several games have multiple rooms and do not need to be solved in a linear way. Therefore, the application could be extended to handle these sort of escape rooms.

The design of our application already took into account that multiple rooms could be made for one game. For instance, the configuration file already has a possibility to extend to multiple rooms, and the back-end is designed in such a way that the information extracted from camera footage is handled per camera. Therefore a room id could be given to the camera objects to link the information to the room. The most adjustments, in the design seen in figure 3.1, need to be done in the front-end part, especially in the controller, figure 3.3. The *RoomController* is designed in a way that only one room is processed, because it only has one progress object with one room object. It should be enough to extend the *RoomController* into handling multiple progress objects to deal with multiple rooms. To support non-linear paths of opening chests, the model of the front-end needs to be adjusted. Currently, the model assumes at most one previous chest and at most one next chest per chest. Extending this to the possibility that multiple chests could be before or after a chest should take care of this. Also, the configuration file and the view, mainly the progress bar, need to be extended to establish this.

Popup-escape indicated the desire to create an extra pane to be able to visit their web page where hints to players can be given. For this, an extra pane needs to be created and it needs to be made sure that all computations need to continue in the background.

To differentiate chests from each other and to assure chests will not be detected multiple times, different colours could be used for each chest. Additionally, QR-codes could be used to differentiate chests from each other. To extend this in the application, just as room ids, chest ids are needed to make a distinction between chests in the back-end. Linking the chests to their colour or QR-code, and sending this information to the back-end is needed, as well as extending the back-end to detect these different colours which need to be set in for example the configuration file. Frames of chests and whether the host confirmed it to be a chest are also saved. That way the detection of chests could be improved by using a classification

algorithm, for example machine learning, to correctly classify objects as chests.

Lastly the detection of activity could be improved. At this moment the activity is compared and decided by splitting all activities into several equal sized parts. To improve distinction between high and low activity, a clustering algorithm could be used to better determine boundaries between levels of activity. Also because the cameras are placed in upper corners of the room, people who are at the bottom of camera footage usually are closer to the camera than people who are at the top of camera footage. Therefore to deal with the previously mentioned problem that close by people seem to have higher activity than distant people, the frame could be divided into multiple parts. In that way, parts below on camera footage and above can be converged to each other. As a matter of fact, our application already has a possibility of dividing frames into multiple parts. Therefore only computations need to be decided upon and created to make this happen.

Appendix A

Original Project Description

The project:

The most popular escape rooms are those who are technically advanced. A challenge for popup-escape is that all their escape rooms are mobile. Furthermore, their escape rooms change continuously.

One of the new ideas of popup-escape is to create an escape-room where the players have interaction with the main-character and or sub-characters in the game. Normally, this would be done by a game-host watching the camera-feeds.

The goal of popup-escape is to automate these interactions. The first step probably is to distinguish players from one-another, secondly, to track them during the game. (Where are they in the room, what are they doing) and thirdly to interact with them based on the status of the game.

Possible research questions:

- What techniques can be used to differ players from one-another?
- What is the impact of using more/less cameras to track people
- How can the tracking + likelihood of correctness be shown to the game-operator?
- What techniques can be used to detect a players actions?
- What is the impact of changing the room the game is played in?

Appendix B

Research Report

The end goal of the project is to support the role of the human game host that watches a group of people in the escape room and gives hints when necessary. The application to be created is responsible for the extraction of information from video footage of escape rooms. Currently, there are multiple cameras available to record what happens in an escape room.

Similar to the game host, the computer wants to keep track of the progress of the game. For this it is relevant to track objects or people inside the escape room. These things allow the computer to detect when hints are needed. The application aims to make interacting with the players easier for the game host.

In order to find out how this can be achieved, some research is needed. The following main research question is constructed:

How can the game host of an escape room be automatically supported in interacting with players in the game by using escape room specific extractable data?

1. How does an escape room work?
2. What techniques can be used to recognize objects on video?
3. How can useful data be extracted from sound in the escape room?
4. How can behaviour and emotions of players be tracked and for what purpose can this be used?
5. How can the progress in an escape room be tracked?

These questions are answered below with help of relevant literature.

B.1 Escape rooms

Escape rooms are live-action team-based games where players discover clues, solve puzzles, and accomplish tasks in one or more rooms in order to accomplish a specific goal in a limited amount of time [1]. In other words, players must complete challenges in order to win the game. Escape rooms differ from each other in the challenges they bring to players. Every escape room however tries to encourage players to think differently and from new perspectives. The challenges players face can be of various categories. Wiemker et al. [9] make a differentiation between mental and physical puzzles. Mental puzzles are solved by using thinking skills and logic, whereas physical puzzles are based on the manipulation of artefacts in the room. Both puzzles can be combined. For the last part of the escape room, often there is a meta puzzle, which is a puzzle that can only be solved after completing all previous puzzles.

Usually, to be able to complete an escape room within the limited time, it is necessary for players to work together. Essentially, escape rooms are team activities. Nicholson [1] reports that the average group exists out of about five people. A 'good' escape room should aim to involve every member of the team in the process.

To get to the end of an escape room, a team of players must follow a certain path of puzzles. There are three main approaches for path creation: *linear path*, *open path*, and *multi-linear path* [9]. In a linear path, puzzles must be solved in a specific order: the solution of one puzzle gives access to the next puzzle, and so on. This approach allows for a fairly simple escape room. Differently, an open path contains puzzles that can be solved in any order. Usually there is a final puzzle that cannot be solved until all the other puzzles have been completed. While an open path allows for all team members to be able to always do something to help the progress, in a linear path however one puzzle may be solved by a single person while the rest cannot do anything. Lastly, a multi-linear path is a combination of the linear and open path approach. It involves a series of linear paths that can be done in parallel.

While the players are locked inside an escape room, the game host monitors the game progress from a different area via cameras. When players get stuck in the game, the game host should provide hints to the players, so that the players can move on and will not think of the game as a bad experience. The game host is able to give hints by voice (for example using an intercom or by using a telephone), in person (the game host can be summoned), or via pen and paper (a note can be slipped into the room) [9]. Another way of giving hints is showing hints on some kind of electric device that is inside the room (a screen for example). These kind of hints can be automated.

Players are fully engaged in a task when they are both challenged and entertained. If a puzzle is too hard, players will get frustrated and might give up. On the other hand, if the puzzle is too easy, players will get bored. It is up to the game host to make sure the game process goes smoothly and that players are having a good time. This means that the game host should keep track of how well a group is performing, and based on this he may or may not give hints. This is something a game host could be supported in.

We now have sufficient understanding of how escape rooms are constructed and what they generically consist of. This enables us to create an application suitable for escape rooms.

B.2 Recognizing objects in video

Recognizing objects such as people and important items in the escape room can help the game host in his task of giving hints to players. In this section, several techniques to recognize objects are explained. A distinction is made between different phases of object recognition. These phases are shown in figure B.1 found in an article by Bedagkar-Gala and Shah [2]. This article is about person recognition. However, any object can be recognized in the same way. To be able to recognize an object in a video, this object must first be detected in the video. Since we are working with videos, the tracking of object movements is also important. Furthermore, a descriptor that describes an object by extracting relevant information from the video is needed. Lastly, the previously constructed descriptor can be used to (re-)identify an object.

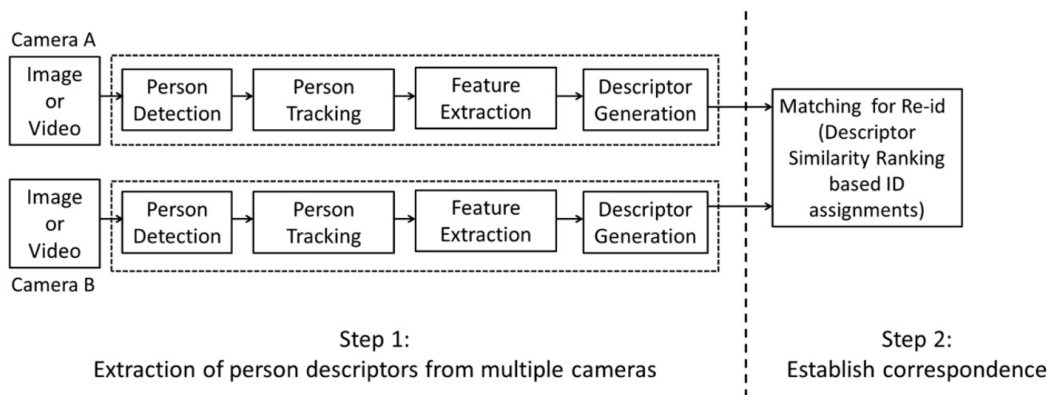


Figure B.1: Identifying a person on multiple cameras [2]

B.2.1 Object detection

There are many different techniques that can be used for the purpose of detecting objects. In this section an overview of these techniques is given.

In the paper by Yilmaz et al. the common object detection methods are described [3]. These techniques are divided in four different categories: point detectors, background subtraction, segmentation, and supervised classifiers. All categories are elaborated on below.

Point detectors

With point detectors interest points can be found in an image with expressive textures. An interest point is invariant to illumination changes and to the camera viewpoint. Some interest point detectors are Moravec's interest operator, Harris interest point detector, KLT detector and SIFT detector [3]. The methods by Moravec, Harris and KLT are invariant to both rotation and translation. However, they are not invariant to transformations. SIFT is. SIFT outperforms most point detectors and is more resilient to image deformations.

Background subtraction

Background subtraction models the background of a scene. At each incoming frame deviations from this background model are detected. Considerable changes in an image frame

compared to previous frames indicate a moving object. The pixels that indicate this change are further processed and an algorithm is used to classify the pixel region as an object [3]. An example of this method is shown in figure B.2. The pixels of the beforehand instantiated background model are subtracted from the pixels of a frame. If the difference in pixel values exceeds a certain threshold, an object is detected.

Background subtraction is used in most state-of-the-art tracking methods for fixed cameras. The reason for this is that it is insensitive to illumination changes, noise and periodic motion of the background region. This means that it can accurately detect objects under different circumstances. Also, the methods are computationally efficient and can therefore detect objects quickly, which is beneficial for real-time tracking. An implementation of background subtraction can be found on the website of OpenCV [4].

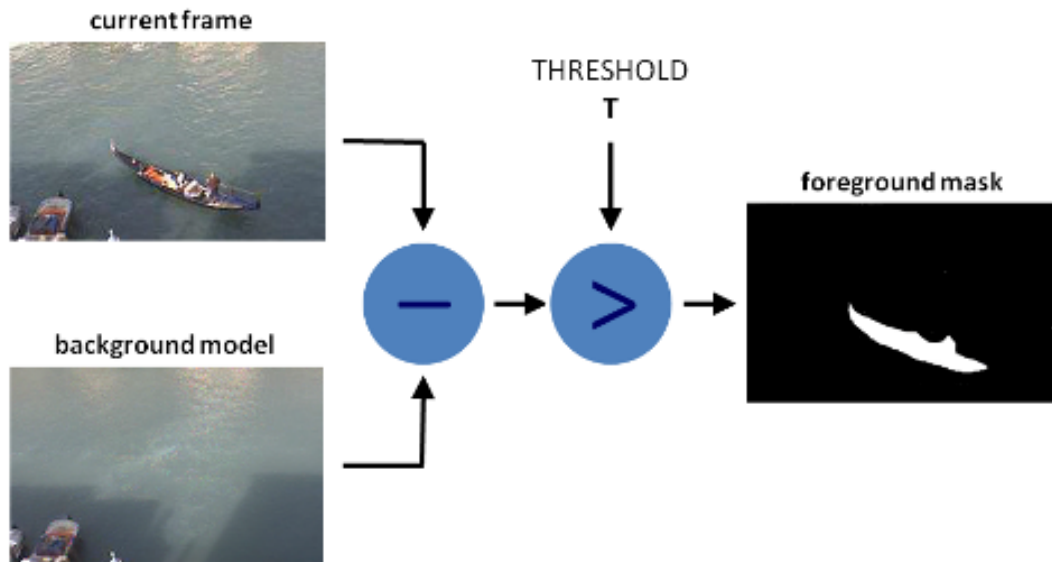


Figure B.2: Background subtraction example [4]

Segmentation

Image segmentation partitions an image into perceptually similar regions. A segmentation algorithm addresses two steps: the criteria for a good partition, and the method for achieving efficient partitioning with these criteria. Two techniques for segmentation are *mean-shift clustering* and *image segmentation using graph-cuts* [3].

The mean-shift clustering algorithm results in a segmentation of the image as shown in figure B.3(b). The results of applying segmentation using graph-cuts are shown in figure B.3(c). Mean-shift clustering requires fine tuning of various parameters in order for the segmentation to work well. For segmentation using graph-cuts this is not needed. However, segmentation using graph-cuts can be expensive in terms of processing and memory requirements. Segmentation is quite a complex technique to detect objects compared to other described techniques.

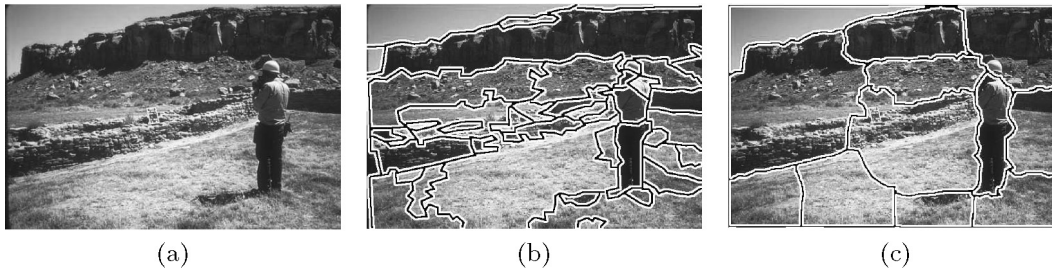


Figure B.3: Segmentation example [3]

Supervised Learning

In supervised learning different views of an object are learned automatically from a set of examples. Supervised learning methods generate a function that maps input to desired output. Some techniques discussed belonging to this category are *adaptive boosting* and *supportive vector machines* (SVM) [3].

Another technique in supervised learning is *You Only Look Once* (YOLO). YOLO is a technique that uses a single trained neural network that predicts bounding boxes around multiple objects simultaneously [10]. YOLO has a couple of advantages in comparison to other object detection methods. First of all, YOLO is fast, which makes it very well suited in a real-time application. Secondly, YOLO uses highly generalizable representations of objects, which makes it rather robust when applied in new scenarios.

B.2.2 Object tracking

The task of object tracking in general can be described as finding out where an object moves to and what path it follows throughout subsequent frames of a video. Every tracking algorithm needs an object detection mechanism either in every frame or when the object appears in the video for the first time. Some common tracking methods are described in Yilmaz et al. [3], where they are divided into three different categories: *point tracking*, *kernel tracking* and *silhouette tracking*.

Point Tracking

Detected objects are represented by points and are associated with the location of these points in the previous state of the object. This shows motion of the object. In order for this tracking method to work in every frame, the objects need to be detected [3]. A visualization of point tracking is shown in figure B.4(a). Point trackers are best suitable for tracking very small objects. For larger objects, this tracking method is not efficient.

Kernel Tracking

This type of tracking requires a shape and appearance information of the object. This combined representation of information is known as a kernel [7]. The object can be for instance a rectangular template, or an elliptical shape with an associated histogram. Objects are tracked by computing the motion of the kernel in consecutive frames. This is shown in figure B.4(b). The motion is usually in the form of a translation, rotation or affine transformation. The only

techniques that are suitable to track multiple objects and handle full occlusion are *layering* [11] and *BraMBLe* [12].

With layering, it is assumed that the background is 2D and in motion. In this project the background is however assumed to be static. Therefore for this project layering is not a suitable technique. BraMBLe, which is a Bayesian multiple blob tracker, seems more promising as it assumes that the background is fixed. This means that positions of 3D objects can be computed. Particle filters are used to determine the 3D position, shape and velocity of all objects in the scene. Also, occlusion between objects is tolerated and it can be used in real-time. However, the maximum number of objects in the scene needs to be predefined. This is no problem for this project as the amount of objects appearing in the scene is known beforehand. Another limitation is that training is required to model the foreground regions and that BraMBLe is developed for single camera usage.

Silhouette Tracking

Silhouette tracking estimates an object region in each frame by using previous frames. Information about the region may be considered as the appearance density and shape model. By using object models, silhouettes are tracked by either shape matching as shown in figure B.4(c) or contour evolution as shown in figure B.4(d). The only techniques that are suitable to track multiple objects and handle full occlusion belong to the subcategory contour evolution. Within this category only the techniques *particle filtering* [5] and *gradient descent* [6] are relevant. It is not known whether these techniques can be used in real-time. The first technique tracks a maximum of only two objects and is therefore not relevant for this project. The second technique works well with background subtraction (for object detection). Furthermore, this technique focuses on detailed analysis of the shape deformation during intense action (such as sports). Therefore, this technique is not completely suitable for this project as such motions are not expected and may make the application unnecessarily computationally inefficient.

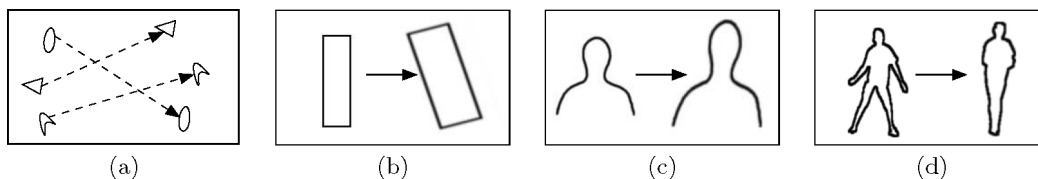


Figure B.4: Tracking approaches: point tracking (a), kernel tracking (b), silhouette tracking by shape matching (c), silhouette tracking by contour evolution (d) [3]

Cameras

When tracking objects one should wonder if it would be a good idea to use multiple cameras as opposed to only one. Using only one camera means a limited field of vision, limited depth determination and a higher chance of occluded objects. However, it may be easier to implement detection and tracking algorithms using one camera than multiple cameras.

Using multiple cameras can increase the field of vision where objects can be found. Multiple cameras can also help with calculating depth of objects. This way one could track the

location of an object in a room more easily. The difficulty here is that a good feature extraction for objects is needed, so that the same object can be recognized on multiple cameras.

Kumar et al. [7] propose a way of using multiple cameras in real-time. When using multiple cameras instead of one, it increases the chance of detecting objects. This is because of the afore mentioned reason that more cameras increase the total field of vision. It also means that objects that get occluded on one camera might appear on a different camera, making it possible to find them. Another technique used to determine the position of an object in a 3D space using two cameras is triangulation. Triangulation determines the position of an object in a 3D space using two known locations (cameras) and angles where the object is detected on these two cameras [13].

For this project two cameras are available for use, supplied by the client. The cameras can be placed opposite of each other in the room. The benefit of these cameras is that the video feeds are saved in a Real-Time Streaming Protocol (RTSP), which can easily be processed with Java as the programming language. Therefore, we will make use of two cameras and use Java as the programming language for this project.

B.2.3 Feature extraction and person descriptor generation

When looking at B.1 it is visible that feature extraction and person descriptor generation are combined in this section. This is due to the fact that all extracted features will be put into an object, and this object is the descriptor of a person or an other object. Therefore this section is focused on techniques used for feature extraction.

The first feature extraction method that is covered is called *Scale Invariant Feature Transform* (SIFT). SIFT is an algorithm that identifies and characterizes local features in an image. There are many versions of the SIFT algorithm that all work in a similar fashion, the so called SIFT-like algorithms. Tao et al. [14] evaluated the performance of some SIFT-like algorithms in their paper. They found the best results for *Gradient Location and Orientation Histogram* (GLOH) and *SIFT*. SIFT is robust to scale, translation and rotation, as shown in the article by Lowe [15]. This means that extracted SIFT-points are rather good to reuse, and therefore also to identify people. SIFT is however quite slow when compared to other SIFT-like algorithms. Isik & Ozkan [16] compared SIFT-like algorithms and found that for example a combination of FAST and BRIEF is relatively fast.

The second feature extraction method considered is a saliency map. As the name hints, this technique searches for salient regions in an image and puts them in a map. Salient regions are areas in an image that stand out. The way in which those areas stand out are often based on colour, intensity or orientation. Rosin [17] discusses a way of implementing saliency maps based on edge density. He then compares his saliency map to saliency maps generated by different algorithms. In another paper by Achanta et al. [18] a technique of acquiring the saliency map by using colour and luminance is used. They state that, because they used these low level features, their algorithm is noise tolerant and fast enough to be used in real-time applications.

The third feature that can be extracted is colour. There are multiple techniques which describe colour in an image. A couple of those techniques are described in a paper by Manjunath et al. [8]. One promising descriptor is the *dominant colour* descriptor, which returns the salient colour distribution in the image. Another descriptor defined is the *scalable colour descriptor* (SCD). Simply put, the SCD defines a HSV colour histogram of the image and

compresses it using the Haar transform. The third technique is called the *colour structure histogram* and aims at identifying localized colour distributions by using an 8x8-structuring element. The resulting histogram is constructed in the HMMD colour space.

Another feature that might be helpful to identify objects with is texture. While colour is great to identify objects with whenever there is enough illumination, it does not really work with grayscale images (or when there is little illumination). Texture is not dependent on colour, which makes it a good option for extracting features in grayscale images. Lin et al. [19] describe a technique that describes textures in any kind of grayscale or colour image. This technique is called *adaptive local binary patterns* (ALBP) and makes use of the classic local binary patterns technique (LBP). The only new feature of ALBP as opposed to LBP is that it considers the smoothness of the region to be important.

B.2.4 Object (re-)identification

The goal of this section is to (re-)identify objects on multiple frames of video. In the previous steps techniques to find objects in an image, track them, and extract features from them to describe them, are discussed. Object (re-)identification is especially useful for tracking people, as they are moving objects. Therefore, in this section the focus is on people. All that is left to do after creating the descriptors, is to label each person with their right name. There was however not much literature on this problem, apart from re-identifying people by matching descriptors [2].

Since the descriptor of a person will be stored in an object, the only thing that needs to be done is to find the best match between the newly found descriptor and a known descriptor. For this, a simple algorithm that returns the nearest neighbour is sufficient. The newly found and labelled descriptor can be added to the dataset of descriptors to improve the nearest neighbour algorithm over time. This could however increase the computation time and therefore might worsen the real-time application. Since this issue is rather specific to the problem at hand, the best option should be decided upon when the application is build.

B.3 Sound

Besides video footage, it is also possible to use sound to gather more data of what is happening inside the escape room. For this, some research is needed on voice recognition systems, as they analyse speech and convert them to text.

Currently, most voice recognition systems are based on the conventional GMM-HMM framework [20, 21]. These systems make use of hidden Markov models (HMMs) and Gaussian mixture models (GMMs). HMMs are statistical models that output a sequence of symbols. A HMM has a number of hidden states, a number of distinct observation symbols per state (from an alphabet), a state transition probability distribution, an observation symbol probability distribution per state and an initial state distribution [22]. With all this, given a sequence of symbols the model can be used to calculate the probability of particular series of that symbol sequence. HMMs are used to deal with the temporal variability of speech. A Gaussian mixture model is a clustering technique. It determines how well each state of each HMM fits a frame of coefficients from the input audio.

A different more newly developed method for speech recognition systems are deep learning techniques. Deep learning uses artificial neural networks. It lets computational models composed of multiple processing layers learn representations of data, with multiple levels

of abstraction [23]. Machine learning is then able to train the neural network such that it exhibits desired behaviour (e.g. audio is correctly classified) [24].

There are several open source libraries for speech recognition available for use. The two most notable that also feature a complete English acoustic model are *Sphinx* and *Kaldi*. Sphinx is a flexible framework written in Java that uses HMMs [25]. It allows for accurate, large-vocabulary speaker-independent, continuous speech recognition. Moreover, it works offline. Another toolkit for speech recognition is Kaldi. Kaldi uses HMMs as well, and is written in C++. Kaldi was mainly intended for acoustic modeling research [26].

B.4 Recognizing Behaviour and Emotions

When detecting people in video, behaviour and emotions can be investigated. In this section, behaviour is distinguished from emotions. The behaviour of a person describes their movements and actions. Emotions show how people experience the game (e.g. the level of frustration can be helpful to keep track of).

B.4.1 Behaviour

The behaviour of players in the game can be detected by looking at actions of players. An article by Moeslund & Granum [27] shows several techniques for detecting a person's motions. The article divides these techniques into several phases shown in figure B.5. At first, there is an initialisation step, which can be seen as a preprocessing step where persons need to be detected on video. This is needed to create silhouettes of people to track (important parts of) the body. Ways to detect and track people are described in section B.2.2. After the body is detected and tracked, the pose of the person is estimated before the motion is recognized.

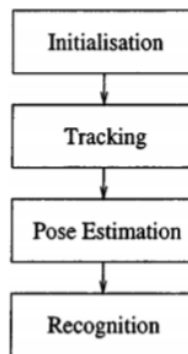


Figure B.5: Extraction of person motion [27]

One of the biggest differences between techniques shown in the article by Moeslund & Granum is that some techniques use a predefined model to compare to the video images, as described by Carranza et al. [28]. In their article an approach is described which first initializes the body parts of the person in the room using a predefined model. These body parts are then tracked to create a model of the person's actions. These techniques are helpful to construct a model of the pose and actions of the person. For this project, only certain actions need to be detected, rather than creating a model of these actions. Also, the actions

are mostly recognized with a single person on camera, while an escape room contains multiple persons.

Other techniques do not use a model up front, but rather use examples of the to be detected actions to make comparisons with. Davis [29] uses motion to detect human actions. A motion history image is used to create motion histograms, which are compared to several training examples of known actions to determine the right action in place. Actions which can be determined by this include walking, sitting down and waving hands. It is however not always verified that the action is actually caused by a human. Detecting that the object is actually a human silhouette could help in this case. The detection of silhouettes is explained in section B.2.2.

Poppe [30] states that using multiple cameras can be helpful to detect actions. Occluded parts not detectable by one camera can be spotted in another, increasing the likelihood of detecting an action.

B.4.2 Emotions

Recognizing emotions can be helpful to determine whether people enjoy the game, or if they are stuck. Using video and sound can help to detect certain emotions in humans. An article by Busso et al. [31] shows that emotion can be recognized with facial expressions and speech. Schindler et al. [32] states that emotions can be recognized with the body pose of humans. Recognition with facial expressions and sound is a difficult task, because there are many assumptions to be made [33]. Facial expressions can only be recognized when there is a (nearly) frontal view of the face. Furthermore, recognition in real-time is an almost impossible task and especially difficult with more than one person in the room.

An article by Koolagudi et al. [34] compares several emotion recognition from speech studies. These studies differ in database use, feature extraction, and use of classification models. There are several emotional speech corpora. Some use acted or simulated speech, while others use natural speech, such as call centre recordings. The downsides of acted speech are that it is not a real world representation, it is individualistic to the actor and the emotion is not natural. Therefore natural speech would be favourable. The problems with natural speech corpora are that there is background noise, it is hard to label and model speeches, and there are copyright and privacy issues. Creating a new dataset for this purpose would take care of the latter, but make the other problems even harder.

Human posture can also be a way to recognize emotions, though it is more subtle. Postures of humans can be recognized by detecting silhouettes and comparing it to key postures defining an emotion [35]. The technique described in the article uses K-Means clustering to classify human poses. Difficulties are creating these key postures and assigning a label of emotion to them. For this project the most important emotion to be detected is frustration.

Knowing the emotional state or the behaviour of a person, the question that rests is for what purpose it can be used. Cohen [36] shows that a feedback system can be useful in stressful situations, for example in the Marine, to reduce the number of errors. Gonzalez [37] states that mostly feedback in the form of showing expert decisions is helpful, giving outcome feedback or showing own past performances is not helpful for improving the performance. However, while it is shown that giving feedback can be helpful, it is not as necessary in an escape room to improve the performance. In the experiments done, the performance is important and it is more preferable to have a high performance. In an escape room it is not

important to have a high performance. What is important is that the players have fun.

B.5 Progress

In this section we would like to answer how the progress in an escape room can be tracked. There was no relevant literature concerning this topic, probably because it is rather specific to escape rooms. Hence instead of analysing literature, the owner of Popup-escape, Jan-Willem Manenschijn, was consulted. The findings are discussed in this section.

Progress in an escape room can be seen as the state of progression in the game. An indication for this is how many of the total amount of puzzles have been solved. Usually when people are about to solve a puzzle, they either walk towards that puzzle or they are already standing near it. This led to the idea that it is necessary to keep track of where puzzles are located in the room and map this. If there is such a map, it could be combined with person tracking by for example showing positions of players on the map. This could give a pretty good idea of the progress of the game, as it is able to show whether players are standing near the next puzzle or not.

Another way of keeping track of the progress of the game is to keep track of player knowledge. With player knowledge, knowledge a certain player has about locations of different key pieces needed to solve puzzles, is meant. This should give an idea of which puzzles can currently be solved by the players. If for example a hammer is needed to complete a puzzle, and no one has seen the hammer, this puzzle can not be solved yet. To do this well, it is needed to keep track of what people have seen. This may be possible with video by mapping the locations of puzzle pieces and adding it to persons' knowledge if they are faced towards that puzzle piece.

To make the tracking of progress in the game easier, constraints on the game elements can be made. An example of a constraint to easily track puzzles and especially chests is to put invisible trackers on them or give them a specific colour. This does not affect the game experience, as it does not bother the players.

It is also possible to track player knowledge by using sound recordings. This can be done by actively listening for key words describing the puzzle piece (like 'hammer'). When such a word has been said, this puzzle piece is added to the knowledge of the group of players.

Furthermore, an important aspect of progress in an escape room is time. The time that has passed since the beginning of the game can be a good indicator of the phase of the game in which the players should be. When players are solving puzzle *A* and 10 minutes have passed, while puzzle *A* can normally be solved within 5 minutes, the group takes too long to solve puzzle *A*. This should then alert the game host to give a hint. If puzzle *A* is normally solved in 20 minutes, and only 10 minutes have passed, no hints need to be given. The time that people are standing still instead of moving around can also be an indicator of the progress of puzzles. When people are not moving for a long time, it probably means that they are stuck at a certain point, especially when they are not looking focused and are not near a puzzle that they can solve. However, standing still does not necessarily mean that the players are stuck in the game, as people might also stand still while solving puzzles. Therefore, all these analyses of deriving the progress of the game are assumptions and must be confirmed by the game host.

B.6 Conclusion

What it comes down to is that this project is aimed to extract valuable information from escape rooms with multiple static cameras. The most important thing for the game host is that the progress of the escape room is tracked, as this indicates whether players are able to finish the game in time.

To track the progress of the game it is needed to know if and when certain puzzles are finished. This can be done by tracking objects that are important for the game progress (such as locked chests). Research showed that the best way to do this is by using background subtraction, as this is an easy solution that can be implemented within the limited time span. Background subtraction can be created using the image processing library of OpenCV. Sound recordings might be used as well in order to track progress by detecting certain words (words that are key for progressing in the game) that are mentioned by players.

Furthermore, for the game host it is essential to detect when players are stuck in the game. A good indication is when players are standing still for quite some time. This is detectable by tracking players and checking how long they have not been physically contributing. Again, same as with objects, persons can be detected by using background subtraction.

In order to track and map the location of objects and people, multiple cameras are used. With each camera the position of the object is determined and mapped. When the object is occluded as viewed from one camera, this camera can still guess its location based on recent information. In combination with the other camera, where the object might not be occluded, mapping the object's location can still be realized. The tracking techniques discussed in section B.2.2 track detected objects by showing their path on the camera feeds and do not visualize their position on a map. This means that the nature of these algorithms is different than what is needed for this project and these algorithms are therefore not used to map the objects. Instead, triangulation is used by using two cameras and mapping the position.

Labelling players with their names might be a nice feature to include in the application, however this is not deemed to be of utmost necessity. It is however useful to differentiate players from one another for mapping purposes. Feature extraction can help us in this task. We plan on extracting colour features per person, because colour is often good enough to differentiate people. When it is dark, the cameras used go into a night vision mode, which gives us back a grayscale image as opposed to a coloured one. This means it is not possible to describe people using colour whenever it is dark. To still be able to differentiate people, ALBP will be used to extract textures/edges. We chose not to include SIFT-like descriptors or a saliency map, because they were either too slow for using in real-time (SIFT), not descriptive enough in real-time (FAST) or worse at its task than colour descriptors (saliency map).

It is not crucial for the game host to know emotions of players. Detecting emotions is quite a difficult task and almost impossible without clear images of people's faces. For these reasons it is decided not to include the task of detecting emotions of players in this project. A player's posture and actions can be tracked by comparing its posture to known postures, but it is not very important for the game host to be aware of players' poses. Emotions of players could be detected with their posture, and whether a player is walking or running can be detected by their actions. However, as mentioned before, this is not all too relevant for this application.

The programming language that will be used is Java because this language is compatible with relevant libraries like OpenCV and Sphinx and it is the language we are most proficient in. Furthermore, Java is compatible with processing the streams returned by the cameras

that will be used.

In summary, the focus of the software project is on tracking people and important objects in escape rooms, with the main goal to be able to keep track of the game progress. For tracking, background subtraction techniques will be used. Furthermore we could keep track of the location of the objects in the room by using triangulation. To differentiate people from one another feature extraction by colour or texture is a good option, depending on whether the room is being illuminated or not. OpenCV, a library for computer vision related tasks, will be used to help with image processing tasks.

Appendix C

Configuration Manual

```
[
{
  //Give room an id, this is used to retrieve the information, like in a DB.
  "roomId": 0,
  // Cameras holds a list of links to some kind of video, this video can be a stream/video/image
  "cameras": [
    {
      "link": "testlink"
    },
    {
      "link": "testlink"
    }
  ],
  //Specify the amount of people that start in this room at people
  "people": 5,
  //Chests are seen as some kind of checkpoint in the escape room, each checkpoint can have subcheckpoints.
  /*
  A chest is formatted in the following way:
  NB: information on what to put at each field is located between < and >
  {
    "sections": <THE AMOUNT OF PUZZLES THAT NEED TO BE SOLVED BEFORE PLAYERS CAN OPEN A CHEST>,
    "targetDuration": <DURATION IN SECONDS
    AFTER THIS TIME HAS PASSED THE PLAYERS TAKE "TOO LONG" TO FINISH THIS SECTION>
    "warningTime": <TIME WHEN A WARNING NEEDS TO BE SHOWN TO THE GAME HOST>
  }
  */
  "chests": [
    {
      "sections": 2,
      "targetDuration": 120,
      "warningTime": 60
    },
    {
      "sections": 3,
      "targetDuration": 220,
      "warningTime": 60
    },
    {
      "sections": 1,
      "targetDuration": 30,
      "warningTime": 10
    }
  ],
  //This target duration is the total amount of time in seconds players get to escape the entire escape room
  "targetDuration": 500
  //The port number the server needs to listen to for API calls. When not set, default is 8080.
  "port": 7070
}
]
```

Appendix D

Test Scenarios

D.1 Menu tests

This section describes scenarios in which the menu options are tested. These are adding video sources, configuring the escape room and the settings. In these scenarios, it is only tested whether the right components pop-up and are shown in the GUI. The real interaction on the progress of the game is tested in section D.3

Scenario 1: Playing a video

Date tested: 21/06/2018

Description: This scenario tests whether a video file can be loaded and played in the application.

Execution:

1. Start the application
2. Click on "Add Video Source"
3. Click on the "Add Video file button"
4. Choose a video of choice in the file chooser and click on "open"
5. Click on the play button

Expectation: A video is played in the video pane of the application.

Results: Works as expected.

Scenario 2: Loading and playing a video stream

Date tested: 22/06/2018

Description: This scenario tests whether a stream can be entered and shown in the application.

Execution:

1. Start the application
2. Click on "Add Video Source"
3. Click on "Add Stream"

4. Fill in the stream URL and click on "submit"
5. Click on the play button

Expectation: The stream is shown in the video pane of the application.

Results: Works as expected. However, at some PCs, after a short period of time, the connection to the stream gets lost and a black tile is shown.

Scenario 3: Load a configuration file

Date tested: 25/06/2018

Description: This scenarios tests whether a configuration file can be loaded from the file chooser.

Execution:

1. Start the application
2. Click on "Configure the Escape Room"
3. Click on "Load Configuration File"
4. Choose the desired configuration file and click on "open"

Expectation: In the middle of the screen, a confirmation message is shown stating the number of cameras that are ready to be activated.

Results: Works as expected. The video/stream can also be started with the same results as in **scenario 5**.

Scenario 4: Load the standard configuration file

Date tested: 25/06/2018

Description: This scenario tests whether a pre-defined configuration file can be loaded directly, without choosing.

Execution:

1. Start the application
2. Click on "Configure the Escape Room"
3. Click on "Use Standard Configuration"

Expectation: In the middle of the screen, a confirmation message is shown stating the number of cameras that are ready to be activated, as in **scenario 3**.

Results: Works as expected. The video/stream can also be started with the same results as in **scenario 5**.

Scenario 5: Correct manual configuration

Date tested: 25/06/2018

Description: This scenario tests whether an escape room can be configured manually inside the application.

Execution:

1. Start the application

2. Click on "Configure the Escape Room"
3. Click on "Manual Configuration"
4. Fill in the amount of players, chests and the total duration of the escape room
5. Click on "Proceed"
6. Fill in all the settings for each chest
7. Click on "Submit"
8. Add and play video source as in **scenario 1** or **scenario 2**

Expectation: One or multiple videos are played with the logger, progress bar and status pane around it.

Results: The progress items pop-up and show the correct numbers as entered in the manual configuration window. Now the application is ready for human input, which is tested in section D.3.

Scenario 6: Change contents of fields in manual configuration

Date tested: 25/06/2018

Description: This scenario tests whether the contents of the chest field can be adjusted and whether the number of chests occurring for which the settings need to be filled in are changed accordingly.

Execution:

1. Start the application
2. Click on "Configure the Escape Room"
3. Click on "Manual Configuration"
4. Fill in the amount of players, chests and do not fill in the total duration of the escape room
5. Click on "Proceed"
6. Change the number of chests
7. Click on "Adjust Settings"

Expectation: The number of chests for which the settings need to be filled in changes.

Results: This number indeed changes, so this scenario works as expected.

Scenario 7: Wrong manual configuration

Date tested: 25/06/2018

Description: This scenario tests whether an error occurs when not all fields in the manual configuration window are filled in.

Execution:

1. Start the application

2. Click on "Configure the Escape Room"
3. Click on "Manual Configuration"
4. Fill in the amount of players, chests and do not fill in the total duration of the escape room
5. Click on "Proceed"

Expectation: An error pops up.

Results: An error indeed pops up, saying "Please fill in a number in each field". This is also shown when one forgets to fill in a number in any of the other fields.

Scenario 8: Reset the application

Date tested: 25/06/2018

Description: This scenario tests whether the application resets.

Execution:

1. Execute for instance **scenario 5**
2. When the video is playing, click on "Settings"
3. Click on "Reset Application"

Expectation: The application resets.

Results: The application indeed resets and the GUI looks as if the application has just started.

Scenario 9: Close the application

Date tested: 25/06/2018

Description: This scenario tests whether the application can be closed.

Execution:

1. Start the application
2. When the video is playing, click on "Settings"
3. Click on "Close Application"

Expectation: The application closes.

Results: The application indeed closes.

D.2 Detection tests with streams

This section describes scenarios in which purely the detection of chests is tested. This is done with a real-time stream. No configuration files are loaded as only the detection needs to be tested. Therefore no progress items are shown in the GUI.

Scenario 10: Chest detection

Date tested: 22/06/2018

Description: This scenario tests whether the application detects an opened chest.

Execution:

1. Execute **scenario 2**
2. Open a chest in the camera's field of vision

Expectation: The chest gets detected. An image for approval will be shown at the bottom left corner of the application.

Results: The chest indeed gets detected. This is due to the fact that the yellow paint inside the chest is detected by the camera. This tint of yellow falls between the HSV values which are used to detect the colour yellow. Afterwards, an image of this chest is shown of which the user's approval is asked.

Scenario 11: Wrong object detection

Date tested: 22/06/2018

Description: A red and yellow ball gets thrown in the detection scope of the cameras.

Execution:

1. Execute **scenario 2**
2. Throw a red ball in the detection field of a camera
3. Throw a yellow ball in the detection field of a camera

Expectation: The yellow ball gets detected, while the red ball does not get detected. For the yellow ball an image for approval will be shown at the bottom left corner of the application.

Results: The reason that the yellow ball gets detected is because its yellow colour falls in the range of HSV values which are used for the detection of the yellow coloured chests. Ideally, we only want the yellow chests to be detected by our application, but when another yellow object with the correct HSV values appears in the detection field of the cameras, this object gets detected. Afterwards, the user can choose to either accept or reject the detected object. The red ball does not get detected as expected.

D.3 Progress tests

This section describes scenarios in which the interaction with the game host and the progress of the game is tested. It tests whether the components in the GUI are updated accordingly and shown properly. In these scenarios a predefined configuration file is used together with one pre-recorded video of the development team opening chests. At the start of each scenario, it is assumed that the video is already playing.

Scenario 12: Approve a chest

Date tested: 25/06/2018

Description: This scenario tests whether the progress changes accordingly whenever a chest is approved.

Execution:

1. Whenever a chest gets detected and an image of it is shown in the application, click on the "tick" symbol

Expectation: The progress of the game is updated accordingly.

Results: The time stamp of when the chest is detected including the text "Found chest 1/i" where i are the amount of chests, is printed in the logger. In the status pane the number of chests opened is added with 1. Also the timer of the concerned chest is stopped and the correct time stamp is shown. The timer of the next chest starts ticking. Furthermore the progress bar is updated in which the concerned chest icon and its sections are coloured green. This all works as expected.

Scenario 13: Disapprove a chest

Date tested: 25/06/2018

Description: This scenario tests whether the progress does not change whenever an object is disapproved.

Execution:

1. Whenever an object gets detected and an image of it is shown in the application, click on the "cancel" symbol

Expectation: The progress of the game is not changed and the image disappears.

Results: Works as expected.

Scenario 14: Wait for warning sign to appear

Date tested: 25/06/2018

Description: This scenario tests whether a warning sign appears whenever the time until warning has exceeded.

Execution:

1. Wait until the time until warning for the first chest has exceeded

Expectation: A warning sign appears.

Results: A warning sign indeed appears, which can be clicked to close it.

Scenario 15: Wait for chest timer to colour red

Date tested: 25/06/2018

Description: This scenario tests whether the timer of a chest turns red whenever the target time is reached.

Execution:

1. Wait until the time until warning for the first chest has exceeded

Expectation: The timer turns red and a warning sign appears.

Results: The timer indeed turns red and a warning sign appears, which can be clicked to close it. If the warning sign, which appeared because the time until warning has exceeded, is still visible, it can be clicked away. After a short time, the warning sign reappears.

Scenario 16: Open chest manually

Date tested: 25/06/2018

Description: This scenario tests whether the progress of the game gets updated when the progress bar is updated manually (open chest simulation).

Execution:

1. Click on the first chest in the progress bar

Expectation: The progress of the game is updated accordingly.

Results: The time stamp of when the chest is detected including the text "Found chest 1/i" where i are the amount of chests, is printed in the logger. In the status pane the number of chests opened is added with 1. Also the timer of the concerned chest is stopped and the correct time stamp is shown. The timer of the next chest starts ticking. Furthermore the progress bar is updated in which the clicked chest icon and its sections are coloured green. This all works as expected.

Scenario 17: Remove opened chest manually

Date tested: 25/06/2018

Description: This scenario tests whether the progress of the game gets updated when the progress bar is updated manually (remove opened chest simulation).

Execution:

1. Click on the first chest in the progress bar
2. Click on the section icon before the first chest

Expectation: The progress of the game is updated accordingly.

Results: The time stamp of when the chest is removed including the text "Removed chest 1" is printed in the logger. In the status pane the number of chests opened is decremented with 1. Also the timer of the previous chest starts ticking again and the timer of the current chest is removed. Furthermore the progress bar is updated in which the concerning chest icon is coloured blank. This all works as expected.

Appendix E

User Test Interview

This final user test was performed in the last sprint. Questions were asked verbally to the two persons using the application to host the escape room. The questions and the answers are documented in this section.

- Was it easy to configure the configuration file for the escape room?
No, they preferred configuring the escape room inside the application. It felt like writing code instead of using an application. Furthermore, they found it quite complex to fill the duration of the game in seconds and thought it would be better to fill in minutes and seconds. Also, the sections that needed to be filled in were not really clear what is meant by them.
- Was it easy to see when a hint was needed during the game?
Yes, they found that the timers of the chests helped in keeping an eye on the progress of the game. Also, the warnings appearing at certain times were helpful to know when hints are needed.
- Do you think the application helped in keeping the overview on the progress of the game?
As said before, the timers of the chests and the warning were helpful. However, the users said that too much chests were detected after each other. They also accidentally pressed on the stop button during the game. We deleted this button afterwards.
- Was the application easy to use after the basic explanation?
Yes, they found it was not very difficult to use. The progress bar was easy to understand. However, they found configuring files a little more difficult, but this can be done differently. Furthermore, they found the application useful if the user knows the escape room.
- Do you think the application is a useful tool for escape room hosts?
Yes, but they wondered what would happen if you use a lot of cameras? They found the images of the streams inside the application pretty small. It might be better if these could be enlarged. The activity has no added value for them, because you can also just look at the streams. They found the detection of the chests useful, if the game host misses this manually. Furthermore, they liked the fact that you do not always have to pay attention whether the players are in time with opening each chest.

Appendix F

Software Improvement Group

F.1 First SIG evaluation

De code van het systeem scoort 4.0 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Module Coupling en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. Dit soort code wordt niet alleen vaak aangeroepen, maar wordt vaak ook vanzelf groter omdat alle functionaliteit owordt gecentraliseerd. Om zowel de grootte als het aantal aanroepen te verminderen zouden deze functionaliteiten gescheiden kunnen worden, wat er ook toe zou leiden dat de afzonderlijke functionaliteiten makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden worden.

In jullie project wordt MainController het vaakst aangeroepen. De naam van die class geeft eigenlijk al aan dat er geen duidelijke verantwoordelijkheid is. Iedereen gebruikt nu MainController, en MainController geeft op zijn beurt weer toegang tot alle andere controllers. Zo'n "iedereen praat met iedereen" architectuur leidt op termijn tot een enorme hoeveelheid afhankelijkheden, wat er voor gaat zorgen dat een aanpassing in class A altijd ook tot een aanpassing in class B, C, en D leidt. Probeer daarom eerst te definiëren welke classes met elkaar zouden mogen praten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.

In jullie project is `MediaBar.createMediaBar()` een goed voorbeeld. Jullie hebben dat nu als n grote methode uitgeschreven, terwijl je aan het commentaar kunt zien dat er eigenlijk verschillende stappen in het proces zijn. Door die stappen met codestructuur aan te geven, bijvoorbeeld door nieuwe methodes te maken per stap, zorg je er voor dat je de code op lange

termijn begrijpelijk houdt.

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

F.2 Second SIG evaluation

In de tweede upload zien we dat het project een stuk groter is geworden. De score voor onderhoudbaarheid is in vergelijking met de eerste upload ongeveer gelijk gebleven.

Wat betreft de aanbevelingen uit de eerste upload, Unit Size en Module Coupling, zien we een wisselend beeld. Bij Unit Size zijn er duidelijk aanpassingen gedaan, en zien we dat ook de nieuwe code beter scoort. Bij Module Coupling is het probleem erger geworden, omdat met meer code er ook meer verwevenheid is ontstaan.

Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie ook nieuwe testcode hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog steeds goed uit.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload grotendeels zijn meegenomen tijdens het ontwikkeltraject.

Appendix G

Info Sheet

Title of the project: Tracking the Progress of an Escape Room to Support the Game Host

Name of the client organization: Popup-escape

Date of the final presentation: July 4, 2018

Description Popup-escape is a company that designs escape rooms. They have asked us to develop an application that supports the game host in the process of observing escape rooms. Hence, we developed an application that displays live video streams and shows valuable information about the progress of the game. The game host can configure the escape room in the application before players enter the escape room. This configuration sets up how the escape room is structured. The game host indicates the number of chests (key points in the game) that need to be unlocked and the time it should take players to open it. The application then processes the incoming video streams and detects chests that have been opened, as well as the level of current activity. The progress is measured against time. When the progress made is falling short compared to the preconfigured time limits, the host gets a warning, alerting him that the players in the escape room need a hint in order to be able to finish the game in time.

Members of the project team:

Name: Ege de Bruin

Interests: Cycling (Giro d'Italia), Netflix, football, food (mainly good Italian pizza), reading

Contributions: Back-end developer

Name: Robin Hurkmans

Interests: Traveling (hitchhiking), music festivals, football

Contributions: Front-end developer

Name: Jasper Kroes

Interests: Sports (mainly athletics), food (mainly cheeses) and memes

Contributions: Back-end developer

Name: Lisette Veldkamp

Interests: Music, reading, Netflix, singing, dancing

Contributions: Front-end developer

Client and Coach

Client: Jan-Willem Manenschijn, Popup-Escape, jw@manenschijn.com

Coach: Willem-Paul Brinkman, Interactive Intelligence group at TU Delft, W.P.Brinkman@tudelft.nl

Contact

Ege de Bruin: egedebruin@gmail.com — Jasper Kroes: jasperkroes@hotmail.com

A digital version of the final report for this project can be found at: <http://repository.tudelft.nl>

Bibliography

- [1] S. Nicholson. Peeking behind the locked door: A survey of escape room facilities. *White Paper available online at <http://scottnicholson.com/pubs/erfacwhite.pdf>*, 2015.
- [2] A. Bedagkar-Gala and S.K. Shah. A survey of approaches and trends in person re-identification. *Image and Vision Computing*, 32(4):270–286, 2014.
- [3] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), 2006.
- [4] How to use background subtraction methods? In *OpenCV Tutorials*, 2016.
- [5] J. MacCormick and A. Blake. A probabilistic exclusion principle for tracking multiple objects. *International Journal of Computer Vision*, 39(1):57–71, 2000.
- [6] A. Yilmaz, Xin Li, and M. Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1531–1536, nov 2004.
- [7] K.S. Kumar, S. Prasad, P.K. Saroj, and R.C. Tripathi. Multiple cameras using real time object tracking for surveillance and security system. *Proceedings - 3rd International Conference on Emerging Trends in Engineering and Technology, ICETET 2010*, pages 213–218, 2010.
- [8] B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, and A. Yamada. Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):703–715, 2001.
- [9] M. Wiemker, E. Elumir, and A. Clare. Escape room games: can you transform an unpleasant situation into a pleasant one. *Game Based Learning-Dialogorientierung & spielerisches Lernen analog und digital, Ikon Verlags Gmbh*, pages 55–68, 2015.
- [10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-January:779–788, 2016.
- [11] Hai Tao, H.S. Sawhney, and R. Kumar. Object tracking with bayesian estimation of dynamic layer representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):75–89, 2002.
- [12] M. Isard and J. MacCormick. BraMBLe: a bayesian multiple-blob tracker. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. IEEE Comput. Soc.
- [13] J. Houssineau, D.E. Clark, S. Ivekovic, C.S. Lee, and J. Franco. A unified approach for multi-object triangulation, tracking and camera calibration. *IEEE Transactions on Signal Processing*, 64(11):2934–2948, 2016.
- [14] Y. Tao, M. Skubic, T. Han, Y. Xia, and X. Chi. Performance evaluation of sift-based descriptors for object recognition. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010, IMECS 2010*, pages 1453–1456, 2010.

- [15] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, 2:1150–1157, 1999.
- [16] S. Isik and K. Ozkan. A comparative evaluation of well-known feature detectors and descriptors. *IJAMEC*, 3(1):1–6, 2015.
- [17] P.L. Rosin. A simple method for detecting salient regions. *Pattern Recognition*, 42(11):2363–2371, 2009.
- [18] R. Achanta, F. Estrada, P. Wils, and S. Ssstrunk. Salient region detection and segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5008 LNCS:66–75, 2008.
- [19] C.H. Lin, C.W. Liu, and H.Y. Chen. Image retrieval and classification using adaptive local binary patterns based on texture features. *IET Image Processing*, 6(7):822–830, 2012.
- [20] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T.N. Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- [21] D. Yu and L. Deng. *Automatic Speech Recognition*. Springer, 2016.
- [22] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [23] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [24] M. Cooke, P. Green, L. Josifovski, and A. Vizinho. Robust automatic speech recognition with missing and unreliable acoustic data. *Speech communication*, 34(3):267–285, 2001.
- [25] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel. Sphinx-4: A flexible open source framework for speech recognition. 2004.
- [26] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, and P. Schwarz. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.
- [27] T.B. Moeslund, A. Hilton, and K. Volker. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, nov 2006.
- [28] J. Carranza, C. Theobalt, M.A. Magnor, and H.P. Seidel. Free-viewpoint video of human actors. *ACM Transactions on Graphics*, 22(3):569, jul 2003.
- [29] J.W. Davis. Recognizing movement using motion histograms. Technical report, 1999.
- [30] R. Poppe. A survey on vision-based human action recognition. *Image and Vision Computing*, 28(6):976–990, jun 2010.
- [31] C. Busso, Z. Deng, S. Yildirim, M. Bulut, C.M. Lee, A. Kazemzadeh, S. Lee, U. Neumann, and S. Narayanan. Analysis of emotion recognition using facial expressions, speech and multimodal information. In *Proceedings of the 6th international conference on Multimodal interfaces - ICMI 04*. ACM Press, 2004.
- [32] K. Schindler, L. Van Gool, and B. de Gelder. Recognizing emotions expressed by body pose: A biologically inspired neural model. *Neural Networks*, 21(9):1238–1246, nov 2008.
- [33] M. Pantic and L.J.M. Rothkrantz. Toward an affect-sensitive multimodal human-computer interaction. *Proceedings of the IEEE*, 91(9):1370–1390, sep 2003.
- [34] S.G. Koolagudi and K.S. Rao. Emotion recognition from speech: a review. *International Journal of Speech Technology*, 15(2):99–117, jan 2012.

- [35] A.A. Chaaoui, P. Climent-Pérez, and F. Flórez-Revuelta. Silhouette-based human action recognition using sequences of key poses. *Pattern Recognition Letters*, 34(15):1799–1807, nov 2013.
- [36] I. Cohen. Improving trainees' performances while under stress using real-time feedback. pages –, 2015.
- [37] C. Gonzalez. Decision support for real-time, dynamic decision-making tasks. *Organizational Behavior and Human Decision Processes*, 96(2):142–154, mar 2005.