PlayPOD

Multi-medial enhancement of audio theatre on Android smartphones

PlayPOD
Multimedial förbättring av ljudteater för android-smartphones

Daniel Larsson
Armin Magnafic
Daniel Steinvall
Martin Wahlberg

Faculty of Health, Natural Science and Technology (HNT)
Computer Science
15
Lothar Fritsch
Karl-Johann Grinnemo
January 2019
Abstract—The development of technology and digital revolution has led to a connection between humans and smartphones. Today smartphones are a big part of our lives and we interact with them every day. Most people are not aware about how powerful algorithms can be, what they can do and how they can affect their privacy. Playpod is a smartphone application created for the Android platform that utilise algorithms and Android permissions to implement side effects that enhance the story telling of six different short audio dramas also known as playpods. The side effects are related to the content of the playpods and use the Android permissions to access sensitive user information and incorporates this information in the side effects.

Index Terms—Podcast, Algorithms, Theatre, Drama, Enhanced podcast experience, Privacy invasion, Sensor manipulation

I. INTRODUCTION

This section will give an introduction to the project, what it is about as well as the goal.

The technology plays a wide role in today’s modern society where the influence from computer algorithms in human lives is greater than ever. It has come to a point where systems can affect and sometimes even control what we do. The purpose of this project is to explore this digitalized influence by means of audio drama pieces reinforced with programmed multimedia feedback from a player app.

The goal of the project is to create a stand-alone smart phone application for the android platform. The application will appear to the user as a podcast application with a number of podcasts (also known as playpods) available. When the user listens to one of the playpods he/she will experience not just the audio, but also a number of so-called side effects. These side effects will range from sending email to the user with links to food or videos, to tracking the user’s whereabouts and classifying them based on different algorithms.

The project was carried out by four students at Karlstad University in Sweden. The timespan for the project was roughly five months starting from week 35 (August) 2018 and ending in week 3 (January) 2019 and the project was done at half speed.

The rest of the report is structured in the following way. Section II explains the background to the project as well as a summary of each playpod, section III describes the development environment and the version control we have used throughout the project, section IV describes the steps towards a final specification, section V presents the design of the application, section VI describes how all parts of the application was implemented as well as how the application was tested, section VII are about the major problems that was encountered during the implementation phase, section VIII describes the final result for all playpods as well as the application in its entirety followed by section IX which discuss what could have been done better as well as what worked especially well during the project.

II. BACKGROUND

This section will explain the background to the project, what it is about as well as a description of the different playpods

We have been given six drama playpods from our project customer Amanda Fromell who is developing the playpod project. She is an Arts and Humanities Research Council (AHRC) Ph.D. student at the department of Drama and Theatre Arts at the University of Birmingham in the UK. Among other things, she is working with the intersection of art and technology and this is where we as program developers come in the picture. Our task is to create a smartphone application that interacts with the playpods. The playpods should be available inside the application and side effects are going to be executed while the users are listening. Amanda provided us with her suggested side effects for each playpod and we also brainstormed some new ones together with her at the beginning of the project. The side effects are related to the playpod, hence there is going to be a set of different side effects for each playpod. After listening to all playpods the goal is to generate critical awareness among the listeners by giving them an experience in how computer algorithms can affect us in daily life. Below is the content of the playpods presented one by one.

A. Drowning

This playpod consists of a conversation between Ali and Izzy. Ali is wondering and searching online whether it is possible to trick Googles search algorithms so negative information can be drowned by new positive information. He is afraid that Google has the ability to govern too much about whether certain information should be showed or not. The character Izzy is trying to sell him a new online lifestyle while answering Alis questions. Almost everything she says is based on statistics. The original question/concern of Ali is drowned out by Izzys ability to predict what he likes. At the end of the playpod Alis question is completely forgotten, and he starts to act in line with Izzys suggestions that comes from algorithms.
B. Falling

Two characters, Alec and Joe, are having a conversation about financial trading in a pub. Alec is concerned about computer algorithms and how they are controlling the market. For example, he mentions a situation where a book price escalated to 24 million dollars just because the vendors algorithms were set to sell the book at a price some percent higher than their competitors price for the same book. Alec used to work with financial trading, but he quit his job after the crash of 2010 on May 6th, known as The Flash Crash. Their dialogue starts, after a while, to imitate this crash with a conversation loop bursting into a crash.

C. FitChip

This playpod is about health and wellbeing. Sarah is the victim in this audio recording where her mother and her mothers colleague constantly are telling her to implement a so-called FitChip. It is a device that is attached directly to your body which gives you information such as calories burned, steps taken, distance covered and sleep quality. They are telling her this because they are worried about her health situation since she is drinking beer and eating pizza during the weekdays. In the playpod, appears advertisement snippets several times and they are saying that with FitChip implemented she does not need to think about her health, the device is doing everything for her.

D. High Risk

This playpod is about the likelihood to become a criminal. It is set in a hospital where the main character, Emma, follows the birth of her daughter. While she is in the maternity ward two officers arrive and start a gentle conversation with her. As time goes the conversation starts to get more unpleasant from Emmas perspective. They are mentioning things like her relationship with her mother and her drug-addicted fathers absence. After a while, it becomes clear that the officers are in fact there on criminal justice errands and alerts the fact that Emmas daughter has a risk of becoming a criminal.

E. Lets Google it

This playpod depicts a conversation between Kay, Helen, and Stanley at a fancy-dress party. They are constantly using Google during the conversation to fill the awkward silences with some content. They are totally relying on Google and says, for instance, that McDonald’s is healthy just because they read it on Google. In the playpod, it appears like they cannot think without Google.

F. Trapped

The playpod trapped is about the humans obsession with the smartphones. Smartphones are taking a considerable part in our lives, but what is going to happen when the interaction with the phone impact our way of living and our thoughts? What happens when the phone is out of battery, will we be able to think without it? This playpod illustrates a conversation between the colleagues, Simon and Anna. It starts like a normal conversation, but then Anna begins to address her concern about Simon losing his reception for some days. The conversation takes a turn and becomes like an interrogation. Anna is worried that he is hiding something, and she is also worried about the fact that something dangerous could have happened to him and he should not be able to reach out to anybody.

See Appendix A for the initial specification from Amanda Fromell.

III. Environment & Version control

This section describes the integrated development environment (IDE) we have worked with during the project, how we chose that IDE and also which version-control software that we utilized to synchronize our work.

A. Environment

The final product of the project is a stand-alone smart phone application for the android platform that is able to play and perform the side effects for each of the playpods, and therefore we had to conduct some research about the available IDE:s in order to be able to pick the one we felt was the best suitting for this project. Each member of the group performed their own research about which IDE:s that was available and then provided the group with their findings.

When the group discussed their findings with each other to decide which IDE to use there was two IDE:s that all members had found and they were both suited for this project. The choice was between Android studio [2] which use Java as programming language, and Xamarin [12] which is included in the Visual studio IDE [11] and use C# as programming language.

The group felt that either one of the two options would have worked, but in the end the group felt more comfortable programming in C# since all group members have more recent hands on experience with it as well as more prior knowledge than with Java. Aside from the programming language another advantage with using Xamarin and Visual studio is that the application that is developed mainly for Android can be ported to iOS if it’s necessary.

B. Version control

During the project the group used Git as version control software which simplified both the synchronisation of the project and distribution of work. In Git there can exists multiple versions of the program and each version is called a branch, the different branches can be used to develop different functionality which is then merged into the main branch when the functionality is fully functional. The main branch typically contains the final product.

In this project the implementation of the side effects for a playpod was done individually by one team member per playpod. I.e. There was one person that worked on the side effects for the playpod Falling simultaneously as another person worked on the side effects for the playpod Trapped and so on. By developing an application skeleton and using it as a
foundation for the project (i.e. in the beginning of the project the master branch consisted of the skeleton) and then creating one branch for each playpod that was currently being worked on it didn’t create any code conflicts. When the side effects for a playpod was completed the relevant branch was merged to the main branch and added to the final product. Using this method combined with good communication within the group removed any problems regarding project synchronisation and work distribution.

IV. Specification

This section will explain how we came up with the specification. This includes the meetings we had with Amanda and also what was said at those meetings. The final specification will also be presented here

A. Method

Before we could start implementing the application we had to agree on a complete specification. This was done through weekly meetings with Amanda. The first meeting was conducted on September 12 2018. The group had prepared for the meeting by listening to the playpods, as well as brainstormed ideas for potential side effects. These ideas were then discussed with Amanda who found the ideas interesting and approved the group to look further into it. We agreed upon diving deeper into the side effects for Fitchip, Trapped, and Drowning. The second meeting was on September 19 2018 where we discussed our findings with Amanda, and talked about alternative solutions for the side effects the group deemed too hard or time consuming to implement and agreed upon to look into the possibility to implement the side effects for the final three playpods, High Risk, Falling and Let’s Google It. The following meeting was on September 26 2018. The group discussed with Amanda which side effects that were deemed too hard or time consuming to implement and looked for alternative solutions as well as briefly talking about the side effects we agreed to be possible to implement. The final meeting was on October 1 2018. The goal of this meeting was to agree with Amanda on a final specification for side effects and layout in which the group would base the application off.

B. Side effects

Drowning: While the pod is playing users will be able to see a slideshow of pictures related to the content of the pod. The longer into the pod the user is the more information will be displayed and the pictures are going to change faster. There will also be Facebook authentication implemented in order to gather information about the user and to get access to their account.

Falling: This side effects for this playpod will use vibrations with different intensities and frequencies, increase the temperature of the mobile as well as send emails to the user that contains Youtube links that is relevant to the content of the playpod. The visual effect will be a video that loops in different speeds depending on which part of the pod the user is currently listening to. This was the final agreement on implementation which differs from the original one where the goal was to give users several questions with limited amount of time to answer. The original suggestion about the visual effect was to have a heart monitor that shows the heartbeats or a stock graph that shows the current state of the stocks but these ideas were removed after the meetings and discussions.

Fitchip: The side effects for this playpod will be able to keep track of how many steps the user takes and will also be able to determine if the user is moving or not. In order to be able to listen to the playpod the user must be moving. The step counter will be active from the first-time user listens to the pod and will later send a result to the user in form of a ranking which will show how active the user is compared to the other people. This comparison will result in tips of food and activities that the user should do. While listening to the pod the user will be able to see the step counter and information about his or her activity improvement.

Highrisk: The purpose of the side effects for the playpod is to gather fundamental information about the user and give them a score which suggests improvements areas to the user. The information will be gathered from the Google plus account if the user has it, otherwise the user will get a pair of general questions. The answers will then be analysed and the user will get a result depending on answered questions and the gathered data.

Trapped: While listening to trapped the user will be able to see a map which show their current location. This information is gathered and saved from the first-time the user is listening to the pod as well as several times after the pod is completed, after some time the user will get an email with this data which informs the user that the app is able to track him. The team originally suggested on implementation where the users would be asked few questions about usage of their social apps where the gathered information would be compared to the real one taken from the app statistics. After some meetings and discussions with Amanda the decision was to send information to the users email about their location instead of gathering statistical information and asking question to the user.

Let’s Google it: The original goal for this side effect was that the user meanwhile listening to the pod will be given a pop-up quiz with the questions about the content of the pod. The user will also be given the chance to use Google implemented as a search bar in the pod. The background of the pod was thought to be the pictures from Google that are relevant to the playpod. It was also expected to retrieve the information of the user’s browsers search history but this will be discussed to see if it is feasible to implement.

All initial suggestions from Amanda can be retrieved in Appendix A

V. Design

This section will present the design of the application. This includes a classification of the side effects for all playpods, the type of components that is present during application runtime, how these components interact as well solutions to side effects that range over multiple playpods.
A. Classification of side effects

In order to get a broad overview of the side effects the playpods was categorised in two groups, Interactive and Non-interactive. A playpod that is categorised in the Interactive group have side effects that requires the user to interact with the application in order to get the side effect to work. The same reasoning is applied to the Non-interactive group, a playpod that is placed in this group have side effects that does not require user interaction in order to work. A playpod must at least be categorised in one group, but can also be present in both. A visual representation can be seen in table I.

<table>
<thead>
<tr>
<th>Playpod/Group</th>
<th>Interactive</th>
<th>Non-interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drowning</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Falling</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>FitChip</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>HighRisk</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Let’s Google it!</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Trapped</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

While the broad categorisation gives an idea about how the side effects work it is not detailed enough to be of any use during implementation and actual understanding of the side effects. Therefor a more detailed classification for each playpod was performed which resulted in a better understanding of the different types of side effects that was supposed to be implemented. The different groups that the playpods are categorised in are the following:

- Classification of user - Place the user in one of several pre-determined groups
- Email - Send the user an email with content related to the playpod
- GPS - Track the user’s location
- Motion detection - Detect whether or not the user is moving around
- Notifications - Give the user a device notification with information
- Personal information gathering - Gather information about the user in any way
- Quiz - Ask the user questions with multiple answers
- Slideshow - Provide the user with a slideshow of pictures
- Vibrations - Make the device vibrate

The more detailed overview of the playpods provides a clear picture of their respective side effects. This more detailed overview can then be used to see if any playpods are overlapping in terms of functionality which can be further used to make design decision in the actual implementation. The detailed classification of the playpods can be seen in table II.

B. Design choices

In order for this section to be understandable the following terms have to be explained.

- View: Consist of the layout for the user interface (UI)
- Activity: Creates a window that is connected to a view in order to display the user interface, also consists of the code that controls the functionality of the different components of in UI
- Service: Performs work in the background without user interaction
- Intent: A request to move to a new activity, start a new service, or interact with an existing service.
- Widget: A button, textbox, layout etc.

Skeleton structure - Martin Wahlberg: The core structure of the skeleton consists of several Activities that each have a corresponding View, there is also a background service that handles the audio playback of the playpods. When the user opens the application from a closed state i.e. there is no current instance of the application, they will be brought to a View that consist of a list with all available playpods. When an entry in the list is pressed the user will be taken to the Activity and View pair that corresponds to the entry they chose. E.g. If the user presses the entry HighRisk they will be brought to the Activity and View pair that contain the logic and UI for that particular playpod. The underlying technique when moving between different Activity and View pairs as well as making requests to the service back is the Intents. The Intents are created and executed each time the user moves between different Views on the device as well when the user want to interact with the audio by pressing the available buttons.

In the skeleton the Views for all playpods look the same, it consists of a button that starts/pauses the corresponding playpod, a button that starts/pauses the playpod that was played most recent, a 15 second fast forward button, a 15 second rewind button and a progress bar that shows the current position as well as the total length of the current playpod. A visual representation can be seen in Appendix B. When the user press one of the play buttons an Intent is sent to the background service that consist of the desired action and the playpod to perform the action on.

The background service contains logic for all available actions and simply executes the code that corresponds to the desired action. When the action has been performed the result is visible in the current View. E.g. The user wants to rewind HighRisk 15 seconds and press the button that corresponds to that action. The Activity creates an Intent with the rewind action and the playpod HighRisk this Intent is then sent to the Service. The service receives the Intent, extracts the rewind action as well as the playpod HighRisk and executes the code that corresponds to the rewind action. The progress bar in the View is connected to the audio, which means that when the rewind action has been executed the progress bar responds by moving back the number of steps that corresponds to 15 seconds. A context diagram of the skeleton structure can be found in appendix Appendix E.
in the playpods respective Activity classes would cause these classes to be large and messy. Instead a separate Service class was created for each of the playpods which scheduled the side effects at their respective time stamps.

For the playpods Trapped and HighRisk we felt that there was no need to create a separate Service class for the side effects as it didn’t require much code to accomplish their side effects. But we should have created a separate side effect class as well as a Service class for them in order to get a mutual structure for how the side effects are handled in each playpod.

VI. IMPLEMENTATION

This section will present how the skeleton, core functions as well as side effects for the playpods was implemented.

A. Skeleton - Martin Wahlberg

Early in the project the group discussed if it was possible to find an already working podcast player and use it as a starting point instead of developing everything from scratch. The group decided that while it will take time to develop a skeleton it is the preferred approach since the group will then have more knowledge about the code structure as well as the functionality instead of using some other developers code that the group had no control over when it was developed.

Before the implementation begun, the core functionality was decided to be the following. Playing podcast audio, basic audio controls (play, pause, forward, backward), an interactive progress bar, a display that shows the current position as well as the total duration of the audio as well as the ability to save information about the different playpod in order for the user to be able to continue listening between application sessions.

To create a simple and intuitive starting point the skeleton consisted of a main activity that contained a list of all available playpods. When one of the playpods is selected in the list the user is taken to a new activity connected to a new view. The new view had a button that could be used to play and pause the selected playpod, a button that moved the current duration of the audio forward 15 seconds, a button that moved the current duration backward 15 seconds and a button that could be used to play and pause the playpod that was currently playing. Six identical activities and views was created because the starting point of all playpods was decided to be the same. A visual representation of a view can be found in [Appendix B].

The audio playback is best suited as a background operation as it doesn’t require any user interaction and therefore was place in a service which can continuously run in the background. The service consists of all logic that is controlling audio playback functionality such as play, pause, forward and backward. The service also contains information about if any audio is currently playing and which the last played playpod is. In order for the user to be able to use any audio controls such as play and pause the different buttons in the UI create intents when they are pressed, the intents request the wanted action to the service which executes the corresponding code for the given action.

Interaction with the progress bar create intents to the audio playback service that corresponds to the desired actions. E.g. Moving the progress bar forward or backward will create intents to move the audio forward or backward the desired time. When the progress bar is dragged the time the audio will move to is displayed as the “current” time.

When a playpod is playing, information about it is saved continuously to a permanent file that reside on the phone. The information contains the name of the playpod, the current duration of the playpod and also that this playpod is the last one to be played in order to continue with it if the application is restarted.

In summary, the skeleton consisted of basic audio playback functionality such as play, pause, forward and backward. The skeleton had a list with all playpods that took the user to the playpod specific view when an entry in the list was pressed. There was also functionality to show the current and total duration of the audio file as well as an interactive progress bar.

B. Sending email - Armin Mangafic

The email functionality consists of two classes, MailMessage which is the composition of email with subject, sender, receiver and message body all of type string and SmtpClient which connects to Google’s smtp server on port 578. For the senders email address we created a new Google email address called “0dysc0nnect0@gmail.com” which is used to automatically send email to the users. The email functionality will only be available if the user has the app running in the background, i.e if the email is scheduled to be sent in 24 hours from the first use then the app must be running in the background in order to be able to send the mail.

C. Trigger events at specific time stamps - Martin Wahlberg

The majority of the playpods have side effects that has to triggered at specific time-stamps, we therefore felt that there was a need for a general event-handler that could handle all sorts of side effects and trigger the at a specific time-stamp. The first thing we did was to search for an already

### TABLE II

DetaiLed classification of playpods

<table>
<thead>
<tr>
<th>Playpod/Group</th>
<th>Classification of user</th>
<th>Email</th>
<th>GPS</th>
<th>Motion detection</th>
<th>Notification</th>
<th>Gather personal information</th>
<th>Quiz</th>
<th>Slideshow</th>
<th>Vibration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drowning</td>
<td>×</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Falling</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FitChip</td>
<td>×</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HighRisk</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Let’s google it!</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trapped</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
existing event-handler that we could include in our project and use, unfortunately we did not find anything that fulfilled our demands. Instead we created our own event-handler that took the desired side effect, a timer with a specified trigger-interval as well as the time-stamp to trigger it as input. The event-handler then started the timer and each time an event was created by the timer it checked if the current duration of the audio was larger or equal to the time that the side effect should be triggered, if this statement is true then the side effect is started.

D. Animation of control panel - Martin Wahlberg

The control panel exists of a play/pause button, rewind button, fast forward button, a progress bar, current duration of the audio and total duration of the audio. This is information that isn’t necessary to display to the user at all times. Therefore we decided to make the whole control panel be able to fade in when the user press the screen and fade out if the user have been idle for a couple of seconds. This feature was implemented by placing all widgets (buttons, text-fields etc) in the same layout and then using existing animation functionality to reduce the transparency value of the entire layout to zero percent for the fade out animation, and to 100 percent in the fade in animation [8][6]. The animations had to be performed in asynchronous functions to avoid blocking the entire UI thread [5].

E. Drowning - Daniel Larsson

The Drowning playpod uses two .axml to display the layout of the activities. The drowning.axml is the handles the main drowning activity and consists of two nested RelativeLayout. Inside the outer layout lies the VideoView and the other RelativeLayout. The inner layout holds multiple randomly placed ImageView. Relative layouts where chosen inorder to allow the views to overlap eachother.

The drowningmatch.axml contains one LinearLayout several TextView and one CircleImageView.

The side effects for drowning are handled in a background service that runs on a different thread from the main UI thread. This is to make sure that the side effects do not freeze the UI. The images used for the slideshow are put inside of a two-dimensional array, where one column contains the timestamp in milliseconds and the other column contains the ID for the images. When the play timer of the playpod reaches the timestamp defined in the array it displays a picture on one of the ImageView. The image view is selected randomly with a few restrictions. An image view is not allowed to be selected if it is already in use by another image, and recently used image views are also not allowed to be chosen. This is to avoid the possibility that the images choose the same location. The images are then displayed for two seconds before disappearing. The original idea was that the images would fade away over a few seconds. This idea was implemented and tested. The fading worked fine, but it caused the GUI to freeze, even if it ran on a different thread. The cause of this is unknown and the feature was scrapped due to time constraints. At around six minutes into the playpod, the user will receive a notification telling them they have a match with YumYum84, if the user taps the notification, it will open a new activity that resemble the layout of commonly used dating applications.

F. Falling - Martin Wahlberg

The side effects for the Falling playpod consist of vibrations with different frequencies and intensities that vibrate in synchronisation with the audio, increase of phone temperature, modification to the visual video and sending email with links to Youtube videos that is related to the playpod content. The subsections below consist of how each of the side effects was implemented and how they turned out.

General solution of the side effects: Since the side effect for this playpod takes up a lot of power in order to function properly they were placed in their own separate service that executes in a separate thread. It was necessary to create a new service for the side effects as it would otherwise cause the GUI to freeze as a lot of work was performed in the thread responsible for the UI. The service containing the side effects is created when the user starts the playpod and destroyed when the user pauses the playpod. Preferably the service should be paused instead of destroyed to avoid unnecessary work for the CPU, but no suiting solution could be implemented and it was decided that the service would be destroyed instead as this caused the side effects to behave as expected.

Synchronisation of vibration and audio: During two sections of the playpod you can hear heartbeats and sounds from a heart-monitor that increase in frequency and intensity as the playpod move forward. The idea was to synchronise vibrations to the heartbeat and heart-monitor audio enabling the user to feel the heartbeats in their hands. In Xamarin it is a simple task to trigger vibrations with the use of the vibration class [14]. The difficult part is to create the vibration pattern with the vibration effect class [15] as it requires complete knowledge about how long each vibration should be and the time between each vibration in milliseconds (ms).

Each heartbeat in the audio consist of two ”pumps” that are simulated by two vibrations with different strengths. This implies that in order to create a synchronised vibration pattern for the heartbeat audio, complete knowledge about how many ms each ”pump” take, the time in ms between the ”pumps” as well as the amount of ms between each heartbeat is required. The audio file was manually inspected in an audio-modification program called Audacity [3] and after enhancing the lower frequencies the heartbeat sound waves became visible. All required times could then be extracted and later used to create the vibration pattern.

The sounds from the heart-monitor was easier to implement as each individual sound could be simulated by a single vibration and the strength and frequency of the vibrations could be increased over time. The audio file containing the heart-monitor sounds was manually inspected in Audacity [3] and the length of each sound as well as the time between each
sound could be extracted and later used to create the vibration pattern.

**Increase of phone temperature:** The side effect that increases the temperature of the phone was decided to be combined with the side effect that synchronise vibrations with the heart-monitor sounds in order to give the user more impressions and possibly get a better reaction from them. There is no function that can be called that simply increase the temperature of the phone on demand, instead the temperature has to be increased by performing heavy work on the phone. After discussion within the group it was decided to create an infinite loop and perform some sort of operation inside it over and over. The first operation that was tested was counting to a specific number over and over, while this increased the temperature of the phone it took too long before it was barely noticeable. It was clear that a heavier operation was needed in order to increase the temperature faster and therefore we tried to fetch the location of the user with the help of high precision GPS once every two seconds.

While this change decreased the time it took to notice the change in temperature the phone did not increase in temperature as much as we would have liked. After adjusting the interval from once every two seconds to twice every second, the high precision GPS not only decreased the time it took to notice a change in temperature but also increased the temperature significantly which is the result we were after. Worth noting about this side effect is that the temperature increase will vary significantly between different devices. This variation is because not every device has the same procedure to deal with an increase of temperature in the phone which means that the result will vary between devices.

**Increasing the playback speed of the video:** The intensity of the content is heavily increased in one part of the playpod and it is therefore suitable to increase the speed of the background video during that section as well. Since the video utilise the mediaplayer class \[10\] to play the video, the speed of the video can be increased by using the playbackParams class \[13\] together with the mediaplayer.setPlaybackParams function call. The playbackParams class \[13\] can be used to alter properties of a mediaplayer such as the playback speed which makes it the optimal choice for this side effect. In total, four different video speeds were used in the solution, normal speed, five times the speed, seven times the speed and 10 times the speed. The reason four different speeds are used is because the intensity of the audio is increased three times and the video have to return to the normal speed afterwards.

**Sending email with Youtube links:** When the user has finished listening to the playpod they will receive two emails over the course of 72 hours. The emails contain text which tells the user that we know that they have listened to Falling and that we think that they should have more information about the content in it. There will also be one of the following youtube links in the email:

- Flash Crash 2010
- The Wall Street Code

An example of an email can be seen in Appendix C.

A visual representation of the side effects in form of a timeline which describes when each of the side effects occur can be seen in Appendix D.

**G. Fitchip - Daniel Steinvall**

There are three major side effects in the FitChip playpod:

- A pedometer that shows current steps while listening (runs non-stop in the background)
- Motion detector to check whether the user is moving or not
- Email sending one day after listening with diet suggestions due to step statistics

Everything explained in this section is valid while the application’s side effect mode is turned on. Otherwise, the playpod is going to execute as a normal audio application without any side effects. The coming sections will explain how the side effects were implemented one by one.

**Pedometer:** The process with calculating the steps was straightforward because Android has a built-in functionality to detect steps that could be used. The most challenging task faced here was how to get the pedometer to run all the time. At first, it was decided to create a sticky service, which basically means a long-time running service. This turned out at great at the start-up of the application, the problem was that the service was automatically destroyed by the Android system after approximately 10 minutes. This was done because the service was inactive, i.e. the user did not walk. Some attempts were made to send a message to a broadcast receiver while the service was being destroyed. The work of the broadcast receiver was then to restart the service. However, this was a failed attempt since Android did not only close the service, but also the broadcast receiver.

After some research, it was decided to try with a job-service instead, that should be a suitable service for a long-running work \[7\]. However, while using this approach the same problem occurred as before, the service was destroyed after roughly 10 minutes. The difference here was that while the service was destroyed Android gave the developer an opportunity to reschedule the job. This option was selected, and the Android system restarted the service as promised, the only issue was the delay. After some testing, it was noticed that delay could be anything from 2 minutes to hours. Of course, that kind of delay is unacceptable when it comes to a pedometer that needs to count the steps all the time. Imagine if the pedometer is inactive while the user is doing some power walking, then many steps would be missed and the statistic in the mail would not be accurate at all.

The issues with the service and job-service were due to the update in Android 8.0. They had some complaints about battery draining because many applications on the cell phone were running all the time. That is why they introduced these constraints like killing inactive applications/services after a certain amount of time. This caused a lot of trouble while implementing the pedometer. Anyhow, a sustainable solution was going to be found. A foreground service did the trick to
make the service run non-stop. Users with Android versions from 8.0 and higher are going to see a notification that the service is running. This notification is going to be there as long as the service runs. The service has not been destroyed by Android even once while testing it, even if the service has been inactive for hours.

**Motion Detector:**

The purpose of this playpod is to get people to be more physically activated. Therefore, a walking requirement is implemented with this side effect, i.e. the listener needs to walk to listen; The user cannot start FitChip with any user interface interaction. If the listener is stopping, the playpod will also be stopped and start to vibrate until the user manually cancels the vibration through the user interface or start walking again.

The first effort made was to implement an accelerometer in its own service to detect the motion of the device. The accelerometer should indicate the acceleration of the device in three-dimensional space. This turned out to be a poor solution since the motion detected was not precise. It was difficult to configure the sensibility of movement, either the sensibility was too high or too low resulting in too much or too little motion detected.

A second and successful effort was to use the already implemented pedometer functionality to detect movement. The motion detection service job is to initiate and keep track of a timer. The timer was after some testing set to the interval of 2.2 seconds. The timer is reset for every taken step in the step-counter service. If the timer is not reset within 2.2 seconds that means inactivity and if the playpod is currently playing it will be stopped and the device is going to vibrate, an annoying and consistent vibration. Motion is detected when the user starts to walk again and the playpod will be automatically started without any user interaction.

The motion detection service is only active if the user is in the FitChip activity, which means the playpod is paused while pressing the Android return button. However, you are still allowed as a user to press the home button and do some other stuff with the cell phone. Putting the phone to sleep will make no difference because the main idea here is to have the device in your pocket while walking and listening to this playpod.

A great concern with this side effect is the opportunity to shake the device. If the user shakes the device while in the FitChip activity, the pedometer class is going to interpret those shakings as steps and that is going to set the motion detector to true, i.e. indicating movement even if it is false. This is going to start the playpod because the device “thinks” the user is walking. Anyhow, a solution was suggested to include some GPS functionality to check whether the device was actually moving. After some research, Xamarin essentials API geolocation was tried to get an accurate location. According to the documentation, it is a bit of trade-off between the time and location accuracy. The highest accuracy is requested here to notify whether the device is changing its location or not. In the most time-consuming mode with the highest accuracy it could only detect the device’s location with a range between 0 and 100 meters. Furthermore, after some testing, it took roughly 25-30 seconds to get a different location. A search for a location within 25 seconds from the previous search only resulted in the same location. This means that GPS functionality still can be used but it will only detect movement every 25-30 seconds. Which means that the user can shake the device and it will stop after 25-30 seconds but then the user can just start shaking their device again and it will take additional 25-30 seconds to stop it again and so on. It was decided that this was not a sustainable solution, therefore the issue with the shaking remains. A further note is that even if the GPS worked great with the highest accuracy and quick responses it would raise a problem if the user walks in circles (can be circles with a diameter of 100 meters since the bad accuracy on Android devices in the best mode).

**Mail functionality:**

This side effect was the simplest to implement in the FitChip playpod. Basically, an alarm is set at the end of the playpod with the alarm manager class (specifically at time 7:50 of the total 8:04). The alarm is set to start a pending broadcast receiver (called pedometerStatistics) intent one day after the alarm was triggered. The logic in the broadcast receiver is going to calculate the average step rate per day since the application’s download date. Then based on that rate it is going to send email to the users with diet suggestions.

**General comments:**

All the interaction between the user interface and the services is done through a broadcast receiver. The receiver receives a message and acts by calling some relevant method in the activity class that will change the user interface.

**H. Trapped - Armin Mangafic**

The implementation of this side effect consists of three major parts: the Google API for implementing the map, the geolocator plugin for getting the data from GPS sensor and a function that stores the coordinates for each day and emails the user after five days with the address that corresponds to these coordinates.

The first step was to implement the visual part of the side effect where the user will be able to see a map of the world when clicking on the playpod. Google provides its own API for this and it could be downloaded into the project from NuGet package manager by searching for Google play services Google maps. In order to be able to use this API I had to get access to the API key from Google. This is done by accessing the Google developers pate and connecting our SHA1 fingerprint and package name to the API key that we are going to use. When the API key is received it is copied into our project as a string in string.xml file which then is used in AndroidManifest.xml as meta data. Additionally, in the manifest file we had to add few permissions which can be observed in the table. After the setup is done the next step was to modify the layout file which will hold the map shown on the screen by creating a map fragment and connecting it to the fragment object in setupMap function in the class of
the playpod. The last step of this part was to create the map object and pass it as a parameter in the onMapReady function which connects the important parts and gets it shown on the user’s screen.

The second part was to actually get the coordinates from GPS sensor that later will use the map from the first part to show the users current position on the map. This was implemented by using Xamarin’s geolocator plugin provided by James Montemagno. This plugin enables us to use cross-geolocator class that can retrieve the latitude and longitude data from the current position of the user. This happens inside the getCoordinates function where the data is stored in variables and sent as parameter to the map object that shows the current position of the user on the map. The map object has an onclick function which means that when the map is clicked the getcoordinates function is called. In order for this functionality to work the user needs to enable GPS permission from the application settings as well as enabling the GPS. The permissions that were required for getting the location data from the device can be seen in table [IV].

The last part of this side effect used shared preferences to store the data after the user has closed the application and that this data can be received again later on. The point here was to create a scheduling task that will run in the background even if the user is not interacting with the application. The first-time a user listens to trapped the alarm manager will schedule the task that will run getcoordinates function 24 hours after the user has listened to the pod and it will store this data in shared preferences depending on which day of the week it is. This scheduling will repeat each 24 hours and after five days from listening to trapped, the user will get an email with a list of addresses that the user has visited during the week. If the user was on the places where the address cannot be found then the user will not get any email i.e the user will only get an email if the app actually converted the coordinates into addresses successfully. This will inform the user that the application is able to track him even if the user is unaware of it and not using the application. The translation from coordinates into addresses is done by using reverse geocoding from the geocoder class. Geocoder class in Xamarin can be used to transform an address or other location description into a latitude and longitude coordinates. The reverse process translates the coordinates into corresponding address which has been used in in this context. The reverse geocoding has not been always so accurate and sometimes it had some troubles finding the location and sometimes it gave the address of the street next to the exact location. From the testing experience we also noticed that the location address may not be retrieved if the user is close to sea/water.

<table>
<thead>
<tr>
<th>Table III</th>
<th>Required permissions for the GoogleMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission</td>
<td>Internet access</td>
</tr>
<tr>
<td></td>
<td>Write to external storage</td>
</tr>
<tr>
<td></td>
<td>Google read gservices</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table IV</th>
<th>Required permissions for the GeoLocator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission</td>
<td>Access coarse location</td>
</tr>
<tr>
<td></td>
<td>Access mock location</td>
</tr>
<tr>
<td></td>
<td>Access fine location</td>
</tr>
</tbody>
</table>

I. Facebook authentication - Armin Mangafic

To be able to send the email to the right user we needed to retrieve the user’s email. One of the solutions that the team together with Amanda agreed on was the implementation of Facebook log in where the user will give the app access to retrieve its email address. We used existing Facebook SDK which we downloaded into our app. The first step here was to get the Facebook app id which will give us the permission to use the Facebook API in our application. The app id can be requested on Facebook developers page by registering our application together with our SHA key hash. The next step was the coding part where we needed to reference our app id and meta data which is specified in the manifest file. We also needed to add internet and write external storage permissions. With Facebook SDK we had access to many useful interfaces and methods that we needed for our purpose. The login button for Facebook was included in the SDK which we used as a part of the start page in our app.

After the login part was implemented and the user was able to successfully login with its Facebook credentials through the Facebook SDK the next step was to retrieve the users email from the account. Facebook has a method for this called graphrequest which allows you to extract the user information from the account. In order to be able to get this information we needed to get a permission from Facebook for this which was requested on the Facebooks developers page where we registered our application. For each field that we wanted to extract for the users account we needed to request the permission from Facebook to do so. Some of the permissions did not require complete description of how the permission would be used and were there by default, while some others required full documentation and video proof on how we would use it in our implementation. Luckily getting email was not that strict and was allowed from Facebook without any troubles. The original plan was sending an email to Amanda containing the link of the user’s profile in order to be able to add him as a friend but since this required the detailed description of our app that would take a lot of time we later decided to use the default permissions which gave us the ability to extract users information such as email, name and profile picture. This is the content of the mail being sent to Amanda letting her know all the needed information for adding the user as a friend on Facebook.

J. Let’s Google it - Daniel Steinwall

The playpod “Let’s Google it” does not have many required user interactions during the play time. The only user interaction required to start the playpod is to press on the video that
is displayed in the centre of the screen. The video depicts a globe that is spinning around with a black background. The video will begin in a slow tempo and increase its speed for every minute. The functionality to display, play, stop and fast forward a video was already implemented when falling was created. Therefore, have a look at the subsection *Increasing the playback speed of the video in Falling – Martin Wahlberg* to see implementation details about the video.

Since this playpod brings up many references from internet was another side effect to show a Google search bar at the top of the screen. Any attempts to google results in the popup message “Don’t google”. This was implemented with a text view surrounded by a border to mimic a google search field as much as possible. A click-handler was attached to the text view to show the popup message when it is pressed. The popup message is done with a simple toast notification.

This is all the functionalities implemented while listening on Let’s Google it. The most essential side effect in this playpod takes place when the playpod is finished playing. Then will the user be redirected to a new page that is going to take them through a quiz. The quiz functionalities were implemented as a separate activity with no strings attached to Let’s Google it, i.e. the quiz activity has no dependency to the other activities. This was considered to be the simplest and most efficient solution since also the playpod *High Risk* is going to use the same quiz activity. Read more about the quiz implementation details in the section *Quiz - Daniel Steinwall*.

A timeline describing when the side effects occur in *Let’s Google it* can be seen in Appendix F.

**K. Quiz - Daniel Steinwall**

There are two different quizzes implemented in this project. First, it is the quiz where multiple answers can be correct that will appear after listening to the playpod *Let’s Google it*. The user will after the quiz have the opportunity to view the right answer(s) and see the final score he/she got. Then there is the second quiz where just one option can be selected and this appears after listening to the playpod *High Risk*. In this quiz there are no right and wrong answers. Instead, the answers contain an integer that will be used to calculate the user’s risk score. I will explain the two quizzes in the subsections below.

**Quiz - Let’s Google it:** This quiz activity contains much logic. Also, the quiz runs with some essential differences depending on which activity called it. In case of *Let’s Google it* the quiz will include a timer at the top right corner and a Google search field for each question. Otherwise, in case of any other activity, there will be unlimited time to answer the questions without any Google search field. At the moment there is no other playpod than *Let’s Google it* that is calling this quiz, but it might be in the future and then it is easy to use the quiz. To avoid confusion, I am going to go through the general implementation from the start of the activity until the end (even if the implementation order was slightly different). Then I will explain the implementation details added when the quiz activity is called by the playpod activity *Let’s Google it*. But before diving into the implementation details here is the general layout idea:

It was decided to have the quiz in just one layout format with different widgets. All the widgets are set to the visibility state “gone” in the beginning, i.e. the screen contains nothing visible. It was considered to use the visibility option “gone” over the alternative “invisible” since the latter alternative does not free the space as “gone” does. On every question, the layout contains a clarifying text view to show the current question number among the total number, for example, if it is the third question out of ten, “3/10” will be displayed at the top of the screen. Under this text view is the question itself together with its options. The options are displayed in form of seven checkbox widgets that can be checked and unchecked. It was decided to use checkboxes over the alternative “radio buttons” since radio buttons only support functionality for a single right answer, while in checkboxes you can check as many options as you want. Hence, our quiz activity can manage multiple right answers. In case a question contains less than seven options, the remaining checkboxes will keep their visibility state gone. At the bottom of the layout is a button to continue to the next question or finish the quiz. This quiz phase layout is also used to show the right answers when the quiz is done. The difference is that the checkboxes will be uncheckable and are going to contain a colour depending on the right answer(s) and the user’s answer(s), and there will be a text that says how many points the user got on the question. More on this later. Furthermore, there will be an extra button at the right-answer-phase to leave the quiz activity and return to the home page with the list of the playpods. Before showing the right answers, there will be a page that tells the user how many points he/she got on the quiz out of the total points. To clarify the layout even more, I am going to include some sample pictures from the quiz that happens after listening to the playpod *Let’s Google it* (for now, ignore the Google search bar and the remaining time at the top, those are specific to *Let’s Google it* and will be explained later). Figure 1 illustrates the quiz-phase and figure 2 illustrates the right-answer-phase:

![Figure 1. Showing the fourth question of five in the Google quiz.](image-url)
Figure 2. The left picture demonstrates a case where the user guessed correctly, and the right picture demonstrates a case where the user guessed wrong.

A colour explanation to the pictures in figure 2:
- **Green**: The user guessed right
- **Red**: The user guessed wrong
- **Yellow**: The right answer(s) that the user missed

Score system explanation:
- **Correct answer** (green): 5 points
- **Wrong answer** (red): -3 points
- **Missed answer** (yellow): 0 points

A colour and point explanation will be displayed at the beginning of the right-answer-phase.

**Implementation:**

The quiz activity takes a list of question objects as a parameter. The question objects contain four attributes:
- An id to separate the questions
- A string with the question
- An array of strings with the answers
- An array of integers with the right answer(s)

The quiz activity contains an essential method called “fillForm” which is called every time a new question should be displayed. It takes the question to be displayed as a parameter and fills in the needed widgets. Furthermore, since this method is used in both the quiz-phase and in the right-answer-phase it needs to check the bool variable “quizPhase” to display the correct layout. When a question is replaced by a new question the user’s answer is saved, and this is done by the method “saveAnswer”. It maintains a 2-dimensional array called “quizAnswers”, to keep track of the question and for each question keep track of the selected answers (can be multiple right answers as mentioned before).

When the quiz is finished the score will be calculated and a score page will be shown. The score will first be calculated for each question based on the array “quizAnswers” and the containing question object attributes. The score was considered to be calculated per question since this will be displayed later if the user selects to view the answers. The scores per question are kept in an array called “score”, index 0 of this array means the score of the question with id 1. After calculating the score for each question, the sum of the score array is calculated to get the final total score for the user; This is done in the method “calculateScore”. In the score page, the user has the option to either leave the quiz or the view the right answers. If choosing the latter an explanation page will be displayed, explaining the colors and the point system. After this the questions are going to appear again in the same format but with colors and a text view telling how many points the user got on the present question.

**Extra features in Let’s Google it:**

As can be seen figure 1 there are a Google search field and a countdown timer at the top of the screen. Those widgets are only going to be present if it is the playpod Let’s Google it that started the quiz. The countdown timer will provide a time limit on each question. The remaining time will tick down in second and if the user reaches end of time, the present answer(s) if any will be saved and the next question will be displayed. This timer is started every time a new question is displayed assuming the user still are in the quiz phase. It will be restarted when the user moves on to the next question and stopped when the user finishes the quiz.

A click on the Google search field takes the user to Google on the device’s preferred browser, but the activity remains active in the background. This means that the counter will still count down even if users are trying to google the answers. Every google search gives the user -7 points, therefore are a bool array called “didGoogle” maintained for each question to keep track of the potential google searches. If the user googled on a question it will be shown in the answer that they got minus points because they googled.

**Quiz - High Risk:** The quiz in High Risk has its own activity and layout and do not interact with the quiz described above. The decision to make this quiz as a separate quiz was done because this quiz only has a single “right” answer and that is already supported in the RadioGroup widgets that exist in Xamarin Android. The word “right” is quoted because in this quiz there are no right and wrong answers. Instead, it is a questionnaire with questions and statements where the user can select the option that he/she feels most satisfied with. Hence, this quiz does not contain a right-answer-phase. After the quiz, the user will receive a risk score on SMS if the user allowed it at the app start. If the user did not allow it, the risk score will be sent to the user’s e-mail. The layout is built on the above quiz’s layout, with the difference that it contains radio buttons instead of checkboxes.

Figure 3. Sample from the High Risk quiz, showing the 10th question of 14.

**Implementation:**
This quiz takes a list of question objects as a parameter, just like the multiple-answer-quiz. The difference here is the way the attribute with an array of integers is used. Before it was used to give the right answer(s) of a question. Now it is used to give every answer an integer value that will be evaluated at the end of the quiz and included in the high-risk score. The higher the value is the higher the risk score will be. The risk score is not only based on the quiz, but it will also include the statistics from the step counter used in the playpod FitChip. The lower daily step rate the user has the higher risk score he/she will get. This is included at the end of the quiz in the method “fitchipScore”. The implementation with filling the form is quite similar to the approach in the multiple-answer-quiz so no need to explain it more here. After every question, the risk score from that question is added to the variable called “riskScore” which keeps track of the current risk score. Anyhow, what happens after the quiz is different. Now the user will receive an SMS or e-mail with the risk score. The email functionality was already implemented in the system, so it was just a call to the method “Email” in the “sendEmail” class. SmsManager is used to send a basic text message. The risk score will be given in percent and say something like “You got 78% risk becoming a criminal”.

General comments:

A decision was made to not have a previous-button, so the user can return to the previous question(s). It created extra work an did not feel essential, especially in the quiz for the user can return to the previous question(s). It created extra work an did not feel essential, especially in the quiz for the playpod Let’s Google it since every question has a time limit.

L. Testing

During the project the group members have tested the application continuously to make sure that it works as intended. It is important for the developers to test the application because certain functionality can momentarily be turned off to allow for easier testing. For example, it should not be possible to fast forward the playpods when the side effects are enabled, but we as the developers can disable this functionality to make it more convenient to test the side effects. Users outside the project can also provide useful feedback by using the application in a certain way that the developers did not consider. The devices used for the testing can be seen in the table below. The column ”in development team” shows the devices used by the developers.

<table>
<thead>
<tr>
<th>Device name</th>
<th>API</th>
<th>Android version</th>
<th>In development team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asus Zenpad</td>
<td>24</td>
<td>7.0</td>
<td>x</td>
</tr>
<tr>
<td>Google Pixel</td>
<td>28</td>
<td>9.0</td>
<td>x</td>
</tr>
<tr>
<td>Honor 8</td>
<td>24</td>
<td>7.0</td>
<td>x</td>
</tr>
<tr>
<td>Nokia 8</td>
<td>27</td>
<td>8.1.0</td>
<td></td>
</tr>
<tr>
<td>Samsung Galaxy s7</td>
<td>26</td>
<td>8.0.0</td>
<td>x</td>
</tr>
</tbody>
</table>

Asus Zenpad, Samsung Galaxy s7, and Google Pixel were used during the whole project and the Honor 8 device was used at the end of the project. The device’s with API:s higher than and equal to 26 were the main targets for testing since the application does not consider API:s lower than 26. Hence, there were some odd bugs that appeared in the testing devices with API lower than that. If the same bugs did not appear in the targeted devices, the bugs were simply ignored. But the devices with lower API:s were still useful to try many parts of the application and they made it easier for the developers to test on physical devices instead of the emulator that had its drawbacks itself. It was unfeasible and inconvenient to test some features on the emulator. For example, the step counter in the playpod FitChip was infeasible to test and it was inconvenient to test the reception of emails that were scheduled to be received more than a day after some interaction with the application. Also, the emulator was resource demanding which made it cumbersome to test with it because of the delays. The Nokia 8 device was used by our customer Amanda.

When the project hit its half-time mark, we had a short demonstration for our supervisor Lothar Fritsch where he had the chance to show the current state of the application. After the demonstration, we discussed the possibility to put together a smaller group of test-persons that during an afternoon would test the application to see how it works and provide feedback to the group. While this was a great idea it unfortunately wasn’t realized because we ran out of time.

Fortunately, our customer Amanda Fromell, who was the only that tested our application without knowledge about the code, had the ability to test multiple versions of the application. She provided us with valuable feedback that was used to improve the application.

VII. Problems

This section will cover the major problems that were encountered during the project and also if there was any additions or Removals from the specification.

A. Trigger side effects at certain time stamps

It was noticed early in the project that we had to implement some sort of timestamp trigger that could handle real-time events. I.e. An event handler that could take in a specific time stamp and run desired code when the playpod arrived at that timestamp. We searched on the web for an already existing solution but didn’t find exactly what we were looking for. Instead we decided to write our own event handler that did exactly what we desired. A description about how the event handler is implemented can be found in section VII. While this event handler works fine it is probably not the best implementation since we are continuously running a timer and reacting to its events. While there is probably a better solution for this problem this particular solution works well with what we are trying to do.

B. Animation of control-panel

Making the control-panel have a smooth animation depending on user actions might seem like an easy thing to accomplish since it exists an animation function you could call on different widgets. But there is a lot of different situations that can occur during an animation. E.g. What if
the user pushes a button when the control-panel does a fade-out animation? Or what if the user is idle for a certain time when the control-panel is visible, what happens then? Taking all of the different situations into account took some time but worked out in the end. The animations of the control-panel work in every situation and if we need to change how the animation looks or behaves it is not hard to do.

C. Animating Images

It was originally planned to make the appearance and disappearance of the images in the slide show of drowning to be animated, either by fading, or by increasing the size in such a way that it looked like they came out of the background. A lot of time was spent trying to implement this feature and we tried both using the integrated animation library in Xamarin and by creating our own animation by dynamically altering the attributes of the image views. Our own implementation of the animations worked when they ran in the UI thread, but it did however freeze the entire application when the animation was running. The build in Xamarin library used for the animation of the control panel did not work with the slide show at all. This feature was eventually scrapped due to time constraints.

VIII. Final result

This section will cover what the final product looks like, which side effects from the specification that was included and which wasn’t. There will be one subsection for each playpod.

A. Drowning

Drowning have two major side effects. The first and most noticeable one is the slide show of images. The images are displayed in random locations on the screen in various sizes. Contrary to the original specification the images are not animated in any way. This was due to unexpected issues during the implementation and the animation feature was ultimately scrapped.

The second side effect matched users to YumYum84, in a way familiar to people who have used common dating applications. This matching is through a notification. How the notification behaves in regards to sound, vibration etc. depends on the user’s notification settings. Once the user presses on the notification they are taken into a new view which resembles those you see in common dating applications.

B. Falling

The side effects that was implemented for the Falling playpod corresponds to the side effects that was described in the final specification in section IV-B. These side effects include vibrations that occur on different frequency and with different intensity, polling of high-precision GPS to increase the temperature of the phone, increase of playback speed for the background video and also the scheduling of email that contains Youtube links.

All side effects in the final specification was implemented but the increase of phone temperature side effect had mixed effects on different phones which unfortunately didn’t live up to our expectations.

C. FitChip

The application records the number of steps taken since the application was started for the first time. After the user has listened to the playpod, the application will calculate how many steps the user has taken per day since the app was installed. This result will four days later be emailed to the user with different recommendations depending on the average number of steps. The user interaction during the playpod is working as expected, i.e. the user can have their smart-phone locked in their pocket and it will be automatically started if the user starts walking and then stopped when the user is not walking anymore.

D. HighRisk

During the playpod there will appear some pictures on the display. Then the user is presented with a survey at the end of the playpod. The survey asks for the user’s information such as age, gender, education and living situation. The result is then gathered and a risk score is calculated. The application then, a day later, attempts to send the result to the user through SMS. If the sending of an SMS fails, the risk score will be delivered in email instead. Unfortunately, to send an SMS will fail in most cases because the attempt to get the device’s phone number was harder than expected.

E. Let’s google it!

During the playpod there will be a video of a globe spinning around. The globe will increase its speed as the playpod plays. The video is replayed several times to match the length of the playpod. Unfortunately, there will be some delay when the video is replayed and it will appear to the user like the video is lagging. Then the user is presented with a quiz at the end of the playpod. The quiz is comprised of trivia questions with the integrated option to “Google” the answer. The questions are timed and the user will get a score based on the result of the quiz, with a heavy penalty if the user googled an answer.

F. Trapped

The final result for the implemented side effect is that when the user clicks on Trapped, a map of the world will be displayed on the screen. Clicking on the map the pod starts to play and the map is zoomed to the user’s current position. Two scheduled tasks are also started where the first one will occur every 24 hours retrieving the latitude and longitude coordinates from the devices GPS sensor and storing them in the memory of the device. Second scheduling task will occur every five days sending an email to the user with the list of addresses translated from the coordinates that were retrieved during the last five days. If the user has been on the same address during these five days only one address will be included in the mail and if the device was not able to retrieve the address during this period then the user will not get any mail. Also, in order to receive the email the user may not force quit the app from running processes.
The final result matched quite well with the original planning where the only change was the time when the user will receive the email. Originally the goal was to send an email in the last day of the week which was later changed to five days after listening. Rest of the results were the same as desired from the start.

IX. Discussion

This section covers what the group did well during the project, but also what could have been done different during the project in order to make the implementation-phase easier and the final result better.

A. Code structure

At the end of the project we noticed that we had a lot of duplicate code in our playpod-classes. Since all six of the playpods have a similar core-structure it would have been a lot better to create a single superclass which would contain all code that is mutual between the playpod-classes. Doing this in the beginning of the project would have saved us a lot of time in the end, mainly because in the end when we sat down and did the design, automatic ordering, control-panel layout etc. we have to add the same code to six different places instead of to a single superclass. It would be possible to change the structure of the project but we noticed it at a time where we had to prioritise other things and there simply wasn’t enough time to do such a major change to the codebase.

B. General work execution

In our first meeting we discussed the importance of dividing the work evenly between all group members and also that each member should be aware of what every other member is currently working on. During the entire project the workload was split evenly between every group member in order to make it as fair as possible. Every member was also fully aware about what he should be doing, the deadline and also what the rest of the group worked on. By doing this we avoided the situation where two or more people worked on the same functionality at the same time and could progress through the project without conflicts.

We also discussed the importance of communication inside of the group, especially when it comes to problem solving. By telling the group that there is a problem in the code that couldn’t be solved other group members could take a look at the problem and see if they had any ideas for a solution. By asking other people for their perspective we find a solution for almost every problem quickly.

C. Testing

In this project there were some testing approaches that were omitted. For example, unit testing which is a great way to test different parts and functions of a system. However, since our application is built on user interaction, we felt that it was not appropriate to make use of unit tests. Also, the system behaves differently depending on the smartphone version, thus it felt difficult and complex to introduce automated testing with respect to the different versions. Therefore, does the application not have any automated tests, instead we tested the application manually by using different smartphone version as mentioned in the testing section.

We were planning on having additional testing by letting teachers and friends interact and try out the application and give us their thoughts. It would most likely have been valuable for us with some feedback on the application from some test users. The main reason for us not doing this was because of the lack of time. We were planning on giving out the first release for a user test and when we did that we wanted the application to be almost finished where we as the developers had tested it and stated that the application was running without considerable bugs. But when we got to the point where we had tested the application without noticing serious bugs it was only a few days left of the project and hence we had to skip this testing approach.

References

DYSCONNECT

Hidden behind ones and zeros, a digital revolution is building a new automated society where our lives are becoming increasingly controlled or influenced by computer algorithms. These mathematical formulas supply us with ‘the truth’ through Internet searches, match us with partners, run the stock market, diagnose patients, direct us through telephone cues or read our facial expressions for signs of suspicious behaviour. But how much do these systems control what we do? Are they able to change how we think and act?

Dysconnect explores the themes of encrypted power and digital control, mind-altering systems and distorted identity, through a fusion of PodPlays and digital hacks.

Set between the blurred lines of humans and technology, it is a mirror of modern society where people’s actions are being mapped and movements judged, safe within the boarders of democracy, in the name of profit. The question is: who is driving this automation? And what will it cost us?

Project outline

I’m looking for a group of students who are interested in working on an artist project using a smartphone app to create gamified, interactive audio drama pieces. The app will enable the listener to access a series of short audio drama recordings (‘PodPlays’), while playing a simple video/series of images, dealing with some aspect of algorithmic control (e.g. online surveillance). Behind the appearance of an audio player, the app will generate various “side effects” related to the PodPlays and the listener’s own location and/or online profile. For example, after listening to a PodPlay about location tracking, the user may be sent a text message suggesting what they should wear, depending on the
weather specific to their location. The app will involve a kind of artistic hacking that mirrors and makes use of commercial practices, but with the aim of generating critical awareness, giving the audience a direct experience of the power of algorithms in everyday life.

**Ideas about the general interface/movement between PodPlays.**

The app will communicate with the user through speech bubbles, to which the user will respond by tapping on the screen or on given options. Each PodPlay will be activated by such an interaction.

**Further background information on the PodPlay art project**

The project is supported by the Arts and Humanities Research Council (AHRC), UK; Curve Theatre, Leicester; and Making Space Arts, London.

The PlayPod project is developed by Amanda Fromell, an AHRC PhD student at the department of Drama and Theatre Arts, University of Birmingham, UK. She also works as a writer and dramaturg in theatre, dance and TV, making work at the intersection of art and technology. To learn more about her work, you can visit her LinkedIn page at: https://www.linkedin.com/in/amanda-fromell-473482149/.

Recordings of the finished PodPlays can be accessed and listen to via a pending invitation to Dropbox.

In terms of the future of the project, we hope to tour the app around theatre festivals in the UK and Europe, before investigating the possibility of making it available for purchase to a wider audience. In fact, we have an invitation to present the work at Dundee Rep's New Writing Festival (Scotland) in April 2019, which we can discuss further.

**Synopsis of each PodPlay and starting points for side-effects/visuals.**

**High Risk**

*Synopsis*

The PodPlay *High Risk* depicts possible dangers of criminal justice algorithms. Set in a dystopian future, the character, Emma, is in hospital following the birth of her daughter. Still in the maternity ward, she is visited by Officer 1 and 2. Initially, the conversation is cordial. Soon, however, it becomes increasingly hostile, as it is revealed that the officers are there because criminal justice...
algorithms have alerted them to the fact that Emma’s baby daughter is running the risk of becoming a criminal.

This idea was inspired by work carried out by Richard Berk, Professor of Criminology and Statistics at University of Pennsylvania. Berk is developing an algorithm that he claims would be able to predict, from birth, the likelihood of a person committing a crime by the time he or she turns 18 (Brustein, 2016).

**Suggested Side Effect (to be discussed/developed)**

Idea: After listening to High Risk, the listener receives a text message, detailing a risk score using a classification algorithm. This is a highly subjective risk score compiled of statistics collected from their phone (such as age, gender, location, occupation) steering towards the absurd. Imagine that it says something like: you have been categorised as xxx... We strongly suggest that you change xxx. The idea would be to present something almost comical/absurd – reflecting the precarious nature of prediction algorithms, how they force people into inadequate categories. Of course, we would need to consider what type of information we are able to access legally etc... Something to brainstorm 😊

**Visuals**

The background sounds in the PodPlay follows an app played by Emma, which contains sounds that are designed to make babies sleep. Visuals follow the background sounds – an image of a washing machine, an image of a busy road, rain, heartbeat, etc. perhaps there is an interface in-between, when Emma changes that sound.

**FitChip**

**Synopsis**

Sarah is being pressured by her mother, Kim, and Kim’s colleague, Sharon, into implementing a ‘FitChip’. A FitChip is a fictionalised version of the wearable and mobile activity tracker ‘Fitbit’, which ‘constantly measures the acceleration of your body and algorithms convert this raw data into useful information about your daily life, such as calories burned, steps, distance and sleep quality’ (Fitbit Inc., 2014). Rather than a wearable fitness tracker, the technology has, in a dystopian twist, morphed into an implant, a ‘fitchip’, which is placed directly into the body.

**Side effect (to be discussed/developed)**

Idea: The side effect is an account, sent via text message, taken from the time when the listener downloads the app, counting how many steps the listener takes per day, generating an average step count. Also think it would be fun if we could make a competition between the different listeners so that it says something like – compared to our other listeners, this puts you in ‘the bottom 10 per cent’...
Trapped

Trapped explores the questions: what happens when our smart phones become part of who we are, our interactions, our thoughts and our way of living? Will our sense of self diminish when we run out of battery? Will we be unable to think without devices mediating our thoughts?

Set in a dystopian future, it depicts a conversation between the two colleagues, Simon and Anna. First, we think they are having a friendly conversation. Soon, however, it transpires that Anna interrogates Simon, after he lost reception for days. What is he hiding?

Side Effect (to be discussed/developed)

Idea: Maybe the visual of the app, as in what the listener is looking at while listening, is the live picture of a map of themselves. A blinking dot, showing that the app has the ability to track them.

Drowning

Synopsis

Ali is searching online for information regarding how one might be able to trick Google’s search algorithms, through a process of drowning out negative information with new positive information. The character, Izzy, gives answers to
Ali’s questions, while simultaneously attempting to sell him a lifestyle that will make him happier. As implied by the title *Drowning*, Ali’s original question is eventually drowned out by Izzy’s ability to predict what he likes, through a process of collecting and mapping his user pattern and then selling these services/products back to him. Finally, Ali’s original intention is forgotten and he acts in line with the suggestions made by the algorithm.

**Side Effect**

After listening, the listener receives a Facebook message from the character YumYum84, who appears in the PodPlay. I would create a profile for this character, who would then interact with the listener on Facebook, posting info about/debating algorithms. If they don’t have a Facebook account – maybe they could be sent something that tells them about what they are missing out... Using the same tactic as that used by companies to connect. The tricky part would be to automate such an invitation...

**Visuals**

Idea: A stream of Instagram-like photos and/or dating profiles, perhaps moving as if someone is scrolling through them. Drowning the listener in images of cravings maybe…? Everything from food to body... in sync, or not, with the audio.

**Falling**

**Synopsis**

The two cousins, Alec and Joe, are having a conversation in a pub about the risks of financial (automated) trading. Soon, the dialogue begins to loop towards a crash, mimicking the structure of an event commonly referred to as ‘The Flash Crash’.

On May 6th, 2010, the US stock market faced the fastest and second-largest percentage-point price decrease in the history of the Dow Jones (Millo and Beunza, 2015). Its losses were close to 862 billion dollars, before it suddenly began to recover, automatically, rebounding within the space of 20 minutes (SEC and CFTC, 2010). One trader described the experience like ‘watching someone get run over by a car’, only to minutes later see the very same person get up and walk away, unharmed (Meerman, 2011). The exact reasons behind the crash remain inclusive, but the speed with which the crash happened indicated a problem with the algorithms set to automatically buy and sell stocks, where one triggered another in a never ending feedback loop.

**Side Effect/Dysconnect**

Idea: Perhaps this is an interphase interaction rather than a side effect. I was thinking something along the lines of speed. Perhaps a page of information about financial trading appears, with a clock ticking down the time the user has to read it. When the time is up, they are asked a question about the content. This moves on to another page, each time the clock ticks faster and faster. It’s an unbeatable circle of speed – drawing on the ideas developed in the PodPlay – the possible danger of speed and how it can affect us.
Visual
I envision a mix between the image of a wet floor with a sign indicating wet floor slippage and a moving graph. Either the graph shows a heart monitor - allowing the ‘cardiac arrest/crash’ moment in the script to happen on screen as a ‘heart attack’, only to reboot as the script ‘reboots’. Alternatively, this could be a replication of a moving graph of the Flash Crash.

Let’s Google it!

Synopsis
Let’s Google it! follows an algorithmic logic, highlighting the often absurd, bias or even racist answers produced by the Google autocomplete algorithm. Set at a fancy dress party, the PodPlay coveys a conversation between the characters Kay, Helen and Stanley. The conversation becomes increasingly absurd as they rely more and more on Google in order to answer their questions and fill their
uncomfortable silences with content.

_Side Effect (to be discussed/developed)_

Idea: Ideally something to do with people’s Google history, but I understand this may be difficult to access... Could we get around this by asking their consent in ‘terms and conditions’? Something interesting here could be to use an idea from ‘Google poetics’ – basically where you take autocomplete and use it to make a poem. I wonder if we could do something similar with peoples search histories and email it to them.

![Google Autocomplete Screenshot](https://www.googlepoetics.com)

**Visual**

Idea 1: maybe allow the Google auto complete to generate images prompted by the text (kind of mimicking the content). We’d need to find images that we’d be allowed to use, but the idea could be to create a bombardment of what Google can produce – kind of image overflow (when Kay Googles ‘bald white men glasses’ that is what we see, when they Google MacDonals, we see burgers etc.) Maybe they appear in a similar format to Google images? They don’t need to be in sync – actually maybe it’s better if they aren’t...

Idea 2: Something completely different – like the live image of a jungle, to give a contrast between nature/technology.

_In addition to these six, I am currently working together with a programmer to develop two more, Connected and Safe. We can talk about these as well, how they relate to the project and how (if at all) we may integrate them within the same app._
Appendix C - Example of email from Falling

playpodproject@gmail.com
Fri 2018-11-30, 12:08

Hi! I believe that you should have more knowledge about the content in the playpod Falling. I know that you have listened to it and I'm therefore going to give you a YouTube link that you can watch.

https://www.youtube.com/watch?v=skFD18xQDHA

The Wall Street Code - VPRO documentary - 2013

A thriller about a genius algorithm builder who dared to stand up against Wall Street, Haim Bodis, aka The Algo Arms Dealer. After Quants: the Alchemists of Wall...

www.youtube.com

Best regards /Playpod
Appendix D - Falling timeline

0:00  Falling starts

2:01  Vibrations in synch with heartbeats starts

3:16  Vibrations in synch with heartbeats stops

6:00  Increase of phone temperature starts

7:05  Vibrations in synch with heart-monitor sounds start

7:05  Increase playback speed of video to 5x

7:24  Increase playback speed of video to 7x

7:32  Increase playback speed of video to 10x

7:38  Increase of phone temperature stops

7:38  Vibrations in synch with heart-monitor sounds stop

7:38  Decrease playback speed of video to 1x

9:28  Falling stops

9:28  Emails are created and scheduled
Appendix E - Visual representation of skeleton functionality
Appendix F - Let's google it timeline

0:00  
Let's Google it starts

1:00  
Increase playback speed of video to 2x

2:00  
Increase playback speed of video to 3x

3:00  
Increase playback speed of video to 4x

4:00  
Increase playback speed of video to 5x

4:55  
Let's google it stops

Quiz appears