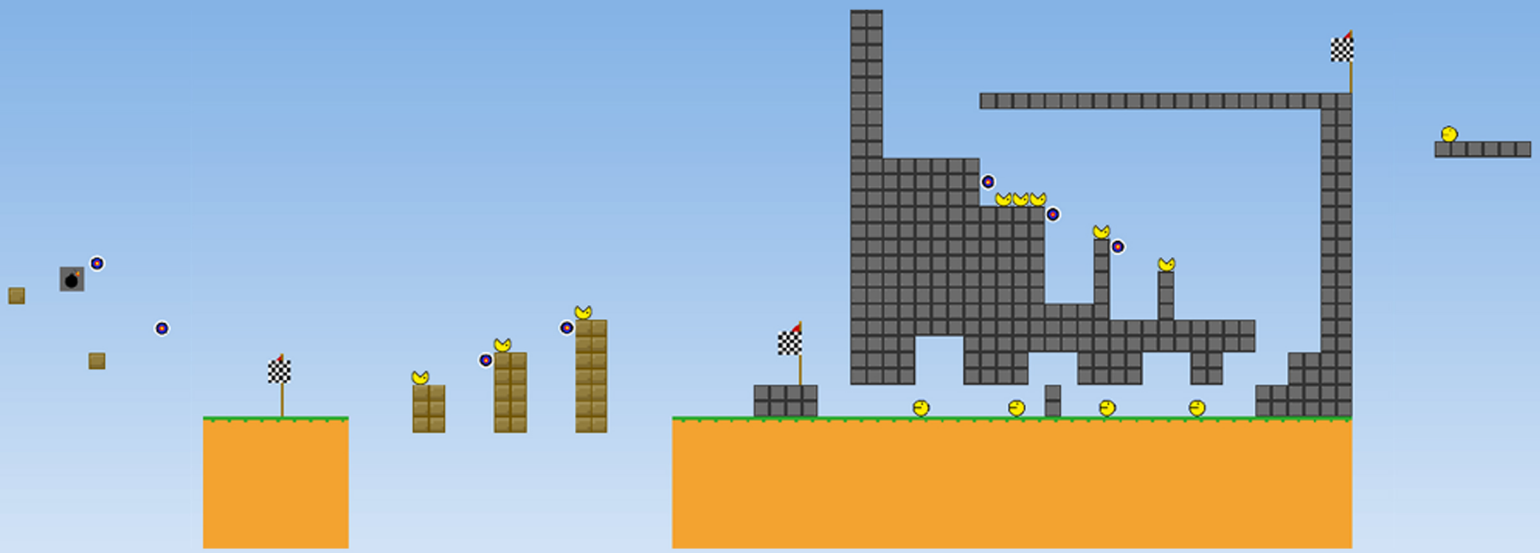
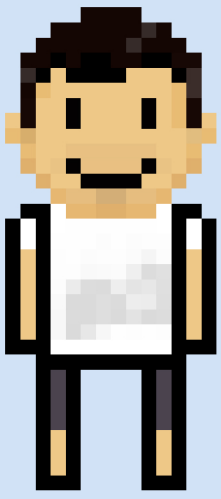
RESEARCHING INTERFACES FOR PLATFORM GAMES ON TOUCHSCREEN DEVICES





CHRISTIAN S. KRAGEGAARD

KIM S. KROG

MEDIALOGY

MASTER THESIS

AALBORG UNIVERSITY

SPRING 2013

AALBORG UNIVERSITET

Medialogi 10th Semester
Medialogi - Aalborg
Sofiendalsvej 9
DK-9000 Aalborg
Tlf. 99 40 24 84
Fax 99 40 97 88
www.sict.aau.dk

Titel: Researching Interfaces for Platform Games on Touchscreen Devices

Theme: Master Thesis
Project Period: 04.02.2012 - 06.06.2012

Project Group: mta131031
Authors:

Christian Skriver Kragegaard

Kim Srirat Krog

Prints: 2
Pages: 78
Portfolio: DVD

Synopsis:

The main focus of this thesis is to research the problem with interfaces that are directly converted from physical controllers, and what impact this has on users' performance in 2D platform games in relation to accuracy, error rates and stress.
Two initial tests are conducted; the first in order to research users' accuracy between a physical controller and a directly converted to touch interface during simple tasks, and the second in order to research the difference in errors between physical and touch interface during rapid repeated actions.
Three new interfaces that included the use of vibration, gestures and a mix of virtual buttons and gestures was designed from knowledge gained from the two initial tests. These were tried and tested in an advanced game prototype and compared to a directly converted interface in order to find any performance differences. Lastly the effect of repeated actions on a physical and directly converted interface in relation to stress was researched.
These tests showed that there was no difference between the two interfaces during simple tasks. On touch interfaces users made more errors and made them faster than on the physical controller when performing rapid repeated actions. The results from the game prototype showed no significant improvement over the directly converted interface. There was also no significance difference in relation to stress during rapid repeated actions.

Chapter 1

Preface

This report is a result of a 10th semester Medialogy project by group mta131031 at Aalborg University spring 2013.

1.1 Reader's guide

The report is divided into several sections that explains the process up to the final test.

1.2 Reference Style

The sources in this report are references using square brackets containing the surname of the author(s) and the publication year. If the reference is placed before a period, it means that only the phrase itself is based on the source from the reference. If the reference is placed after a period, the whole section above is based on the source. All reference can be found in alphabetical order at the end of the report.

1.3 DVD Content

The DVD contains the following:

- Game prototype and source files
- Documentation (Report, product movie and questionnaire answers)

1 Contents

Chapter 2

Introduction

Games on smartphones are massively popular and are today a multi-million dollar business. With over 800.000 application in just one of the biggest online application store the App Store by Apple [2013] and with nearly 17% of downloads being games, the games business has a major impact on the smartphone market.

Researching the smartphone games market in relation to 2D-platform games [Wikipedia, 2013a] it does, however seem that game developers often neglect the fact that they are developing for a touchscreen and not a physical game controller with buttons. This paradigm of converting the controls of the physical controller directly to a touch interface control scheme, might therefore often distrupt the gaming experience on touchscreens.
The main theme of this thesis is therefore to study the problems with interfaces directly converted from a physical controller, and what problems this can create. The differences of a physical controller and a touchscreen, what errors users make and how accurate users are will therefore be researched throughout this thesis. The motivation for this research is to see if we can find any methods to improve touchscreen interfaces in relation to games on smartphones. These methods could in turn have benefits for developers of smartphone games.

This study will start in Chapter 3 where we will research and discuss the basic concepts of 2D platform games on touchscreens. In Chapter 4 we will present the project formulation, delimitations and statement. In Chapter 5 we will provide device specifications and make a delay test in relation to the devices used. In Chapter 6 we will research the differences in accuracy between a touchscreen and a physical controller when performing simple tasks. In Chapter 7 the differences in errors between a touchscreen interface and a physical controller when performing rapid repeated actions will be researched. In Chapter 8 we will use the knowledge gained from Chapter 6 and 7 and use it to design new interfaces and a game prototype will be created. In Chapter 9 the newly designed interfaces will be tried and tested in the prototype. In Chapter 10 we will research the effects of repeated actions in relation to stress. Lastly Chapter 11 will conclude this thesis with a discussion of the collected results.

Chapter 3

Problem analysis

In this chapter we will analyse the problems associated with virtual interfaces on touchscreens, which seems to have been directly mapped from a physical controller. The current market of mobile games will be analysed to give insight on the frequency of directly mapped interfaces and will look into research that have relevance to this issue.

3.1 Market Analysis

3.1.1 Platform Game History

2D platform games [Wikipedia, 2013a] dates back to the 1980s with titles like Donkey Kong, Bubble Bobble and Mario Bros. Games in this genre generally had a static playing field, where the player was confined to run, climb, jump and accomplish tasks within a single screen.
In the following years side-scrolling video games was invented. In these games the player views the game from the side and must with his character generally move from the left of the screen to the right to encounter the games challenges. The early platform games were mainly played in arcades on stationary machines with a stick control and buttons to do various actions. Later on home gaming consoles became very popular, for example with the Nintendo Entertainment System or the Sony Playstation, where the games were played with a controller containing a D-pad (directional pad) and a set of buttons, see Figure 3.1.

Figure 3.1: Sony PlayStation 3 Controller [Wikipedia, 2013c].

Among the most famous side-scrolling games is the game Super Mario Bros., which has become the second best-selling game in the world with 40.24 million copies sold [Wikipedia, 2013d]. Since the eighties 2D side-scrollers has been renewed several times and is still a very popular game genre on the market, for example with titles like Limbo [Develop-Online.net, 2013] and Super Meat Boy [ComputerAndVideoGames.com, 2013]. Though the genre has in some ways been reinvented the general principles in the game style remains the same, where the player has to control his avatar by moving left or right, jump and shoot to overcome obstacles and complete tasks to reach the goal of the game.

3.1.2 Smartphone and Application Market Shares

Gaming on mobile devices is a relatively new form of entertainment, but it is an industry that has evidently come to stay with its massive growth during the past decade. Increased accessibility and computing power of mobile devices along with a more structured developers community, has made making games a relatively simple, but potentially very lucrative job.

Software for today's mobile market is highly dominated by Apple and Android, and these two companies will in the future have an even higher market share with an estimated 81% by 2015 combined. [FutureBooks, 2013]

There is a tendency for consumers in these virtual stores to mainly buy games. With an application count of nearly 800.000 on Apple's App Store and 16.78% of all downloads being in the games category, with the runner up being educational applications with 10.64%, the games industry could be said to have a great impact on the mobile market. [148apps, 2013]

A reason why games on mobile devices are so popular today could be that games are far more accessible than before mobile gaming became so widespread. In a study by Cunningham et al. [2008] where people were asked "what is the main reason you don't play video games?" 40% of all asked answered "I don't have time". Mobile devices have already come a long way since 2008 and there is today over 1 billion smartphone users worldwide [CBSnews, 2013]. This could mean that instead of having to turn on the console or pc to play a game, users can just reach into their pockets, even when travelling, and begin playing on the smartphone as it is always carried around.

3.1.3 Platform Games on Touchscreen Devices

As touch interfaces has become increasingly popular and has overtaken some of the console games market share, it would be interesting to see how well players perform in a 2D platform game controlled with a D-pad controller compared to controls on a touchscreen.
In order to find the state of the art within the field of controls for the 2D platform games genre, an analysis of the existing market will be carried out.
As we in this thesis focus on 2D platform games, restrictions for the games that will be analysed must be set. The game should therefore:

- Be a side scrolling platform game.
- Have similar controls to the D-pad.
- Require the player to jump, move left and right and to use an action.

When browsing the two most popular mobile phone application markets, Apple's AppStore and Google's Android Market., a few 2D platform game titles that meet these criteria are found in the top 100 of the games categories; these are "Cordy 2" [GooglePlay, 2013] and "Lep's World" [NerByte GmbH, 2013].

Title 27/2	Cordy 2	Lep's World 2
Game Type	2D Platform	2D Platform
Control Type	L, R, Jump, (action)	L, R, Jump, shoot
Dimensions	2.5D	2.5D
Vibration	No	No
Release Date	Feb 7, 2013	Sep 21, 2012
Ranking (iOS / Android)	21 / 22	14 / 29
Rating (iOS / Android)	4 (27) / 4.2 (3,039)	4.5 (155) / 4.6 (20,129)
Downloads	1-5m	1-5m

Figure 3.2: Table displaying atrributes and information about Cordy 2 and Lep's World 2 [GooglePlay, 2013] [NerByte GmbH, 2013].

What these two games have in common is that they are both sidescrolling platform games where the player has to jump to and from platforms, collect coins and defeat the enemies. The player views the game from a side perspective and though both games have some depth perspective this does not affect the game play in any way.

There are four main controls in **Cordy 2**; Left and right movement button, jump button and an action button to either teleport, grab onto selected things or boost his jetpack according to which level the player is in. As seen in Figure 3.3 the left and right movement buttons are with a slight displacement grouped together in the bottom left of the screen, and the action and jump buttons are with a slight displacement grouped together in the bottom right of the screen. This means that the player must use his left thumb to activate the movement buttons and his right thumb to activate the action and jump buttons. When the player presses one of the virtual buttons it will light up to indicate that the button has been pressed. When the buttons are not pressed they

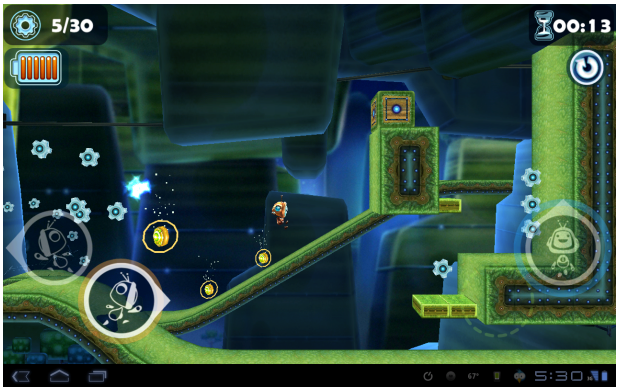


Figure 3.3: Screenshot of Cordy 2 [Androidtapp.com, 2013].

are semi-transparent to allow the player to see some of the level through the buttons. In Figure 3.3 the player is currently pressing the right movement button. Though this gives the user some feedback to which button is pressed, one must keep in mind that the buttons could be entirely occluded due to the fact that the player is covering this part of the screen with his finger when pressing the button.

When holding the left or the right movement button down the character moves accordingly until he is obstructed by a wall. By pressing the jump button the character will jump once, and by lifting the finger and pressing once more while the character is in the air he will jump once more though still in midair.

Figure 3.4: Screenshot of Lep's World 2 [Freeapps.ws, 2013].

In **Lep's World 2** the controls are like in Cordy 2 grouped together to be pressed with the left and the right thumb. While the left and right movement buttons are aligned horizontally, the jump and shoot button in the right side of the screen are with a slight displacement placed above one another, see Figure 3.4. When pressing one of the buttons it turns green to give the user feedback. When the buttons are not pressed they are, as in Cordy 2, semi-transparent in order to give the

player a better view of the level. When holding down for example the right movement button the character will continue to move in the right direction until hitting an obstacle. When pressing the jump button swiftly the character will do a small jump, but holding the button down the character will continue upwards until a certain point. Double jumping is however also possible, as long as the character is not at his maximum jump height.

As the gameplay of Cordy 2 and Lep's World 2 are very similar, so is the actions required to complete level. Six actions required to successfully play these two games is displayed in Figure 3.5:

#	Action	Button(s) pressed	Thumb movement
1	Move right	R	Left
2	Move right then left	R + L	Left then Left on different button
3	Move right and jump	R + J	Left + Right simultaneously
4	Jump and move right	J + R	Right + Left simultaneously
5	Double Jump	J + J	Right then Right same button
6	Move right, jump and shoot	R + J + A	Left simultaneously with Right then Right on different button

Figure 3.5: Table displaying the six actions required to successfully play Cordy 2 and Lep's world 2.

3.1.4 Games Specifically Designed For Touch Devices

Some game developers have tried to overcome the D-pad converted control paradigm, and have come up with different solutions to this problem. Some of the titles where the controls are fitted the touch interface are titles like "Monsters Inc. Run" [iTunes, 2013b] and "Jetpack Joyride" [iTunes, 2013a].

Title 27/2	Monsters Inc. Run	Jetpack Joyride
Game Type	2D Platform	2D Platform
Control Type	Touch to jump	Touch to fly
Dimensions	2.5D	2D
Vibration	No	No
Release Date	Dec 13, 2012	Sep 1, 2011
Ranking (iOS / Android)	4 / -	31 / 10
Rating (iOS / Android)	4 (50) / -	4.5 (4,539) / 4.6 (204,991)
Downloads (Android)	-	10-50m

Figure 3.6: Table displaying atrributes and information about Monsters Inc Run and Jetpack Joyride.

(a) Monsters Inc Run (b) Jetpack Joyride

Figure 3.7: a) Screenshot of Monsters Inc Run [iTunes, 2013b], b) Screenshot of Jetpack Joyride [iTunes, 2013a].

What these two games have in common is that the stage is automatically scrolling, meaning that the character is perceived to be moving to the right, or that you follow the character with smooth pursuit eye movement [Wikipedia, 2013b]. The character will therefore always stay in the left side of the screen and it is the stage itself that is moving, as if it was an infinite treadmill. As the character moves by itself, the player is therefore denied one degree of freedom, meaning that he cannot decide at what pace he want to move through the level or move backwards to explore a previous part of the level. This does however also mean that the player does not have to worry about pressing several buttons at once, as the only thing he will have to do is to jump to avoid falling into pits (see Figure 3.7a) or boost his jetpack to avoid being electrified or hit by rockets (see Figure 3.7b). Both games use almost the whole screen as controller, except the small pause button in the top right of the screen that brings the menu forward and the boost button to the left of the pause button in Monsters Inc. Run. This means that the player do not have to worry about hitting small regional buttons and can press wherever he want to make the character jump or fly.

3.2 Background Research

The market analysis established that there are a lot of games that converts the physical control paradigm directly to touch devices. The following section will discuss the pros and cons about converting physical controls to touchscreens. It will also discuss related works and measurement tools.

3.2.1 Advantages and disadvantages of Touchscreens

Advantages of Touchscreens: The advantages of a touch device is that there is no need to dedicate physical space for any buttons. The screen is the interactive space so it allows more of the device's surface to act as display instead of sacrificing space for physical buttons. Touchscreens also allow for gestures to be used, which can give a lot of interaction possibilities, unlike physical buttons that can only use the dedicated buttons or a combination thereof [Hynninen, 2012]. An example of a touch gesture could be a swiping motion in an upwards direction done by the finger across the screen, which in this case could be mapped to be the same as using the 'up' button on a controller.

Touchscreens are also easier for novice users to learn, as they do not have to use physical controls for manipulating the screen, but operates directly on the screen without looking at the controls [Albinsson and Zhai, 2003]. This eases the learning curve (e.g in games) and users might be more compelled to try games, where they might have turned it down because of the a seemingly harder learning curve on a physical controller.

Disadvantages of Touchscreens: When using the fingers to manipulate the user interface, some of the screen will be blocked. This is called occlusion [Albinsson and Zhai, 2003] and can be a problem in some games if you have to look forward in a level and the finger occludes an important part. This could also cause problems in games that has a few touch buttons to use, where the user can not be sure if he presses the right button if the finger occludes the area. This problem can also be caused by of the lack of tactile feedback [Hoggan et al., 2008]. On a physical controller the user is able to feel the buttons constantly, whereas on a touchscreen the user is only able to feel the screen. On touchscreens, the user can only receive aural, visual, or haptic feedback (phone vibration) in order to know if the button was pressed.

Small targets can also be a problem if the users finger is bigger than the target (also called the "fat finger"-problem) [Wigdor and Wixon, 2011] and several techniques have been developed to counter this problem. One example is back-of-device interaction [Baudisch and Chu, 2006] that enables touching the device from the back, enabling users to see where they are touching by a pointer on the screen. Another example is the shift technique [Vogel and Baudisch, 2007] that will display the content from under the user's finger next to the finger, but only if he remains in the same area for a short amount of time. However it seems like these techniques have not been made with games in mind, but for everyday application uses.

3.2.2 Controls setup

The touch controls for this project is set up as close to the PS3 controller as possible (by spacing and size) in order to make them comparable as can be seen in Figure 3.8. Though the controls is very similar this way, there is a couple of tradeoffs that must be kept in mind.

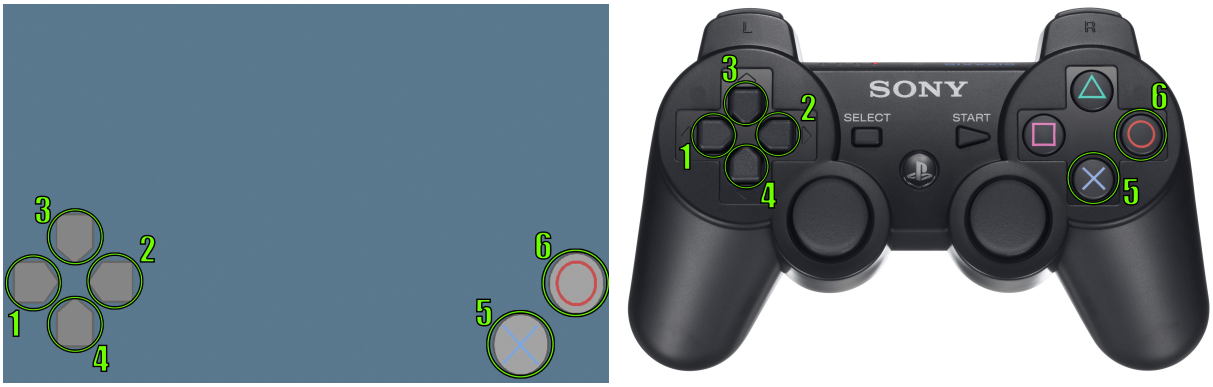
With controls on the touchscreen the user will occlude part of the screen with his fingers when interaction with it. Furthermore there is no physical feedback as on the physical controller. However, the positive with the touchscreen is that the controls are in the same direction as the users field of vision and that the reaction from hitting the button is instant unlike a physical button that needs to move a distance in order to be activated. With the physical controller the user often has his attention to the screen, away from the controls, so the controls must be remembered or there must be occasional gazes to the controls. The positive with the physical controller is, however, that the user at no point occludes parts of the screen with his finger and that he can at all times feel where the buttons are located.

Touchscreen:

- Cons:
  - Occlusion of screen
  - No physical feedback
- Pros:
  - Instant reaction when hitting the screen
  - Controls are in the same direction as the field of vision

PS3 Controller:

- Cons:
  - Delay from the button is touched to the spring is down at registration point
  - Controls not in the direction of the screen - User has to remember controls
- Pros:
  - Physical buttons - User can feel the buttons underneath his fingers
  - No occlusion of screen



(a) The touch controls

(b) The PS3 controller

Figure 3.8: The controls for the Touchscreen and the Physical controller. Overlay to indicate button function has been added to the illustrations; (1) Left, (2) right, (3) Up, (4) Down, (5) Jump and (6) Shoot.

3.2.3 Related Work

There have been several experiments that has studied physical buttons compared to using a touchscreen. An experiment conducted by Hoggan et al. [2008] found that using tactile feedback (phone vibrations) can significantly increase the fingertip interaction and the speed of entering phrases when writing on a touchscreen keyboard. From the market analysis it seems as if games made for touchscreens does not use vibration on buttons. Vibrations will therefore be tested to see if it can make a difference in performance when playing a mobile game. Obrist et al. [2013] conducted a study to learn about the intensities of a 16 Hz vibration and a 250 Hz vibration. The 250 Hz showed to have a strong tactile effect, which can possibly help in giving the user feedback when touching the buttons. A 250 Hz vibration should be therefore be used if possible.

A few studies have also researched players game performance, with the same game using a touch device and a physical controller. Zaman et al. [2010] made a study where they tested the same game using an iPhone 3G and a Nintendo DS. Their study showed that all subjects preferred the physical controls on the Nintendo DS and the performance in terms of death and level completion time was higher on the iPhone. However the authors does not state in their qualitative results whether the participants (n = 12) had used an iPhone or Nintendo DS before and did not control for experience with either device, which could have an impact on the results. It does also not appear that the authors included ratings (such as on a scale from 1 to 10) about the different interfaces (They just asked which they prefered), neither does it appear that prior game experience (e.g hours played per week) was taken into account.

In a study by Hynninen [2012] he states:

> *It is clear that touchscreen devices offer increased usability especially when the target size is bigger than the user's finger.*

This indicates that if the interface is directly convert from the physical controller, you must make sure the virtual buttons are larger than the finger. This makes sense in that the user will have less chance of hitting outside the button area. However with several buttons stacked next to each other, it might not always be possible to have a large collision area for a button. Controls that are spaced close together (left, right and sometimes up and down) might require the interface to automatically adapt to user's fingers in a specified area instead. The direction could then be relative to the starting point of the finger. He notes, after looking at how different participants performed trying to acquire and follow targets in a few *First Person Shooter* games for an iPod:

> *Especially target leading and acquisition tasks were difficult to complete. Judging by the results, mapping PC or gamepad controls to a virtual touchscreen does not seem to be a good fit.*

He states that the solution would be for developers to try and design for the touch device and use the device's strongest points, which he suggests is by using gestures. Albinsson and Zhai [2003] also note that:

> *The zero displacement between input and output, control and feedback, hand action and eye gaze, makes touchscreens very intuitive to use, particulary for novice users.*

It follows then that it could be interesting to see if gestures can perform better in a game than a directly converted interface.

Claypool and Claypool [2007] studied the effect of different frames per second (FPS) in First Person Shooters. They noted that First Person Shooters that required precise, rapid response (such as shooting) was severely worse (with performance measured in targets killed and player deaths) with FPS less than 15. They also noted that a 60 FPS provides seven times increase in shooting performance (kills and deaths) compared to 3 FPS. We assume that the mobile device used will run atleast 60 FPS, but that the framerate will be saved in order to see if any errors should occur that might have affected the controls during play.
Another thing that must be considered is if the physical controller and touch device differs in delay from pressing a button on either device. Liukkonenl [2009] states that receiving light to the eyes and then process it takes about 190-215 milliseconds for light stimuli. Thus it must be checked if the devices are comparable in delay for pressing buttons.

3.2.4 Researching Measurement Tools

Before performing any test it is important to establish if there is any existing measurement tools available that could be useful for measuring, recording and analysing the different dimensions of data that would be obtained from the tests. The tools reseached is Fitt's Law and GOMS, which is both used for analysing human movement in relation to human-computer interaction.

Fitt's Law
What Fitt's Law in short terms does is to by a logarithmic function (D/W) measure humans perform movements over different distances (D) to targets of different sizes (W) combined with the movement time (MT). This is known as Fitt's Law and is formulated as: $MT = a + b\log_2(2D/W)$, where a and b are empirical constants [Bertucco et al., 2013]. Fitt's Law can be a very useful tool when having to measure simple and consistent tasks, but is not found suitable for game measurements as several variables, such as the players initial finger position and moving game objects, constantly change, as the inputs in this formula should be uniform successive responses [Fitts, 1992]. As this could potentially give an unwanted noisy data output Fitt's Law is opted out as a measurement tool in this project.

GOMS
GOMS (Goals, Operators, Methods, and Selection rules) is a model for observing and analysing human-computer interaction, by restricting the user's interaction with a computer to its elementary and core actions [Card et al., 1983]. Restricting the user's actions does that a particular interaction and its time measurement can be easily calculated, but its does however render this measurement tool unsuitable in relation to games as GOMS do not take user unpredictability into account. There are several GOMS models, but none of these allow for any type of error and should therefore only be used with expert users [Sharp et al., 2007]. As games are often highly unpredictable and allows for several interaction methods GOMS is therefore unsuitable for this project.

Physiological Measures
One of the things that seem to be missing from touchscreen research is the physiological aspects, e.g. the measure of stress and the impact it has on using a touchscreen or a physical controller. An interesting aspect would then be if the users are more relaxed physical, because of this. The stress might also relate to the score the users receive in a game. A way of measuring relaxation would be using a heartbeat monitor and a measure called RMSSD (Difference between adjacent Inter Beat Interval) [Pradhan and Islam, 2010]. Relaxation can be measured as having a specific pattern in a heart rate graph [McCraty, 2006].

3.3 Gap of Knowledge

Different research have been presented that touches to the topic of touchscreens and the comparison to a physical controller. However, there seems to be a lot of gaps in the research that have not been explored. The following will first summarize what is known from related work and then present the gaps in these works and furthermore present what we want to study in this project.

Summary of Related Work:
Zaman et al. [2010] stated that the physical device (a Nintendo DS in this case) was preferred by the users and that it outperformed the touch device in performance (in terms of deaths and time to complete). However, there was no data from the participants in terms of ratings of the devices or to experience with the devices. An experiment by Hoggan et al. [2008] showed that vibrations improves the standard touch keyboard on a touch device. However, it seems as if vibration have not been tried on buttons in games to see if it can improve directly mapped buttons as well.
Hynninen [2012] stated that for a FPS game on a touch device, the target leading (following a target) and target acquisition was difficult when buttons had been directly converted. Moreover he suggested that controls should be built using the touch device's strongest points, suggesting that using gestures would be feasible. However he did not get into how one would use this for improving the directly converted interface.

Moreover it does not appear that there is research on the difference between physical controls and an interface that has been directly applied to the touchscreen, performing in simple tasks. Neither does it appear that continuous repeated actions and how fast users make errors on a touchscreen compared to a physical counterpart have been tested. It does not appear that there is research for trying to improve a directly converted interface, in terms of trying gestures or vibration as mentioned earlier. Finally, we assume that a touch and physical device might have an effect on how stressed the user is when playing. There might also be other unknown physiological reasons that the physical controls outperform the touch controls in most cases, but there seems to be no research on testing this difference with psychological data. These gaps of knowledge will therefore be researched throughout the report.

Gaps:
The following will describe the fields that will be researched in this project:

- **Test 1 (Chapter 6):** The differences in accuracy between a physical controller and a directly converted interface during simple tasks will be researched.
- **Test 2 (Chapter 7):** The difference in errors and time to first error between a physical controller and a directly converted interface during rapid repeated actions will be researched.
- **Test 3 (Chapter 9):** The effects of using interfaces with vibrations, gestures and a mix of these in a game compared to the directly converted interface will be researched.
- **Test 4 (Chapter 10):** The effects repeated actions has between a physical controller and a directly converted interface on relaxation will be researched.

Chapter 4

Problem Formulation

The last chapter showed that the problem of mapping physical controllers onto a touchscreen, comes with a lot of difficulties. The research showed that in most cases the physical controls has better performance than their touchscreen counterpart. However in daily use, it would be very redundant to walk around with a physical controller in order to play games on a smartphone. In this project we will focus on solutions with virtual interfaces, without the need for a physical controller or any addons to the phone. We assume that we cannot reach the same kind of performance with the touch interface as with a physical controller. The goal of the project will be to understand limitations of the touchscreen during different tasks and use this information to design interfaces that intends to improve the game performance compared to an interface that has been directly mapped from a physical device.

Project Delimitations:
The devices that will be used in this project have been deliminated to be a Samsung Galaxy S3 mobile [Samsung, 2013] phone for running the game and for touch input. The physical controller is a Sony Dualshock3 controller [Wikipedia, 2013c], that is made for the PlayStation 3 [Sony, 2013]. The important parts of the controller is that it has a directional pad and at least two buttons that can be used to compare it to games with directly converted interfaces.

Project statement:
How to design interfaces that improves the players control in a game on a touch device compared to an interface that has been directly mapped to a touchscreen from a physical controller?

Chapter 5

Technical limits Test

The project has been delimitated to use a PS3 controller together with Unity3D [Technologies, 2013] for prototyping. In order to have a proper comparison in a test, the devices must roughly have the same delay when performing inputs. This test will aim to figure out the delay of the PS3 controller and the onscreen buttons.

5.1 Device specifications

The following is specifications for the Samsung Galaxy 3 and the PlayStation 3 controller.

Samsung Galaxy 3 specifications:

- Display: 4.8 inch HD Super AMOLED (1280x720 pixels at 306 ppi)
- Dimension: 136.6 x 70.6 x 8.6 mm
- Weight: 133g
- CPU: Quad-core 1.4 GHz Cortex-A9
- GPU: Mali-400MP

Figure 5.1: The Samsung Galaxy S3 [HDwallpaperspk.com, 2013]

Sony Dual Shock 3 specifications:

- Dimensions: 160 x 97 x 55 mm
- Connectivity: USB, Bluetooth
- Weight: 192g
- Input:
  - Motion sensing (6 axes)
  - Analog sticks (10-bit precision)
  - Analog trigger (L2, R2)
  - Pressure sensitive buttons (Triangle, Circle, Cross, Square, L1, R1)
  - Pressure sensitive directional buttons
  - Digital buttons (Start, Select, 'PS', L3, R3)

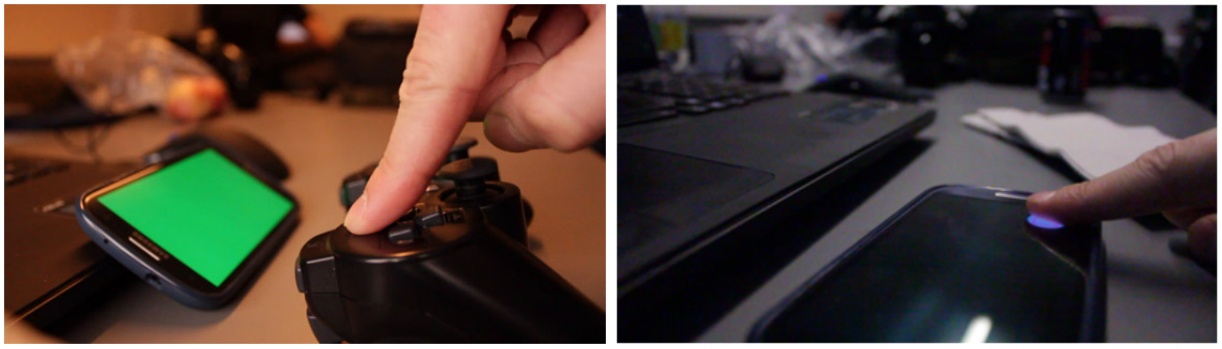
Figure 5.2: The Sony Dual Shock 3 for Playstation 3 [Modify.com, 2013]

5.2 Testing the delay

These tests of the screen and camera was made to learn the minimum refresh rate of the screen and the delay of the two different devices.

5.2.1 Screen refresh rate

To test the refresh rate of the screen, a counter was made in Unity that count seconds and milliseconds, which was then filmed by the camera. The test was conducted using a Canon 550D camera [Dpreview.com, 2013] and the video was filmed with 60 frames per second.



(a) Screenshot of controller delay test (b) Screenshot of touchscreen delay test

Figure 5.3: The test setup for using the physical (a) and touch (b) buttons. The screenshots are from the recorded camera feed. Display turning green when button is pressed in order to analyse the recording of the delay in a videoediting program.

When a button is pressed, the smartphone's display turns green and when pressed again the green disappears (as seen in Figure 5.3a). During the test, the finger hit the button at a steady pace and the finger was kept down on the button until the green display turned either on or off. By keeping the finger down, one can count the camera's frames from when the finger is down, until a green light shows or disappears on screen. On the touch screen, just a single button was made and tested (which would simply correspond to a touch on the screen) while the physical controller was tested on the X button and UP button on the directional pad. The X and UP button are normally analog buttons, but in this prototype they only activate when the button is pressed fully down, thus acting as a digital button.

The test results showed that the counter displayed very close to the same time as what was shown on the camera. This means that the refresh rate is at least 16.6 milliseconds (1000 milliseconds / 60 frames per second). This means that the camera can be used to find the delay of the controller and the onscreen touch buttons on chosen devices within a small margin of error. This in turn can let us know whether the difference between the delays are too high, which would then have to be taken into account when the users play.

The results show that there is a slight delay for the touch controls. The delay for the physical controller is about 125 milliseconds and the delay for the onscreen touch button is about 136 milliseconds. Because of the margin of error of 16.6 milisecond per frame, the touch time lies together with the physical and can they can all be said to be about the same amount of delay.

5.2.2 Conclusion:

The delay for the physical controller and the touch device appears to lie between the same amount of delay. As stated in the background research, see Section 3.2, the delay for receiving and processing light stimuli takes about 190-215, thus there should not be a problem with the delay of the controller.

Now that it has been established that the PS3 controller and the device used can be compared, the devices can be used to test simple tasks performed by the user.

Chapter 6

Test 1: Differences between touchscreen and physical controller in simple tasks

In this study we aim to measure the accuracy of a touchscreen and a physical controller. We will do this by making test subjects carry out simple tasks, that are button combinations commonly found in 2D platform games, with either the smartphone's touchscreen or the PlayStation 3 controller.

6.1 Test Setup

In order to find out how precise users could perform in simple assignments with controls on the touchscreen and with the PlayStation controller, a series of simple tasks was constructed. These tasks would test the user for precision with the D-pad alone (used with left thumb), precision with two action buttons (used with right thumb) and finally precision with tasks that combined the D-pad and action buttons.
All result were measured in percentage in distance of the circle objects starting point to the goal.
All participants in this study were Medialogy students from Aalborg University. There were 6 male participants with their age ranging from 22 to 28. Prior to the experiment the participants completed a questionnaire where they were asked which game genres they preferred and how many hours they spend on pc, console and smartphone games, as this could have an impact on how well they performed in the test.
After each task they furthermore had to answer how well they liked the physical and the touchscreen controls on a scale from 1 to 10. The order of the tasks, as well as in which order they tried the physical or the touchscreen interface were furthermore randomized, so that the results would not get bias due to learning.
Prior to the experiment the subjects were asked to find a natural gaming position and sit as comfortable as possible. Without further instruction the test subjects all held both the smartphone and the PlayStation3 controller in similar ways. The distance from the participants' eyes to the smartphone is therefore in a distance that we assume is common when utilizing smartphones in the everyday life.

Objective of Study:

- To find the difference in accuracy between a physical controller and a touch screen device during simple tasks.

Participants:

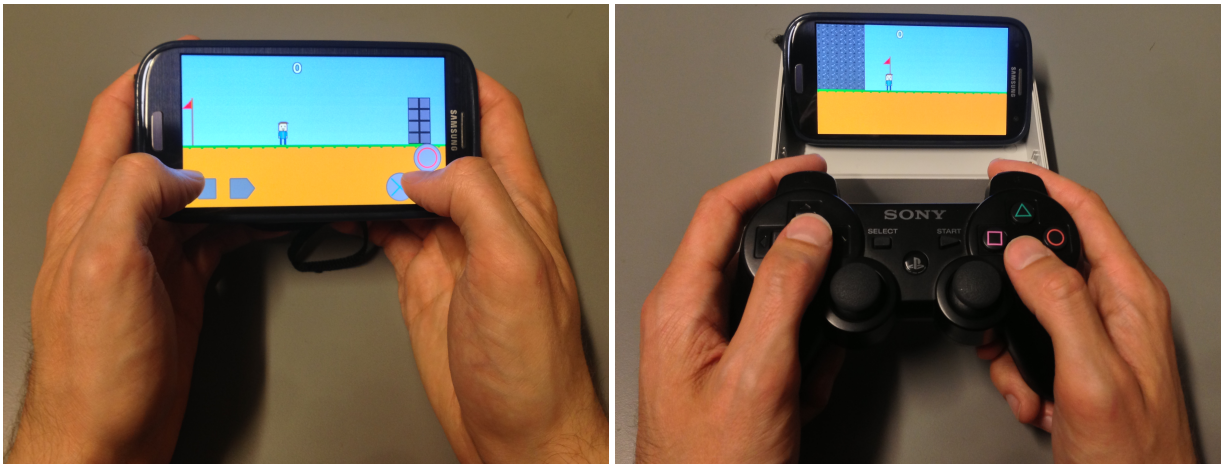
- 6 male participants
- Age 23 to 28 with an average of 24.6
- 5 right-handed and 1 left-handed
- Recruited at Aalborg University

Test Apperatus:

- Samsung Galaxy 3, see specifications in section 5.1.
- Sony Dual Shock 3 Controller, see specifications in section 5.1.

Experiment Procedure:

- All tasks were video recorded.
- Before the tasks the test subjects were given an unlimited training period.
- 5 tasks carried out, see the individual task for further detail.

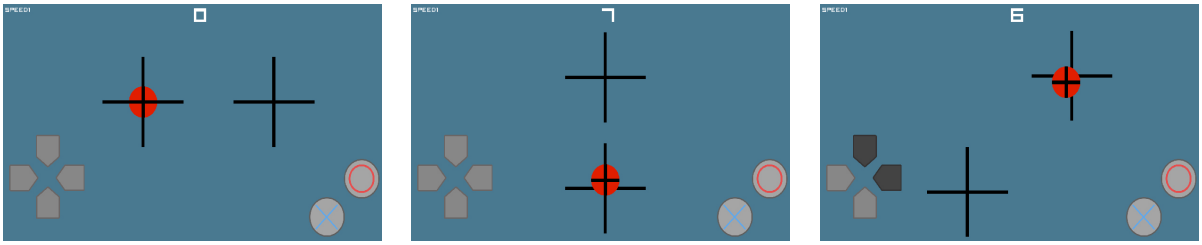


(a) Test subject's point of view when testing the touch-screen interface.

(b) Test subject's point of view when testing the physical controller.

Figure 6.1: To position the screen in a natural way when the subjects used the PS3 controller, the phone was placed in a stand to angle it naturally to the user. Both screenshots are from later tests. The setup is, however, the same for all tests throughout the report.

As can be seen in Figure 6.1 the smartphone was placed on a stand, so all participants were able to see the screen when using the physical controller as they would have when holding the phone.



(a) Horizontal task (b) Vertical task (c) Diagonal task

Figure 6.2: Tasks where the test subjects had to move the red circle from the first cross to the other in its respectable direction using the D-pad. The subject began with the first speed (150 px/s), then tried the horizontal, vertical and diagonal task. After trying three trials, the speed is changed to speed 2 (100 px/s), then repeat the same three task and then change the speed to speed 3 (150 px/s).

6.1.1 Task 1: Directional Pad

In this task the test subjects were asked to move the red circle from its start position at the first black cross to the second and then back again by only using the directional pad, see Figure 6.2. This task was carried out in three different directions; Left to right and back (Figure 6.2a), down to up and back (Figure 6.2b), and right-up diagonally and back (Figure 6.2c). These were tested at different speeds; 100 pixels per second (px/s) and 150 px/s.

Task 1 Results

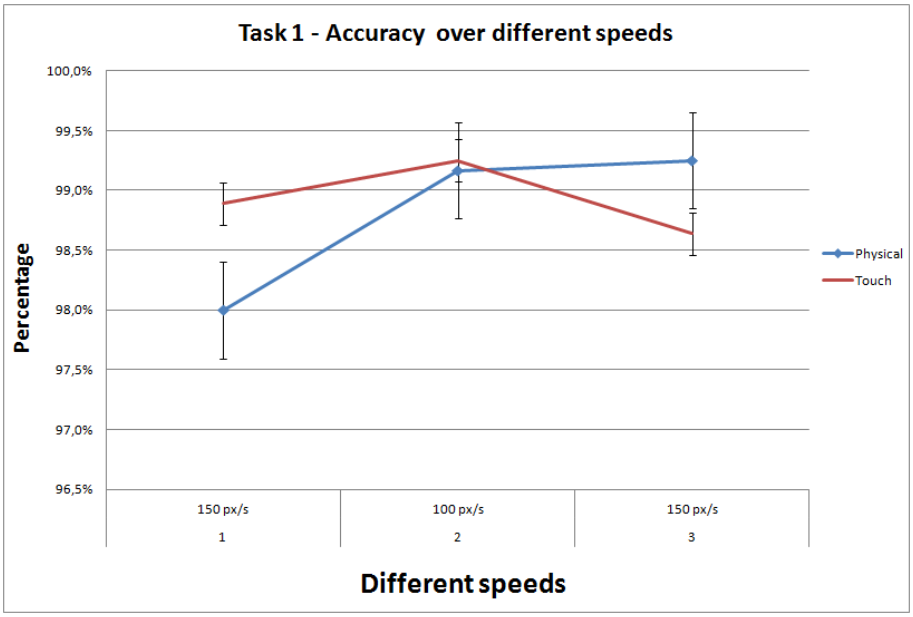


Figure 6.3: The accuracy over three different speeds. The participants try 150 px/s first, then 100 px/s and lastly 150 px/s.

Looking at Figure 6.3, where Physical and Touch has been separated, we can see that they meet

at the same accuracy at the middle speed, but then the touch interface decreases in accuracy when coming back to a speed of 150 px/s whereas the physical goes slowly upwards. Statistical tests show no differences between the touch and physical in turn three at 150 px/s ($P > 0.05$).

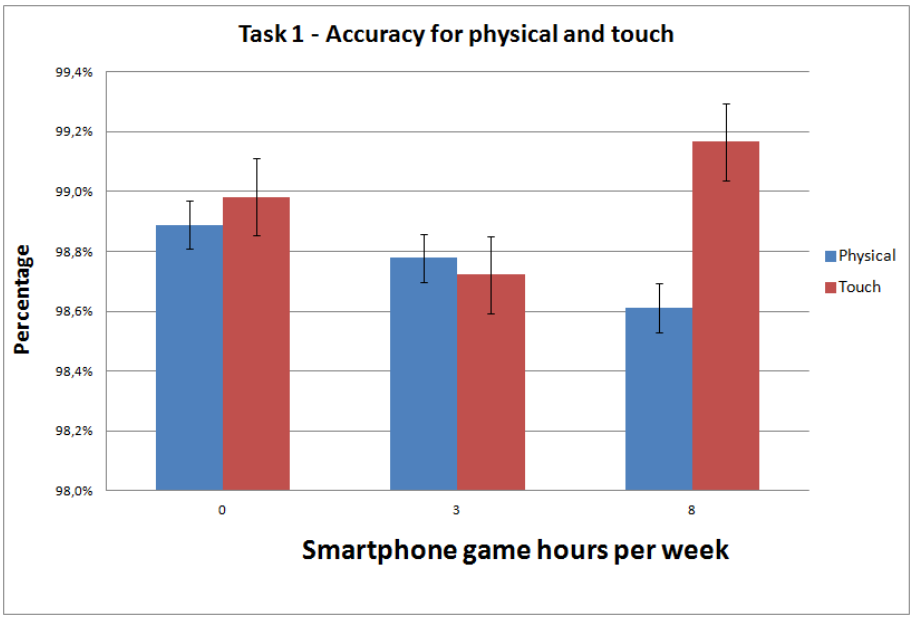


Figure 6.4: Shows the accuracy for the participants that plays 0, 3 and 8 hours of smartphone games per week.

In Figure 6.4 the smartphones games played per week has been included. The participants who plays at least 8 hours per week, has an overall higher accuracy for the three turns, compared to only 3 hours per week.

6.1.2 Task 2: One Button Timing

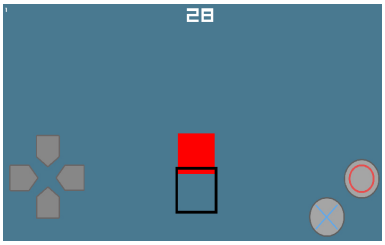


Figure 6.5: Red square falling from the top of the screen, when red square was center of the black square, then user had to press X.

In this task the test subjects simply had to press the action button X, when the red square was in the center of the black square, see Figure 6.5. The red box was falling from the top of the screen

at three different speeds, first five moving at 250 px/s, five with 350 px/s and then five with 450 px/s.

Task 2 Results

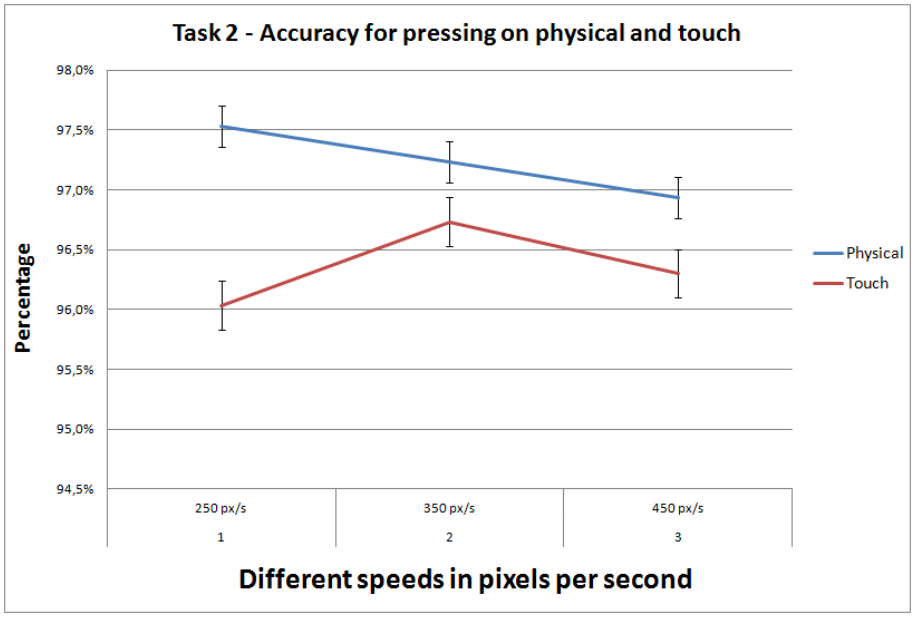


Figure 6.6: The accuracy for the controls over the three different speeds with standard error bars.

As seen in Figure 6.6 when looking at the accuracy of the two control types in task 2, we can see a drop for both from speed 2 (350px/s) to speed 3 (450px/s). The difference between the physical and touch interface in speed 3 (450 px/s) is not significant ($P < 0.05$).

6.1.3 Task 3: Repeated Press Timing

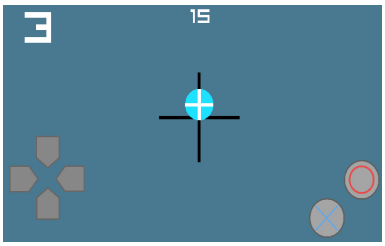


Figure 6.7: Test subjects press X to make the circle jump and must press X again when in the center of the cross. The circle start position is in the bottom of the screen.

This task simulates a jump situation in a game. The test subjects had to make the circle "jump" from the bottom of the screen by pressing action button X and then pressing X again when the circle was in the center of the black cross. The circle "jumped" from the bottom of the screen

with three different speeds (778 px/s, 1490 px/s and 1890 px/s), five times at each speed.

Task 3 Results

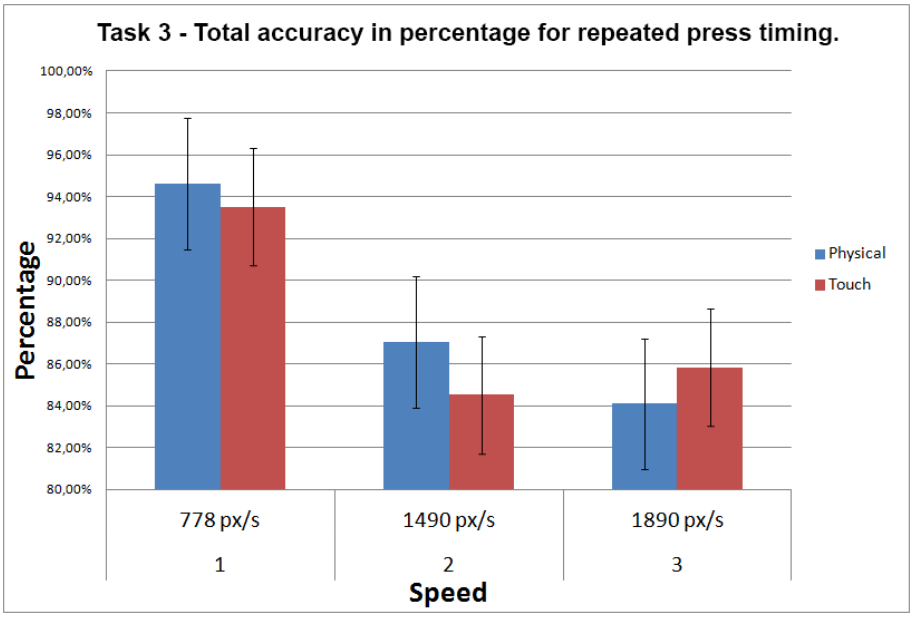


Figure 6.8: Shows the accuracy over speed for both interfaces with standard error bars.

As can be seen on Figure 6.8 the accuracy drops for both the touch and physical controls when a higher speed is applied.

6.1.4 Task 4: Two Button Timing

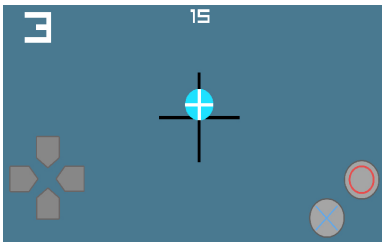


Figure 6.9: Jump task: test subjects press X to make the circle jump and must press O when in the center of the cross

In this task the test subjects had to do as in task 3, but in this task they had to press action button X to make the circle "jump" and then action button 'O' when the circle was in the center of the cross.

Task 4 Results

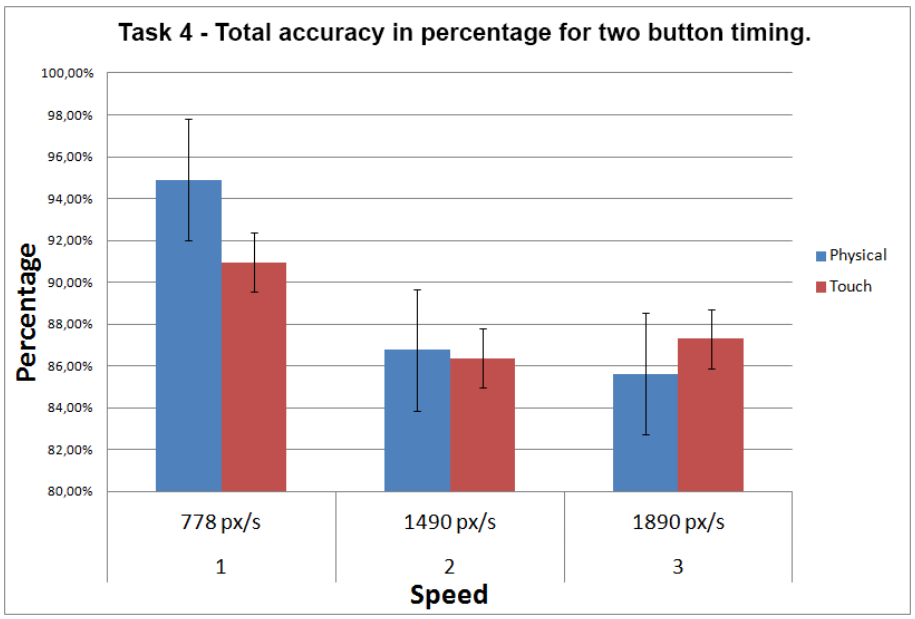


Figure 6.10: Shows the accuracy over speed for both interfaces with standard error bars.

As can be seen on Figure 6.10 the accuracy also drops for both the touch and physical controls when a higher speed is applied.

6.1.5 Task 5: Combined Coordination and Timing

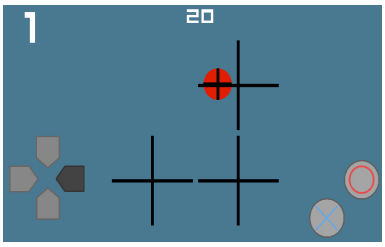


Figure 6.11: Starting at the lower left cross, the test subject must move the red circle to the lower right cross with the D-pad and then make the circle "jump" using the X button and then lastly use O button when the circle is in the center of the top cross.

In this task the test subjects had to move the red circle from its starting point at the lower left cross to the lower right cross using the D-pad, then had to make the red circle "jump" by pressing action button X and then pressing action button O, when the red circle was in the center of the upper black cross, in a total of five times. The speed for moving the red circle was 200 px/s and the speed for jumping was 707,69 px/s.

Task 5 Results

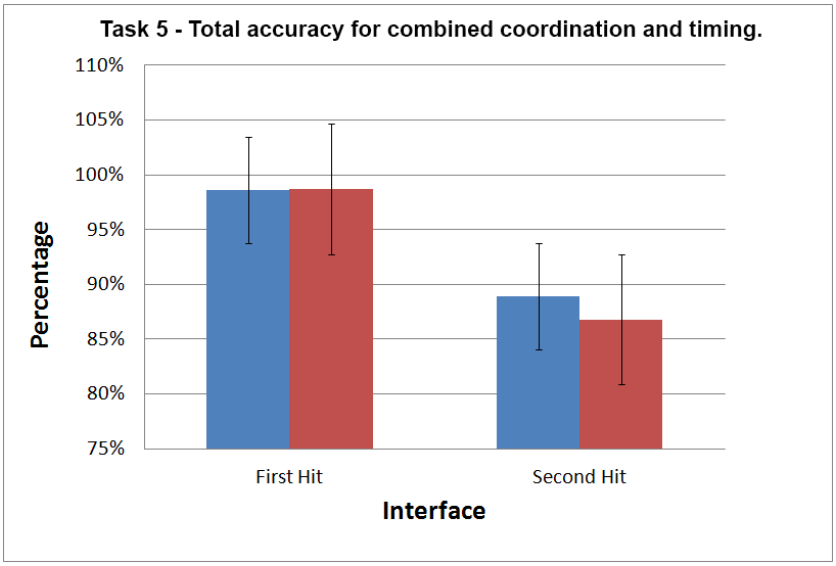


Figure 6.12: The graph displays the average accuracy for the first and second hit with standard error bars. The first hit is where the participants moved the ball to the right. The second hit is the accuracy when the ball has jumped and the participant has pressed the button.

In this task, it does not appear that there is much different in the first speed (where the action is to move right to the target) as seen in Figure 6.12. There does not seem to be any changed in the second hit either.

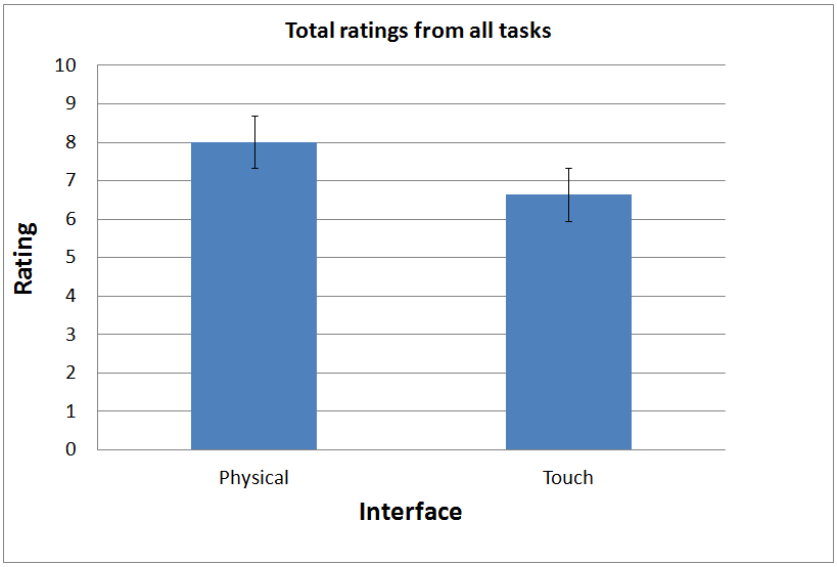


Figure 6.13: The ratings from the participants for each interface

As can be seen in Figure 6.13 the ratings of the two interfaces show that the participants rated the physical controller higher.

6.2 Discussion

The test was performed to see if there was any differences in accuracy between the physical controller and the touch interface when performing simple tasks. The results show that there is no noticeable difference between the interfaces when comparing accuracy. When looking at the first task the results show that the touch interface has a lower accuracy when coming from a 100 px/s speed to a 150 px/s. There was an error that changed the first speed to be 150 px/s where it should have been 50 px/s. However, all three of these speeds appeared to be rather slow when the participants were trying it out and thus it could be interesting to try it out with higher speeds to see if there is a larger difference.
The results also show that participants who plays around 8 hours of smartphone games per week, has a higher accuracy for task 1. This makes sense as the increased usage and experience with the smartphone should make it easier for them in the task. However since the participants were not equally balanced in how much they played per week, the increased accuracy might fluctuate more. Thus more participants with a greater variety of hours per week would help get more precise data.
Task 2 shows that both interfaces seem to decreases as the speed increases, with the physical a bit less than the touch interface. The difference in the beginning of the task is possibly due to the participant getting used to the touch device. Trying a faster speed than 450 px/s might be interesting to examine, especially more participants with a broader amount of experience for each interface.
For task 3, the differences in accuracy for the different speed when pressing the same button twice seem to make a large difference from 778 px/s to 1490 px/s. But comparing the 1490 px/s to the 1890 px/s does not seem to make a huge difference. A reason for this could be that the 1490 px/s is the speed that is fast enough to make errors more likely. This can be seen by the standard error, that vary a lot. More samples could have been taken from each speed to reduce the difference in the beginning of approaching a new speed. The same thing applies to task 4, where the only difference seems to be a bit better accuracy for speed 2 and speed 3 for the touch interface.
In task 5 there was no sign of difference between the interfaces. The participants ratings shows that they preferred the physical controller (with a rating of 8), but the touch interface was only rated 1.5 points lower. A possible reason is the lack of tactile feedback on the touch interface, since it otherwise looked much the same as its physical counterpart.
Another problem might be the low sample size and that the participants tried the interfaces from random sequences instead of randomly assigning them to an ordered sequence. This could have balanced the chance of learning the interfaces over the different tasks. The sample size was rather low (n = 6) and might have impacted the results depending on the amount of experience from the participants. Another error with the prototype was that we wanted as similar interaction and as comparable results in each task across the different test subjects, we had to restrict the users to certain interaction with the controls. This was for example that they in some tasks could not press two buttons at the same time. This could have the impact on their results that they would keep this information in mind and maybe not fully focus on the task at hand. Furthermore, though the

tasks seemed fairly simple, they could have a quite elaborate description, which could maybe for some of the test subject seem rather complicated. An example of this is for example in task 5 where the test subjects were explained every step they had to take and which buttons to press.

From all the results gathered it does not appear that simple tasks makes a difference with a physical controller and a directly converted interface. As it does appear that faster speeds could show a difference between the interfaces, the next goal is to test it on faster actions.

Chapter 7

Test 2: Errors when repeating rapid actions

7.1 Test Setup

In this test the difference in errors and the time to the first error made between a physical controller and a directly converted interface during rapid repeated actions. These tasks were carried out with onscreen controls and as well as with a physical controller in order to compare the two control methods.

All test subjects' performance was recorded in 1080p using a video camera and was then later analysed. As this is a test to give an overview of gamers' performance no quantitative data will be recorded and only qualitative data will be analysed. This data is for example if the test subjects misses a targeted button.
In this test the participants were given a set of simple repetitive tasks to carry out within a limited period of time. These tasks will in further detail be explained individually below. As these task are relatively short and does not simulate a player playing through a whole level, it only simulates a small parts of game situations. These small tasks are therefore repeated several times within a small time frame varying from 10 to 20 seconds, in order to observe if the players make the same errors repeatedly.

Objective of Study:

- To find the difference in error between a physical controller and a directly converted interface during repeated actions.

Participants:

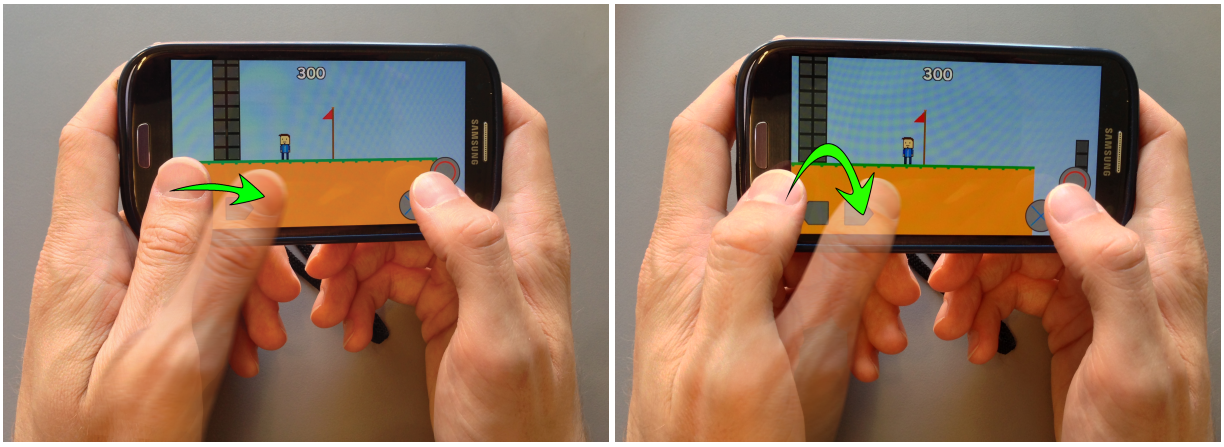
- 6 male participants
- Age 21 to 28 with an average of 23.3.
- All right-handed
- Recruited at Aalborg University

Test Apperatus:

- Samsung Galaxy 3, see specifications in section 5.1.
- Sony Dual Shock 3 Controller, see specifications in section 5.1.

Experiment Procedure:

- All tasks video recorded for analysis.
- Before the tasks the test subjects were given an unlimited training period.
- 7 repetitive tasks carried out for 10 to 20 seconds, see the individual task for further detail.



(a) Swipe on touchscreen (b) Tap on touchscreen

Figure 7.1: Illustration of the different input methods the test subjects were asked to make. Same input methods were done on the physical controller.

Gunman Clive

To carry out this observation test a 2D platform game called "Gunman Clive" (see Figure 7.2) created by Bertil Hörberg was chosen [Hörberg, 2013]. This game has a traditional controls scheme involving an onscreen D-pad and action buttons to control jumping and shooting. Overall this game has a stylistic, but very clear design, giving it an uncluttered appearance for the test subjects.

Figure 7.2: Screenshot of Gunman Clive [Hörberg, 2013].

7.1.1 Task 1: Left-Right swiping

In this task the test subjects were asked to make the game character turn left and right as many times as possible within 10 seconds by swiping on the touchscreen or doing a swipe movement on the controller, see Figure 7.3. No restraints was given for the test subjects to keep the character in one place.
This task requires the user to move his left thumb from side to side with his finger constantly touching the screen (As illustrated in Figure 7.1a) and then repeat this process for 10 seconds.

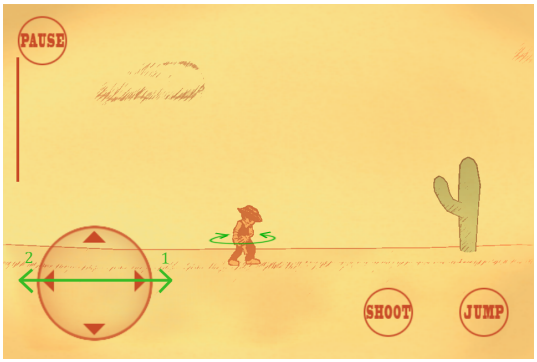


Figure 7.3: Test where the test subject in rapid succession should make the game character turn by swiping.

Touchscreen Observations

When performing this task it was observed that the players often swiped their fingers too short of the target area, the consequence being that the character did not turn. This might be caused by the friction between the players finger and the screen, combined with the fact that there and no physical feedback other than the flat screen.
Furthermore the test subjects often did not swipe equally to the right and the left, making the game character move to the right on the screen. This did however happen on both the touchscreen and the controller.

Though this task was only carried out for 10 seconds, some of the test subjects switched hand position during the 10 seconds. This could be because they got fatigued in their hand or arm, or that they thought the task could be carried out better with another hand position.

Controller Observations

When using the controller in this task, the test subjects often seemed to have more control over the game character, in the sense that the character made a full rotation from side to side. This could however be due to the fact that the swipe motion on the controller is harder to execute.
Some made the swiping motion as on the touch, while other carried out the task with a rolling motion of the thumb - having the tip of the thumb on the right movement button and when having to move left uses the upper part of the thumb with a rolling motion while releasing pressure on the right movement button.

7.1.2 Task 2: Left-Right tapping

In this task the test subjects were asked to make the game character turn left and right as many times as possible within 10 seconds by in turn pressing the left and right button, see Figure 7.4. No restraints was given to the test subjects to keep the character in one place.
This task requires the user to press the left and right button by lifting his left thumb between each button press and then repeat this process for 10 seconds (As illustrated in Figure 7.1b).

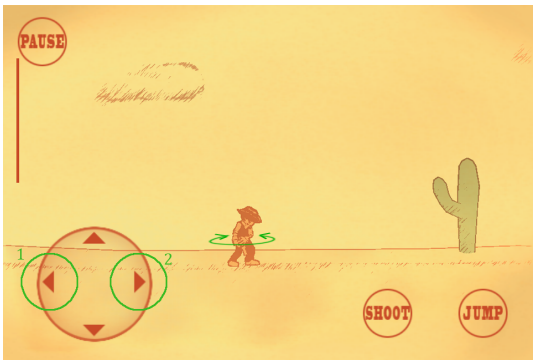


Figure 7.4: Test where the test subject in rapid succession should make the game character turn by tapping.

Touchscreen Observations

Because of the repetitiveness in this task several of the test subjects often failed to alternate between the left and right-buttons and sometimes accidentally pressed the same button twice in a row.
In this task some of the test subject missed their targeted left or right movement button. This could either indicate that occlusion of the buttons has some significance or it could mean that when focussing on a task that the test subjects does not pay attention to the controls.

Controller Observations

The test subjects made same coordination errors as with the touchscreen, and sometimes pressed same button twice in a row.

7.1.3 Task 3: Jump and Shoot

In this task the test subjects were asked to jump to the characters max height and shoot as many times a possible and repeat this in a period of 10 seconds, see Figure 7.5.
This task requires the user to hold the jump button until the game character is at his max height with right thumb, then lift the finger and press the shoot button up to three times in a row and then repeat this process for 10 seconds.

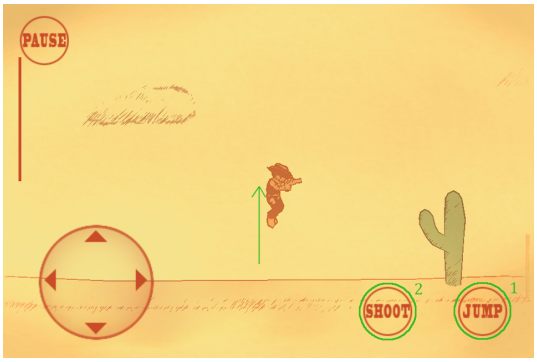


Figure 7.5: Test where the test subject should make the game character jump and shoot by pressing the "jump"-button (1) and then press the "shoot"-button (2).

Touchscreen Observations

When doing this task the test subjects sometimes failed to target the jump-button. This could be because they were too focused on to game character and the assignment or because they did not pay attention to where the on-screen buttons are located. While some test subject made three shots in the air, some only made two and some only one.

Controller Observations

With the controller the players did not manage to get as many sequences of actions through the 10 seconds as on the touchscreen.

7.1.4 Task 5: Jump, turn and shoot

In this task the test subjects were asked to jump from one platform to another while turning and shooting midair, and then repeat this process 5 times.
As seen in Figure 7.6 this task requires the user to hold the left movement button shortly (1), then simultaneously hold the jump button shortly (2), then press the right movement button (3) and the lastly press the shoot button (4).

Touchscreen Observations

Though this task required more coordination of the test subjects, most of them carried out the task without errors, while some fell down the platform several times out of the five trials. This was often caused by the test subjects' inability to precisely navigate the character with the on-screen controls.

Controller Observations

With the controller only one out of the six test subjects fell down the platform. This happened only once and was because he mixed up the jump and shoot buttons.

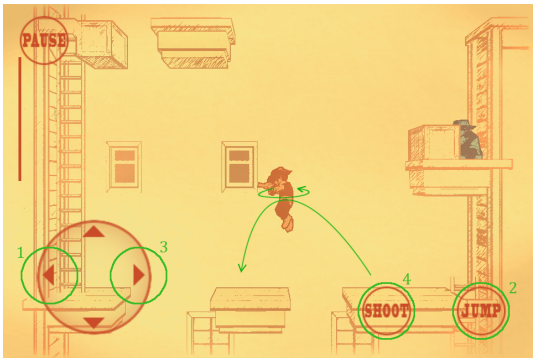


Figure 7.6: Test where the test subject had to make the character jump from one platform to another while mid-air making the character turn and shoot. The green numbers indicate the order in which test subject should press the buttons to successfully complete the task.

7.1.5 Task 6: Blind left-right tapping

In this task the test subjects were blindfolded and was then asked make the character turn from side to side by tapping the left and right movement buttons for 20 seconds, see Figure 7.7. Before being blindfolded and beginning the task the test subjects were asked if they were satisfied with their finger position and if they was confident of the button positions.
This task requires the user remember the button positions and guess where the buttons are located, while trying to hit the left and right movement buttons with his left thumb.

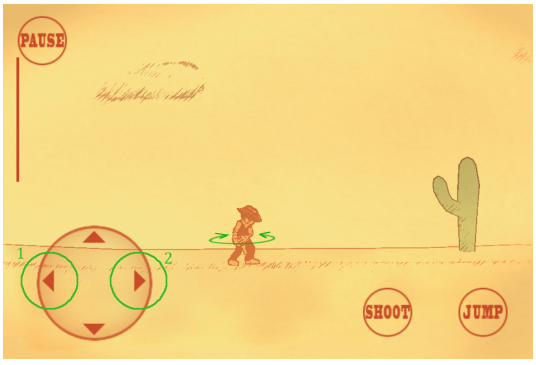


Figure 7.7: Test where the test subject in rapid succession while blindfolded should make the game character turn by tapping.

Touchscreen Observations

As in the non-blindfolded left-right tap task the test subjects sometimes failed to coordinate their finger and therefore pressed the same button two times in a row. This could as in the non-blindfolded task be because of the repetitiveness of the task.
When having no vision some of the test subjects only hit the left movement button and hit slightly

above the right in a diagonal movement of the finger. Though the test subjects also made some mistakes in the non-blindfolded task, they did not do it repeatedly as in this task. This could in the non-blindfolded tasks indicate that players, though paying attention to a task, does also subconsciously keep an eye on the button positions.

Controller Observations
As players have physical buttons underneath their finger and therefore knows where to press they do not make as many mistakes as with the touch interface. They furthermore manages to press equally on the left and the right movement button and thereby making the game character stay in place.

7.1.6 Task 7: Blind turn and shoot

In this task the blindfolded user was asked to make the game character turn left and shoot, then turn right and shoot and repeat this for 20 seconds, see Figure 7.8. Before being blindfolded and beginning the task the test subjects were asked if they were satisfied with their finger position and if they were confident of the button positions.

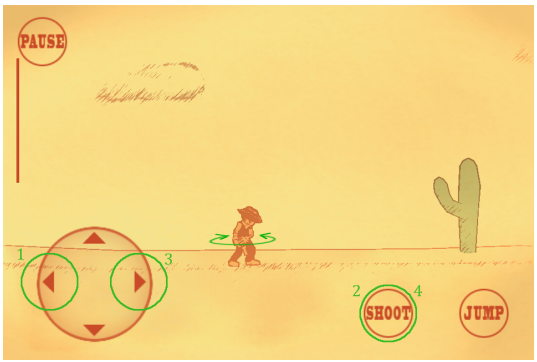


Figure 7.8: Test where the test subject in rapid succession while blindfolded should make the game character turn by tapping and after each turn make the character shoot.

Touchscreen Observations
As this task requires slightly more coordination from the test subjects more mistakes were made. Some of the test subjects often pressed several times in a row on for example the right or left button or the shoot button.
Though the test subjects all had a look at the touchscreen interface before being blindfolded, they all drifted off the button with their fingers within a few seconds. This could indicate that people generally consciously or not (without being blindfolded) will glance at the controls on a touch interface when having to target small button sizes.
Because the test subjects was blindfolded some test subjects repeatedly missed the left or the right movement buttons, some even managed to press outside the screen area.

Controller Observations

With the use of the controller some of the test subjects sometimes missed a shot. This could be due to the fact that one does not know when the physical button is fully pressed.
Because this task requires a bit more coordination while blindfolded, one of the test subjects had a slight pause a couple of times to coordinate right or to think about the assignment.

7.2 Results

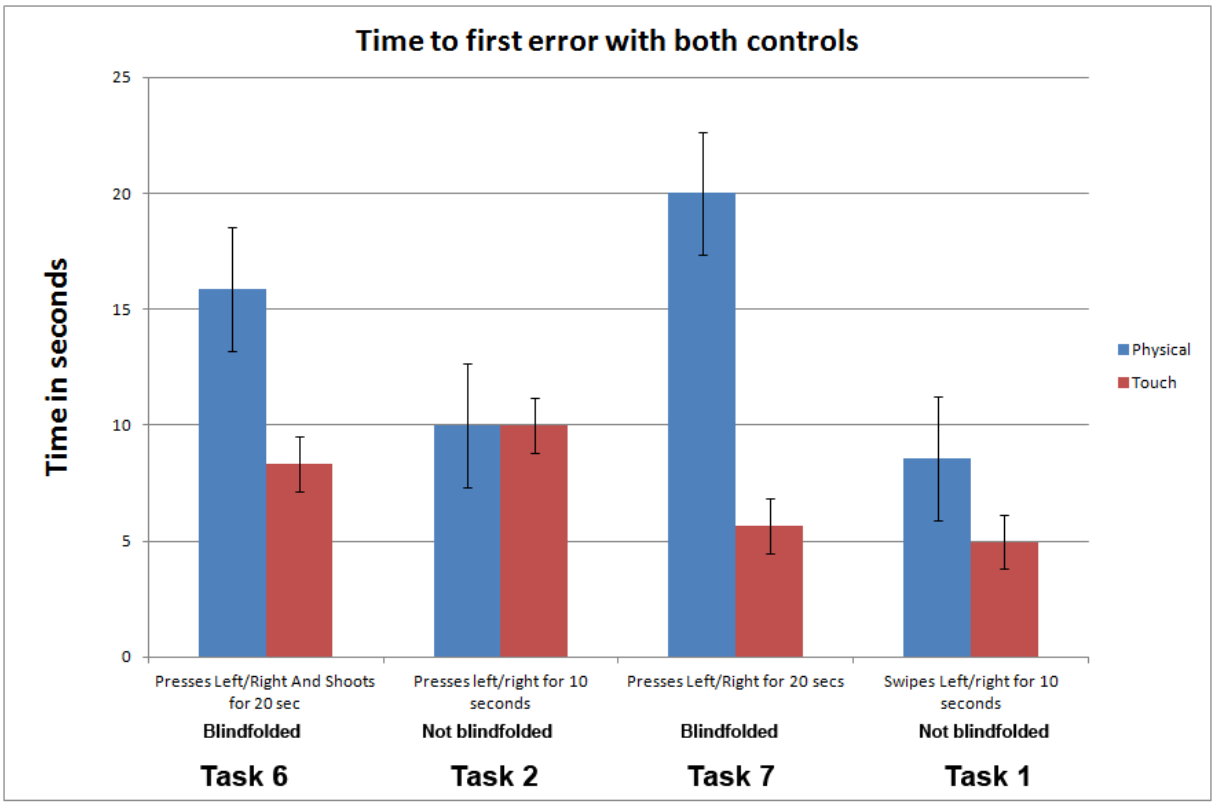


Figure 7.9: Graph showing how long it takes for the test subjects to make an error. 4 tests are displayed in this graph - from left; (Task 6) Press left/right and shoot for 20 seconds while being blindfolded, (Task 2) press left/right for 10 seconds, (Task 7) press left/right for 20 seconds while being blindfolded and (Task 1) swipe left/right for 10 seconds.

As seen in Figure 7.9 it seems that the test subjects generally makes mistakes faster when using the touch device opposed to the physical device while begin blindfolded. This is also the case in one of the non-blindfolded tasks, but as seen on the graph the difference is however much narrower.

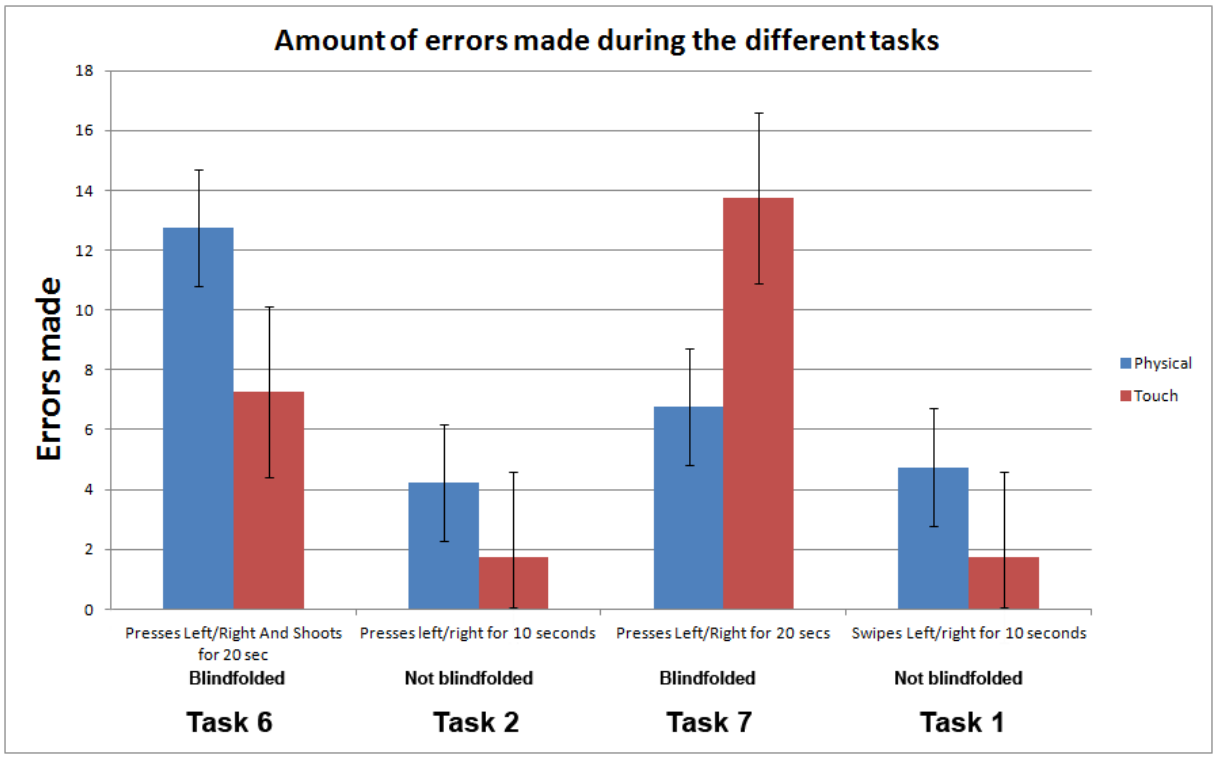


Figure 7.10: Graph showing the number of errors made within the given task time. Blue columns represent the physical device and red columns represent the touch device. 4 tests are displayed in this graph - from left; (Task 6) Press left/right and shoot for 20 seconds while being blindfolded, (Task 2) press left/right for 10 seconds, (Task 7) press left/right for 20 seconds while being blindfolded and (Task 1) swipe left/right for 10 seconds.

As seen in Figure 7.10 the physical makes more errors in the left/right and shoot 20 seconds test when blindfolded. The left/right for 20 seconds while blindedfolded has a larger error rate for the touch interface. The other bars does not appear to have any difference (not blindfolded).

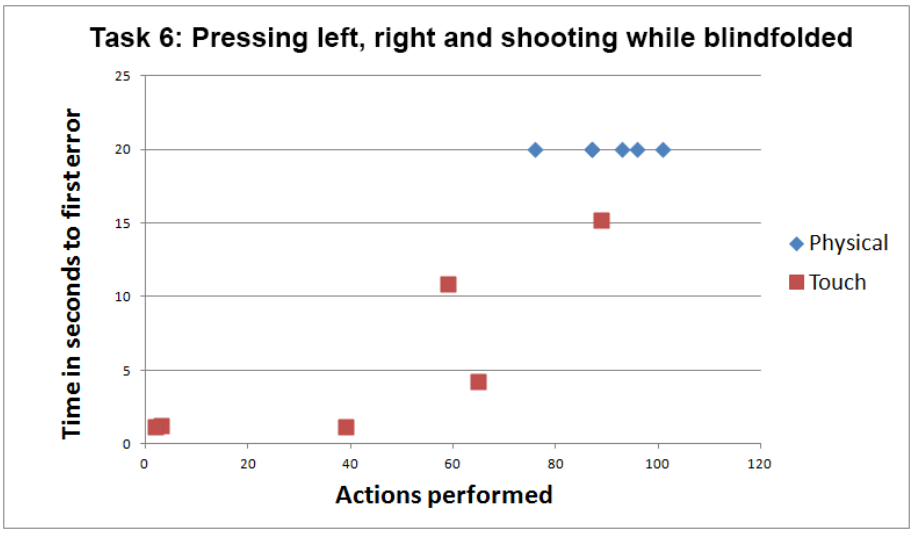


Figure 7.11: Graph showing how many left/right actions each test subject manages to perform before making an error while blindfolded. Blue graphics represent the physical device and red graphics represent the touch device.

As seen in Figure 7.11 it does not appears that making more actions has an effect on how fast an error is made while blindfolded. Generally in this task the participants only made errors with the touchscreen and moreover made errors very quickly.

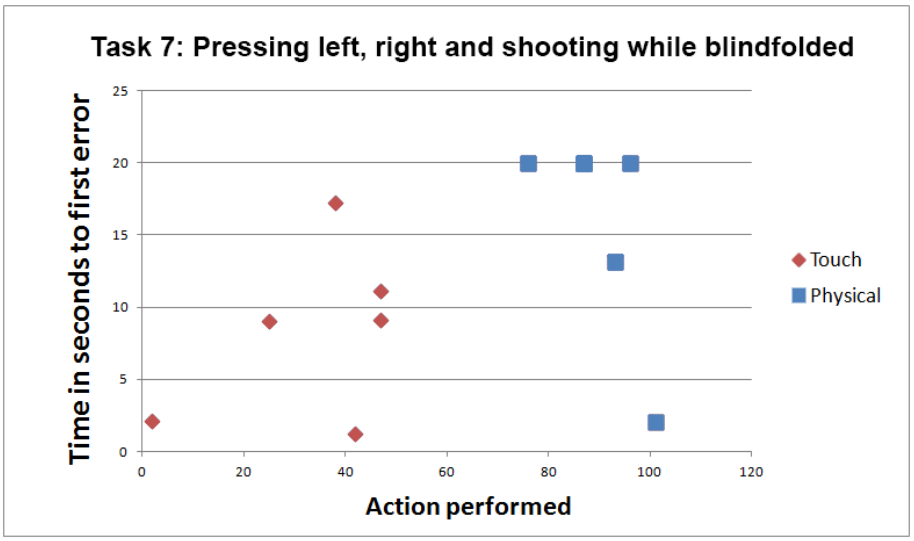


Figure 7.12: Graph showing how many left/right and shoot actions each test subject manages to perform before making an error while blindfolded. Blue graphics represent the physical device and red graphics represent the touch device.

Looking at Figure 7.12, it appears that some test subjects make error faster when performing more actions over 20 seconds, but the same amount does it a bit later on.
The graphs from the physical controls shows us that the limit for the physical controller was not even reached for many of the participants, 20 seconds was not enough to make an error.

7.3 Discussion

The results show that the time in seconds to make the first error while doing repeated tasks blindfolded is a lot longer on the physical controller than on the touch interface. It makes sense as the participants would not be able to know if they were actually hitting the buttons or not. The interesting thing is that it can be used to get some knowledge about the time it would take to recalibrate when playing a touchscreen game. If they were playing a real game, they might have to recalibrate their fingers as they are playing, because they are not able to feel any buttons. The physical controller can always be felt, but still some of the participants made errors before the 20 seconds had elapsed. Some of the reasons for this might be that the participants accidentally pressed the shoot button too fast (before they had pressed either left or right again) and this was counted as an error. This might have been caused by being stressed due to the fact that they had to do it as fast as possible.
When looking at the errors made, the user makes a lot more errors on touchscreen except for in the left/right and shoot task. This is because they accidentally performed the extra shooting, which could not be removed as an outlier (as they were still touching the button). Taking this into account, it appears that the physical controller still outperforms the touch controller, but that the stress factor might be a reason that they did not perform as well with the physical controller.
Looking at the time to first error and actions performed graphs (which would give an idea if extra actions is the same as making errors earlier) we can see that the physical controller hits the ceiling for most of the participants. The time to make an error with the physical controller is not high enough for them to actually make an error in the time. However the touch controls does not appear to make errors faster by doing more actions, rather it is the other way around.
It does appear that the physical controller performs better in terms of taking longer to make errors than the touch device. The conclusion is that users makes less errors with the physical controller than with the touch device when making repeated actions.
Now that repeated actions have been tried out, new designs will be designed for the touch device from the information collected and tested.

Chapter 8

Implementation

In order to have control over every aspect of the desired test variables it was decided to produce a 2D platform game in style with what is currently on the game market. This would enable us to record different data as times, death counts, shots fired, enemies killed, targets shot, jump count for every section of a level, and further more enable us to create our own interfaces for this game. This section describes the implementation of the different aspects of the game, such as interface design, data recording, level design. All code for the prototype can be seen in the appendix 11.1.

8.1 Interface Design

In order to find if there is any improvements in performance by having new ways of interacting with a touchscreen in smartphone games, three new interfaces based on previous findings will be designed. In this section we will describe the three new interface, as well as the two familiar ones; the physical controller and the standard converted to touchscreen interface.

Physical Controller
As in previous tests we will use the physical PlayStation controller to have a baseline to compare how well the test subjects will perform. Furthermore the buttons used on the controller for this test will also be the same as in previous tests; left and right on the D-pad, and "X" for jump and "O" for shooting, see Figure 8.1.

Figure 8.1: Physical controller: The PlayStation 3 Controller [Modify.com, 2013]. The green overlay indicates the buttons used in this test; (1) Left, (2) Right, (3) Jump and (4) Shoot.

Standard Touch Interface

In this test we will as in previous tests have the interface with the controls directly mapped from the physical controller in order to have an idea of how well people perform on this interface. As mentioned earlier a lot of games on the market use this form of interface that has been directly converted from the layout of the physical controller. The layout of the controls will therefore be similar to the physical and will furthermore have slightly larger colliders than what is actually displayed on the screen, see Figure 8.2. This means that the player will have a slight margin of error, meaning that he can actually press slightly beside the displayed button and still execute the intended action. This applies to all of the virtual interfaces that has button graphics displayed onscreen.

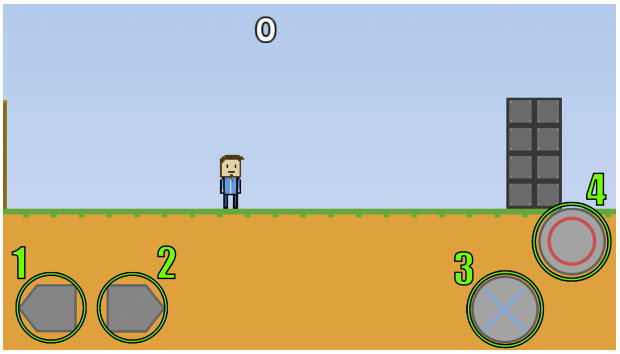


Figure 8.2: Standard interface design as close to the PlayStation Controller as possible. The green overlay indicates the buttons used for this test; (1) Left, (2) Right, (3) Jump and (4) Shoot.

Standard Touch Interface with Vibration

This interface has exactly the same attributes as the "Standard interface", but has vibration to every buttonpress the players makes. As researched in Section 3.2.3 haptic feedback should help the user to analyse if he has actually hit the button and give him a sensation of having physical buttons underneath his fingers.

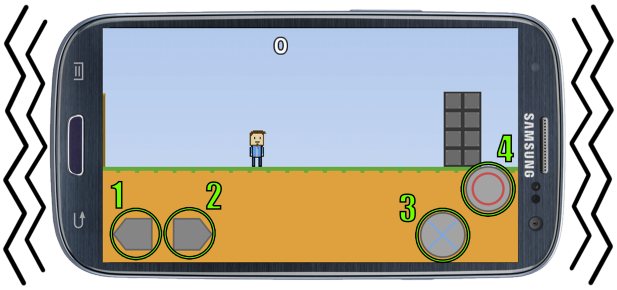


Figure 8.3: Vibration interface: Same as the Standard, but with vibration feedback to every button press. The green overlay indicates the buttons used for this test; (1) Left, (2) Right, (3) Jump and (4) Shoot.

Gesture Touch Interface

As found in Section 7 players often missed to intended button, though the buttons were relatively big. Furthermore when having to swipe between two buttons within a region they often swiped outside this limited area. In order to overcome these mistakes we have therefore proposed a gesture-based interface, where the whole screen is the controller. This way the player should never focus on hitting small button, but strictly speaking only focus on not occluding important parts of the screen. As seen in Figure 8.4 the screen is split in the middle, where the left side of the screen is intended for the left thumb for left and right movement. From the origin or contact point between the screen and the players finger the player can swipe right to make the game character move right and can instantly move left with a small thumb-movement to the left. This applies for movement in both left and right direction.

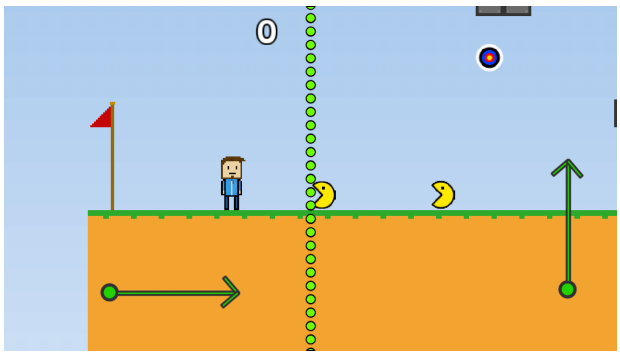


Figure 8.4: Gesturebased interface: Users swipe left or right on the left half of the screen to move the character in the respectful direction. On the right side of the screen they swipe upwards to make the character jump or tap to shoot. The green graphics overlay is only to indicate the middle of the screen and what possible interaction methods this interface offers.

Mixed Touch Interface

This interface is a mix of the standard and the gesture-based interface. It combines the players' familiarity of buttons and the more intuitive gestural way of interacting with a touch device. With this interface the player moves left by pressing the left movement button with his left thumb, and accordingly to the right, while he anywhere on the screen jumps by swiping upwards and shoots with a tap, see Figure 8.5. This way the player obstructs as little as possible of the screen in the movement direction as he for example when moving right uses his right thumb in the lower right corner. As action and obstacles is generally encountered in the direction the player is running, keeping the players finger in the lower part of the screen is a good way of avoiding occlusion in the most important part of the screen.

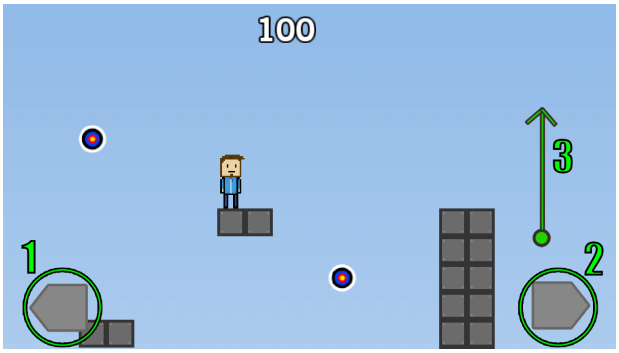


Figure 8.5: Mixed interface: Users move left (1) or right (2) with the button in the left and right lower corner. Jump and shoot can be achieved anywhere on the screen; Swipe upwards for jump (3) and tap to shoot.

8.2 Level Design

In order to test the different interfaces a realistic game world was needed. A level similar to other 2D platform games on the market was therefore designed. Through this level the player would experience different degrees of difficulty, encounter obstacles, pits and enemies as they would in most other 2D platform games. The level has been designed so that every player-action possible (as discovered in Section 3.1.3) is covered throughout the level.
To avoid too much frustration checkpoints has been distributed throughout the level after each level-subsection (see Figure 8.6, also see larger version in Appendix 11.1). If the player dies at a certain point of the level he will be brought back to the nearest encountered checkpoint. Having checkpoints furthermore does that recorded data can be divided into the different sections of the level.

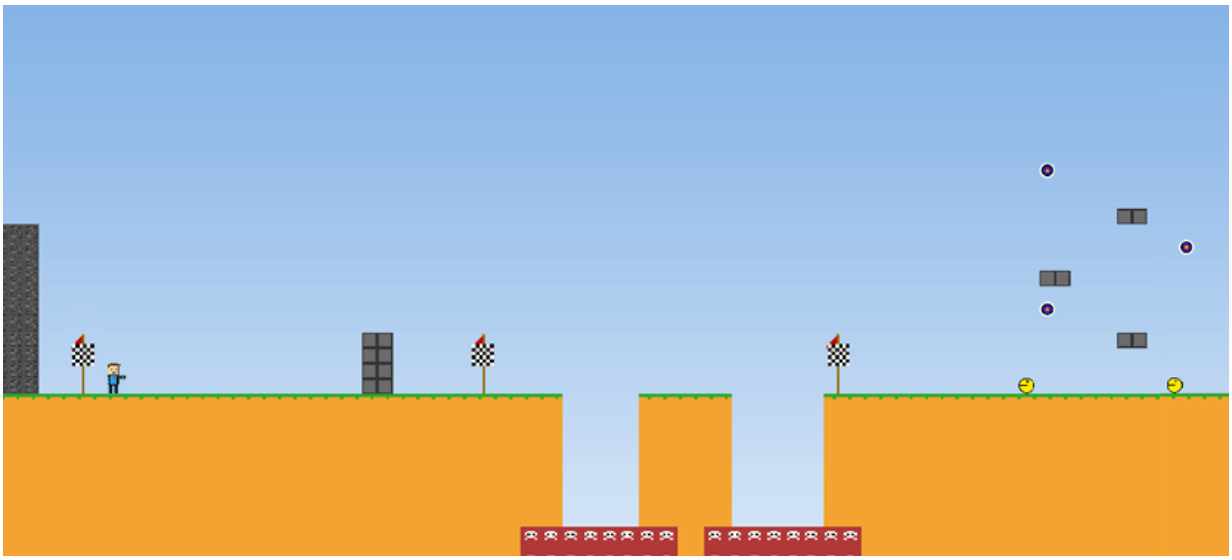


Figure 8.6: A cutted screen capture of the level (See whole level in Appendix 11.1).

In Table 8.1 the different level objects the checkpoint, the target and the different enemies are displayed along with their different properties:

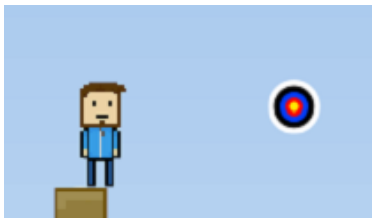
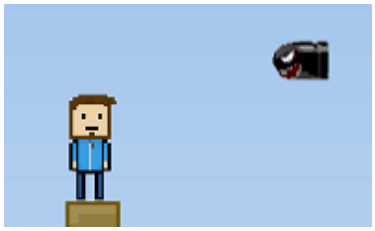
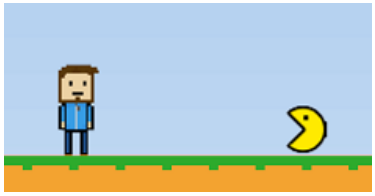
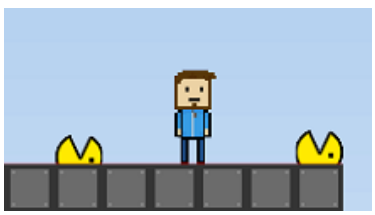
Level Objects	Properties
	• Checkpoint: – Spawns the game character at the last encountered checkpoint when dying – Player cannot move back from this point
	• Target: – Can be shot – Gives Points – Player cannot jump in this object (no colliders)
	• Enemy 1: – Player dies from this object if touched, unless it is jumped on – Cannot be shot and does not give point if jumped on – Flies through the air
	• Enemy 2: – Player dies from this object if touched, unless it is jumped on – Cannot be shot and does not give point if jumped on – Moves back and forth on the ground
	• Enemy 3: – Player dies from this object if touched – Moves up and down – Cannot be killed

Table 8.1: Level objects and its properties.

Chapter 9

Test 3: Testing Interfaces in an Advanced Game Prototype

9.1 Test Design

In this test the participants will play through the custom level (see section 8.2) with five different interfaces, two well known and three newly designed interface (see section 8.1); Physical controller, standard directly converted interface, standard directly converted interface with vibration, gesture interface and a mix between the standard and the gesture interface. There are two purposes of this study. The first is to see if there is any differences between the standard interface and the physical controller when playing through an advanced game prototype. The second purpose is to find if any of the newly designed interfaces show improvements in performance (when looking at time to complete, deaths and targets shot) compared to the standard directly converted interface.
The devices used for this test is the PlayStation 3 controller and the Samsung Galaxy S3, see specifications in section 5.
All participants in this test were students recuited from Aalborg University. There were a total of 12 test participants, 11 male and 1 female, all but one right-handed, with their age ranging from 22 to 30 with and average of 24.25.
Prior to the experiment the participants were asked to fill out a questionnaire where they were asked how many hours weekly they on average spend on pc, console and smartphone games (See full table of answers in Appendix 11.1). After each interface task they were furthermore asked to answer how well they like the interface on a scale from 1 to 10 and give comments of their experience with the given interface (See full table of answers in Appendix 11.1). The order of the tried interfaces were randomized to avoid bias of the test subjects becoming increasingly better at the level for each tested interface.

Prior to the experiment the test subjects were asked to sit as comfortable as possible in a natural gaming position. The test subject were then given the rules of the game; That enemies could not be shot, but only jumped on and would not give any points, that targets could be shot and would give points, that they could not go back once they had reached a new checkpoint, and finally that they could quit at anytime if the level was too difficult, but that we would like the to try as hard as possible to finish the level. After being given the rules of the game the test subject were given an unlimited test period before beginning the actual game.

Objective of Study:

- To find the effects of using interfaces with vibrations, gestures and a mix of these in a game compared to the directly converted interface.

Participants:

- 12 participants; 11 male and 1 female.
- Age 22 to 30 with an average of 24.25.
- 11 right-handed and 1 left-handed.
- Recruited at Aalborg University.

Test Apperatus:

- Samsung Galaxy 3, see specifications in section 5.1.
- Sony Dual Shock 3 Controller, see specifications in section 5.1.

Experiment Procedure:

- All data recorded to a file locally on the device.
- Recorded data:
  - Time.
  - Targets destroyed.
  - Shots Fired.
  - Jump count.
  - Enemies killed.
  - Death count.
- Before beginning the actual level, the test subjects were given am unlimited training period.
- The test subjects were asked to complete the level (see section 8.2) with the 5 different interface (see section 8.1) in randomized order as best possible.

9.2 Results

Question 1: Is there any difference between the standard interface and the physical controller?

The standard interface and the physical controller has been shown to lie close together on the time spent to complete the level on the hard and easy part. Outliers that did not finish the level completely have been removed (n = 10). Statistical tests show that the difference between the physical and standard touch interface is not significant (P = 0.05)

Question 2: Does the subjects perform better with any of the new interfaces, than the standard interface?

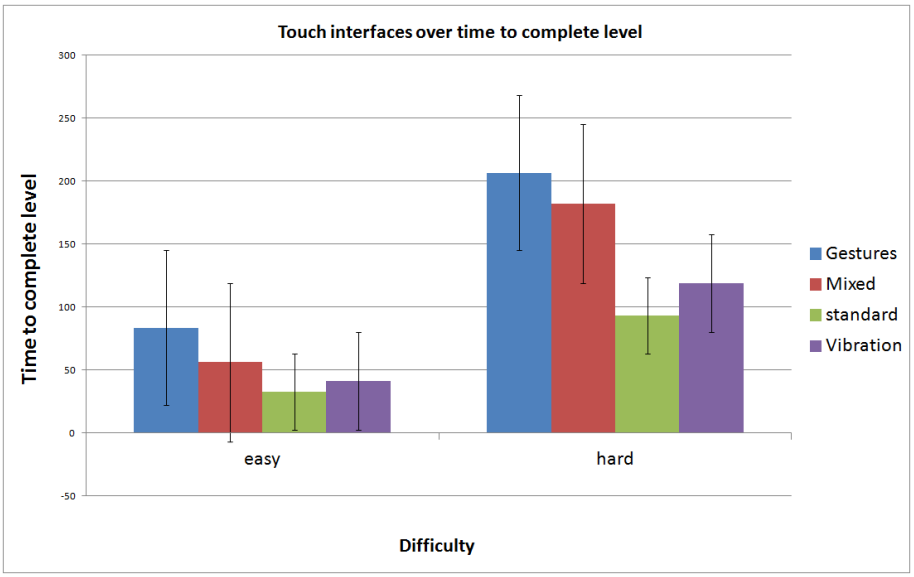


Figure 9.1: The different time to complete the easy and hard part of the level for all touch controls. Five outliers that did not complete the level for all interfaces has been omitted (n = 7).

As seen in Figure 9.1 the subjects that did not complete the level has been omitted. The graph shows that the time spent on the easy and hard part was slower with all of the new interfaces.

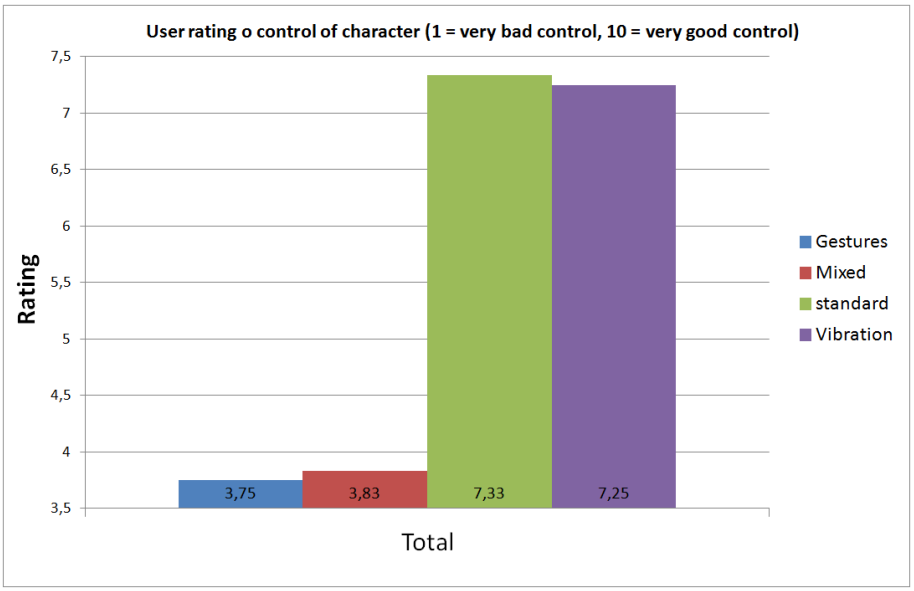


Figure 9.2: The user ratings for each touch interface. The bars display the average rating from all participants.

The Figure 9.2 shows that the participants rated the the standard and vibration interface about the same and both Mixed and Gestures interface lower.

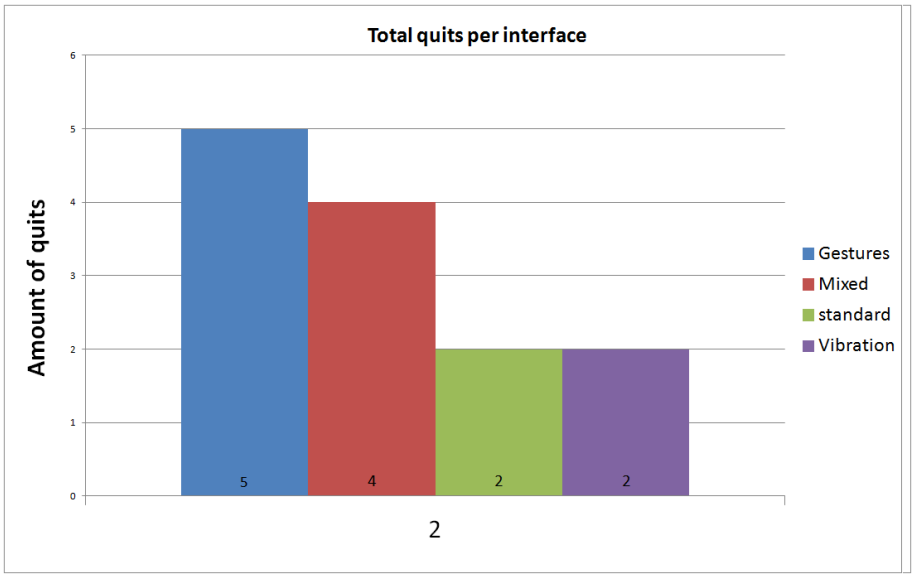


Figure 9.3: The count of quits per touch interface.

As can be seen in the Figure 9.3 the amount of times quitted per interface was highest for the Gestures and Mixed interface. The standard and vibration interface has both the lowest amounts of quits.

Comments from participants: A general view from the participants, when looking at their comments in the questionaire, is that there is generally a problem with jumping in the prototype. Many of the participants also commented that the vibrations interface, assisted a lot to the standard interface. One participant commented: 'I would like to put it somewhere between 9 and 10, better than classic, but not as good as the controller. But the vibration did A LOT'. Overall the participants also commented that the physical controller was more smooth and responsive and that 'it felt good'.

9.3 Discussion

The results showed no difference between the physical and touch interface. The results from the different interfaces furthermore showed that none of the new interfaces performed better than the standard interface when looking at time. When looking at the ratings for each interface, the standard and vibration interface had around the same score. The participants preferred to have the standard interface as shown in their ratings, but their comments also show that the vibrations can assist in the feeling of control, but that many vibrations over the level was almost too much for their fingers (which might be why we were not able to find games with vibrations in their buttons).
There might also be other reasons that the interfaces, that were designed for the touch platform, did not work. For example, the programming of the interfaces might not have worked as well as the standard interface. Sometimes the gesture to jump could take some time to get activated,

which the participants commented was a general problem for jumping in the prototype. Another reason might be that they did not have time to fully learn the interfaces, but already had a 'known' interface in form of a directly converted interface. The reason could perhaps also be found in the amount of quits per interface. The highest quits were made on the gestures and mixed interface, where the participants stopped midgame because they found it too hard or too frustrating. Now that we have discussed the differences between the newly designed interfaces we will now research if stress has any an impact on performance between the physical controller and the touch interface.

Chapter 10

Test 4: Differences between the physical and touch interface in relation to stress.

In this test, we will try to see if there is any differences between the physical and touch interface in relation to stress. The stress when performing on a touch might be higher, since the users can not rest the thumbs on the screen without interacting with it (unlike a physical interface). Touch devices might also cause more frustration as the error rate has been shown to be higher in the previous tests.

Objective of Study

- To see if there is any differences related to stress in participants when using either a physical controller or a touchscreen during repeated actions.

Participants:

- 11 male and 1 female participants
- Age 21 to 25 with an average of 22.8.
- 11 right-handed and 1 lefthanded
- Recruited at Aalborg University

Test Apperatus:

- Samsung Galaxy 3, see specifications in section 5.1.
- Sony Dual Shock 3 Controller, see specifications in section 5.1.

Experiment procedure

- The tasks are video recorded for analysis.
- Before the tasks the test subjects were given an unlimited training period.
- 1 repetitive task carried out for 20 seconds for both interfaces.

10.1 Results

From test 2 we could see that the time to making the first error is much higher for the touch interface. The performance differences could potentially have something to do with how relaxed they are while they are completing the repeated actions.

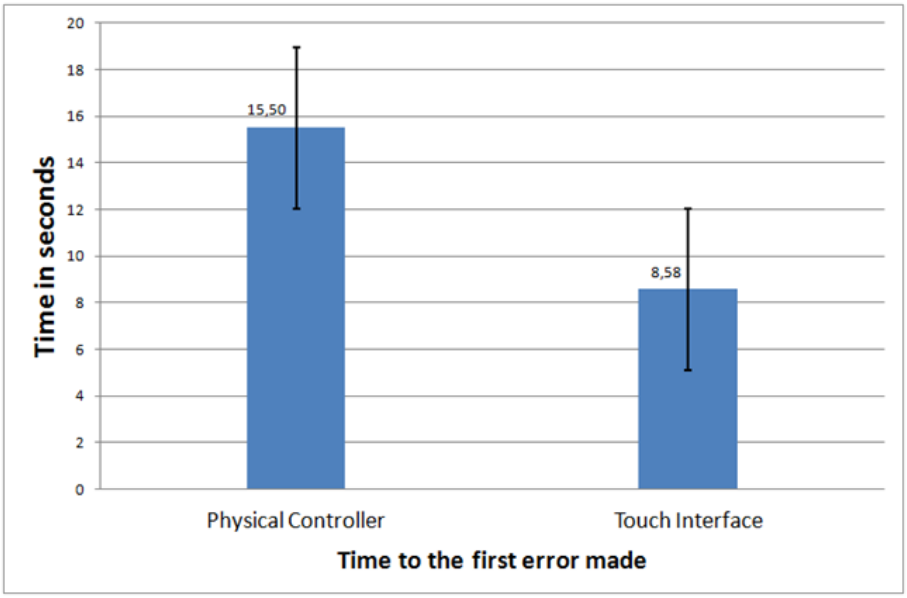


Figure 10.1: The time it takes for participants to make their first error using either the physical controller or the touch interface with standard error bars.

The graph shows that there is a large span in the time to first errors between the interfaces. The difference is significantly different from zero ($p = 0.05$).

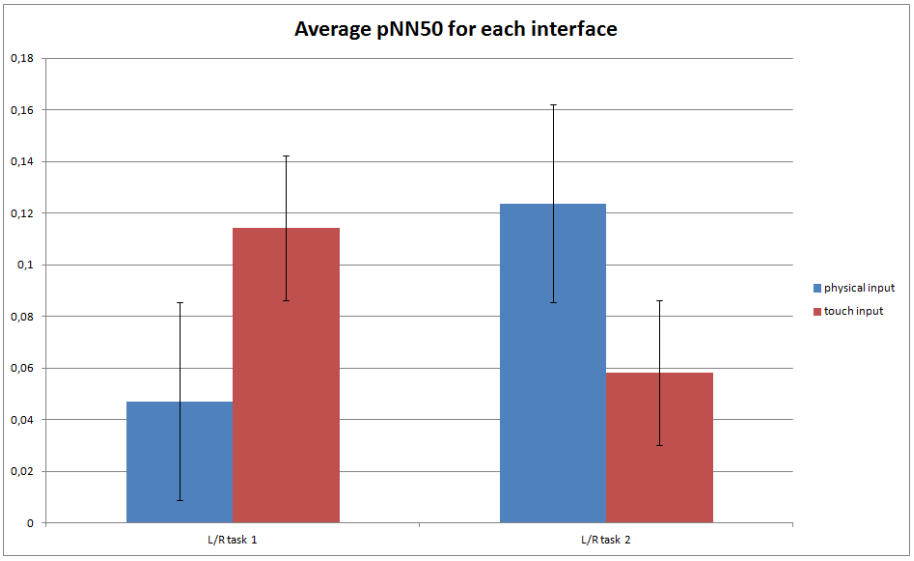


Figure 10.2: The average pnn50 in the first and second round for the physical and touch interface.

As seen in Figure 10.2 the pnn50 is lower when the participants start out with a physical or touch controller. In the second round, the opposite interface of what they were using in round 1, becomes less relaxing.

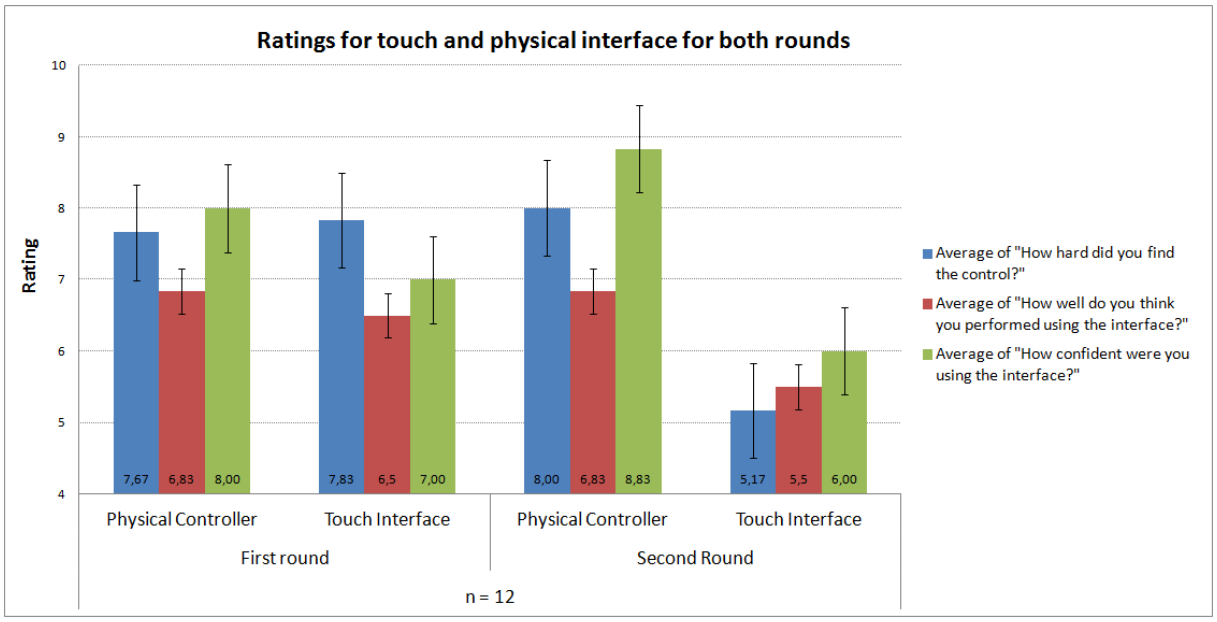


Figure 10.3: The average ratings from the participants. The ratings were giving for each interface right after they had performed the task. The ratings for 'How hard did you find the control?' was from 1 = very hard and 10 = very easy. The rating for 'How well do you think you performed using the interface?' was 1 = very bad and 10 = very good. The rating for 'How confident were you using the interface?' was 1 = Not confident at all and 10 = very confident.

When looking at the participants ratings as seen in Figure 10.3 there is a large drop in rating for the touch interface in the second round, unlike the physical controller, which looks like it is increasing from the last round (which would be touch interface).

10.2 Discussion

The results from this test showed that the touch interface made errors faster than the physical interface. The pnn50 (which can indicate levels of relaxation) was lower in the second round for the touch interface, which indicates that it becomes less relaxing for the participants to play for both rounds. Overall it appears that there is no differences between the two interfaces when related to stress. However it is interesting to note that the participants rated the touch interface lower when trying the physical interface first, where they seem to rate the physical higher when they have just tried the touch interface. So even if there was no statistical difference between the touch and physical order on stress, the participants were still a lot less confident and thought it was harder and that they performed worse when going from a physical to a touch interface. A problem for the participants is that none of them played more than 3 hours of console or smartphone games per week. Thus it could be a good idea to get more samples that could indicate whether the amount of playtime has any effect on the relaxation.

Chapter 11

Evaluation

11.1 Discussion

Through this project several factors revolving touchscreens and user tasks has been evaluated. The first test showed that there are no difference in accuracy when comparing a physical controller and a directly converted interface on a touchscreen. The second test showed that touchscreens make more errors and faster in comparison to a physical controller, when performing repeated actions. The results led to the design of new interfaces that was tried out in a game prototype. However, at first it did not appear that the new designs had any improving effect in time to complete the game when looking at how the participants performed when using these new interfaces.

Adding vibration was the only interface commented as having a potential, but after a period of time some of the test subjects argued that the vibration was too much for their fingers for a longer period of time. The last test showed that there was no difference between using a physical controller or touch interface when relating to stress, which rejected the idea of stress being an issue when having to use an interface with no tactile feedback. The first assumption was that the physical controller when playing would be significantly different than the touch interface could not be confirmed. This could easily have been due to the low number of subjects (n = 6) trying out the initial tests (test 1 and test 2). With such a low number of participants and a few outliers in those tests it is hard to find the difference, as background research showed.

Another problem was that the participants were not balanced in the amount of time played per week for smartphones, consoles and PC. A better balance in experience with each device could lead to interesting results, such as how much experience is needed for a bigger difference between the interfaces. One other problem was that we accidentally randomised the order of interfaces played, rather than structure different sequences such that all orders would have been tried out. This led to holes in the data, which could have had interesting results as for the difference in time on each interface.

This is just a step in understanding how touchscreen interfaces can be improved to give better control to the user. One possibly way of improving them might lie in using vibrations in another way than was done in this project. As the project lacked the necessary amount of participants with a broad amount of game experience, further research should be conducted.

11 References

148apps (2013). Application category distribution. http://148apps.biz/app-store-metrics/?mpage=catcount. [Last accessed: 03/07/2013].

Albinsson, P.-A. and Zhai, S. (2003). High precision touch screen interaction. In *Proceedings of the SIGCHI conference on Human factors in computing CHI '03*, pages 105–112. ACM Press.

Androidtapp.com (2013). Cordy 2 picture. http://cdn.androidtapp.com/wp-content/uploads/2011/06/Cordy-Jumping-2.png. [Last accessed: 05/26/2013].

Apple (2013). Apple's app store. http://www.apple.com/iphone/from-the-app-store/. [Last accessed: 05/29/2013].

Baudisch, P. and Chu, G. (2006). Back-of-Device Interaction Allows Creating Very Small Touch Devices. *Back-of-Device Interaction Allows Creating Very Small Touch Devices*.

Bertucco, M., Cesari, P., and Latash, M. (2013). Fitts law in early postural adjustments. *Neuroscience*, 231:61–69.

Card, S. K., Newell, A., and Moran, T. P. (1983). *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc.

CBSnews (2013). Number of smartphone users tops 1 billion. http://www.cbsnews.com/8301-205_162-57534583/study-number-of-smartphone-users-tops-1-billion/. [Last accessed: 05/26/2013].

Claypool, K. T. and Claypool, M. (2007). On frame rate and player performance in the first person shooter games.

ComputerAndVideoGames.com (2013). Super meat boy. http://www.computerandvideogames.com/330990/super-meat-boy-passes-1-million-sales. [Last accessed: 05/26/2013].

Cunningham, A., Langlotz, H., Rhode, M., and Whaley, C. (2008). Video games industry overview: An analysis of the current market and future growth trends. http://holgerlanglotz.de/downloads/BU4510_VideoGamesIndustry_LanglotzEtAl.pdf. [Last accessed: 05/26/2013].

Develop-Online.net (2013). Limbo. http://www.develop-online.net/news/39109/Limbo-sales-fly-past-1-million-mark. [Last accessed: 05/26/2013].

Dpreview.com (2013). Canon eos 550d. http://www.dpreview.com/reviews/canoneos550d. [Last accessed: 03/29/2013].

Fitts, P. M. (1992). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 121(3):262–269.

Freeapps.ws (2013). Lep's world picture. http://www.freeapps.ws/2013/01/leps-world-2-app.html. [Last accessed: 05/26/2013].

FutureBooks (2013). Mobile game industry analysis 2011. http://www.slideshare.net/futureb00ks/mobile-game-industry-analysis-2011. [Last accessed: 03/07/2013].

GooglePlay (2013). Cordy 2. https://play.google.com/store/apps/details?id=com.silvertree.cordy2. [Last accessed: 05/26/2013].

HDwallpaperspk.com (2013). Samsung. http://www.hdwallpaperspk.com/wp-content/uploads/2013/02/292630-samsung-galaxy-s-iii-at-t.jpg. [Last accessed: 03/29/2013].

Hoggan, E., Brewster, S. A., and Jognston, J. (2008). Investigating the effectiveness of tactile feedback for mobile touchscreen. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems CHI '08*, pages 1573–1582. ACM Press.

Hörberg, B. (2013). Gunman clive. http://www.gunmanclive.com/. [Last accessed: 05/28/2013].

Hynninen, T. (2012). M.Sc. Thesis. *First-Person Shooter Controls on Touchscreen Devices a Heuristic Evaluation of Three Games on the Ipod Touch*.

iTunes (2013a). Jetpack joyride. https://itunes.apple.com/us/app/jetpack-joyride/id457446957?mt=8. [Last accessed: 05/26/2013].

iTunes (2013b). Monsters inc. run. https://itunes.apple.com/us/app/monsters-inc.-run/id555724449?mt=8. [Last accessed: 05/26/2013].

Litobyte Softworks (2013). Native vibrator plugin. http://u3d.as/content/litobyte-softworks/native-vibrator-plugin/2rd. [Last accessed: 05/26/2013].

Liukkonenl, T. N. (2009). Human Reaction Times as a Response to Delays in Control Systems.

McCraty, R. (2006). Emotional Stress, Positive Emotions and Psychophysiological Coherence. *Emotional Stress, Positive Emotions and Psychophysiological Coherence*.

Modify.com (2013). Sony dualshock 3 picture. http://www.modiify.com/store/images/watermarked/detailed/0/black1.jpg. [Last accessed: 03/29/2013].

NerByte GmbH (2013). Lep's world. http://www.lepsworld2.com/. [Last accessed: 05/26/2013].

Obrist, M., Seah, S. A., and Subramanian, S. (2013). Talking about tactile experiences. In *Talking about tactile experiences*, page 7. CHI'13.

Pradhan, L. and Islam, M. S. (2010). Replacing outliers with existing data in Inter Beat Interval Signal for Heart Rate Variability analysis. *Replacing outliers with existing data in Inter Beat Interval Signal for Heart Rate Variability analysise*.

Samsung (2013). Samsung electronics co., ltd. http://www.samsung.com/global/galaxys3/. [Last accessed: 05/26/2013].

Sharp, H., Rogers, Y., and Preece, J. (2007). *Interaction design : beyond human-computer interaction*. Hoboken, NJ : Wiley.

Sony (2013). Sony computer entertainment europe, playstation 3 controller. http://dk.playstation.com/ps3/. [Last accessed: 05/26/2013].

Technologies, U. (2013). Unity technologies, unity 3d. http://unity3d.com/. [Last accessed: 05/26/2013].

Vogel, D. and Baudisch, P. S. (2007). A technique for operating pen-based interfaces using touch. In *A Technique for Operating Pen-Based Interfaces Using Touch*, pages 657–666. CHI'07.

Wigdor, D. and Wixon, D. (2011). *Brave NUI World: Designing Natural User Interfaces For Touch and Gesture*. Morgan Kaufmann.

Wikipedia (2013a). Platform games. http://en.wikipedia.org/wiki/Platform_game. [Last accessed: 03/07/2013].

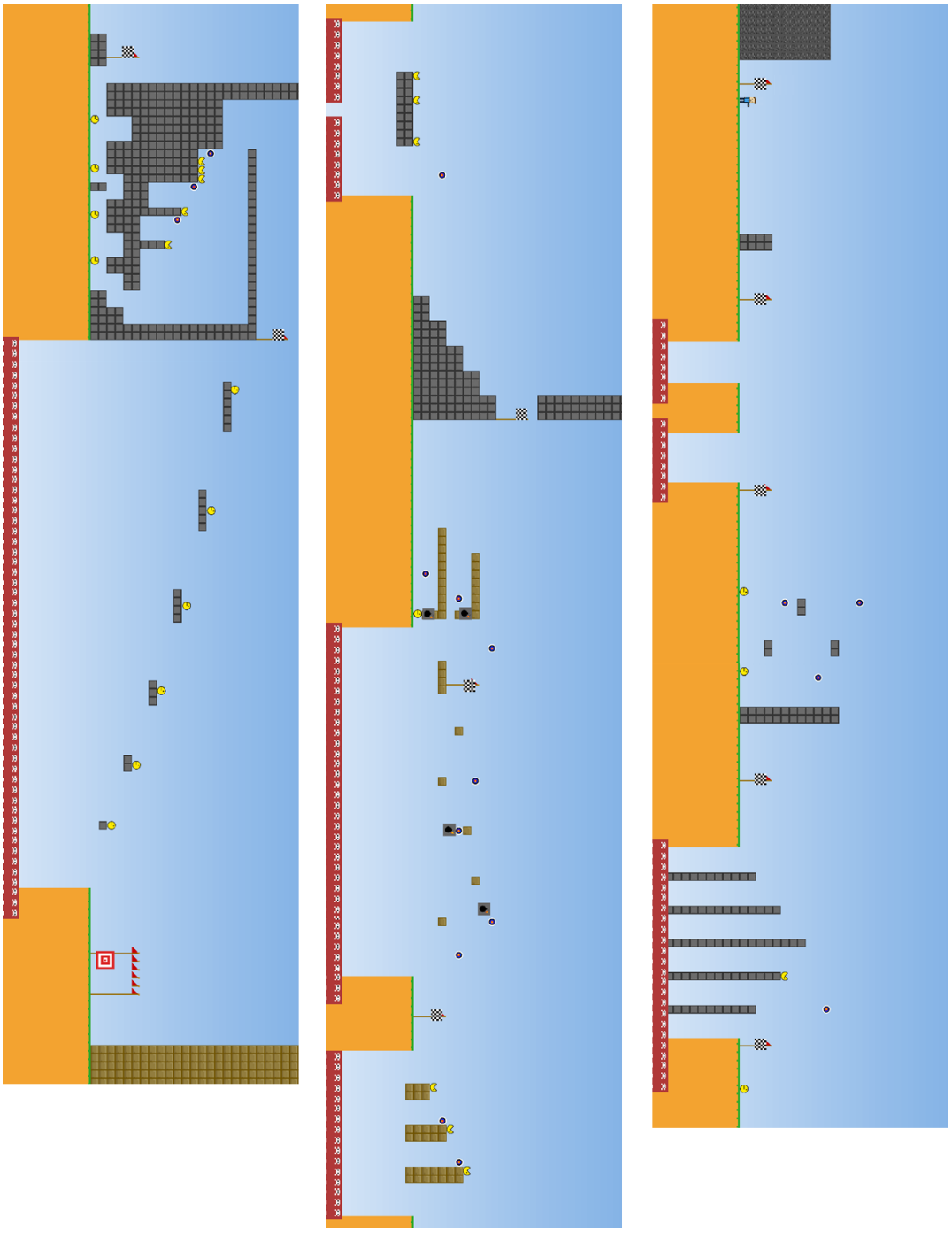
Wikipedia (2013b). Smooth pursuit eye movement. http://en.wikipedia.org/wiki/Smooth_pursuit. [Last accessed: 05/26/2013].

Wikipedia (2013c). Sony dualshock 3. http://en.wikipedia.org/wiki/Dualshock_3#DualShock_3. [Last accessed: 05/26/2013].

Wikipedia (2013d). Super mario bros. http://en.wikipedia.org/wiki/Super_Mario_Bros. [Last accessed: 03/07/2013].

Zaman, L., Natapov, D., and Teather, R. J. (2010). Touchscreens vs. traditional controllers in handheld gaming. In *Futureplay '10 Proceedings of the International Academic Conference on the Future of Game Design and Technology*, pages 183–190. ACM Press.

11 Appendices



Task Order	Gender?	Age?	Handedness?	On average, how many hours do you weekly spend on PC games?	On average, how many hours do you weekly spend on console games?	On average, how many hours do you weekly spend on smartphone games?	Interface 1 - Standard	Comments	Interface 2 - Physical	Comments	Interface 3 - Gesture	Comments	Interface 4 - Vibration	Comments	Interface 5 - Mixed	Comments
1 5 3 4 2	Male	23	Right-handed	6	<1	0	6	jumping felt delayed.	10	smooth	4	with no tactile or visible feedback when changing directions or trying to do so. control feels very soft	5	as good as control gets on a touch screen - probably	2	swipe-jump takes a lot of time to get used to. Shooting was difficult in stressed situations.
1 4 3 5 2	Female	23	Right-handed	10	1	0	5		8		2		8		4	
3 1 4 2 5	Male	25	Right-handed	20	0	2	8	jump bug	10	seems more responsive with the controller.	2	simple touch/not touched would have been better	9	terrible slow jumping is done.	1	terrible slow responsive jump.
2 4 3 5 1	Male	22	Left-handed	0	0	7	9	Gik godt. Måske har jeg vænnet mig til det?	7	Han hoppede ikke altid når jeg synes han skulle	1	kunne slet ikke koordinere at skyde samtidig med bevægelse/hop. Stadig ikke hop når jeg synes han skal hoppe. Han hopper også "efter"	6	Hopper ikke når jeg synes han skal	2	Skulle hele tiden tænke over koordinationen mellem hænderne. Motiverede skyd bedre end gestures, men sikrede ofres ved et uheld.
								missing feedback from buttons, which made it easier to misplace them.				Like the idea, but it is extremely hard move a few pixels. The jumping and shooting part worked fine.				I would have like this mixed interface to be a combination of the button in the left side for walking and the gesture on the right side for jumping/shooting.
4 1 3 2 5	Male	24	Right-handed	7	2	0	5		9		3		7		3	
								Only problem was to learn how to time the jumps.		With out a doubt the best. I'm used to the PS controller and nothing beats Hard to time				I would like to put it somewhere between 1 and 10, because it's too Classic, but not as good as the controller. But the vibration did A LOT.		Had control, but was forced to shoot every time i wanted to move. Sometimes you started swiping inside one of the arrow, which resulted in death
5 1 3 2 4	Male	24	Right-handed	7	7	1	9		10	tactile buttons. Much easier than 3 and 4. It was no problem to time running jumps, and switching directions from left to right while standing still was possible, rather than running off	5	tactile buttons. It was rather difficult to time running to the side and jump. Sometimes it was better simply to jump straight and afterwards, however that didn't	9		4	sometimes. Dear God make it stop. Worst method hands down. Very confusing as the mechanisms controlling the jump changed from left to right, depending on which way you wanted to go or
										incidently off thin poles.		allow as long a jump.				shoot.
3 4 2 1 5	Male	25	Right-handed	10	0	1	9	a while.	10		7	the jump swipe caused trouble because it was not the motion but the release after the motion that seemed to make	9	button didnt work.	5	Even though i had trouble with swiping to jump earlier, i know found it much easier, must be
								felt confident using these controls for a		not controls im used to, but i do have controller experience so i felt				still had a little trouble timing the jumps. but was able to adapt		
3 4 5 2 1	Male	30	Right-handed	20+	0	0	9	second time	9	mostly like playing on home field	3	the character jump	7	Quicker than 3 Det var	8	adaptation
3 2 4 5 1	Male	27	Right-handed	8	0	1	8		8	Det føles godt	6		9	overraskende godt	5	
2 5 4 3 1	Male	23	Right-handed	0	0	0	7	Hoppeknappen var rimelig broken. den reagerede ikke hvor man trykkede hvis man kilkekde på bevæge sig fremad og hop, så virkede den ikke. Hvis man ikke hoppe så virkede den fint, eller omvendt så	8	Det føles godt	5		7	Det var	3	
										DEr var stadig lidt problemer med at hoppe nogen gange. ellers virkede det meget		Synes stadig der var problemer med		Det var meget mere behagelig at skyde med vibrationer. men det begyndte at irritere lil sidst. Det		Det var sjovt. det var svært i starten, men det blev nemt. en af dem der var bedst . Det var lidt
1 2 3 4 5	Male	22	Right-handed	6	4	3	5	meste. Shots seemed timed not when i wanted to shot but	8	flydende.	6	hop	6	var for kraftige vibrationer.	7	sværere at skyde. det var irriterende.
2 4 1 3 5	Male	23	Right-handed	20	1	0	8	a little after.	10		1		5	Vibration... why....	2	det var irriterende.
Average:		**24.25**		**8.55**	**1.36**	**1.25**	**7.33**		**8.92**		**3.75**		**7.25**		**3.83**	

Code Implementation

This section will describe how the game was implemented and code examples will be explained to the most interesting parts of the prototype. The game was made using Unity3d [Technologies, 2013] that enables quick prototyping and easily porting to a mobile device.

Player controls:

The player controls script contains the code for the different interfaces. Together with Unity's input manager, the script makes sure the player can be controlled the same way for all the interfaces.

At the beginning of the game, the static enum 'CurrentInterface' is set from the previous scene (the menu):

```
if (CurrentInterface == ControllerType.Gestures)
        {
            stdOnSreenButtons.SetActive(false);
            MixedInterface.SetActive(false);
        }
    ....
```

The program is then able to diffientiate between the checks needed for each interface later in the code.

The player gameobject uses Unity's character controller, which helps in detecting collision with the game world. If rigidbodies were used instead, the player might translate into colliders or out of the game world. In the function 'ListenForControlInput' the controls and gravity are applied to the player and if the player shoots or jump, it is detected in this function.

```
  if (_playerCharacterController.isGrounded && !isDead)
        {
            _moveDirection = new Vector3(ReturnInput(ControllerInput.
                RightLeft) * PlayerSpeed, 0, 0);

            // Add playerjump to the y component aka. jump
            if (((Input.GetButton("Jump")) || _onScreenJump) && !
                _physicalJumpLock)
            {
                _moveDirection.y = PlayerJumpSpeed;
                _physicalJumpLock = true;
                playerinfo.AddJump();
            }
            else if (!Input.GetButton("Jump"))
            {
                _physicalJumpLock = false;
            }
            _onScreenJump = false;
        }
```

First it checks if the player is on the ground and is not dead. '_moveDirection' is the vector that is applied to the player every frame. The 'ReturnInput()' function returns either a -1, 0 or 1 which will let the player go to the left, stand still or go to the right. While the player is still on the ground the function checks if any jump input is detected from either the Input.GetButton() (which is from a controller) or from the _onScreenJump (which is the virtual buttons). If a jump is detected, the _moveDirection vector's y component is changed to the predetermined jump height. This will make the player instantly jump that amount in the air. A lock is set to true so that is can not be used again before the player is on the ground or the button is lifted. The jump count is also increased in the playerinfo gameobject.

When the player shoots the following are run later in the code:

```
if (((Input.GetButton("Shoot") || _onScreenShoot) && ((Time.time -
    _timeSinceLastBullet) > 0.3f)) && !_physicalShootLock)
        {
            if (!isDead)
            {
                GameObject tmpbullet = Instantiate(FireBall) as GameObject;
                Projectile tmpProjectile = tmpbullet.GetComponent<Projectile
                    >();

                // bullet speed
                tmpProjectile.bulletspeed = 40000f;

                // Fly time before destroy
                tmpProjectile.bulletDelay = 3f;

                // Change players arm
                _playerRightArmShooting.SetActive(true);
                _playerRightArmIdle.SetActive(false);

                // check if bullet should go left or right
                if (_isFacingRight)
                {
                    tmpProjectile.moveRight = true;
                    // Put bullet in front of player
                    tmpbullet.transform.position = new Vector3(transform.
                        position.x + 20, transform.position.y+30, transform.
                        position.z);
                }
                else
                {
                    tmpProjectile.moveRight = false;
                    tmpbullet.transform.position = new Vector3(transform.
                        position.x - 20, transform.position.y+30, transform.
                        position.z);
                }
                _timeSinceLastBullet = Time.time;
                _physicalShootLock = true;
                //  _shootLock = true;
```

```
33              }
34          }
35          else if (!Input.GetButton("Shoot") || _physicalShootLock)
36          {
37              if (!isDead)
38              {
39                  _physicalShootLock = false;
40
41                  // Change players arm
42                  _playerRightArmShooting.SetActive(false);
43                  _playerRightArmIdle.SetActive(true);
44              }
45          }
46          _onScreenShoot = false;
```

First is checks whether input has been received from the controller or from the virtual buttons. It also checks if enough time (0.3 secs) has run since the last fire (to avoid spamming the fire button). If the player is not dead, a bullet is instantiated and speed plus fly time before destruction is set. To make it look like the player is shooting the first arm of the character is disabled while the shooting arm is activated. The direction of the bullet is determined by checking the beforementioned _moveDirection. If the x component is bigger than zero the '_isFacingRight' boolean is set to true. The following contains the rest of the ListenForControlInput script:

```
1
2           if (!isDead)
3           {
4               if (!_playerCharacterController.isGrounded)
5               {
6                   // Move in the air!
7                   _moveDirection.x = ReturnInput(ControllerInput.RightLeft) *
                        PlayerSpeed;
8               }
9           }
10
11          // If he hits something above him
12          if (_playerCharacterController.collisionFlags == CollisionFlags.
                Above)
13          {
14              _moveDirection.y = 0;
15              _moveDirection.y -= afterHitForceDown;
16          }
17
18          if (!isDead)
19          {
20              // Face the right direction
21              if (ReturnInput(ControllerInput.RightLeft) > 0)
22              {
23                  transform.localScale = new Vector3(Mathf.Abs(transform.
                        localScale.x), transform.localScale.y, transform.
                        localScale.z);
```

```
24                  _isFacingRight = true;
25              }
26              else if (ReturnInput(ControllerInput.RightLeft) < 0)
27              {
28                  transform.localScale = new Vector3(-Mathf.Abs(transform.
                        localScale.x), transform.localScale.y, transform.
                        localScale.z);
29                  _isFacingRight = false;
30              }
31          }
32
33          if (extrajump)
34          {
35              _moveDirection.y += PlayerJumpSpeed*2f;
36              extrajump = false;
37          }
38
39
40          // add the gravity
41          _moveDirection.y -= PlayerGravity * Time.deltaTime;
42          _moveDirection.y = Mathf.Clamp(_moveDirection.y, -1000f,
                PlayerJumpSpeed);
43
44          if (!isDead)
45          {
46              _playerCharacterController.Move(_moveDirection * Time.deltaTime)
                    ;
47          }
48      }
```

For the rest of the script, it checks if the player is not grounded and allows the player to move in the air. The _moveDirection.x component is changed as if it had been grounded. To make it more realistic when jumping up and hitting a collider, the script checks the charactercontroller if its CollisionFlags.Above is true. If it is, it means that the player jumped into something in the ceiling and the y component of the _movementDirection is first set to 0 and then a negative force is applied to it. The scripts also checks if the player transform is facing the right direction. If the ReturnInput function is larger than 0 the player's localscale x component is set to a positive value (as the player is facing right in this direction). If the value is less than 0, the player is going to the left and the localscale x component is set to always have a negative value, causing the player to look to the left.

The script also checks if extrajump is enabled, which occurs if the player jumps onto an enemy. Last, gravity is applied to the _moveDirection vector and the _moveDirection is finally applied to the player gameobject. The Time.deltatime is multiplied to add the values per second instead of per frame.
Gestures: When the gestures interface is selected, the function Gestures() is run. The function runs inside inside a loop that runs through all possible finger touches.

```
void Gestures()
    {
        if (Input.touchCount > 0)
        {
            for (int i = 0; i < Input.touchCount; ++i)
            {
      ...
```

This is done as when the fingers are touching the screen, the order of which they are placed are the order of number id's they will receive. So running through the fingers every frame enables the possibility to control the output for the different fingers.

First of, the gestures have been designed to have the left and right movement controls on the left side of the screen and jumping and shooting on the right side of the screen. The left side is checked first:

```
                if (Input.GetTouch(i).position.x < Screen.width / 2f)
                {
                    float Velocity = Mathf.Abs(Input.GetTouch(i).position.x
                       - GesturePrev.x);
                    if (Input.GetTouch(i).phase == TouchPhase.Began)
                    {
                        GesturePrev = Input.GetTouch(i).position;
                    }

                    if (Velocity > ThresholdForLeftRightGesture)
                    {
                        // If right
                        if (Input.GetTouch(i).position.x > GesturePrev.x)
                        {
                          //  Debug.Log("Right");
                            OnScreenLeftUp();
                            OnScreenRightDown();
                            GestureRightFinger = Input.GetTouch(i).fingerId;
                        }
                        // if left
                        if (Input.GetTouch(i).position.x < GesturePrev.x)
                        {
                          //  Debug.Log("Left");
                            OnScreenRightUp();
                            OnScreenLeftDown();
                            GestureLeftFinger = Input.GetTouch(i).fingerId;
                        }
                    }

                    if (Input.GetTouch(i).phase == TouchPhase.Ended)
                    {
                        OnScreenRightUp();
                        OnScreenLeftUp();
            GestureLeftFinger = -1;
```

```
                    GestureRightFinger = -1;

                }

                GesturePrev = Input.GetTouch(i).position;

            }
```

Here the area is only tracked if it is inside the left area of the screen. It finds the velocity by comparing the current position with the last position of the finger that is being used. GesturePrev is set to a vector with zero components before the code is run and when the first finger is placed the position is recorded. The position is recorded at the end of this segment every frame. The reason for doing this during the placement of the finger, is that if the finger is lifted and placed another place, the character would move as if the previous position worked as a reference point, which is not intended. The velocity's direction in the x axis is checked for either right or left and the corresponding functions is run. When the finger is lifted, the buttonup functions are run to reset the button. Two varibles are set to -1 in order to avoid a bug in the next part of the code. The right side of the screen that controls jumping and shooting is set here:

```
            if (Input.GetTouch(i).position.x > Screen.width / 2f)
            {
                // Avoiding bug when going over half the screen with
                    left right
                if (Input.GetTouch(i).fingerId == GestureRightFinger)
                {
                    OnScreenRightUp();
                }
                if (Input.GetTouch(i).fingerId == GestureLeftFinger)
                {
                    OnScreenLeftUp();
                }

                if (Input.GetTouch(i).phase == TouchPhase.Began)
                {
                    GestureJumpPrev = Input.GetTouch(i).position;
                    GestureShootTapStartPos = Input.GetTouch(i).position
                        ;
                }
                float Velocity = Mathf.Abs(Input.GetTouch(i).position.y
                    - GestureJumpPrev.y);

                if (Velocity > ThresholdForJumpGesture)
                {
                    if (Input.GetTouch(i).position.y > GestureJumpPrev.y
                        )
                    {
                        OnScreenJumpDown();
                    }
                }
```

```
27                      OnScreenJumpUp();
28
29                      if (Input.GetTouch(i).phase == TouchPhase.Ended)
30                      {
31                          OnScreenJumpUp();
32                      }
33
34                      GestureJumpPrev = Input.GetTouch(i).position;
35
36                      // Shoot
37                      if (Input.GetTouch(i).phase == TouchPhase.Ended && (
                            Mathf.Abs(Input.GetTouch(i).position.magnitude-
                            GestureShootTapStartPos.magnitude)) <
                            ThresholdTapGesture)
38                      {
39                          OnScreenShootDown();
40                          OnScreenShootUp();
41                      }
```

The first part of the script checks if the fingerid is the same as the current right or left variable. If it is, it means that the finger from the left side of the screen has crossed half the screen, which is not allowed. The action is reset. Afterwards the script checks the velocity in the y direction and if the velocity is big enough, a jump will be triggered (only during an upwards velocity). Shooting happens if the user lifts his finger and has not moved over a certain threshold since he placed the finger. In the mixed interface, gestures are also used. The interface uses a function that only contains the jump and shoot from the above code and are run instead when the mixed interface is selected.

Raycast hack:

The buttons in 2d toolkit works as intended when they are used for pressing. But if the user places his finger outside the buttons and swipes over the button, it is not activated. To resolve this a hack was made. Since there are similar code for the left and right button, only the right button will be displayed. The code runs in a loop that runs through all fingers every frame. There are three segments of the function. The first segment checks if the finger enters the button area:

```
1  if (Physics.Raycast(ray, out hit))
2                  {
3                      // If the raycast hits one of the btns, we save the
                          fingerID. The fingerid can be used to check if the
                          finger is lifted.
4                      if (hit.transform.name == "btn_right")
5                      {
6                          lastfingerOnRight = Input.GetTouch(i).fingerId;
7
8                          RayCastLock_left_up = false;
9
10                         if (!RayCastLock_right)
11                         {
12                             OnScreenRightDown();
```

```
                RayCastLock_right = true;
                RayCastLock_right_up = false;
            }
        }
```

If it hits the button area, the finger is saved to check if the finger is lifted and the function for right button press down is run. The locks are made to prevent the function from running more than once. The second segment checks if the finger has slipped outside the button:

```

        // if you slide your finger outside the button area
        if (Input.GetTouch(i).fingerId == lastfingerOnRight &&
            hit.transform.name != "btn_right")
        {
            if (!RayCastLock_right_up)
            {
                lastfingerOnRight = -1;
                OnScreenRightUp();
                RayCastLock_right_up = true;
                RayCastLock_right = false;
            }
        }
```

It checks if the fingerid is the same as the last finger on the right button and checks if it is still inside the area. If true, then the last finger is set to -1 (a number it will not ever set itself) to 'reset' it and the button up function are called. The last segment checks if the finger is lifted inside the button:

```

        // if you let go your finger inside the area
        if (Input.GetTouch(i).fingerId == lastfingerOnRight &&
            Input.GetTouch(i).phase == TouchPhase.Ended)
        {
            Debug.LogWarning("Lifted Finger from Right Button");
            lastfingerOnRight = -1;
            OnScreenRightUp();
            RayCastLock_right = false;
        }

    }
```

This segment runs if the finger is lifted and the finger is still the same as the last finger on the right button. If the touch phase is equal to TouchPhase.Ended then it must still be inside the button and thus up functions are called.

Saving Data: For testing purposes, several variables had to be saved to the device, that could later be transfered to a pc. First, all the different variables are saved to a script called PlayerInfo.

```
    public float GameTime;
    public int TargetsDestroyed;
```

```
    public int AmountOfDeaths;
    public int BulletShot;
    public int Jumps;
    public float PlayerLengthTraveled;
    public float EnemiesKilled;
```

The above values are inside the PlayerInfo script. Every time a certain event has been made (e.g jumping) a function is called in the script to increment the certain value. When a checkpoint is reached, the values are saved to a list of strings by calling this line:

```
SavedAtInterval.Add("Checkpoint" + Current_Level_Interval + ";" +
    AmountOfDeaths + ";" + TargetsDestroyed + "; Time: " + GameTime + ";" +
     BulletShot + ";" + Jumps + ";" + PlayerLengthTraveled + ";" +
    EnemiesKilled +"\n");
```

Thus a single string is a checkpoint containing all infomation since last checkpoint. When the player reaches the goal or chooses to quit, a function is called in a script called DataHandler called SavePlayerInfo:

```
public void SavePlayerInfo(bool quit)
 {
     string typeofcontroller = (PlayerControls.CurrentInterface).ToString
         ();
     string TestSubjectnumber = PlayerPrefs.GetInt("Subject").ToString();
     string ifquit;
     if (quit)
     {
         ifquit = "_quitted";
     }
     else
     {
         ifquit = "";
     }
     string Filename = "PlayerInfo_" + typeofcontroller + ifquit + ".csv"
         ;

     // If the folder for this subject doesn't exist, then make it
     if (!System.IO.Directory.Exists(PathToSDCard + "/Subject" +
         TestSubjectnumber))
     {
         System.IO.Directory.CreateDirectory(PathToSDCard + "/Subject" +
             TestSubjectnumber);
         Debug.Log("A folder was created at " + PathToSDCard + "/Subject"
              + TestSubjectnumber);
     }


     if (!File.Exists(PathToSDCard + "/Subject" + TestSubjectnumber + "/"
          + Filename))
     {
```

```
26
27              File.AppendAllText(PathToSDCard + "/Subject" + TestSubjectnumber
                    + "/" + Filename, ";Deaths;TargetsDestroyed;Time;Bulletsshot
                    ;Jump;LengthTraveled;EnemiesKilled" + "\n");
28
29              foreach (string k in playerinfo.SavedAtInterval)
30              {
31                  File.AppendAllText(PathToSDCard + "/Subject" +
                        TestSubjectnumber + "/" + Filename, k);
32              }
33
34          }
35      }
```

The function recevies a boolean, to tell whether or not the player quitted. If he did, '_quitted' is appended to the filename. When the script is first run it checks if folder exists on the sdcard - if it does not, it will create a folder to the corresponding test subject number. The test subject number comes from Unity's PlayerPrefs, which can be used to store variables even after the application has quitted. Afterwards it checks if the file already exist, which it should not, unless the user went through the whole level again. However as a safeguard, the script will increment the testsubject number and add the strings to a new file. If the file did not already exist, all the strings are saved to a file, with file naming so that it can be seen what type of controller is used and if the user quitted.

Enemies

There are three different enemies in the game. The pacman that runs from side to side, the pacman that appears up and down and a bullet that flies from the right. The player are able to jump on two of these in order to destroy them, but when using a charactercontroller, physic collisions can not be checked as if it was rigidbodies. The following will describe how the three different enemies work and how to overcome the collision detection problem.

Pacman ground: This enemy's function is to run to left or right, until it hits something and then turn to the other direction. The enemy script has a function to translate the gameobject a certain speed every frame, which it does continually. It also uses a charactercontroller, which means that only some collisions can be checked. To check if it has to turn around, this code is run:

```
1       void OnControllerColliderHit(ControllerColliderHit hit)
2       {
3           CharacterController ccontrol = GetComponent<CharacterController>();
4           if (ccontrol.collisionFlags == CollisionFlags.CollidedSides)
5           {
6               if (MoveRight)
7               {
8                   MoveRight = false;
9               }
10              else
11              {
12                  MoveRight = true;
```

```
            }

        }
    }
```

The above code checks if there is a collision on the side of the enemy. If there is, the MoveRight boolean is changed to either false or true. The orientation of the enemy is left startup, so moveright is false from the beginning. This code is not optimal, as any collision will trigger it to go side to side.

There was a problem of using the character controllers as the hit variable did not activate the player object properly. Another solution was made instead. Two colliders was attached to the enemy object. One box for the top area of the enemy that was intended for jumping in (to kill the enemy). Another box collider at the bottom (and a bit to the sides) to check if the player did not jump on the enemy. As the enemy script can not check collisions on several child objects, two scripts were made for the child objects, which activates two different functions in the parent gameobject. If the bottom is hit by the player, the player is killed (and a respawn function is called in the player controls). If the above collider is hit first, the enemy is killed and a extra jump factor is added to the player's movement.

Pacman up down: The function of this enemy is to hide for a certain period and then appear. The code is as following:

```
void Update ()
    {
        if (moveup)
        {
            transform.position = Vector3.Lerp(transform.position, moveto,
                speed * Time.deltaTime);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, movefrom,
                speed * Time.deltaTime);
        }
 }

    IEnumerator UpDown()
    {
        if (useonce)
        {
            yield return new WaitForSeconds(StartTimeOffset);
            useonce = false;
        }
        moveup = true;
        yield return new WaitForSeconds(TimeOffset);

        moveup = false;

```

```
25          yield return new WaitForSeconds(TimeOffset);
26
27          StartCoroutine(UpDown());
28          yield break;
29      }
```

The moveto and movefrom are public variables, with positions that can be seen in the editor (because of a DrawGizmo function). In the beginning of the game in the start() function, the IEnumerator UpDown() is run. First the start of the function is delayed with a certain amount (that can be changed in the editor) if an offset is desired from other enemies of the same type. Then moveup is set to true, which will let update move the gameobject to the 'moveto' position, until enough time has passed. When enough time has passed, the boolean is set to false, the function is paused for a certain time and a new corountine of the same function is run. This makes the enemy move up and down until the game is stopped. The enemy also has a trigger on a child gameobject to check if the player hits it. If the player hits, the player dies.

Bullet: The bullet almost has the same functions as the pacman on the ground, except that it will fly to the left for a certain amount of time. The amount of time it moves, depends on the bullet spawner that shoots the bullets.

```
1       IEnumerator SpawnBullet()
2       {
3           if (!once)
4           {
5               yield return new WaitForSeconds(StartTimeOffset);
6               once = true;
7           }
8
9           GameObject tmp = Instantiate(Bullet) as GameObject;
10          tmp.GetComponent<EnemyBullet>().Speed = bulletSpeeds;
11          tmp.GetComponent<EnemyBullet>().flytime = BulletFlyTime;
12          tmp.transform.position = transform.position;
13
14
15          yield return new WaitForSeconds(BulletSpawnInterval);
16          StartCoroutine(SpawnBullet());
17          yield break;
18
19      }
```

When the game starts, this function is called from the BulletSpawner script, which can be set with different startoffset, timeoffset and bullet speed and fly time.

Vibration: Since Unity can only create a 500 miliseconds vibration for phones (due to iPhone limitations) a way of accessing the native android was done by using Native Vibrator Plugin [Litobyte Softworks, 2013] from the Unity asset store.