Designing Touch Interaction for a 3D Game

Mattias Bergström

January 31, 2015
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Mikael Rännar
Examiner: Anders Broberg

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

The marketshare of touch devices is increasing, and so is the number of games on that platform. As touch devices have become almost as powerful as game consoles, games from the PC and console platforms are being adapted to these touch devices. Console and PC games often feature quite complex controls which can be difficult to implement on touch devices.

This thesis details the design, implementation, and evaluation of methods of interacting with a 3D game on touch devices. The project was performed at Dohi Sweden on the game MilMo, which is a 3D game currently available on the web.

The results of this projects was two prototypes that were evaluated using both user tests as well as heuristic evaluation. The evaluation show that one of the prototypes was preferred because it was more similar to other existing solutions.
4.5.2 Heuristic Evaluation ........................................... 22

5 Results .............................................................. 25
  5.1 Design ......................................................... 25
    5.1.1 Early Prototypes ........................................... 25
    5.1.2 Prototype 1 ............................................... 26
    5.1.3 Prototype 2 ............................................... 26
  5.2 Implementation ............................................... 27
    5.2.1 Gesture Recognition ..................................... 27
    5.2.2 Virtual Thumbstick ...................................... 29
  5.3 Evaluation Results ........................................... 30
    5.3.1 Usability Testing ........................................ 31
    5.3.2 Heuristic Evaluation .................................... 31

6 Conclusions ........................................................ 35
  6.1 Discussion ................................................... 35
  6.2 Limitations .................................................. 35
  6.3 Future work .................................................. 35

References .......................................................... 39
Chapter 1

Introduction

As the performance of tablet devices have increased, advanced 3D games have become popular on this platform. Aside from putting a high demand on graphics performance, these type of games often feature quite complicated controls that can be tricky to implement on a tablet device. The user can often move their character, interact with the environment (jump, pick up items, etc.) and move the camera in these types of games. On personal computers this is often solved using a combination of keyboard buttons and the computer mouse, but games on tablet devices are expected to be playable without keyboard and mouse. Because of this, complex controls often need to be adjusted or simplified to better fit a tablet device.

MilMo\textsuperscript{1} is a Massive Multiplayer Online (MMO) game developed by Dohi Sweden\textsuperscript{2}. MilMo is built on Unity\textsuperscript{3}, a game engine with support for multiple platforms, including iOS and Android. Dohi Sweden wants to investigate how MilMo could be brought to tablet devices and a big part of this is figuring out how to control the game without keyboard and mouse.

1.1 Outline

- **Chapter 1 Introduction:**
  A short introduction to the thesis.

- **Chapter 2 Problem Description:**
  A description of the problem, the goals and the purpose of the thesis work.

- **Chapter 3 Interaction in 3D Games on Touch Devices:**
  In-depth study on methods of interacting with 3D games on touch devices, but also how to evaluate games in particular.

- **Chapter 4 Accomplishment:**
  This chapter describes the work process and explains why some choices were made.

- **Chapter 5 Results:**
  The results of the implementation and evaluation is presented in this chapter.

\textsuperscript{1}https://www.milmogame.com/
\textsuperscript{2}http://www.dohi.se/
\textsuperscript{3}https://unity3d.com/
– Chapter 6 Conclusions:
  In this chapter limitations of this work and future work is discussed.
Chapter 2

Problem Description

In this chapter the problem statement is presented as well as the goal and purpose. The methods to achieve this is also described.

2.1 Goals and Purposes

The goal of this thesis is to develop a method of interacting with MilMo on touch devices. The interaction method should meet the needs of interaction methods for games, meaning that it should be easy to learn, and hard to make mistakes. It is also very important that it is fun to play using the interaction method. To accomplish this, the following sub goals will be completed:

– Study the current solution and determine how it can be adapted to touch devices. Previous work should also be evaluated.

– Prototypes should be built based on the study.

– These prototypes should then be evaluated.

The purpose of this thesis project is to develop a method of interaction for MilMo that works well on touch devices. This is one step on the many required to bring MilMo to the tablet market.

2.2 Methods

The project is divided into three steps, where each step is completed using different methods:

– The first step is the study, this will be performed by studying previous work, and how the interaction with the game currently works, and creating a list of what is important to focus on.

– The second step is building prototypes. These will be built in Unity with C# and integrated into the MilMo project. Building the prototypes in MilMo instead of as separate projects makes sure the interaction methods actually work in the game.
– The third and last step is evaluation. The prototypes will be evaluated using two methods: Heuristic evaluation and usability tests in the form of play testing while using the “think aloud”-method followed by a short interview.
Chapter 3

Interaction in 3D Games on Touch Devices

This chapter is a study in interacting with 3D games on touch devices. The goal is to gain insight into how to design interaction for 3D games. The chapter starts out with defining usability and how to evaluate the usability of games. Further, interaction with direct manipulation and gestures is studied, and the advantages and disadvantages as well as means of implementing these methods are presented. At the end of the chapter, different metaphors used for navigating 3D space and examples of where these are found in games are studied.

3.1 Usability and Evaluation of Games

Nielsen[19] divides usability into five components, learnability, efficiency of use, memorability, errors, and satisfaction. Nielsen’s definition is probably the best known definition, however, a definition from ISO 9241-11 is becoming the main reference of usability. According to the International Standard ISO 9241-11, usability is a measure of how effectively a user can achieve specified goals in a specified context of use with respect to three aspects: effectiveness, efficiency and satisfaction[18]. The common factor in Nielsen’s model and the ISO Standard’s model of describing usability is that they both break the term usability into aspects that can be evaluated to measure the usability of a product.

The ISO 9241-11 standard defines the aspects of usability as follows[16]:

- **Effectiveness** defines the accuracy and completeness with which users achieve specified goals.

- **Efficiency** defines the resources expended in relation to the accuracy and completeness with which users achieve goals.

- **Satisfaction** defines the freedom from discomfort, and positive attitudes towards the use of the product.

- **Context of Use** defines the users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used.
The design considerations of video games differ from other types of software. Other software may be purchased to perform necessary tasks, but games are purchased primarily for entertainment. If the game is not fun to play, no one will buy it. Therefore, in the case of video games, the satisfaction aspect in the ISO 9241-11 standard must be the primary consideration[10].

Federoff[10] goes on to suggest that other software industries could benefit from having a better understanding of principles in game design, because games often succeed in areas that other software struggle or fail. This may originate from the fact that the satisfaction aspect of usability is so central to game design. Going back to the three aspects of usability; Frøkjær, Hertzum, and Horbaek[12] argue that these aspects should be considered as separate and independent aspects of usability, because all three measures are not equally important or applicable depending on the context.

Effectiveness and efficiency typically measure productivity. These measurements are not useful for games, since they are not meant to be productive, but rather an escape from productivity. Thus, using these measurements in games directly contradict the philosophy of game design, since the goal is not to allow the player to finish the game with as little resources as possible, but rather to spend as much time as possible in the game[10]. However, there are some areas of games where effectiveness and efficiency may be preferred, one of these areas being the user interface in the game.

Satisfaction is the one aspect of usability that is easy to relate to game design philosophies. Federoff[10] suggests this should be central to the evaluation of the usability of games since they aim to entertain, not provide productivity.

Using heuristics to evaluate software is usually quick and inexpensive. The heuristics can be used by usability evaluators to assess the goodness of software design by going through the interface and producing a list of heuristics violations[10]. In game development, this could be used to produce successful games more consistently.

Nielsen’s 10 usability heuristics is often used when performing evaluations on software design[20]. However, some of these do not apply to games and, therefore, alternative heuristics has to be created. Federoff[10] has compiled a list of heuristics of game design, and related them to three areas of usability identified by Clanton[5]. These areas are, Game Interface, Game Mechanics, and Game Play. Game Mechanics and Game Play consists of what the player can do in the game, and the problems and challenges the player faces in the game. Game Interface, on the other hand is the physical interaction with the game, the topic of this study fits right into this area, and will therefore be the focus. The following list are the heuristics within the Game Interface area compiled by Federoff[10]:

- Controls should be customizable and default to industry standard settings
- The interface should be as non-intrusive as possible
- A player should always be able to identify their score/status in the game
- Follow the trends set by the gaming community to shorten the learning curve
- Interfaces should be consistent in control, color, typography, and dialog design
- For PC games, consider hiding the main computer interface during game play
- Minimize the menu layers of an interface
- Minimize control options
- Use sound to provide meaningful feedback
- Do not expect the user to read a manual

Sweetser and Wyeth[26] argue that player enjoyment is the single most important goal for video games and that current heuristics for designing and evaluating games are not focused enough on player enjoyment, but instead focus on the three aspects mentioned earlier: Game
3.2 Multi-Touch Interface, Game Mechanics, and Game Play. They also note that there are many separate heuristics for game design, but they are isolated, repetitive and often contradictory. Because of this, they propose a new model of player enjoyment in games called GameFlow[26].

As indicated by the name, GameFlow is based on flow experiences. Csikszentmihalyi[6] has conducted extensive research into what makes experiences enjoyable. He found that very different enjoyable experiences are described in similar ways, irrespective of social class, age, or gender. He describes flow as an experience “so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it, even when it is difficult or dangerous”[6]. Based on Csikszentmihalyi’s work, GameFlow was constructed as a model of enjoyment in games. It consists of eight elements: Concentration, Challenge, Skills, Control, Clear Goals, Feedback, Immersion, and Social.

Interaction models directly influence the sense of Control that the player feels in a game. The rest of the eight elements may also be influenced by the interaction in one way or another. The player’s Concentration may be dependent on how complicated the controls are, since complex controls have the potential to distract the player from immersing themselves into the game. Complex controls also have the potential of increasing the Challenge of the game by making certain task harder to perform interaction-wise.

To illustrate a hypothetical situation where changing interaction model influences both the challenge and concentration required, imagine a gap between two platforms that the player has to jump over. To accomplish this task the player needs to run towards the gap and press the jump button at the right time, this requires both concentration and provides a challenge to the player. Now imagine another interaction model where jumping is automated. Whenever the player is about to fall off the platform the character automatically jumps. This results in the following: the player does not have to concentrate to not fall off the platforms, and the challenge of correctly timing a jump is also removed.

The GameFlow model includes overall criteria for all eight elements that can be used to design and evaluate games with respect to player enjoyment[26], the goals for the elements of Concentration, Challenge, and Control are shown in Table 3.1.

Sweetser and Wyeth come to the conclusion that some of the GameFlow criteria are more relevant in some game genres while some are less relevant or not applicable at all[26]. Some criteria were shown to be difficult to measure through an expert review and would instead require player-testing. In the current form, GameFlow could be used as a guideline for expert reviews or used as a basis for constructing other types of evaluations. They conclude that future work would involve developing the GameFlow criteria into usable design and evaluation tools for game developers and researchers. Alternative models like Pervasive GameFlow[17], EGameFlow[13] and RTS-GameFlow[7] have been developed based on GameFlow to accommodate to the differences between genres.

3.2 Multi-Touch

Multi-touch technology offers multiple interaction advantages compared to other interaction mechanics, there is a notion that the interaction is more “natural” and “compelling”[3]. In contrast to other mechanisms, when using touch input the user moves and interacts with the interface elements in a natural fashion giving the user the impression of “gripping” real objects. This form of interaction is called direct manipulation. Multi-touch interaction allows for multiple fingers to be used simultaneously, which allows for a wider diversity of gestures compared to technologies which only allow one finger at a time.

One particular weakness of touch devices is that it is difficult to provide feedback to the user. There is no way for users to utilize their fingers to feel where buttons are on the screen
<table>
<thead>
<tr>
<th>Element</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concentration</td>
<td>- games should provide a lot of stimuli from different sources</td>
</tr>
<tr>
<td></td>
<td>- games must provide stimuli that are worth attending to</td>
</tr>
<tr>
<td></td>
<td>- games should quickly grab the players’ attention and maintain their focus throughout the game</td>
</tr>
<tr>
<td></td>
<td>- players should not be burdened with tasks that do not feel important</td>
</tr>
<tr>
<td></td>
<td>- games should have a high workload, while still being appropriate for the players’ perceptual, cognitive, and memory limits</td>
</tr>
<tr>
<td></td>
<td>- players should not be distracted from tasks that they want or need to concentrate on</td>
</tr>
<tr>
<td>Challenge</td>
<td>- challenges in games must match the players’ skill levels</td>
</tr>
<tr>
<td></td>
<td>- games should provide different levels of challenge for different players</td>
</tr>
<tr>
<td></td>
<td>- the level of challenge should increase as the player progresses through the game and increases their skill level</td>
</tr>
<tr>
<td></td>
<td>- games should provide new challenges at an appropriate pace</td>
</tr>
<tr>
<td>Control</td>
<td>- players should feel a sense of control over their characters or units and their movements and interactions in the game world</td>
</tr>
<tr>
<td></td>
<td>- players should feel a sense of control over the game interface and input devices</td>
</tr>
<tr>
<td></td>
<td>- players should feel a sense of control over the game shell (starting, stopping, saving, etc.)</td>
</tr>
<tr>
<td></td>
<td>- players should not be able to make errors that are detrimental to the game and should be supported in recovering from errors</td>
</tr>
<tr>
<td></td>
<td>- players should feel a sense of control and impact onto the game world (like their actions matter and they are shaping the game world)</td>
</tr>
<tr>
<td></td>
<td>- players should feel a sense of control over the actions that they take and the strategies that they use and that they are free to play the game the way that they want (not simply discovering actions and strategies planned by the game developers)</td>
</tr>
</tbody>
</table>

Table 3.1: Elements and criteria in GameFlow

for example. A lot of devices can vibrate, and this is sometimes used when the user presses buttons on the screen. However, this can get annoying in games if the game requires a lot of inputs.

An inherent disadvantage with touch input is that the fingers obscure the screen while
interacting. In games, this is particularly important to avoid since the user most likely wants to see what is going on in the game. Therefore, the interaction should mainly be kept close to the edges of the screen.

### 3.3 Direct Manipulation

Direct manipulation is a term first introduced by Schneiderman[11]. He referred to it as a certain style of interaction fulfilling the following requirements[25]:

1. Continuous representation of the object of interest
2. Physical actions or labelled button presses instead of complex syntax
3. Rapid incremental reversible operations whose impact on the object of interest is immediately visible.
4. Layered/Spiral approach to learning. Permits usage with minimal knowledge and expansion of knowledge as familiarity with system increases.

Direct manipulation is used in many modern user interfaces, some examples include graphical buttons, and draggable objects[3] that the user can interact with directly (see Figure 3.1). One important characteristic of direct manipulation is that it provides instant feedback in contrast to gestures, which must be completed before they can be translated into an action[24].

![Figure 3.1: Example of direct manipulation where the user is moving a rectangle by dragging it with a finger.](image)

#### 3.3.1 In Games

This style of interaction lends itself well to games, perhaps mainly because of the layered approach to learning mentioned earlier. It is easy to play around with the interface and learn by doing, instead of needing to read the manual. Another reason why direct manipulation is often used in games might be that a lot of games require precise and fast input. This requires fast feedback, which is one of the advantages of direct manipulation compared to gestures.

Direct manipulation can be seen in multiple games across several genres. Selecting units by tapping on them in strategy games is one example of direct manipulation in games. Another example in strategy games with a top-down view is panning the viewport by dragging a finger across the screen.
3.3.2 Implementing Direct Manipulation

A direct manipulation implementation can be very simple. It can be as simple as triggering an event when a finger tap is recognized. This can be extended to implement dragging functionality by triggering an event every time a finger is moved on the screen.

Based on Buxton’s three-state model[4], Figure 3.2 illustrates a state machine with three states - tracking one, two, or no fingers. State 0 is the initial state, the machine stays in this state until a finger is detected. Given a touch as input, the state machine transitions into State 1 in which it tracks the finger that was detected until the finger is released, or until a second finger is detected. Given that the machine is in State 1 and a second finger is detected, the machine transitions into State 2. From State 2, the machine can transition back into State 1, this happens when one of the two fingers are released.

Applying this state machine to a hypothetical application, State 1 could represent dragging of objects and State 2 could represent pinching to zoom. The user could then use two fingers to scale an object and then release one of them, thereby transitioning into dragging mode and moving the object somewhere.

![State machine that handles one and two finger touch.](image)

3.4 Gestures

Rubine defines a gesture as a handmade mark used to give a command to a computer[24]. In contrast to direct manipulation, gestures do not necessarily have any relation with an object[23], and typically needs to be completed before being classified to execute the intended command. Because of this, it is not possible to do continuous interaction in the same way that is possible with direct manipulation. This makes gestures inappropriate for performing operations that require continuous feedback, such as the drag operations described in the direct manipulation section (see Figure 3.1).

A clarification might be needed for what is considered a gesture. This report considers a gesture to be an interaction method that strictly adheres to Rubine’s definition[24]. Nygård calls this type of gesture a symbolic gesture[23]. Pinching to zoom on touch devices is commonly called a pinch gesture, but according to Nygård[23] it belongs to the category direct gesture and falls into direct manipulation rather than gestures.

Gestures can be as simple as drawing a straight line, typically called swipe gestures, but it can also be more complex shapes. A gesture is not limited to only one finger, one
3.4. Gestures

example being a swipe gesture with multiple fingers. Gestures are often used to simplify tasks, such as avoiding having to dive deep into menu structures[23]. An example of this could be having a save feature where drawing a s-shape performs a save action (Figure 3.3 illustrates an s-shaped gesture).

![Figure 3.3: Example of a gesture. This gesture is performed by drawing a s-shape.](image)

3.4.1 Limitations

The biggest limitation of using gestures is that they usually need to be completed before they can be classified and perform an action[3]. Even the simplest gestures are often not simple enough, for example a simple swipe gesture, where the user moves their finger across the screen in one direction, is more complex than pressing a button[3].

Compared to direct manipulation it is harder to provide good metaphors for gestures, partly because of limitations of what shapes can be classified with an acceptable accuracy[3]. The limited visual feedback with gestures is also a factor.

Norman and Nielsen considers gesture interfaces a step back in usability[22]. One of their main criticisms is that interfaces that depend on gestures lack visual clues, meaning there is nothing visible on the screen indicating possible actions. Based on this, it seems important that the gestures have clear metaphors and visual clues that the users can utilize to remember the gestures, and to perform user tests to establish that the users do in fact understand the interface.

3.4.2 In Games

The usage of gestures can remove the need for buttons and menu structures, which can allow for a deeper immersion and more enjoyable game experience[3].

Since a gesture generally needs to be completed before it is classified, gestures are typically not used for actions that need to be instantly performed. The complexity of the gesture must be considered depending on what action it should perform. Something used very often should perhaps be a simple gesture, while, on the other hand, a complex gesture could be used to represent an action that should be challenging to perform. It is also important to consider how hard the gestures are to remember[3], players should not be expected to look in the manual to be reminded of how to perform an action.

It is important to avoid competing gestures, they need to be easily distinguishable to minimize wrongly classified gestures. This is extra important in games where the user might be under stress, and therefore not drawing the shape perfectly.
3.4.3 Gesture Recognition

Gesture Recognition refers to the process of classifying user input into an intended action. Different methods of classifying gestures can be used depending on how complex the gestures are, and how many possible gestures the user can perform.

*Dot product based recognition* is a fairly simple method of recognizing gestures based on the dot product between two gestures, described in an article by Dopertchouk[8]. A dot product close to 1.0 indicates a close match, while a low or negative value indicate completely different gestures. This assumes that the gesture strokes have been normalized. According to Nygård[23], this method will often have problems separating circles and squares, but notes that this is the price to pay for speed and simplicity.

*Direction based recognition* recognizes gestures based on the direction of the points in the gesture stroke[23]. The algorithm calculates the angle between the current and previous point for each point while the user is performing a gesture stroke, and determines which of the possible directions it is closest to. The possible directions are defined by the implementor, see Figure 3.4 for two examples of possible directions. In the example to the left, there are four possible directions, up, down, left, and right. Nygård[23] suggests dividing the possible directions into 8 zones, that would result in something similar to the example to the right. After the gesture has been completed, the sequence of directions is compared to possible gestures. For example, using the 4-direction example in Figure 3.4, an s-shape could be described as the sequence [Left, Down, Right, Down, Left].

![Figure 3.4: Two examples of possible directions](image)

*Region based recognition* is similar to the direction based recognition but splits the gesture points into regions instead of directions[23]. The resulting chain of regions is then compared to possible gestures. The way this works is that the area where gestures should be performed is split into a grid of regions, see Figure 3.5 for an example of a labeled 3x3 grid. The regions that the users finger travels over are stored as a sequence. This chain is then compared to possible gestures. An s-shape could for example be defined as [C, B, A, D, E, F, I, H, G].

*Hidden Markov Model (HMM) based recognition* is a statistical method, and more complicated than the previously described methods[23], and only a brief description is given here. HMM is widely used in areas such as handwriting, speech, and character recognition[9]. In this method feature sets are extracted from gestures, each of these feature sets describe the essence of a gesture while still allowing for variability in scale, orientation, and style. According to Anderson et al.[2] variability is important for a HMM method to be successful. A classifier is then built which can classify gestures based on the features.
3.5 3D Navigation with Touch

Navigating 3D spaces for computer graphics and visualization applications is a long standing interaction challenge[27]. Touch devices today offer touch input on a flat rectangular surface, this is translated into 2D coordinates with 2 degrees of freedom (DOF) - movement along both the x and y axis. Placing a viewpoint in 3D space has 6 DOF, three for positioning in x, y, and z, and an additional three for the rotation along each axis (see Figure 3.6). This inherently creates problems, since somehow 2 DOF manipulations must be translated into 6 DOF manipulations to fully navigate 3D space with touch controls.

Most games only have 5 DOF at most. It is very rare for games to allow tilting, or more specifically, rotation around the z-axis, to be controlled by the player.

![Figure 3.6: Degrees of Freedom](image)

Illustrates “evolution” from one degree of freedom to two degrees of freedom, and lastly into six degrees of freedom.

3.5.1 3D Navigation Metaphors

Designers often use metaphors when designing complex interactions to ground actions, tasks, and goals in concepts that the user might already understand. This is called a user interface metaphor[15].

Metaphors give users an internal model. Based on this internal model, the user will make assumptions of what can and can not be done in this user interface. Therefore, it is important that the design provides meaningful metaphors that the user can relate to. Ware and Osborne implemented three metaphors for 3D navigation - eyeball in hand, scene in hand, and flying vehicle control[27]. Although these metaphors are rarely used as-is, variations of these are commonly seen in video games.
**Eyeball in Hand**

Eyeball in hand is an egocentric metaphor. What this means is that moving the viewpoint is performed with the user in centre, this is similar to our experience in everyday locomotion[14]. With this technique users can move and rotate the viewpoint in a behavior that is similar to holding and moving around with a video camera (see Figure 3.7).

![Figure 3.7: Eyeball in Hand. x indicates how the camera rotates when the user turns left. The line labeled y indicates the direction the camera is currently looking at.](image)

This metaphor is commonly used in First Person Shooter games. In this category of games, the player can control their character from a first person perspective, meaning they see the world from the characters point of view[1]. *Minecraft - Pocket Edition* is an example of a video game for iOS devices that is played from a first person perspective (see Figure 3.8).

![Figure 3.8: Minecraft - Pocket Edition uses the Eyeball in Hand metaphor](image)

Movement is performed using on-screen buttons on the bottom left corner, by tapping somewhere on the screen the player uses their equipped tool, or punches with their hand. The game has an auto-jump feature, this helps the player move along the uneven terrain. Rotating the viewpoint is performed by dragging a finger in the intended direction somewhere on the screen.
Scene in Hand

Hand describes Scene in Hand as an exocentric metaphor [14]. In contrast to an egocentric metaphor, exocentric metaphors give the feeling of looking in from the outside. In an exocentric metaphor, the manipulated objects are the centre of attention instead of the user. In this metaphor the viewpoint is always looking at a certain point of interest, the user can move this point, and orbit around it (see Figure 3.9).

![Figure 3.9: Scene in Hand. x indicates how the camera moves when the user orbits to the left. y indicates the point of interest.](image)

A variation of this metaphor is common in Third Person games. The player sees their character from outside with the character being the point of interest [1]. In some games, the camera is locked behind the character, but a lot of third person games allow the player to freely orbit around the character.

*Star Wars: Knights of the Old Republic* is a third person game for iOS (see Figure 3.10). In contrast to Minecraft, Knights of the Old Republic has no on-screen buttons for movement. Instead, to move, the player touches the screen and drags forward to move forward and backward to move backward. When standing still, the view can be rotated by dragging left or right. When running, the view is locked behind the character and dragging left or right turns the character in the respective direction.

![Figure 3.10: Star Wars: Knights of the Old Republic uses the Scene in Hand metaphor](image)
Flying Vehicle Control

The flying vehicle control is also an egocentric metaphor. As the name suggests it is similar to controlling a vehicle. This metaphor allows the user to control the forward velocity and steering left and right, and also up and down (see Figure 3.11).

![Flying Vehicle Control Diagram](image)

**Figure 3.11:** Flying Vehicle Control. $x$ indicates a possible path the camera would follow when the user steers to the left. The line labeled $y$ indicates the forward direction.

As the name suggests this metaphor is often used when vehicles are involved. In games, that includes racing games, vehicle simulators, and other games where vehicles are controlled.

*School Driving 3D* (see Figure 3.12) is a car driving game for touch devices that uses the flying vehicle control metaphor as a basis for the interior view. The player can move the car forward and backwards by using on-screen buttons shaped as pedals. There are three alternative controls for turning left and right, the first alternative is two on-screen buttons for left and right. The second alternative is a steering-wheel in the bottom left corner that is controlled by touching and dragging to rotate it. The last alternative is using the accelerometer on the device by tilting it left and right.

![School Driving 3D Interior](image)

**Figure 3.12:** School Driving 3D
3.6 Conclusions

Usability is important in all areas of design. Usability and evaluation methods for games differ from ordinary software. Mainly because in contrast to ordinary software, games are not meant to be productive. The focus should be on making sure the game is fun to play, easy to learn and not cause frustration.

Because touch interaction inherently means the screen will be obscured, games are often designed to keep most of the controls on the edges of the screen. Direct manipulation is often used in games as the primary means of input. Perhaps mainly because the interaction often feels very intuitive, but also that it offers instant feedback, which is often needed in games. Gestures, which it is difficult to provide instant feedback can be used as a complement to remove the need for menus, or on-screen buttons in some cases.

Depending on the type of game, different navigation metaphors may be used. There are three common 3D navigation metaphors that have been adapted for games in multiple ways, eyeball in hand, scene in hand, and flying vehicle control. These are commonly found in first person games, third person games, and racing games respectively.
Chapter 4

Accomplishments

This section describes how the work was planned, and then how the work was actually performed.

4.1 Design Process Overview

The following steps describe the iterative design process used in this project:

– Study the problem. This includes an in-depth study on the area, observing pre-existing solutions in other games, and determining how to implement the prototypes.

– Build prototypes that try to solve the problem. This is done in iterations. After each iteration small usability test are performed to assert that the prototypes are moving in the right direction.

– Evaluate the prototypes using a heuristic evaluation and play testing.

4.2 Plan

Table 4.1 below illustrates the initial schedule. While it was mostly followed, a lot of testing was done during the prototype building phase, so the total time spent doing tests is likely higher than the planned one week.

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Pre-study, research for the in-depth study, writing in the report, and evaluating how the prototypes should be built</td>
</tr>
<tr>
<td>7</td>
<td>Building prototypes</td>
</tr>
<tr>
<td>1</td>
<td>Testing the prototypes and evaluating the results</td>
</tr>
<tr>
<td>3</td>
<td>Writing the rest of the report</td>
</tr>
<tr>
<td>1</td>
<td>Prepare presentation and opposition</td>
</tr>
</tbody>
</table>

Table 4.1: Plan for the project
4.3 Pre-study

The majority of time in the pre-study was spent on the in-depth study. Second to that was the time spent on evaluating how the implementation should be built, identifying the target group and determining needs and requirements. Lastly, some time was spent on testing existing games on the platform.

4.3.1 Target Group

Defining the target group is important early on in the design process. It may be of importance for design decisions made during the early stages, but it is also important when considering test subjects. It is preferable to have test subjects that fall into the target group category.

Because MilMo already has a player base it is easy to use player statistics from the existing game to predict the target group for the tablet version. Based on Facebook statistics between May 10th and August 7th (90 days), it is clear that the two largest age groups are 13-17 and 18-24. The former age group represents about 39% of the player base and the latter represents around 35% (see Figure 4.1). About 66% are male, and 30% are female, the last 4% has been identified as other/unknown.

Figure 4.1: Age and gender demographics for MilMo between May 10th and August 7th

4.3.2 Needs and Requirements

Defining needs and requirements was the final step before starting prototyping. These acted as guidelines and where helpful to look back at to remember what to focus on.

The needs and requirements are listed below:

- *Easy to learn*. No user manual should be needed to learn how to play, essentially being able to start playing and learning on the go.
- The interaction should be *designed to avoid input mistakes*.
4.4 Prototyping

– Offer the *same functionality as the original game*. Playing the game on tablets should be equivalent to playing it on the web.

4.3.3 Existing Solutions

Testing existing solutions essentially meant playing some games on tablets to achieve some understanding on how others have solved this problem. This was not done in any formal matter, no comparisons or notes where done in this step.

4.4 Prototyping

It was decided that the prototypes would be built on MilMo instead of as separate projects. There were several reasons, but the main ones where:

– This minimizes the difference between the actual product and the prototypes.

– Assets and real scenarios that can be used in the play testing already exists.

An early prototype was first developed to test functionality that had to be built as groundwork for the prototypes. This allowed for a playground where different settings could be tested. Later on the prototype was separated into two alternatives, this was done after identifying two promising alternative methods of interacting. These prototypes will from now on be referred to as Prototype 1 and Prototype 2, it is important to note that 1 and 2 does not mean version 1 and 2, but rather alternative 1 and 2.

4.5 Usability Testing

This section describes how and when the usability testing was performed, but also the rationale behind why it was done the way it was done.

User tests where made weekly during the implementation phase. These tests were informal and short, the subject essentially got the prototype in their hands and had a chance to try it for a couple of minutes, afterwards a short discussion was held about the positives and negatives. The goal with these early tests was to indentify problems and try to solve them as early as possible.

Because the testing phase occurred during the summer, the age group between 13-17 which would otherwise be reachable trough schools were hard to reach. Because of this, but also because of time limits it was decided that the tests would be performed on employees of Dohi sweden. The youngest employees were chosen, as they fall into, or are at least close to the age group 18-24, which, according to player statistics is the second largest age group with just a few percentages under the age group 13-17 years.

The testing was performed on 6 subjects. The relatively low number of tests ran was a combination of time constraints, but also based on a paper by Nielsen and Landauer[21]. They found that about 5 subjects is the optimal amount in terms of benefit/cost. They also found that 5 subjects will show around 80% of the usability problems.

The biggest argument for having more test subjects is for quantitative studies. Because of the limited amount of subjects in this project the collection of qualitative data was emphasized before quantitative data.

Other arguments that can be made for having more test subjects:
– Large projects need more test subjects because there are so many aspects to test.
– Having several different target groups require more test subjects.

The first argument is not really applicable to this project, since it would most likely not be considered to be a large project because it is not spread across several views and all aspects of it can be tested in a fairly short amount of time. One can also insist that a large project is not a valid argument for more test subjects, but rather for more different tests on a smaller set of features in each test.\footnote{http://www.nngroup.com/articles/how-many-test-users/}

Having several different target groups can be a valid reason for having more test subject because it is optimal to have representatives from each target group. However, there is bound to be some overlap between these groups unless they are expected to behave entirely different, and therefore it may be enough having only a few from each target group.

The subjects were split into two groups to suppress bias from which order they tried the prototypes. One group tried prototype 1 first, while the second group started with prototype 2.

After the subjects had tested both prototypes, they were asked to answer a questionnaire consisting of 5 questions.

When the subjects had filled in the questionnaire, a discussion was had where they had the chance to describe their experience, and further talk about their answers in the questionnaire.

### 4.5.1 About Quantitative Data

Initially the plan was to collect quantitative data, this plan however was scrapped as it became clear that the number of tests performed would be quite low. The main reason for collecting quantitative data is to see patterns and draw statistical conclusions, however, such conclusions are hard to draw with such a small sample size. Therefore focus was put on the qualitative data instead.

The plan was to count the number of failures, and how long the players took to accomplish their goals. The rationale was that the number of failures would indicate how well the prototype was designed to avoid input mistakes which was one of the guidelines specified in the section Needs and Requirements. The time it took for the players to accomplish their goal would quantify how easy to learn the prototype was.

#### Failures

One of the goals was that it should be hard to make mistakes. Therefore it makes sense to measure the amount of failures to see if any significant differences can be measured.

#### Timing

The time it takes from start to goal would give an indication of how fast a user can start accomplishing goals in the game from putting it in their hands.

### 4.5.2 Heuristic Evaluation

The final prototypes were evaluated not only by the usability tests performed on test subjects, but also using a heuristic evaluation. A heuristic evaluation is a review where a
usability expert reviews an interface and compares it against accepted usability principles. The goal is to identify usability problems in the interface. This is one of the most informal methods of evaluating interaction since it is essentially the reviewers opinion of how well the interface conforms to the accepted usability principles.

For this project a selection of Federoff’s\cite{10} heuristics that seemed relevant to this project were used. These were listed in a table and each heuristic was given a score indicating how well the design conforms to them.
Chapter 5

Results

The outcome of this project was two implemented prototypes as well as the results from the evaluation of these prototypes. The following sections describe the design and implementation details of the prototypes, and presents the results of the evaluation.

5.1 Design

It was important that the interaction with the game obscured as little of the screen as possible. As such, visual elements were kept to a minimum. Actions such as jumping and attacking are placed close to the edge of the screen easily accessible with the right hand. The virtual thumbstick used for movement is only visible when the user has their finger on the screen. This way it stays out of the way when it is not used.

5.1.1 Early Prototypes

The first prototype used a camera that the player could not control, instead it would automatically move behind the player. The rationale was that it would be too complicated to move both the camera and the character using touch. However the camera proved to be problematic because players had problems predicting how the camera would move, and because of that sometimes runs in the wrong direction, sometimes causing their character to die as a consequence.

At this time all actions such as jumping, attacking, and switching weapons was performed using gestures. An upwards swipe would make the character jump, and swiping sideways would make the character switch weapons. Simply tapping on the right side of the screen would cause the character to attack enemies close by. Some more complicated gesture shapes were tested, but they proved to be too time consuming to perform. Even the upwards swipe proved to be problematic as some players had problems timing the jump correctly.

When it was decided that the automatically moving camera would have to be dropped, a way of moving the camera would have to be introduced. This would have to be performed on the right side of the screen since the left side was already used for movement. This caused interference with the gestures that were already on the right side. The end result was that the gestures were dropped, and on-screen buttons were created instead.
5.1.2 Prototype 1

This prototype uses a method of interaction that is perhaps more similar to existing solutions than the second prototype. The screen is separated into two zones - left and right (illustrated in Figure 5.1). The left side is for moving the character, and the right is for looking around and performing actions such as jumping and attacking.

![Figure 5.1: Illustration of the prototype](image)

By touching the left or right side a half transparent circle would appear on the screen (see circle 1 and 2 in Figure 5.1). Its center would be at the point your finger touched the screen first and it would stay on screen for as long as the finger is touching the screen. This circle is from here on referred to as a Virtual Thumbstick, the name comes from how it acts similarly to a thumbstick on a game controller for a video game console.

As mentioned earlier, the left side is for moving the character. When the circle has appeared, the character will move in the direction that the finger has moved relative to the virtual thumbsticks center. The interaction for looking around is similar, instead of moving the character, the camera is rotated in the direction that the finger has moved relative to the virtual thumbsticks center.

Jumping and attacking is simply performed using on-screen buttons.

5.1.3 Prototype 2

Prototype 2 has some similarities to controlling a car. Similarly to Prototype 1, the screen is divided into two areas. The difference on the left side is that moving the finger sideways turns the player left or right instead of taking left or right steps. When moving forward and
turning the similarities to controlling a car is clear, with the difference that the character can also turn while standing still.

On the right side of the screen, the camera can be adjusted vertically. This is essential in some levels where the ability to look down or up is important.

Jumping and attacking is performed the same way as in Prototype 1.

5.2 Implementation

Some modules had to be implemented before it was possible to test different interaction methods. Some of them came to play big parts in the final design while some were not needed anymore by the time the final prototypes were finished. One such example is the Gesture recognizer which came to be excluded because no gestures were used in the final prototypes, but it may still be interesting to see how it was implemented.

5.2.1 Gesture Recognition

Gesture recognition can be implemented in a number of ways. For this project a direction-based method was chosen, this method is fairly easy to implement. The recognition can be divided into the following steps:

1. **Input Registration** collects 2d coordinates describing how the users finger travels to form a shape.

2. **Normalization** takes the raw 2d coordinates and normalizes it into a format that can be used for classification.

3. **Classification** uses the normalized input to match it against possible gestures.

4. **Notify listeners of gesture**

Input Registration

As the user moves their finger the finger position is registered. A new position is only saved if the distance between the last and the new point is at least 10 pixels on the screen. This filters noise that is not relevant for the identification of the gestures.

The result from the input registration is a list of 2d coordinates describing how the finger traveled.

Normalization

The normalization of the input is done in two steps. First, the length of the list of coordinates is normalized into 32 elements where the distance between each element is equal. Second, this normalized list of coordinates is converted to a list of angles.

The first step removes the differences in shapes that might occur depending on the speed that they were drawn. Figure 5.2 illustrates an S-shape and how the distance between each point is greater in the top because the user moved their finger quicker at the beginning, the left side of the figure shows how the normalization erases any traces of this. This means the recognizer will be able to identify the shape no matter how the user alternates the speed of their finger movement. This process is not intended to modify the shape, however some sharp corners may be softened a bit during this step. In Figure 5.2 this can be seen in the sharpest corner of the S-shape.
In the second step the list of coordinates is converted into directions. The direction is classified based on the angle of each segment. This implementation has 8 possible directions (shown in Figure 5.3). Algorithm 1 shows the pseudo code for how the direction is classified.

Algorithm 1 Pseudo code of direction classification

```plaintext
function AngleToDirection(angle)
    if angle < 0 then
        angle ← 360 − abs(angle)
    end if
    direction ← round(angle/45)
    if direction = 8 then
        direction ← 0
    end if
    return direction
end function
```

Classification

When the gesture stroke has been normalized, it must be classified. This recognizer uses a Levenshtein distance string comparison. Essentially it compares each point in the gesture stroke to a reference stroke determines the distance between them. It then returns a score of how similar these two strokes are. There are many implementations of this algorithm, but this one uses an iterative implementation with two matrix rows (see Algorithm 2 for a pseudo code implementation of the algorithm).
5.2. Implementation

Figure 5.3: Possible gesture directions.

Early Recognition

Sometimes it may be preferable to identify and perform a gestures action before it has been fully completed. Consider the jumping gesture described earlier where the player performs a swipe gesture upwards making their character jump. Since this is sometimes a time critical action, being able to classify and perform the gestures action as soon as it can be recognized may improve the user experience.

This was implemented by simply doing the normalization and comparison every time new input is registered by the recognizer. If a gesture is recognized, and it is allowed to be early recognized, its action will be performed and the started gesture will be ended. Why are only some gestures allowed to be early recognized?

In some cases early recognition of gestures might confuse the user more than it would help. Consider the S-shaped gesture described earlier. Even if this could be identified halfway through, the user experience would suffer. Halfway through the S-shape from the user’s point of view they have only drawn a C-shape.

5.2.2 Virtual Thumbstick

A thumbstick on a game controller for video game consoles is essentially a tiny joystick. It is a stick that can be pushed forward, backwards, left, and right. It can also be pushed in all angles between these four, this is often called an analog thumbstick compared to digital ones that can not be pushed in angles between those four.

The Virtual Thumbstick that was implemented was inspired, and works similarly to these thumbsticks. When the player puts their finger down on the screen, that initial point is the center of the virtual thumbstick. Moving the finger in any direction relative to that will translate into input to the game (moving the character or rotating the camera).
Algorithm 2 Pseudo code of the levenshtein distance implementation.

```plaintext
function LEVENSHTEINDISTANCE(s[1..m], t[1..n])
    matrix ← [2, m + 1]
    for i ← 1, i ≤ m, i ← i + 1 do
        matrix[0, i] ← j
    end for
    currentRow ← 0
    for i ← 1, i ≤ n, i ← i + 1 do
        currentRow ← i & 1
        matrix[currentRow, 0] ← i
        previousRow ← currentRow ∧ 1
        for j ← 1, j ≤ m, j ← j + 1 do
            if s[j - 1] == t[i - 1] then
                cost ← 0
            else
                cost ← 1
            end if
            min1 ← min(matrix[previousRow, j] + 1, matrix[currentRow, j - 1] + 1)
            matrix[currentRow, j] ← min(min1, matrix[previousRow, j - 1] + cost)
        end for
    end for
    return matrix[currentRow, m]
end function
```

However, there are limitations with touch that make it hard to fully recreate a thumbstick on touch devices, and all boils down to limitations in feedback on touch devices.

The first problem experienced in early tests was that players started running, but when they wanted to stop and run backwards they underestimated how far back they would have to move their finger. The consequence was sometimes that they fell down from a platform because they did not stop in time. This problem does not exist on physical thumbsticks since they have an upper limit on how far you can push them. Since it is not possible to physically stop the player’s finger on screen something else would have to be done.

The solution to this was a virtual thumbstick that would follow the player’s finger as the finger is about to go outside of the thumbstick area. This way the distance to start going the reverse direction would always be the same no matter how far the player moves their finger.

The second biggest problem was that players constantly needed to look at the virtual thumbstick to see where their finger was and where they would need to move it to move their character where they wanted. On physical thumbsticks this is rarely a problem, because the thumbstick is always resisting the movement. In a sense it can guide the player’s finger back to the initial state just by letting the controller fall back into the center. This sort of resistance is not possible on touch screens, however being able to reset the thumbstick to its initial state is possible by simply releasing the finger and putting it down again.

5.3 Evaluation Results

This section describes the results from the play testing and heuristic evaluation.
5.3.1 Usability Testing

As mentioned earlier, each test subject was asked 5 questions in a questionnaire. The questionnaire consisted of the following questions (translated from Swedish):

- *Have you played MilMo before?* (Yes/No)
- *Do you own a tablet device?* (Yes/No)
- *Do you play games on your tablet device?* (Yes/No. Skipped if they answered no on the previous question)
- *About how many hours do you play games in a week?* (Number of hours)
- *Which of the prototypes did you prefer?* (Prototype 1/Prototype 2)

The questionnaire showed that about 83% (5 out of 6 test subjects) preferred Prototype 1. This makes sense since it is also most similar to other games on the platform. Most users seemed to prefer Prototype 1 since it was easier to navigate, especially on narrow paths because it was harder to accidentally move sideways.

The questionnaire also showed that the game was not as easy to play as initially hoped. Players had some trouble performing some of the tasks that required more precision, for example jumping between narrow platforms. Though, some of the blame can most likely be put on the technical problems experienced during the testing.

The ambition of these questions was to see patterns between subject that own tablet devices and actually play them compared to subjects that do not own tablet devices. Maybe one of the groups would prefer one prototype more than the other group. There were no clear patterns however other than the preference for Prototype 1, and considering the small sample size of 6 test subjects, any patterns could be a mere coincidence.

Technical Problems

There were some technical problems during the tests that may have influenced the test results.

Since MilMo has not been fully ported to tablets the tests had to be performed on something called Unity Remote. With Unity Remote the game is run on a computer while the input is read from a tablet running the Unity Remote App. This had worked fairly well during development on my stationary computer at the office. The tests however were run on a laptop, since they were performed in a conference room. This laptop proved not to be as capable as the stationary computer, and as such the performance was worse.

This caused some problems during the tests. The Unity Remote app proved to be much more unreliable, sometimes *forgetting* that a finger was pressed down causing the character to stop running. The game was also running at a much lower frame rate which may have made the game more difficult to play.

5.3.2 Heuristic Evaluation

Because both prototypes were so similar, they where evaluated together, and any differences would have been noted. The results from the heuristic evaluation has been summarized in Table 5.1. Each point was given a score from 1-10 where 1 is not fulfilled and 10 is completely fulfilled.
The heuristic evaluation provided an average of 8.6 out of 10. The most notable, and lowest scores are given on the first and sixth heuristic, which are given 6 and 3 respectively. Regarding the first, no customization of the controls is available because there was no time to implement it. The sixth received a very low score because the game relies heavily on menu navigation. Even though the menu navigation was outside of the scope for this project, it seemed inappropriate to skip this heuristic since it still relates to the overall interaction with the game.
### 5.3. Evaluation Results

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Score</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls should be customizable and default to industry standard settings</td>
<td>6</td>
<td>Controls are inspired by industry standards. No customization is available at the moment.</td>
</tr>
<tr>
<td>The interface should be as non-intrusive as possible</td>
<td>8</td>
<td>Two buttons are visible while the player is not interacting. Elements are shown depending on context. Pre-existing elements are always visible at the top, might be considered intrusive.</td>
</tr>
<tr>
<td>A player should always be able to identify their score/status in the game</td>
<td>10</td>
<td>Score/status is visible at the top left corner.</td>
</tr>
<tr>
<td>Follow the trends set by the gaming community to shorten the learning curve</td>
<td>10</td>
<td>The touch interaction is inspired by existing solutions while trying to improve on shortcomings in them.</td>
</tr>
<tr>
<td>Interfaces should be consistent in control, color, typography, and dialog design</td>
<td>10</td>
<td>The added buttons follow the overall style of the game interface.</td>
</tr>
<tr>
<td>Minimize the menu layers of an interface</td>
<td>3</td>
<td>Many tasks in the game requires a fair bit of menu navigation.</td>
</tr>
<tr>
<td>Minimize control options</td>
<td>10</td>
<td>Only one option currently exist.</td>
</tr>
<tr>
<td>Use sound to provide meaningful feedback</td>
<td>10</td>
<td>Sounds are played when actions are performed (jumping, swinging swords, etc).</td>
</tr>
<tr>
<td>Do not expect the user to read a manual</td>
<td>10</td>
<td>Players are expected to be able to understand the controls in under 5 minutes.</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>8.6</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Results from heuristic evaluation
Chapter 6  
Conclusions

This chapter will conclude this thesis discussing the achievements, limitations, and what may be done in the future.

6.1 Discussion

It was difficult to estimate how much time everything would take in the startup phase. The initial feeling was that this would be possible to do fairly quickly. But I was proven wrong, the iterative process of coming up with ideas, implementing them and testing them proved to be a very time consuming task from start to finish.

Early on I had thoughts about redesigning the user interface for touch devices, but this would only be possible if Unity had released their update for graphical user interfaces before my thesis started, since that update would fix some limitations with the current user interface system. When it became clear that the update would not be released in time I decided to focus on the interaction method instead, but I kept the user interface redesign as a possible bonus that I would do if I had time. In the end I did neither have the time nor did Unity release their update during the time I was working on my thesis.

Given more time, I would like to have performed a quantitative study. This could show other results or allow me to draw statistical conclusions on which prototypes are better, something that is not really possible with the usability testing that I performed. Because the opinions of the test subjects might not represent a majority of the demography of MilMo.

6.2 Limitations

As MilMo has not been fully ported to tablets, these prototypes are limited to running on either touch screen capable PCs, or by using the Unity Remote app.

Some actions must be performed using the pre-existing menus, these have not been touch optimized and therefore some interaction does not work very well and some touch targets are very small.

6.3 Future work

As this project does not yet run on tablets future work would include optimizing the performance for tablets, updating the user interface to be more tablet friendly, and possibly
making gameplay changes to accommodate to the tablet platform. Because this project is already running on MilMo it is possible to use this project as a basis for the rest of the work on porting MilMo to the mobile platform.
Acknowledgements

I would like to thank all the great people at Dohi Sweden and a special thanks to my external supervisor Erik Lövdahl for great feedback and advice. I would also like to say thanks to my internal supervisor Mikael Rännar at the department of Computing Science at Umeå University. In addition I would like to thank my family and friends for their support.
References


