Development of a real-time multiplayer game for the computer tablet

Johan Lindström and Tobias Malm

LIU-IDA/LITH-EX-A–12/052–SE
Linköping 2012
Master’s Thesis

Development of a real-time multiplayer game for the computer tablet

Johan Lindström and Tobias Malm

LIU-IDA/LITH-EX-A–12/052–SE
Linköping 2012

Supervisor:  
Anders Fröberg  
IDA, Linköpings universitet

Jonas Texas Peterson  
Effektfabriken

Examiner:  
Erik Berglund  
IDA, Linköpings universitet
<table>
<thead>
<tr>
<th>Språk</th>
<th>Language</th>
<th>Rapporttyp</th>
<th>Report category</th>
<th>ISBN</th>
<th>ISRN</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ Svenska/Swedish</td>
<td>☑ Engelska/English</td>
<td>☐ Licentiatavhandling</td>
<td>☑ Examensarbete</td>
<td>☐ C-uppsats</td>
<td>☐ D-uppsats</td>
</tr>
<tr>
<td>☐ Övrig rapport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL för elektronisk version</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-XXXXX">http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-XXXXX</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Titel</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utveckling av ett realtids multiplayer spel för surfplatta</td>
<td>Development of a real-time multiplayer game for the computer tablet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Författare</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johan Lindström and Tobias Malm</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sammanfattning</th>
<th>Abstract</th>
</tr>
</thead>
</table>
| This master's thesis discusses game development on the computer tablet, with Apple's iPad as the target platform. Its main focus is development of non-trivial components such as online multiplayer and touch(screen) controls for a real-time action game, using frameworks and APIs that are mainly free and open source. For each non-trivial component problems are pointed out and possible solutions are presented, the resulting game, Battle Angels, is evaluated along with the game design and development. Battle Angels is a 2D real-time multiplayer action game, the development of it lead to the conclusions and results in this thesis.

The abstract nature of the subject of this thesis makes it impossible to present a set of numerical values that can be compared to existing research. The result is therefore presented by showing images from the actual game with a discussion. It could be concluded that designing and implementing a fast paced multiplayer game on a computer tablet system is a major challenge. It typically requires developers to incorporate a variety of different technologies into their implementation in order to succeed. It could also be concluded that each technology in its isolation were not a major issue, however combining them and allowing them to co-exist were. |

<table>
<thead>
<tr>
<th>Nyckelord</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>game, tablet, android, ios, ipad, iphone, touchscreen, games, multiplayer</td>
<td></td>
</tr>
</tbody>
</table>
Abstract

This master's thesis discusses game development on the computer tablet, with Apple’s iPad as the target platform. Its main focus is development of non-trivial components such as online multiplayer and touch(screen) controls for a real-time action game, using frameworks and APIs that are mainly free and open source. For each non-trivial component problems are pointed out and possible solutions are presented, the resulting game, Battle Angels, is evaluated along with the game design and development. Battle Angels is a 2D real-time multiplayer action game, the development of it lead to the conclusions and results in this thesis.

The abstract nature of the subject of this thesis makes it impossible to present a set of numerical values that can be compared to existing research. The result is therefore presented by showing images from the actual game with a discussion. It could be concluded that designing and implementing a fast paced multiplayer game on a computer tablet system is a major challenge. It typically requires developers to incorporate a variety of different technologies into their implementation in order to succeed. It could also be concluded that each technology in its isolation were not a major issue, however combining them and allowing them to co-exist were.
Acknowledgments

I would like to express my special thanks of gratitude to the employees of Effektfabriken and our examiner, Erik Berglund, for making this master’s thesis possible. I especially want to thank Mattias Hallström for putting endless hours into the project and Jonas Texas Petterson for helping us with technical details of iOS development.

Linköping, August 2012
Tobias Malm

I would like to thank my wonderful Wictoria for all support and all endless hours you have allowed me to put into this master’s thesis. I also wish to thank Effektfabriken, especially Mattias Hallström and our examiner, Erik Berglund.

Linköping, August 2012
Johan Lindström
## Contents

List of Figures x

### I Background

#### 1 Introduction
- 1.1 Purpose 4
- 1.2 Limitations 4
- 1.3 Methods and sources 5
- 1.4 Structure 5
- 1.5 Language 6

#### 2 Pre-study
- 2.1 What is a game? 7
- 2.2 Platforms and the digital game evolution 7
- 2.3 Gaming on smart mobile devices 8
- 2.4 Planing for development 10
  - 2.4.1 Platform, tools and game frameworks 10
  - 2.4.2 The game idea 11

### II Methods

#### 3 Multiplayer
- 3.1 Network fundamentals 17
  - 3.1.1 Internet Protocol 18
  - 3.1.2 Transmission Control Protocol 18
  - 3.1.3 User Datagram Protocol 19
  - 3.1.4 TCP vs. UDP 19
- 3.2 Network topologies 21
  - 3.2.1 Client/Server model 22
- 3.3 Networked physics 22
  - 3.3.1 Fixed timestep 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2 Synchronize the world</td>
<td>25</td>
</tr>
<tr>
<td>3.3.3 Network traffic and masking</td>
<td>27</td>
</tr>
<tr>
<td>3.3.4 The synchronization process</td>
<td>28</td>
</tr>
<tr>
<td>3.3.5 Entity interpolation</td>
<td>29</td>
</tr>
<tr>
<td>3.4 Input</td>
<td>32</td>
</tr>
<tr>
<td>3.4.1 Client-Side Prediction</td>
<td>32</td>
</tr>
<tr>
<td>3.4.2 Lag compensation</td>
<td>34</td>
</tr>
<tr>
<td>3.5 Conclusion</td>
<td>36</td>
</tr>
<tr>
<td>4 User created content</td>
<td>39</td>
</tr>
<tr>
<td>4.1 Tools</td>
<td>40</td>
</tr>
<tr>
<td>4.2 Content sharing</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Conclusion</td>
<td>42</td>
</tr>
<tr>
<td>5 Controls</td>
<td>45</td>
</tr>
<tr>
<td>5.1 NUI</td>
<td>47</td>
</tr>
<tr>
<td>5.2 The freedom of touchscreens - opportunity or curse?</td>
<td>48</td>
</tr>
<tr>
<td>5.3 Tools for interaction</td>
<td>48</td>
</tr>
<tr>
<td>5.3.1 Gestures</td>
<td>49</td>
</tr>
<tr>
<td>5.3.2 Software buttons and joysticks</td>
<td>49</td>
</tr>
<tr>
<td>5.4 Common problems</td>
<td>50</td>
</tr>
<tr>
<td>5.4.1 Fat finger problem</td>
<td>50</td>
</tr>
<tr>
<td>5.4.2 Feedback</td>
<td>51</td>
</tr>
<tr>
<td>5.4.3 Precision</td>
<td>52</td>
</tr>
<tr>
<td>5.4.4 Speed</td>
<td>52</td>
</tr>
<tr>
<td>5.5 An example solution</td>
<td>53</td>
</tr>
<tr>
<td>5.6 Physics based controls</td>
<td>53</td>
</tr>
<tr>
<td>5.7 Conclusion</td>
<td>54</td>
</tr>
<tr>
<td>III Results</td>
<td></td>
</tr>
<tr>
<td>6 Discussion</td>
<td>57</td>
</tr>
<tr>
<td>6.1 Platform</td>
<td>57</td>
</tr>
<tr>
<td>6.1.1 Tools</td>
<td>57</td>
</tr>
<tr>
<td>6.2 Game design</td>
<td>58</td>
</tr>
<tr>
<td>6.2.1 Major choices</td>
<td>58</td>
</tr>
<tr>
<td>6.2.2 Alternatives</td>
<td>59</td>
</tr>
<tr>
<td>6.2.3 In practice</td>
<td>60</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>65</td>
</tr>
<tr>
<td>7.1 Future of Battle Angels</td>
<td>65</td>
</tr>
<tr>
<td>7.2 Suggestions for future work</td>
<td>66</td>
</tr>
<tr>
<td>A Internet survey</td>
<td>67</td>
</tr>
<tr>
<td>A.1 Question 1</td>
<td>68</td>
</tr>
<tr>
<td>A.2 Question 2</td>
<td>68</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Stupid A.I. behavior in the game Crysis .......................... 16
3.2 Illustration of a TCP communication across a network. ....... 20
3.3 Examples of different network topologies. ......................... 21
3.4 Interpolation and extrapolation ................................. 25
3.5 Illustration of the filtering process in server/client synchronization. 26
3.6 Illustration of a entity interpolation process. ................. 30
3.7 With or without lag compensation. ........................... 36

4.1 Top selling games on Steam. ................................. 40
4.2 Valve Hammer Editor ........................................ 41
4.3 Battle Angels’s level editor. ..................................... 42

5.1 Modern input devices. ........................................... 46
5.2 Software controls on a touchscreen. ............................. 47
5.3 Android keyboard, showing a solution to the fat-finger problem. 51

6.1 A series of screen shots from the menu of Battle Angels...... 62
6.2 A series of screen captures taken while playing single-player. 63
6.3 A series of images taken while initiating and playing a multiplayer match. ........................................... 64
Part I

Background
The smartphone and tablet market is huge and is getting larger by the second. Google announced\(^1\), at Google I/O 2012 [Google, 2012], there are 400 million Android devices and 1 million are activated each day, while Apple announced [Apple, b] that 37.04 million iPhones, 15.43 million iPads and 15.4 iPods had been sold during the first quarter of 2012. Games exists on most digital platforms, from the perfected gaming PC, which is able to run the latest games on maximum settings, to a T83 plus (calculator) running a Mario clone. Games are everywhere and it is not surprising that with the success of smartphones and tablets they followed. Even people who see themselves as non-gamers knows about games such as Angry Birds, which have evolved from a pretty simple physics game to fully fledged culture phenomena. There is no real doubt that smartphones and tablets represents a large market for games, which will most likely grow. Making it possible for developers to make games that reaches millions of users. The really tricky question is how one can make an application, or further still a game, that does not drown in the hundreds of thousands of other applications already available to the users [Fingas, 2012] and reach these millions of users.

Gaming, the act of playing digital games, is something that we always have found a great deal of interest in, both when it comes to creating and playing them. While playing a game we often find ourselves asking “How do one do that?” and making statements along the line of “Why did they do that, when this should be a lot more fun!”. Therefore it was natural for us to do a Master’s thesis that somehow involved games. In late 2011 we got in contact with a company in Norrköping called Effektfabriken. They were immediately interested when we told them about our plans of writing a Master’s thesis in the area of games.

\(^1\)http://www.youtube.com/watch?v=TquwLpk8N8Q
After some brainstorming together a suitable project, or at least an idea of a project developed. As some of the their work involves creating applications for smartphones and tables they found it interesting to create a game on such a platform. Together we forged an idea of a game, which were given the project name Flicking Fantasy, which later changed to Battle Angels. Apart from us one employee of Effektfabriken were working, part-time, on the project, mainly on game design and graphics.

1.1 Purpose

The purpose of this thesis is to highlight some of the more important factors and problems for games, some that are more or less unique to the platform of smart devices\(^2\). The thesis will also show our initial choices, when developing Battle Angels, and what they resulted in, what could have been done differently and similar. It will answer questions that arises when developing a game purpose-made for the, exciting and relatively new, platforms of smart devices.

Games on smart devices consists of mostly small games, larger games are rare and real-time multiplayer games are overall very few in number. Which begs to question; real-time multiplayer games are very common and overall popular on more established platforms (PC and consoles), how difficult is it to create a real-time multiplayer action game for a smart device? Is there any unique problems one have to overcome on this specific platform? Will the touchscreen input hinder or create opportunities? And above all is it worth to try to make one? These are the major questions that this thesis hopefully will answer.

1.2 Limitations

This thesis covers some techniques and ides for games in general as well as some unique for games on smart devices. However please note the even though the main focus lies upon games on smart devices the scope will be limited to games for a specific genre, namely action games. Action games makes a good reference point since they are required to be fast and responsive, which pushes some techniques to their limits, e.g. real-time multiplayer and touchscreen based input/controls.

As of the time limit of the thesis only one platform (see section 2.4.1) were used when exploring ideas and techniques, however the majority of the thesis should be relevant for most smart device platforms, as they tend to be quite similar.

\(^2\)A smart device are a smartphones or tables with a touchscreen as their input device.
1.3 Methods and sources

As the goal with the actual project were to create a unique game for smart devices, the idea were to start by looking at what others have done and how people felt about that. By using information about what works, what does not and what people felt is missing from games today we could avoid doing fundamental mistakes already discovered by others in our development. The design process of Battle Angels were in no way perfect but at least an agile process working in iterations were used, in this way could the game be made more fine-tuned and become progressively better over time.

It can be very difficult to find (good) sources on the subject of games on smart devices. Some information could be found in books concerning games in general or books about user interfaces for touchscreens, however there are not many of those either. Which boils down to that one will have to look elsewhere, mostly on articles found on the Internet, where the best are written by people that works in the industry e.g. articles by Valve employees. However these articles becomes out-dated pretty quickly because of the fast pace of the industry in general.

1.4 Structure

The thesis consists of three parts; background, methods and results. The background part consists of the introduction and a pre-study, methods consists of chapters concerning multiplayer, user created content and controls and results consists of discussion and future work. The pre-study chapter explains choices that were made before the development began as well as some initial research of existing platforms, tools and similar. The chapters about multiplayer, user created content and controls vary a lot in the length and depth but brings up ideas and techniques that are non-trivial when it comes to design and programming. In the results part a discussion about the choices made before and during the project will be held, future work will brought up e.g. how one would continue exploring the subject of games for smart devices.

Chapter 1 - Describes the background to the thesis, its purpose, limitations and other factors worth mentioning.

Chapter 2 - Describes what were done before the project began as well as initial data gathering.

Chapter 3 - The difficulties and solutions for real-time multiplayer.

Chapter 4 - What can be gained by letting the user create content.

Chapter 5 - Controls design for a game on a smart device.

Chapter 6 - Results and discussion.

Chapter 7 - Conclusion.
1.5 Language

The language of this thesis as you very well have noticed is in English. Writing in English is a natural choice when writing articles, books etc. concerning this subject as most of the terminology about games is given in English. Another positive side-effect is that a lot more people in the world will be able to read and understand.
2 Pre-study

2.1 What is a game?

A game can be many things, it is essentially different things for different people, some may find a game fun while others find it a chore. According to Adams [2009b] a game is a very abstract thing that consists of a few essential elements; rules, goals, play and pretending. A place where real-life actions have very different consequences, where any human is able to fly by the push of a button. A game can be many things, games are only limited by the creativity and imagination of the developers creating them. This is one of the reasons why games are such an interesting subject.

2.2 Platforms and the digital game evolution

Digital games are a fairly new type of entertainment, with a history that stretches back only about 40 years according to Edwards [2007]. The first interactive applications that could be called games were made with the purpose of simulating reality, such as chess or missile simulators. However it did not take long before games were made just to be games and were the purpose were to have fun. These early games were not accessible for the general public, it was only with the introduction of the console in the 70s and later the PC (personal computer) that digital games become accessible and overall affordable for most.

Since then the market has grown and today people of all ages seems to enjoy digital games, as stated by Entertainment Software Association [2011]:

72 percent of American households play computer and video games
The average gamer is 37 years old and has been playing for 12 years.

Eighty-two percent of gamers are 18 years of age or older.

Forty-two percent of all players are women and women over 18 years of age are one of the industry’s fastest growing demographics.

Today, adult women represent a greater portion of the game-playing population (37 percent) than boys age 17 or younger (13 percent).

Twenty-nine percent of game players are over the age of 50, an increase from nine percent in 1999. This figure is sure to rise in coming years with nursing homes and senior centers across the nation now incorporating video games into their activities.

Even as these statistics are from the USA they should at least give a hint about gaming in the (western) world. While the more so-called hardcore gamers play games on platforms designed with gaming in mind such as the gaming PC or consoles such as Microsoft’s Xbox 360 or Sony’s Playstation 3, there is a large group of people that enjoy the so-called casual games. A casual game is a game that almost anyone can learn and enjoy to play pretty quickly, a game which is generally simple, and factors that makes the game as easy as possible to pick-up-and-play can be considered good. While there are some casual games on platforms that are somewhat dominated by hardcore games, casual games seems to be found mostly on platforms that were not designed to be pure gaming platforms, such as the regular PC, the webbrowser, smartphone and tablet. Which makes these platforms really interesting for game developers, whom have the purpose of making games for the masses and spreading their games as far and wide as possible.

### 2.3 Gaming on smart mobile devices

As with the explosion of smartphones and tablets sales [Alto, 2012] a whole new, fairly large, market for digital games were born. Even though smartphones main purpose is to be phones and in general social devices (email, Facebook), when someone buys a smartphone he/she are probably not buying it because there are games made for that kind of device, there are (of course) some exceptions to this statement, such as the Sony Ericsson Xperia Play. To further support this statement think about the following; someone that owns a PC made for gaming or a game console (Xbox 360, Playstation 3 or Wii) bought them for the sole purpose of playing games, when a user turns on an Xbox it is for the reason to game. Whereas the smartphone and the tablet are multi-purpose devices.

Given that basically anyone can gain access to the needed tools and release their own application on Google Play (former Android Market) by Google [b], App Store by Apple [a] or similar it have become quite easy to create and release a game for smaller, independent, teams. Which means that the market is flooded with some good, some (arguably most) bad and some extremely good or at least popular games.
To get a general feel about what people think about gaming on touchscreen-based devices an Internet survey were created and conducted, both the questions and the answers can be seen in Appendix A. An Internet survey were deemed a good approach as it is easier to target a specific group of people (by asking people from gaming, Android and iOS communities to conduct the survey), in our case gamers and specifically gamers with smartphones or tablets. As can be seen in the appendix only one of the precipitants answered that he/she does not play games on smartphones and/or tablets and then skipped every other question.

From the results of the Internet survey one can conclude that one of the biggest issues are the controls, as can be seen from the answers from question A.5, A.7 and A.8. Some participants gave some more specific answers to the last question A.9:

- Nearly all those games are crap, which is no wonder as most are developed by one or a small couple of persons and the devices are smaller than the hand which uses it - which makes every game crap because its either seeing or using...

- Spectacular! But the lack of a well-functioning control experience is really holding back the platforms ability to really start hosting well made, and larger games.

- If I wanted to use a controller, I would. Games should utilize the unique capabilities of touch for interaction.

- It’s important to me that they’re made to be played on a screen instead of emulating buttons and controllers.

- "Bad controls" really hit home. If a game isn’t responsive enough it’s pointless

Some other issues are addressed as well, such as the lack of (good) multiplayer and that many games on the specified platform seems to be ports of classic games. However most seems to find that controls of touchscreen based devices is the largest obstacle for a good gaming experience, which from a developers point of view can be a good, or at least interesting as it opens the door for new types of games which truly utilizes the potential of the touchscreen and perhaps create something that corresponds to more NUI\(^1\)-like control-scheme, instead of merely port regular controls without touchscreens in mind.

As stated above most of the games made for smartphones and tablets seems to be casual, short and small games made for the general user. Which means that many games belong to the so-called trivia and puzzle genera, were the user is given the time he/she needs to think before acting, games like Wordfuad [Wordfeud] and Angry birds [Rovio] comes to mind. This in turn means that there are

\(^1\)Natural User Interface, a user interface that is natural to use and learn.
very few fast games that require the users full concentration when playing, i.e. there are very few so-called hard-core games on this platform. This may be result of the input method, touchscreens, which makes precise and fast input a pain for the general user (see chapter 5). Which may mean that there are a potential market for faster games on this platform to exploit.

A game genre that grows and thrives today is multiplayer, and the smartphone/tablets are no exception, there are plenty of games were a user can challenge a friend or someone else to a game of something. But where there are many turn-based games there are few real-time multilayer games, which is probably due to unstable network connections, as many users uses 3G or the simple fact that smartphones and tablets were not made for games.

Further on, a smartphone or tablet may have some problems as gaming devices due to that fact of their limited hardware, e.g. slower CPUs and GPUs, not that much of RAM compared to other platform. However the biggest hardware limitation of such device is probably the input and controls, as already stated. A human’s strongest sense is her touch, and when input gives no kind of feedback a lot is lost [Wigdor and Wixon, 2011c]. Compared to a user playing on his/her computer using a mouse and keyboard or the console gamer using the analog sticks of the modern gamepad as input a lot is left to desire when it comes to feedback, responsiveness, and precision. Things like the so-called fat-finger-problem [Wigdor and Wixon, 2011b] is a unique problem to platforms using touchscreens as input which in turn makes games on these platforms suffer from these fundamental inherited problems.

To conclude; smartphones and tablets are fairly new platforms to be used as gaming platforms, were many games seems to be of the casual and slow paced kind and some suffers from bad controls, which is an essential part of any game. If a game were made that utilizes the touchscreen interface instead of trying to port control-schemes from other platforms a lot may be gained. That is one reason why a project to create a real-time multiplayer game for a touchscreen based device were chosen to be conducted.

2.4 Planing for development

2.4.1 Platform, tools and game frameworks

There are a number of different engines, frameworks, etc. which are good and can be used to create a game on most smartphones and tablets, therefore a small evaluation were conducted, see Appendix B, were also a sub-platform were decided. It was concluded that Apples iPad were the physical platform to be used, the game engine of choice were Cocos2D [Cocos2D] (game framework), which together with Box2D [Box2D] (physics) and Game Center (online multiplayer) [Apple, 2012] were considered powerful and above all accessible enough to be used in the project.
2.4.2 The game idea

The main idea for the game were that users should be able to create their own content and share their content with others, it should be simple enough for anyone to get content from anyone and use it. The game should also utilize and exploit the unique control interface of the touchscreen, which many games fails to do. Last but not least users should be able to play together, e.g. online multiplayer. This were the main idea of the project to be realized and explored.

Some parts of this game idea stand out as non-trivial and/or implementation dependent, i.e. there may be help, guides or tutorial to be found but one can not simply copy an example and be done. These parts are mainly:

- Multiplayer - is very implementation dependent and it may vary a lot between games. Especially when using tools, e.g. GameCenter, that differs from other implementations.
- User created content - and content sharing, built into the game, have to be very simple so that anyone can share and get other users content.
- Controls - utilizing touchscreens are not a very explored area, as the technology is quite new.

The rest of the game, such as menu interface, rendering, sounds, etc. is, of course, at least as important as the parts mentioned above, but they are seen as explored areas which can be read about in many books and will therefore not be as thoroughly examined.

The plan were to create the game using the language Objective-C++, which enables programing in Objective-C and C++, and mixing them, to a degree. This is good as Cocos2D is written in Objective-C and Box2D in C++. C++ also means that STL could be used, which were deemed good. As the target platform were iPad and therefore iOS the IDE of "choice" were quite simple; Xcode. The development it-self were conducted on OSX for the sake of simplicity.
Part II

Methods
Multiplayer games brings gaming into a new dimension, as they open up the possibility of making games more social and even competitive. According to Entertainment Software Association [2012] 62 % of gamers play multiplayer games, making multiplayer games more popular than singleplayer. Multiplayer is, in a nutshell, a mode of play for computer and video games, where two or more players are participating in the same game simultaneously. Multiplayer in itself do not state anything about how the game is played other than it is more than one player. It may be on a single system using a fraction of the screen for each participant, commonly known as splitscreen, as in Golden Eye 007 [Empey, 2001] or on multiple connected systems using some form of intercommunication as in Quake [chirinea, 2003] or a mix of both. This chapter will consider the case with multiple systems able to intercommunicate in real time, what is commonly known as online multiplayer.

The current trends of social experiences in games and other social services constantly increases what an application should be able to do and users also expects more and more. Even though, it is not the case that the implementation of features required for a good social experience have become trivial, there are still a lot of challenges to overcome in order to produce satisfiable results but, as stated by Adams [2009a] the social aspect will enhances the players’ enjoyment of the game experience and might as well attract more female users, since female users represents a much greater proportion of the online game market than the single-player game market.

Everyone that have played an FPS (first person shooter) game in single-player mode, probably have at some point felt how stupid or extremely unrealistic the
Multiplayer opponents can be. In such a case the question about playing against other human players often raise as a mitigation. What could possibly be better? No A.I. have to be implemented since each participate will be a real person.

![Figure 3.1: From a section about A.I in a review of Crysis [Jörgen Langer and Tomaszewski, 2007]. This image shows stupid A.I. behavior and decision making. Two soldiers are trying to take cover after being fired upon, though both do it in the wrong direction, instead of cover they get exposed.](image)

Creating a game where multiple human players participating simultaneously can not be that difficult, right? Well in fact it is all about what one is trying to achieve but especially in fast paced games there are simply a lot of things which have to fit in. Consider a multiplayer game for two players, in order for the players to be able to share the same experience of the game both have to view the same virtual game world, this would not been a problem if the world never changed. Unfortunately for programmers this is seldom the case with modern action games which tends to become more and more dynamic. All or at least the majority of all action games also have some kind of physical simulation to further increase realism and player immersion. A physics simulation is cool but is it really needed?

Networking a physics simulation is the holy grail of multiplayer gaming and the massive popularity of first person shooters on the PC is a testament to the just how immersive a networked physics simulation can be. - Fiedler [2006-09-02]

However no case is the same so one might ask - "would my game become more fun using physics?". If no then simply ignore it and go with whatever suits the case.

Sharing the same experiences i.e. viewing the approximate same virtual world,
have the same feeling in controls etc. is the single most important and also most challenging part for a online multiplayer game’s development. Hopefully the coming sections will shed some light upon those issues as well as give some solutions and hints where to start (if it is desired to build a multiplayer game or at least experiment with it). But before that, some project specific issues should be addressed, namely existing tools and frameworks for the platform of the project.

It felt as a great idea to search for tools and/or frameworks already existing, for iOS. It turned out pretty quickly that Apple had created an application with an API called Game Center which is bundled with every iOS device running iOS 4.2 or later [Apple, 2012]. Game Center contains functionalities such as leaderboards, achievements, collections of a player’s friends, online multiplayer capabilities, voice chat and more. Actually it turns out that it is possible to either choose Game Center or to implement it all by yourself from scratch, one should note that there are other options, such as RakNet by LLC, which is not free and therefore not an option for this project. Doing it all from scratch is really pointless, unless it is in educational purpose. Readers should note that even though the choice to use Game Center were made issues described later will apply to other domains as well.

Game Center’s multiplayer module allows iOS devices to connect to each other in a network, this network can send data either reliable or unreliable (as any other network). Which of the modes (reliable/unreliable) that should be chosen can not be decided generally, because it depends a lot on what kind of game one is trying to implement.

### 3.1 Network fundamentals

To understand problems, limitation and possibilities that may occur, it is necessary to start with the very basic definitions for some commonly used expressions [Jamsa and Cape, 1995]:

**Computer network** is a collection of hardware components and/or computers interconnected by communication channels that allow communication (i.e. sharing of resources and information).

**Network socket** is an endpoint of an inter-process communication across a computer network.

**Protocol** defines message formats i.e. rules for exchanging messages in or between computing systems.

To develop a smooth and well behaving online multiplayer game it is not sufficient to use a network one must also know how it works. The most fundamental part is sending and receiving data across the network, this is achieved by using network sockets. There are a several Internet socket types available, here are some of them:
**Datagram sockets** A connectionless socket, which uses User Datagram Protocol (UDP) as its transport layer.

**Stream sockets** A connection oriented socket, which uses Transmission Control Protocol (TCP) as its transport layer.

**Raw sockets** Uses raw data, bypasses any transport layer and publicly expose the packet header, used mainly by routers and other similar devices.

From a game programmer’s point of view only the two first are of interest and the best choice depends entirely on what sort of game that is considered. However even considering building a multiplayer game one must understand differences between both to be able to make a good choice.

### 3.1.1 Internet Protocol

The Internet Protocol (IP) is a communication protocol used for transmitting data-grams (known as packets) across inter-networks. The protocol is responsible for routing packets from a sender to a receiver. The routing process may be visualized as passing a note between two persons in a crowded classroom during class (so you can not for obvious reasons stand up and walk over with the note) instead you pass it to someone close to you which in turns passes it along, if every person is nice the note eventually reaches the intended receiver. The one that sent the note have no idea whatsoever if the note actually reached the intended target unless that person that received it decides to send a note back.

Of course, in reality it is a bit more complicated because no network node knows where to send the packet in order for it to quickly reach its destination. Additionally the Internet protocol was designed to be self-organizing, self-repairing and able to route around connectivity problems therefore it may duplicate packets and send it along different paths of the network.

### 3.1.2 Transmission Control Protocol

Transmission Control Protocol commonly known as TCP is a reliable connection based protocol. TCP provides a reliable, ordered delivery of a stream of bytes from one point (computer) to another. TCP is used in conjunction with the previously discussed IP and together (TCP/IP) they form the backbone of almost everything one may do online. Some features of TCP are:

**Reliable** Manages packet acknowledgment, retransmissions, timeouts, duplications when packets are lost or multiple copies of the same packet arrives.

**Ordered** Packet send-order will be preserved. If packets are received out-of-order the packets will be buffered until sufficient packets have arrived so data can be properly re-ordered and delivered to the application.

**Large packets** automatically breaks up large packets into smaller.

**Heavyweight** Require handshaking, retransmission and other complex operations on top of IP.
Flow control  Data is never sent faster than the Internet connection can handle and thus avoiding congestion problems.

Streaming  Data is read like it is a file, no distinguishing between packets are used or needed (in a distinct transmission).

All this according to Tanenbaum [1995].

### 3.1.3 User Datagram Protocol

User Datagram Protocol commonly known as UDP is a connectionless protocol, which is, as TCP, built on top of IP, however in contrast to TCP it is a very thin layer which basically only gives one guarantee; received packets will be whole. If a packet is sent containing 128 bytes of data a receiver will receive exactly 128 bytes or no packet at all. In practice using UDP computers in the network do not know about each other before communicating. A packet is simply sent using an IP-address and a port number and it will get passed from computer to computer until it arrives or is lost. While the receiver is listening to a specific port (where it expects data to arrive) and when any other computer sends data (to that port) it passes the packet along to the application. The receiver have no idea whatsoever who (which computer) that actually sent the packet only that it did receive it.

The characteristics of UDP is as follows:

**Unreliable**  When a packet is sent it is unknown whether the packet actually travels to the destination, get lost or arrives multiple times.

**Not ordered**  Packet send order can not be predicated, in practice packets are received in-order but it can not be guaranteed.

**Lightweight**  A very thin layer on top of IP.

**Small packets**  Packets must be small to be accepted by intermediate routing devices, typically less than 1000 bytes.

**Datagrams**  Packets are sent individually with definite boundaries. Data can thus not be read like a file!

**No Flow control**  If considerably amount of bandwidth is used it may trigger a congestion collapse, typically causing a temporary traffic jam until the load is reduced.

All this according to Tanenbaum [1995].

### 3.1.4 TCP vs. UDP

At a first glance it might seem obvious which one to choose but actually it is not. As a rule of thumb TCP is reliable and slow, UDP is unreliable and fast (compared to TCP). TCP is often a good choice for applications like web-browsers, emailing services and similar, however for online multiplayer games where speed is needed UDP will probably be a better choice. TCP is designed as an abstract layer on top of IP with the goal to make sending and receiving data robust, not necessarily fast. TCP works as an ordered stream of bytes, an application writes
any number of bytes to an underlying TCP data buffer. At some point when enough data have been passed to the buffer its containing bytes are split into chunks (packets) and sent across the network. This can be problematic for applications that only send small amounts of data infrequently because the TCP implementation might decide that it needs more data in order to send any, however it is possible to turn off the buffering so data is always sent directly.

A much greater problem with TCP, for real time applications, is what happens when a packet is lost. TCP will always deliver an ordered stream of data but to do so it must handle losses, out-of-order and duplication of data. If a packet gets lost TCP must request the packet and wait for it to arrive in order to deliver any data after it to the application. This will cause the connection to temporarily stall while a packet is being resent.

![Diagram](image)

**Figure 3.2:** Assume that the initial negotiation for TCP have been carried out successfully and an application have pushed data corresponding to five packets labeled A, B, C, D, E, (sent from right to left) then a possible scenario when a packet is lost is shown. Note that C were sent but it did not make its way through in the first attempt.

As one may see in figure 3.2 the time required for all data to arrive is $t_T$ seconds which is significantly greater then $t_A$ seconds which is the time it would take if no data were lost. In order to put some numbers to it, assume a player have a RTT (round trip time), commonly known as ping, of 80 ms, then it would take approximately twice that time (160 ms) at best, in worst case the packet is lost again and have to be resent a second time resulting in a quite long delay (up to half a second) especially when considering an application which have a real time requirement, this is bad. UDP on the other hand would always send the data in $t_A$ seconds however all packets may not be present since they might been lost or delayed. Generally for a real time application it is better to accept a loss than wait, because when the packet finally arrives its content will be old and out-dated. A lot of games in the FPS/action game genre such as Quake, CS, Half-Life etc. uses UDP for sending and receiving data.

One should also note that using TCP in conjunction with UDP may induce packet
losses as stated by Hidenari Sawashima and Sunahara. Fiedler [2008-10-01] recommend using UDP, and UDP only, if the application is time critical. However today, capacities of networks are far better and faster then it were when Hidenari Sawashima and Sunahara published their measurements and thus the impact less noticeable but it have not disappeared. If a game should work using cellular networks e.g. 3GSM this may be an important factor.

### 3.2 Network topologies

A network topology is the arrangement of elements in a network, it can be either physical or logical. A physical topology refers to the actual placement of elements, e.g. there are one element in the middle of the room and others are placed along the walls. A logical topology is the virtual flow of data created by an application without regard to the physical topology. Jamsa and Cape [1995] states that each network have at least one topology such as bus, star, ring etc. generally one do not have to consider the network topology for the network itself because it is sufficient to decide which type of topology the game should use on top of that.

Smart devices are capable of connecting and communicating through different network types e.g. WiFi, cellular networks or bluetooth. Some of these are capable of using peer-to-peer for data transmission, the immediate benefit of peer-to-peer is that one not necessarily have to be connected to the Internet to play. Peer-to-peer can for example be setup in a LAN or via bluetooth bridging.

![Figure 3.3](image-url)

(a) Shows a peer-to-peer network topology. In a peer-to-peer network each element is connected to every other element. Each element can both act as client and server.

(b) Shows a client/server network topology. One element is a central server which all other connects through.

*Figure 3.3*
3.2.1 Client/Server model

For a fast paced action game it makes sense to use the client/server model, preferable on top of a peer-to-peer network. Smart device users will most likely not want to have to setup their own computer server in order to play a game, instead they probably wish to connect and play as painlessly as possible. One possible solution for this is to make a smart device act as a server. There are both positive and negative aspects of using a smart device as a server:

**Easy to play** - No need to setup, restart or manage an external server.

**No costs** - No extra costs are induced since the server relay on users existing hardware.

**Performance** - The assigned server must be able to act as both client and server, thus it may affect its overall performance.

**Network traffic** - The assigned server must be able to handle the network traffic, it especially requires a higher upload speed compared to clients in order to send snapshot states.

In a pure client/server model no game rules runs locally, instead inputs are sent such as touches, gestures, key presses, mouse motions to the server. The server process the inputs sent and respond with an updated state. Clients use the state to visualize objects position, rotation and similar. The problem with this kind of input driven updates are that they become subject for latency. A player performing an action causes an input to be created and sent to the server, however the effect of the input will not be available for the player until it has traveled to the server, been processed and traveled back again, more about this in section 3.4.1.

What if everyone, clients as well as server, would run the same code locally? Then no one would have to wait for inputs to be processed. Even though this is perfectly true it has security issues involved which makes it non-practical in games where cheating can not be allowed. If clients can be trusted an authoritative scheme could be used to synchronize the game state while still running all actions locally. Readers should note that this paper will only consider the case where clients are considered unreliable. In the case of unreliable clients one is in need of a totally authoritative server. Totally authoritative means that the server will tell each client their current position and not the other way around.

3.3 Networked physics

Games since their inception have in general become more and more dynamic, one way to add dynamic behaviour to a game is adding a physics simulation to it. A physics simulation is a quite complex piece of software, fortunately there exists a lot of implementations which can quite easily be incorporated, this project used Box2D, for more information see section 2.4.
When working with a physics engine as a tool for creating immersing games there are a lot of issues that have to be considered in order to get things right. First of all, one have to decide what parts of the game that actually should make use of the physics. It is very tempting to say that all objects in the game should be physical in some sense but it is likely that there are objects like scenery that no player really interacts with. The point is that the computational power required to step a physical simulation ahead in time (i.e. run the simulation) heavily depends on the number of physical objects being simulated. Keeping the number of objects at a reasonable level is critical for the application’s performance.

Adding a physics simulation to a single-player game is one thing but how can this be achieved for a multiplayer game? For a single-player game the physical simulation runs locally, but for a multiplayer game it is not really obvious where the simulation should be executed anymore, as it depends heavily on the game in question and what one is trying to accomplish. For a client-server model a good initial guess is that the simulation should be placed on the server and that is in most cases true. The server is authoritative and in charge of the rules and the physics simulation contribute a lot to the rules since it imposes limitations, for example for things like movement. However in practice each client also need a simulation, more accurately the very same simulation the server executes. Each simulation have to be approximately the same for things to work, both synchronization and a player’s visual experience depends a lot on this.

3.3.1 Fixed timestep

Physics engines often use a computational algorithm called integrator to advance the physical simulation in time. The integrator simulate physical equations using discrete points of time. Basically one feed the physics simulation with the delta time\(^1\) from the game’s run loop and the simulation will run but doing so will not yield the same result from run to run. The main problem is the fluctuation in the delta time. The delta time value is used to approximate physical equations but, different value will give slightly different results and the propagation, i.e. the result of a step is used as input in to the next, will fast become noticeable if one compare different runs.

In order to reduce the problem the used delta time should be made constant. A good way to do this is to introduce an accumulator which accumulates delta time and consumes it in steps which each are constant. Listing 3.1 shows how this can be achieved:

---
\(^1\)Delta time is the time that have elapsed since the last update.
```c
-(void)update:(double)dt {
  // NOTE: Max frame time to avoid spiral of death
  double frameTime = max(dt, MAX_FRAME_TIME);
  accumulator += frameTime;

  while (accumulator >= PHYS_DELTA ||
         isEqual(accumulator, PHYS_DELTA, UPDATE_FREQUENCY / 2)) {
    world->Step(PHYS_DELTA);
    // Do something
    simulationTime += PHYS_DELTA;
    accumulator = PHYS_DELTA;
  }
  // NOTE: If needed alpha value might be used for a linear
  // interpolation of object’s visual position.
  // const double alpha = accumulator / PHYS_DELTA;
  // State state = curState*alpha + preState*(1.0-alpha); // Per object
}
```

**Listing 3.1:** Shows a possible run loop which calls a physical step method based on an accumulated delta time. *MAX_FRAME_TIME* is the maximum time allowed for a frame, *PHYS_DELTA* is the desired update frequency of the physics simulation, *UPDATE_FREQUENCY* is the normal update frequency of the run loop (e.g. 60 Hz). Note that multiple steps might be required in a single update if enough time have been accumulated. *isEqual()* is a function that measure equality for floating-points, the third parameter specifies accurateness, and will return true (or rather YES as this is Objective-C) if the accumulator is approximate the same as PHYS_DELTA, it also makes sure that the step happens in the best possible frame.

The last comments in listing 3.1 hints how an alpha value may be calculated and used for an linear interpolation, for the graphics. This may be needed as the updated frames for the graphics and the physics engine may miss each other, and micro stuttering may become visible for the user. An observant reader might have noticed that the interpolation which is performed using the current state and the previous actually shift the time back one update cycle, even for a fast paced game one cycle is not noticeable. It is critical to understand that one really want interpolation and not extrapolation. Extrapolation would be calculating the new, non-existing, state based on two or more, known, previous states. Which means that it have to guess the next state since it is not yet known. In the section 3.3.4 this concept will be brought even further.
3.3 Networked physics

Interpolation finds a new value within a set of known discrete values.

Extrapolation also finds a new value given a set of known discrete values but contrary to interpolation, this new value is outside the known set.

Consider the time-line where the dots represent known positions and crosses represent positions in the future. Assume the accumulator contains some "unused time", instead of extrapolating and finding a value at the black line it is far more visually appealing interpolating and finding a value at the red dot. Yes it is in the past.

Even with a fixed timestep a physical simulation is not guaranteed to produce the same results on different systems. Actually it goes down as deep to the level of floating-point calculations. Generally floating-point calculations are not guaranteed to be exactly the same, down to bit-level, on different architectures. For a fast paced game where some kind of server side syncing is involved it rarely becomes a problem but for some types of game, mainly RTS games it may. Readers are encouraged to read more about "Floating Point Determinism".

3.3.2 Synchronize the world

In a game which uses a server/client model it makes sense to regularly synchronize the state of the server’s game world with each client. If no synchronization would be performed it is highly likely that every participant would see slightly, sometimes even very, different worlds. Since the server is authoritative the state of its game world can be considered the “true game world”.

Synchronization may be divided into two categories; partial and full synchronization. Partial synchronization means that only a portion of the game world is synchronized and full synchronization means the entire world is synchronized.
Generally it is desirable to only use partial synchronization but depending on game design sometimes it might be necessary to use a full synchronization for example when a player joins a game session or lags behind.

Partial synchronization will greatly reduce the amount of data that have to be sent in order to stay consistent. The portion to send is the part of the game world that currently is relevant for a player. Relevance can be measured in different ways, a common one is by distance. A player can for example define two ranges $R_v$ which is its "seeing" range and $R_a$ which is its "hearing" range. For each object this gives three different possibilities per player:

**Figure 3.5:** Shows the world from a client’s perspective. The blue circle represents the seeing range, i.e. objects within this range ($R_v$) are fully synchronized. The red circle represents the hearing range, objects within this range ($R_a$) only send sounds and are not fully synchronized. The dotted green box represents what can be seen on the actual screen.

**In seeing range** The object must be fully synchronized.

**In hearing range** The object must be partially synchronized.

**Out of range** The object can be excluded.

Listing 3.2 and 3.3 defines two example structs which could be used as messages for in-range snapshots. The idea is to send AudioEffectPacket when an object is inside the hearing-range but outside the seeing-range. When an object is inside the seeing-range, RigidBodySnapshotPacket will be sent, it is preferable to leave some bits in the body-mask in which one can add the effect-mask.

```c
struct AudioEffectPacket
{
    int   effectMask;
    float positionX;
    float positionY;
};
```
3.3 Networked physics

Listing 3.2: Example of data that can be sent when a sound-effect should be generated.

```c
struct RigidBodySnapshotPacket {
    int bodyMask;
    float rotation;
    float positionX;
    float positionY;
    float linearVelocityX;
    float linearVelocityY;
    float angularVelocity;
};
```

Listing 3.3: Example of data that can be sent as a physical object snapshot. This is the minimum data required to represent a movable physical entity across a network.

3.3.3 Network traffic and masking

At a first glance to listing 3.2 and 3.3 it do not seem to be a whole lot of data, but this is only the very minimum one have to send for objects in-range. In a full-featured implementation there will probably be a lot more variables needed. Adding all possible variables to one packet is not generally a good idea, because it will unavoidable send a lot of unwanted data in each snapshot. What one have to do is to carefully design one packet (e.g. a struct) for each snapshot that the system should be able to handle.

Designing different snapshot packets for different objects is a necessity for reducing network traffic, however it is often possible to squeeze things even more by using masking. Assume that objects have properties such as health, type and id. Instead of using for example three int variables one can use one by adding limitations to objects’ properties. For example health may be an integral value in \([0, 255]\), type in \([0, 255]\) and id in \([0, 65535]\), that would reduce the data needed by 8 bytes. Well 8 bytes is not that much, why should one bother?

Assume that at particular location in the game world there are 50 objects of which 2 are players at a given time, further assume that snapshots are generated each 50 ms. Then the required upload transmission speed for the server is given by:

\[
O_{\text{bytes}} = \text{sizeof}(\text{RigidBodySnapshotPacket}) = 28 \text{ bytes} \\
T_{\text{bytes}} = 2 \cdot 50 \cdot O_{\text{bytes}} = 2800 \text{ bytes} \\
T_{\text{speed}} = 8 \cdot 20 \cdot T_{\text{bytes}} = 4.48 \text{ Mbits/s} \tag{3.1}
\]

The result is a bit higher than one first anticipates, the fact is, it quickly adds up. What if two more players were to come into the range? In section 3.3 it were
mentioned that it is critical to keeping the number of objects at a reasonable level for the application's performance. Calculation 3.1 also clearly highlights this and the need for programmers to enforce it.

### 3.3.4 The synchronization process

The synchronization process is overall not that hard to grasp neither is its implementation, however the amount of code that is required depends on the number of object types the game have. This is cumbersome and it can be hard to manage all the code when the number of different object types grow. Trying to generalize all types of objects tends to be quite complicated, it may even be impossible in some cases.

The goals with the synchronization process are:

- Each player should experience the same world, events and so on.
- The game should be self-healing, i.e. if there is an error the game should correct it by continue the execution of the game.
- The game should be robust.

A important factor of the synchronization process is finding inconsistent states and fixing them. A system that is self-healing will become more robust and less special cases have to be provided to solve inconsistency issues. A common reason for an inconsistency to occur is a missed message. Assume one is building an action game where players have the possibility to respawn some time after they have died. In such a case it makes sense to have a message generated by the server at the time when a player is about to respawn. However if a client never receives the respawn message it is not acceptable that the player, that did indeed respawn, stays in a non-respawned state on that client. In this project there were similar problems which occasionally rendered opponents invisible. One could make use of a TCP-like system that can resend a message if it were not received, however it is strongly recommended to avoid that as much as possible. Such a system would induce additional data traffic since each message will require a response in order for a sender to know that the message were received.

In the best of all possible worlds the synchronization snapshot will already contain enough data to deduce the correct state for an object. Given the respawn example one could fetch the current client state of an object and compare it to what one received. For example if an object is not respawned but it does currently have health greater then zero and is moving around it is probably something that went amiss. In case the snapshot does not contain enough data to fully deduce differences in objects’ states, a first measure would be to add it, if possible (preferable by using masking).

A problem with synchronization is to determine when and how often it should be performed. Should the server generate a snapshot each frame and send to all connected players or should it be every other frame? There are no real answers to
this question because it all depends on how good it feels and looks. As a rule of thumb, it should be as infrequent as possible. Modern games usually uses values around 50 ms [Valve, c], e.g. Counter-Strike Source uses a snapshot update speed of 50 ms. A lower value allow games to detect and fix inconsistencies faster and propagation of visual errors become less noticeable but it requires more computational power as well a higher network throughput capacity compared to a higher value.

The process of synchronization may in steps be illustrated as:

1. Wait for enough time to pass.
2. Filter objects for each player.
3. Split each filtered object set into chunks.
4. Translate to binary representation.
5. Send all chunks.

Some notes about the process:
The first step is about choosing an update frequency, normally a value between 50 - 100 ms. Filtering objects is performed per player for example by keeping all objects closer than some limit. Splitting the list of objects into chunks is necessary to avoid overflowing the maximum UDP-packet size which may vary a bit between different implementations and the required header size but, using a value below 1000 bytes should be fine.

Also remember that the binary representation should only contain the absolute minimum data which stills manage to describes the object correctly. In Battle Angels we discovered that it is practical if each chuck only contains the same object types because that will minimize the required header information required for identifying what were sent from the server on a client.

### 3.3.5 Entity interpolation

Synchronization will keep objects up-to-date on clients, but if one were to immediately accept and change the position and rotation for objects as snapshots arrives one would soon realize that this would give a rather jerky visual appearance. Objects will seem to teleport short distances when snapshot corrections are applied. However instead of directly setting each value one may make use of entity interpolation.

Entity interpolation is a method which will make the visual appearance of entities (i.e. objects) look smooth by using interpolations. Entity interpolation is an expansion of the concept describe in section 3.4. The trick to make entities appear smooth are to "go back in time" and render a state of the world that the client have more information about. The goal with entity interpolation is to continuously interpolate between two recently received snapshots to fake a smooth
appearance.

Theoretically it would be sufficient to go back one snapshot in time for the interpolation to work, but this is not practical since in reality snapshots are exposed to network related issues which can cause some snapshots to be delayed or completely dropped. A better approach is to store a history of several snapshots, the amount to use may vary but should be at least three. The benefit for using three snapshots is that even if one is delayed or completed dropped there will still be two valid which can be used for the interpolation. Storing more snapshots before starting the interpolation will make the process more resilient for snapshots being lost, but for each snapshot that is added the difference between the client and the server grows by another snapshot update time (50 ms in our case). Which may make the user experience feel less responsive.

Entity interpolation causes a constant view lag of \( V_l = UT \cdot (HS - 1) \) seconds, where \( UT \) is the snapshot update time (e.g. 50 ms) and \( HS \) is the number of snapshots in the history when the interpolation is started. This will, to the contrary from the example in section 3.4, both have visual and gameplay noticeable effects but even with delays such as 100 ms it is hard to tell the difference, though it is possible.

![Diagram](image)

**Figure 3.6:** Shows a graph for some snapshots in an program using entity interpolation. The actual "real" client time is \( t \) though what a player sees is based on information from time \( t - 2S \). The character above 420 and 421 are the known poses for the character at those times, however the interpolated result can be seen above time \( t - 2S \). \( S \) is the snapshot update time.

For clarification an example will be given:
Consider figure 3.6 and assume \( S = 0.05 \) seconds, the last snapshot received on the client were at tick 422, assume that \( t = 21.1 \) at that time. The client’s time continues to increase based on the client’s frame rate. When a new frame is about to be rendered the rendering time to use is the current time \( t = 21.125 \) (this is, of course, an assumption) minus the view interpolation delay \( 2S = 0.1 \) seconds.
That would be \( t = 21.125 - 0.1 = 21.025 \) in this example. The decimal part of 21.025 tells the fraction that should be used for the interpolation, 0.025 in this case, which happen to be right in between snapshot 420 and 421. Thus snapshot 420 and 421 should be used with a interpolation fraction of 0.5.

In this case three snapshots were stored before the interpolation were started, this means even if snapshot 421 would be missing it is possible to use snapshot 422 instead. However if more snapshots should be dropped (in a row) it would not be possible to interpolate because the snapshot history buffer would be empty. In such a case one have to make a decision such as using extrapolation for a while and hope snapshots will arrive in near future. Extrapolation is achieved by letting the local physics simulation advance all entities in time, based on the latest received snapshot. Objects which are controlled by the local player should additionally have its input delayed with the view interpolation time (0.1) unless input predication (more in section 3.4.1) is used. Delaying input will give a better approximation which hopefully can avoid a choppy and jittery behaviour when the application is returning to interpolation.

Extrapolation should not be used for long time periods because the prediction error will grow rapidly and will become very big. In case a snapshot is not received in reasonable time an action have to be performed. Possible action can for example be:

- Force the client to disconnect.
- Freeze the client and wait for it to recover, eventually (if not recovered) force it to disconnect.
- Disconnect the client and immediately try to reconnect it.

In most cases a new snapshot is received rather quickly after the extrapolation have started, in such a case the ongoing extrapolation must be stopped to allow a transition to interpolation. The transition should be made as smooth as possible for the best gameplay experience. The local prediction will most likely not be perfect, which means that the current position of objects might be wrong. In order to determine the error it makes sense to analyse the current position and the current correct position provided by the new snapshot. If the two positions are significantly apart it is best to just snap the current position to the current correct position, however it is likely that these two positions will actually be quite close. If the positions are close enough (experimentation needed) it will visually look better if the current position is moved only a fraction of the distance between the two positions for example 10%. Also note if the two positions are very close, the best is to do nothing. This method is sometimes called exponential smoothing average.

Fiedler [2006-09-02] recommend that this type of smoothing should be used for immediate quantities such as positions and orientations, while derivative quantities such as velocities and angular velocities can be immediately snapped because
the effect of abruptly changing them are not as noticeable.

```cpp
Vec distanceVec = currentPosition - correctPosition;
float distance = distanceVec.length();

if (distance > UPPER_BOUND)
currentPosition = correctPosition;
else if (distance > LOWER_BOUND)
currentPosition += 0.1 * distanceVec;
else
    ; // Do nothing
```

*Listing 3.4: Shows a simple if-else construct for position smoothing that can be used when recovering from extrapolation.*

### 3.4 Input

Thus far, little have been mentioned about how players control their character and which difficulties it imposes for game designers and programmers; however, this section will hopefully give some insights. This section will not be about how one can make use of input sources to control a character in the game rather how inputs can be represented and sent between different networked instances.

As stated in section 3.2.1 in a pure client/server model inputs are generated in the client and sent to the server where they are processed and then sent back. Basically a client only forward inputs to the server which continuously sends lists of objects to the client for rendering. Of course, a real application would have a lot more components to it. However most online action games today are not using a pure client/server model, instead they are using a somewhat modified version which allow client-side predictions. The reason is simply that a pure client/server model do not work well under real world situations, where players can experience significant network latency.

In order for a player’s character to be moved visually, the local client have to acknowledge the input given from the player then send it to the server and have its result fetched before the updated position could be reflected visually on the clients machine. This is probably the simplest technique but it is very sensitive for delays caused by the network connection. However what if a client could instantaneously react to a player’s input and move the character appropriately? Then no frustrating network latency would hinder the player from feeling their character’s movements directly. Which is called client-side prediction.

#### 3.4.1 Client-Side Prediction

Client-side prediction is trying to conceal the negative effects of high latency network connections by use of a predication algorithm. Predictions are often best suited and used for movements of different kinds. To be able to make use of predictions, clients have to do considerable more work as they must have the pos-
sibility to simulate the world exactly as the server would. Even though each client
is given a certain degree of freedom the authoritative server still remain to run
the simulation as previously. Even if one client would mangle to cheat by altering
its simulation it would only affect the cheaters client, not the server or any other
client, which is essential.

In proceedings of the Game Developers Conference, February 2001, Bernier [2001]
described how half-life implemented predictions for a player’s movements. As in
the pure client/server model each client generate inputs to be sent to the server,
but before actually sending the input to the server, inputs are turned into a com-
mand like representation which is locally stored. Each command contains the
exact time it were generated, a unique identifier and the actual input it encap-
sulates. Client-side predication starts when the first acknowledgment, sent from
the server, is received by a client. An acknowledgment is like a snapshot which
were described in section 3.3.4 but is now called acknowledgment to avoid con-
fusion as they appear identical but are used in different contexts.

An acknowledgment contains state data for a player (its exact position etc) but
more importantly it contains which player-command that the server last acted
upon, or to be more precis; the last handled command’s unique identifier. Assum-
ing that acknowledgments are generated as fast as possible than the last received
acknowledgment is reflecting the past if there is any latency in the network con-
nection. Given that one input-command is generated each frame a client with
X seconds latency rendering Y frames per second will have stored $\frac{X}{Y}$ commands
ahead of time assuming X and Y are constant.

It is appropriate for a client to store input commands in a linked-list, ordered
as they were generated. This type of storing will have the immediate benefit that
it will make it easier to handle an incoming acknowledgment. When an acknowl-
edgment is detected the client do the following:

1. Remove all commands older than the received command.
2. Move the players character to the position received in the command.
3. Re-run the commands that still remains after step 1 on the moved player
   character.

Following those steps should produce a final position that is the position where
the player should be. The predication error is given by comparing the final po-
sition with the actual position of the player. If the positions are different one
should consider snapping the position or apply the exponential smoothing aver-
age method described in section 3.3.4.

To be able to re-run commands it is appropriate to design commands so they
may contain a flag which determine if it is the first time the command is being
ran. In such a way a client can make sure to only invoke effects such as sound and
specific animations the first time the command is ran. Prediction for a player’s
movements significant increases responsiveness for the controlling player in a high latency network. A player’s movement is not the only thing that can use predictions, basically all type of actions can use it but the far most important are movements and weapon firing. But predictions could for example be used for changing weapon, activate weapon functions such as a bi-pod, holstering weapon, throwing weapons on the ground, etc.

Predictions are generally only need when players are using network with high latency. However a good network with low latency will occasionally have peaks of high latency especially WiFi networks in crowded environments as traffic have to fight its way through the air. It is also important to realize that step 3 can be quite hard to implement as it requires fully deterministic calculations or, in the case of a physics simulation, being possible to re-simulate. Re-simulation of a physics simulation is possible but it will require significant computational power as it have to be performed each time the prediction code runs. Assume a client have a latency of 100 ms and a frame rate of 60 Hz than a client must handle 6 physical steps just to determine the location where the player should be. The trickiest part of re-simulation is to figure out which objects actually needs to be re-simulated. Re-simulating too many objects will have severe performance implications on the other hand selecting too few object might cause collision interactions to be missed an thus produce a miss-leading result.

3.4.2 Lag compensation

Client-side predictions are often a good way to increase responsiveness of the game, but it do not come entirely without problems. One problem is that players have to acquire a certain feeling for their latency in order to determine how they should fire their weapons in order to score a hit. Even that players will hear and see their movements and weapon firing immediately those action will be subject to network latency because this information must in some fashion be synchronized with the server and all other connected clients. Consider two players in a game, one is standing still and the other is running perpendicular to the first player’s aiming direction. In order for the first player to hit the player running, bullets fired from its weapon must be fired in front of the running player by some factor which depend on the players current latency.

The reason for this inconvenience happens because when a local player fires a weapon bullets are not really fired, at least not direct as the player is deceived to believe. The fire-input still have to travel to the server where it must be processed and the firing have not really happened before the server have acknowledge it. This is not acceptable and must in some way be fixed.

Bernier [2001] describes a method called Lag Compensation. Lag compensation is a method which, server-side, normalizes the game world individually for each player as input commands are executed. Lag compensation will take a step back in time in order to evaluate if a player scored a hit by examine the game world as it looked the exact instant the player performed some action. In order for
lag compensation to work the server must maintain a history for each connected player that is large enough to contain all snapshots generated in a certain time frame. This time frame must at least be equal to the maximum allowed latency for a player.

Lag compensation is performed as following:

1. Calculate an accurate latency for the player, that should be compensated for.

2. Select the best possible states from the state history based on the calculated latency.

3. Move all players to their previous locations, read from the state history.

4. Perform ray-casts in order to determine if the player did hit.

5. Move all players back to their real positions.

Step 2 will most likely require an interpolation to be performed in order to calculate an intermediate state unless a state snapshot is ”hit perfectly”.

This type of lag compensation is generally good when the ballistics of the bullet is known as a mathematical function and the hit is fairly direct e.g. a sniper rifle will almost fire bullets in a straight line with a high speed. However for general projectile weapons where bullets are subject for more realistic circumstances and moving slower it will become relatively hard to perform this kind of compensation. Consider when a weapon fires a projectile, the server will have to create a physical body to simulate the projectiles movements toward its destination. Since the projectile is a physical object it have to be stepped through the physical simulation like any other object, every time the projectile is moved it have a chance to collide with something, however collisions depends from which players point of view the object is moved. Remember that lag compensation were about compensating for one player’s lag, but in the case of a projectile it is really hard to tell in which time-space this object should reside additionally should players be moved each time the projectile is ready to be simulated?

If a game supports projectile based weapons it might be more appropriate to use a heuristic method that for example is compensating by leading the players aim by a factor based on the players latency. In Battle Angels, which uses physic-based projectile weapons, this were adopted by altering a players aiming angle to allow the player to better hit targets while moving.

Readers should note that if entity interpolation (described in section 3.3.4) is used, one must do the compensation based on players latency as well as the used interpolation time.
3.5 Conclusion

This chapter have presented solutions and discussions for some common problems in online multiplayer action games. The very foundation of networking have been discussed and finally it could be concluded that the user datagram protocol (UDP) were the most appropriate communication protocol for real-time applications, like an action game. Different network topologies have been presented and the client/server model or a modified version showed to be the most common setup for action games today.
Networked physics have been brought up as a central part. Networked physics will give games a more dynamic world where players will be able to interact with a lot of objects, however it has been shown that it brings a lot of problems with it, which can be difficult to solve, but not impossible. Strategies for handling player input and faking the appearance of responsiveness have been presented.

One can conclude that online multiplayer is a very difficult area where programmers and designers have to make a lot of tough decisions and sacrifices in order to design and build a working game.
For a game to be successful it does not only have to attract new users and be popular when it is first released, it has to keep its user-base for as long as possible. By keeping the initial users playing the game, a lot can be gained, e.g. exposure by *word of mouth*. But how do one keep users from getting bored with the game? There is a good chance that if the content of the game stays the same users will eventually get bored and stop playing, which means that games that renew their content are more likely to keep their user-base for a longer period of time than games that keeps its content the same, forever, from the initial release. Developers can of-course release expansion packs as well as DLCs (downloadable content), but a more cost-effective and, from the users point of view, better solution would be to make the game easy to modify, according to Adams [2009c].

Take some *older* games as examples, games such as Half-Life, Quake and Unreal, they all live to this very day because of their user created content, according to Turpin [2004], and this is a testimony how important user created content can be for a game’s longevity. Some have speculated that mods (modification of complete games) are dying, but one short inspection of the top most selling games on Steam\(^1\) (see figure 4.1), can conclude that it is not the case, as the most selling game Arma 2 is doing well because of a mod called DayZ [Rose, 2012]. Another example would be the Steam Workshop which makes it possible for users to share their content and more importantly makes it easy for other users to get to that content and use it.

At this point one can note that all examples have been of game for the PC, a platform that is known for its openness to modifications. So the question becomes; can one create a game for any given platform, or more specifically for a mobile

\(^1\)Steam is a digital distribution service, see [http://store.steampowered.com/](http://store.steampowered.com/)
Figure 4.1: The top selling games of 2012-07-05 on the popular digital distribution service Steam by Valve [a], showing Arma 2 (combined operations) as the top selling game. A game that is more than 2 years old (released 2010-06-29), or more specifically 3 years as vanilla Arma 2 were released 2009-06-29.

platform such as the smart phone or computer tablet, with the feature of modification. The answer is that it is, with some limitations. As some platforms are very restricted and closed to users, such as iOS, things can get a bit complicated, as for example configuration files can be difficult, if not impossible, to get to and modify. But if developers include tools for modification within the actual game and makes it possible to share that content between users, some of the goals of user created content is meet. This is precisely what were done in Battle Angels. An editor for the creation of worlds (or rather levels), together with the service of sharing, storing and make peoples’ creations available to other users, a complete ecosystem were created which have the possibility to grown and hopefully, flourish.

4.1 Tools

In order for users to be able to create anything there have to exist tools. The most common tool is the so-called level editor. Which can be anything from simple to hugely complex. An example of a common level editor is the Valve Hammer Editor (or more commonly: Hammer), which can be seen in figure 4.2. Hammer offers a lot of flexibility for users, and is heavily involved with level events and scripting [Valve, b]. Hammer is a great tool, but for the novice user it can be a bit overwhelming, and therefore less accessible for the regular user/gamer. To make a tool that enables flexibility as well as simplicity is very difficult, as these properties tend to contradict one and other. E.g. making it impossible to create a level that is broken, broken as it is missing spawn locations, flags or similar things, may make it easier for new users to create levels but for the professional user it may harm flexibility.
4.2 Content sharing

When designing and implementing the level editor (see figure 4.3) in Battle Angels the very goal were to create a tool that everyone could use, as well as make use of some of the features that the target platform, which uses a touch-screen as an input device, enables. Such as pinch to zoom in/out and drag to move around. Of course this simplifies the tool a bit, making it less flexible. However by making that choice many users, arguably, most users should be able to wrap their heads around it and use it with as little frustration as possible. Therefore it seemed justifiable to create an editor that were more limited but easier to use for the broad masses than to allow the users total freedom.

4.2 Content sharing

For PC users and gamers it may seem trivial to share, find and download content such as levels of a game, as well as add levels to the actual game (with the assumption that the game were made with all these purposes in mind), compared doing the same thing on more closed platforms. To enable users to share, download levels etc. on platforms as iOS one would have to build sharing functionality into the actual game. Which may be a good thing for the users as everything becomes centered around the actual application and not on external tools (such as Window’s Explorer or OSX’s Finder), which enables the application to become even more streamlined and easier to use. On the other hand from the developers point of view more functionality have to be designed and implemented. This is arguably the largest problem to overcome, as it tend to grow and become quite large and complex.
Because of this users have no actual control of their content, they are at the mercy of the functionality the game provides towards the sharing service. They can only do what the game allows them to do and together with fundamental design choices, this may cause problems. However such a system can be designed to make content flow easily between users and make it accessible to all, i.e. anyone can download a shared level and play or even modify it at any given time. This may make a user’s creation inspire others to create great content. However by allowing content flow that easily and making it accessible to all the question of ownership must be considered, who owns the content? To whom shall credit be given for a specific piece of content? And how? There are no clear-cut answers, e.g. one would have to chose between making sure that credit is given where it is due or make it easy to share, modify etc.

### 4.3 Conclusion

Creating good tools which are carefully balanced is the single most important thing when utilizing the idea of user created content. The experience must be created in such a way that users finds it fun and interesting to create content for others. Another problem is that there is no perfect or straight way to create a large (enough) user base to get the ball rolling. i.e. to attract people to create content some good content must exist for inspiration. However when there are enough users, other users will be attracted simply to see what the fuss is all about. By looking at older games which were designed to be easily modifiable it can be concluded that this a very good method to enable a game to live longer and,
indeed, prosper.
To be able to control a game in a non-frustrating manner it is essential that a user’s experience is smooth and consistent. This is generally a problem in all games and even regular user interfaces. With the given platform, a number of new problems and opportunities materializes.

Some which have already been exploited in existing games, such as the accelerometer (measures linear acceleration) and gyroscope (rotation) sensors. There is also the possibility of connecting external controls, either pure digital gamepads such as the Xbox 360 or Playstation 3 controllers or extensions that are placed on the touch-screen which gives the user additional feedback. However these control types will not be explored further in this thesis. The reason is that a smart-device is not guaranteed to have all of them. Another issue is that these kind of controls are a bit too experimental, when it comes to casual games. External controls are very interesting but defeats the purpose of gaming on a mobile smart device, because of their bulk.

When exploring the possibilities of touch-screens the assumption of multitouch\(^1\) is made, as it is today, the norm as well as it opens up possibilities where single-touch is very restricted.

---

\(^{1}\)Hard- and software that support multiple touches on the screen simultaneously, which enables gestures such as pinching to zoom or similar.
(a) Xbox 360 Wireless Controller — The standard controller for the Xbox 360, can be used together with some other devices (the PC, Android) as well.

(b) Cyborg R.A.T. 5 — A mouse designed with gaming in mind.

(c) QPAD® MK-50 — Pro Gaming Mechanical Keyboard.

(d) Samsung Galaxy S3 — A high-end smartphone.

Figure 5.1: A number of different controllers, where the gamepad (a), the mouse (b) and the keyboard (c) were created with the purpose of being used as controllers for digital games. The smartphone (d), on the other hand, uses a touchscreen as its input interface which enables the creation of a more NUI-like interaction, but does not guarantee it.
As can be seen in figure 5.1 there are a numerous ways to interact with a game. Where the smartphone is the only one where the interaction is not immediately clear. When developing a game to a smartphone or computer-tablet the creation of virtual controls that utilizes the touchscreen is included in the development process, which in a way is unique to this platform. This is because the touchscreen enables input trough pure touches which can be turned into so-called gestures, which in turn have to be interpreted and used to control a game, either directly or by building controls upon the gestures. Two examples of this can be seen in figure 5.2. Where one uses gestures such as swipe, tap/touch and drag to control the game, and the other uses gestures to create a virtual joystick on the screen. Using gestures to control a game is a good way to making the controls more natural and intuitive and therefore more NUI-like.

(a) Grand Theft Auto 3 — This game uses the virtual joystick (to move around) and buttons (direct actions such as shooting) for controls, on a larger screen (in this case 4.8 inch) it works quite well.

(b) Love Love Love — This game is gesture-based, e.g. to steer the blocks left or right one drags a finger on the screen, to make the blocks fall down one swipes a finger downwards, a tap/touch rotates the blocks around each other.

Figure 5.2

5.1 NUI

NUI means Natural User Interface, a NUI is a user interface that feels, from a user’s point of view, very natural to learn and use. Technology such as the touchscreen and Microsoft Kinect makes the creation of NUIs possible. A NUI can be
seen as the goal, i.e. something to strive towards, as it is not a type of interface but rather a state of it, according to Wigdor and Wixon [2011d]. As smart-devices uses touchscreens, which opens up the possibility of creating a NUI, it is desirable to try to create games which strive for the very same thing. If one creates a game that feels natural to learn and play one would gain much when it comes to gameplay and the possibility of immersion.

5.2 The freedom of touchscreens - opportunity or curse?

As the controls of a game for smartphones or tablet computers have to be built in software in a more comprehensive way compared to controls to the PC, console or the like, it gives a lot of freedom to the developers. Which creates opportunities to create cool, new, ways to play games. However this makes it hard to create games of some genres, as there is no kind of standard. If for instance one were to create an FPS (fire person shooter) game to the PC than one would probably use the WASD-keys on the keyboard as input for moving around, as every gamer knows, the same applies to RTS (real time strategy) games were usually one's army or the like is controlled by clicking on them with the mouse. I.e. when it comes to PC or console game one can usually predict how to control a game before one actually plays the game by its genre or similar. This does not exist when it comes to touchscreen-based devices, but as a standardized behavior helps games by making them easier to learn and play they also in some ways hinder games to develop and create more innovating control schemes, as developers do not want to scare away potential costumers by introducing them to strange new ideas. To successfully create games that uses a non-standardized way of controlling it one would have to make the controls simple and more importantly as naturally as possible.

One very important thing is also to introduce the controls in steps, to make a novice user, a user that is new to the interface, to become a expert user, a user that is used to the interface and knows most of its features and similar. These steps shall build upon each other creating what Wigdor and Wixon [2011e] calls a “scaffold”. They also state that by learning the basics a user shall be able to use the interface more efficiently and perform more complex actions and build upon that knowledge. By introducing the controls in steps the user is more likely to not become overwhelmed. It is important the steps are not creating a too steep learning curve because that could contradict the positive effects. The user is often, or always, the one determine if the learning curve is too steep, thus user testing can be used as a good measurement for how well the interaction works.

5.3 Tools for interaction

As input is given at such low level from the touchscreen, tool(s) are needed to make use of the touchscreens potential. One such tool is the gesture, a gesture is
a motion or a series of motions one can make on the screen, using one or more fingers. They are frequently used on most smart devices, and by using them even higher level input can be created such as buttons or even joysticks.

5.3.1 Gestures

Gestures are used heavily on smart devices, and they range from the simple tap/touch to the more complex pinch gesture. Google [a] describes the following gestures in their developers’ guide for Android:

- **Tap/Touch** - Press, Lift
- **Long press** - Press, Wait, Lift
- **Swipe** - Press, Move, Lift
- **Drag** - Long press, Move, Lift
- **Double touch** - Two quick touches
- **Pinch** - 2-finger press, Move in- or out-wards, Lift

Gestures are generally great for creating intuitive interaction for games, e.g. if the user swipes over an object it gains force in the direction of the swipe, which makes the interaction feel natural. Gestures also have a couple of drawbacks for example using several gestures simultaneously may results in collisions. An example of such a case is if a finger is pressed to the screen and then moved a bit, the gesture can be one of two, it could either be a swipe where the finger leaves the screen with some speed or a drag where it stops and then leaves the screen. This may force the game to wait before it can create a responds and in turn makes the life for developers harder.

A larger problem is that not all gestures are intuitive, the user have to learn them somehow. When creating a more complex game the number of gestures may not be enough, which means that they have to be mixed and mean different things when the game is in different states, which is never a good thing.

However it is possible to use these kind of low-level gestures to create more high-level gestures, for example the drag gesture could be used to create an interface where the user can draw symbols on the screen, and where each symbol can be seen as high-level gestures.

5.3.2 Software buttons and joysticks

Even that gestures can be very powerful, intuitive and natural they may not be enough for a more complex game. A common way to create controls for more complex game is to create software- or virtual controls, i.e. controls that are drawn on the screen and respond to touches, as can be seen in figure 5.2a. Gestures are regularly used to build these high level software controls.
Many games with a wide range of controls seem to make use of software controls, rather than gestures. These types of controls can be very flexible as they can be moved around to suit a specific cause, e.g. one may have a joystick on-screen when the player is walking around and another set of controls when the player is driving. However, these types of controls have a tendency to be used a lot in games that have been ported to this platform from another, without adapting them to the platform, making the controls of the game bad and cumbersome.

If done right they can be a great tool for more complex controls, but one should be very careful as they suffer from many problems such as the fat finger problem (see 5.4.1) and feedback problems (see 5.4.2). It seems that many games that were ported to this platform from another use these kinds of controls without taking the touchscreens potential in regard, which makes the controls, in general, cumbersome. These kind of games feel ported, i.e. non-native.

5.4 Common problems

Controls and interfaces based on the touchscreen usually suffer from a number of different problems. Some are unique to the platform while some exist on other platforms as well as on smart devices.

5.4.1 Fat finger problem

A very famous problem with touchscreens is the so-called fat finger problem, which is according to Wigdor and Wixon [2011b] consists of two sub-problems; when a user touches a touchscreen he/she actually touches an area of the screen and not a single point which is how the system interprets the touch-event, the finger that touches the screen is in the way of the actual touch-point which makes it hard for the user to determined if he/she actually hit the chosen target.

From the Internet survey (see Appendix A) one can conclude that games seams to suffer from this specific problem, as well as the more general problems of non-responsive, inaccurate "ported" controls. As one precipitant answered:

Good. I wish my fingers didn’t get in the way.

Nearly all those games are crap, which is no wonder as most are developed by one or a small couple of persons and the devices are smaller than the hand which uses it - which makes every game crap because its either seeing or using...

There are a number of solutions for this problem according to Wigdor and Wixon [2011b], which includes making the interface, buttons and similar interface elements large enough to be touched without covering them entirely, use more input than just where the touch started and ended, e.g. use the direction a finger slides over a button, make the actual touch area larger than the visual button (so-called iceberg targets) and many more. One can also solve the problem in the same way
as Microsoft, Google and Apple have solved it in their virtual keyboard design, when a user touches a key the key is enlarged and shown above the key as can be seen in figure 5.3.

![Android Keyboard Screenshot](image)

**Figure 5.3:** A screenshot of the Android (4.0.4) keyboard, here one can observe that the key the user have selected is also shown above the actual key. This is done because otherwise the finger (visualized as the red dot) would be in the way to see the targeted key. I.e. this solves the fat finger problem the keyboard is faced with.

The fat finger problem is likely a problem one will be faced with when creating an interface for a touchscreen based device, which includes games as well. However one could hope that by using gestures or similar the problem would be totally avoided or at least minimized.

### 5.4.2 Feedback

When a user pushes a button or in someway interacts with a game he/she needs feedback. Feedback that the users actions were transmitted to the game, which received it and created the appropriate response/reaction. Feedback is essential for any interface and games are no exception, if anything they need it more then most other interfaces. On the PC or consoles there are two types of feedbacks; One when the user clicks the mouse-button or moves the joystick whereas he/she can feel that the mouse-button is down or up, performing a click, or that the joystick moved a bit hitting the edge and can therefore not be moved further in that direction, this is the feedback of the input device. The other type of feedback is given from the game, commonly visually or audibly, were the action of the user can be seen or heard.

As the controls of a game on a smart device is in many ways a part of the game and not something external as the mouse, keyboard or gamepad is. Feedback can
only be give through the game, i.e. when a user touches the screen he/she feels
the touch of the screen but not the button. The user have to see or hear a confirm-
ation that the buttons was indeed pushed. There are, of-course, the possibility
of using the vibrator to give feedback, however that kind of feedback is not very
accurate and not all devices have them, especially not tablets.

The end result is usually that a gamer playing on a smart device have to look at
the controls to see what he/she is doing, were a PC or console gamer can spend
the majority of their time looking at the screen while feeling their way around the
keyboard or gamepad. This (very probable) annoyance, may lead to frustration
and break the immersion of the game, making the game not fun.

5.4.3 Precision

As with console games, smartphone and tablet computer games are faced with a
precision problem when it comes to the controls. Compared to PC games which
uses the mouse as a pointing input device for most input that needs precision con-
soles usually uses joysticks as its pointing device and the touchscreen depends
on the users touches for precision. Joysticks are what Wigdor and Wixon [2011a]
calls a "rate control device" which suffers from being a generally bad input device
when it comes to precision. One of the two sub-problem the fat finger problem
consists of (see section 5.4.1), namely that the input the system detects when a
user touches the screen is a single point and not an area, leads to bad precision
for smart devices as well.

A potential solution to the problem is to either not demand high precision or
do as many console games does and guess what the user wanted to do. E.g. if the
user fires a spray of (very much virtual) bullet towards the vicinity of an enemy
the system can detect that and “fix” the problem by ”cheating” and make the
bullets hit the enemy instead.

5.4.4 Speed

Speed and particularly reaction speed may become a problem because of the as-
pects of the fat finger problem (see section 5.4.1), as fingers and even hands are
in the way for the user. For instance if a user performs a gesture, e.g. a downward
swipe on the screen, to be able to observe the direct consequence of that gesture
the hand, or at least fingers, have to be moved out of the way. By moving the
finger to and from the screen for every action performed in the game a lot of time
is lost. Which to a game that is fast paced may result in a bad score, GAME OVER
and consequently unhappy users.

Another issue which may result in bad reaction times is in the mobile nature
of touchscreen based devices, as they can be a bit cumbersome because they were
made to be mobile and not perfectly tailored for hands and fingers, e.g. compare
the Xbox 360’s gamepad (a) in figure 5.1 and the Samsung Galaxy S3 (d) and
one can argue that the gamepad is much more ergonomic than the smartphone.
Using something that can be cumbersome for precis and fast input will result in slower reaction speed from the user.

There is probably no real solution, expect for changing the game itself. The same way that some PC and console games differ, take Quake 3 Arena by id software and Call of Duty 3 Modern Warfare by infinity ward as an example; Quake is a really fast paced game which demands a lot from the user when it comes to reaction speed and precision compared to Call of Duty, which is by no means a slow paced game, but it is a lot slower than Quake. Even though the heyday of fast paced FPS games are over the faster games are found on the PC, another example would be Tribes Ascend by Studios which is a quite new, really fast, game for the PC. This is probably because of the controls, as the mouse and keyboard are far superior compare to the gamepad when it comes to speed and precision.

5.5 An example solution

As stated in section 5.3 gestures have the potential to be great controls for a game. However if the game demands more complex controls one would have to look at other ways of controlling the game. A potential solution to this is to combined gestures and software controls, e.g. when the user begins a swipe or drag gesture this is detected and the game creates a software joystick at the touch-point of the gesture. This way of utilizing the intuitively of gestures and the flexibility of the software controls are a way to combined the best of the two.

There are, of-course, limitations as the game have to guess which control to draw under the users finger. I.e. if the user should be able to control a characters movements and at what direction it fires its weapon, one would run into problems when the user would like to e.g. move and fire, stand still and fire. As if the movement joystick is initiated when the user drags his/her finger on the screen, the system needs to know when to initiate the joystick to control the fire-angle and the joystick to control movement.

5.6 Physics based controls

If the game is using a physics engine there are the possibility that the user is controlling something that has mass and a physical body, i.e. a body with mass will try to contradict change as it is bound to the law of inertia. Take a, somewhat realistic, racing game as an example; the car is very much a physical body, which can not stop, turn, reach the top speed at an instance. This behavior is expected for such a game. However in games where the user can not associate with something real one can run into problems where the user thinks that the controls are not responsive enough where it is in fact the nature of the physics engine and in turn the game.

One can argue that by adding mass to the controlled character so it become slug-
gish it will become a part of the core game mechanics of the game, but some users may find that the resulting non-responsiveness annoying. During the development of *Battle Angels* a lot of testing and experimentation it were conducted and it could be concluded that it is very difficult to justify the existence of sluggishness and non-responsive controls in a game, even if it is expected. By contradicting the physics, e.g. removing side-way drifting when moving in a specific direction or set the characters movement acceleration very high to make it reach its top-speed almost directly, the controls became much more direct and responsive, which in turn make users happier. As stated previously; games on touchscreen based devices suffers from a lot of problems were non-responsiveness is one of them, and by adding another, caused by the use of a physics engine, which add even more non-responsiveness would general be a bad move on this platform.

### 5.7 Conclusion

One can conclude that good controls are a difficult part of a game to create and implement, arguably the hardest (at least from a designer's point of view). As they are essential for gameplay and compared to other parts, such as graphics and audio, have a very large impact on core game mechanics. If an otherwise good game have bad graphics it can be overlooked and still be quite fun in many cases, however bad controls can kill an otherwise good game.

Touchscreens as an input device for games is hardly explored, which makes this an area of possibilities when it comes to new, intuitive and fun controls. However one needs to be aware of the many problems that exists in this area, and avoid to fall into the trap of many existing games that uses ported controls and in general lazy implementations, that does not try to utilize the potential of the touchscreen.
Part III

Resultat
In this chapter the results of the project will be evaluated, choices along the way will be discussed and important factors will be pointed out for the reader. In its essence; what have been learned during the project and what could have been done better.

### 6.1 Platform

Creating a game and trying to utilize the possibilities that the tablet offers as a platform was very interesting. But was this platform a good choice? One can make the assumption that developing anything to a new platform requires experimentation and exploration, which may prolong the development time. However even if it takes longer time to develop a game to tablets the number of users is huge and growing, as mentioned in the beginning of chapter 1, making the platform a possible goldmine for developers.

The chosen platform were a tablet, or to be more precis Apple’s iPad, which makes the game together with the chosen tools, iOS exclusive. This may harm the success of the game, in the future. As today the iPad is the best selling tablet on the market, which means that one can conclude that a game tailored for the tablet should be made available to at least the iPad, however one should also be aware that this can and probably will change in the future.

### 6.1.1 Tools

The choice of tools to be used in the development of the game were made in the very beginning of the project, it was done by conducting a small pre-study (see Appendix B). Now, at the end of the project, the question if that choice was
correct can be asked. Many of the chosen tools such as the game framework Cocos2D, physics engine Box2D and Apples multiplayer API Game Center are quite low-level, compared to other tools. Tools like Unity3D, which, should be pointed out, is not free for mobile platforms, that may make more complex games easier to create, and make the game, more or less automatically, multi-platform. However high-level tools have a tendency to be harder to learn, which were not the case with the chosen tools, they where found to be quite easy to learn and use. The most time consuming and complex parts of the development of the game were when we had to work outside of the tools, doing things that they did not directly support or even some things that even worked against the tools. Such as when synchronizing physical objects on the client from the server, or trying to utilize more than one core when loading a level. We have come to the, not surprising, conclusion that a low-level tools gives an edge when creating smaller, simpler games, however they can be found lacking when trying to create medium to large games that uses non-trivial components such as multiplayer.

An argument for the initial choice is that both Cocos2D and Box2D are open source and free, making them very accessible, as well give developers the possibility to make changes to the source code (which was done in the project). It is very difficult, if not impossible, to determined if the choice of using low-level tools were the correct choice. However we feel that one should research available tools a bit longer and more extensively, i.e. it may be a good idea do conduct a more comprehensive pre-study, even if it meant that it would have taken time from the actual development.

6.2 Game design

Most software projects uses agile processes for development, games are no different neither were Battle Angels. At the very beginning a pure singleplayer version of the game were considered, an idea which were kept for an extensive period of time. However when creating games one must be prepared to sacrifice partially completed work, ideas, time and sometime make bold moves in order to become proficient.

6.2.1 Major choices

Designing a game will mainly be about making choices, all choices will together form, structure and become the game. Some choices will be easy to make while others will be more difficult as they will influence the rest of the project. Battle Angels did in the end become a multiplayer game, this were not the initial plan but a choice along the way turned it all up-side-down. A suitable question then becomes — was it really worth it?

First, a major sudden change will most likely render a lot of work unusable so it must be carefully considered before it is executed. But external circumstance or new findings might have changed the possibilities and thus made the goal dif-
difficult to reach within the project’s time frame. Generally a such drastic change would be worth it if the end result would turn out considerably better, otherwise not.

Online multiplayer will drastically increase the burden for a programmer but will on the other hand open possibilities for players which previously had been closed e.g. playing together and against one and other. The greatest limitation for online multiplayer is imposed by the network which it uses, the network is the glue that keeps the game together, without it no (multiplayer) game would exist. A networked game will not only be limited by the hardware of the device onto which the graphics of the game are processed and displayed, it will also be limited by the "pipe" (i.e. network) through which players’ inputs and the server’s snapshot are transported. During the project with Battle Angels we realized that implications of networking can be rather time consuming, which have lead us to believe that simplicity is what one needs.

Simplicity can mean a lot of things, probably different things depending who one asks, however in Battle Angels’s case it were about making the visual design clear-cut for players. The game should be intuitive to use, learn and play. A player should not have to struggle with too many choices. This probably helped reduce complex situations which made the game easier to handle across a network.

Sometimes choices are influenced by feedback from people testing the game, maybe something is observed or the user testing say something which neither designers nor programmers have thought about. In Battle Angels case users testing the game had different expectations how the controlling system should work. Those expectations probably originated from other platforms were they had been playing games. This specific issue lead us to introduce alternative control for firing the character’s weapon. This alternative control went more in hand with console games, as the device could be held as if it were an actual hand-control.

6.2.2 Alternatives

Before, during and after the development of Battle Angels different alternatives were considered, some more exotic than others. They basically fall into three categories; turn-based, co-op and pure single player. They were all considered at the initial stages of the project and were rejected of different reasons.

Creating a turn-based games would require a total rework of the actual game, but it would open up the possibility of making the game more accessible and easier to play as player have all the time (at least more than a couple of seconds) in the world to complete actions (in a similar fashion to Wordfueld). A turn-based game would be easier to create compared to a real-time game as real-time network code is very difficult and complex to implement. However creating a real-time game were deemed more interesting and above all more unique as there exists some turn-based games but very few real-time games for smart devices.
Adding co-op components, as in human versus AI, to the game as it is now would probably add a lot of depth and make the game even more versatile and above all fun. It would, however, be a lot of work. Where the AI is probably the biggest challenge. It may be something that can be added in the future.

As previously stated Battle Angels were initially intended to be a singleplayer game, and more of a classical platform game. Making such a game for smart devices is not a bad idea, as there are always the problem with bad network connections and similar. However we felt that it would be a lot more compelling to create a multiplayer game. Perhaps because we tend to play multiplayer games ourselves and find them, usually more fun than singleplayer games. They are, however, not mutually exclusive, i.e. it is possible to add a singleplayer mode to Battle Angels.

6.2.3 In practice

In the end we asked ourselves; are we happy with the resulting game? Short answer; yes. Long answer; to a degree, as there are many aspects of the game which turned out better than others. The overall feeling of the game is quite good and it does definitely have a potential to be very fun.

Things that we are especially happy with are; controls, modes and the editor. Controlling the game is quite easy and intuitive, touch the screen and pull the joystick to move, tap on the screen to fire or use the alternative fire joystick in the lower right corner. The different game modes composes great variation to gameplay for example in the mode treasure hunt it becomes more important to move around fast and precise, to catch as many treasures as possible, than killing each other. Editing and building levels became surprisingly easy. One can build a level which is quite playable in a short amount of time but it is also possible to spend hours perfecting and building to get every little detail to look good and feel right.

Even though most of the game turned out good, there are still a few things which may be improved which mainly are; client-side predictions, delays for the client acting server, more content and 3G/GSM, local wifi (LAN), bluetooth connections. Battle Angels currently have predictions for firing the weapon but including it for other things like a player’s movement would probably help increasing the perceived responsiveness of the game. Another improvement for equalizing gameplay for all players would be to impose an interpolation delay to the client acting server. As discussed in section 3.3.4 entity interpolation imposes a constant view lag for a client, this view lag is not visible on the client acting server as it never receives any snapshots for interpolation. This gives the player playing on the client that acts server a slight advantage as things literally happens more direct.

Perhaps the most important improvement would be to add support for 3G/GSM based networks to increase the game’s accessibility, but that would require major
redesigns and data traffic limitations. Basically in order to support such a network, the synchronization process must send snapshots less frequently and/or the number of objects must be reduced and that would be a major challenge to achieve.

There are also some things in Battle Angels which we would like to improve but can not because of limitations in external libraries being used, these are; joining games, searching for matches and requirement of having an Internet connection to be able to play. Game Center which were used as the multiplayer API only support lobby-based match making which efficiently ruin every possibility for players to join a game in progress. This type of joining do not really go well in hand with action games because they seldom requires the server to be full in order for the game to start. Another limitation is the process of auto-matching, searching for a game, it does not really give the player fine-grained control. A player can for example not search for a game playing any mode and/or a specific map.

The greatest of limitations are the online requirement. It means that it will not be possible for players to connect their devices through a home-network such as a local-area network or directly via peer-to-peer networks. Game Center provides possibilities to mitigate this by using an alternative API for direct communication. However the application have to handle a lot which makes it cumbersome to use.

Even though there are a few problems and improvements which can be made, the overall gameplay is good and lag is only present occasionally depending on the network used. For obvious reasons it is not possible to present numerical values as results, the only possible way is to play the game. Figures 6.1, 6.2 and 6.3 show screen captures or images taken while playing the game.
This is the main menu in Battle Angels.

This is the place where levels that appear on the server can be downloaded.

This is a player’s workspace, levels which are in progress. Can be the player entirely own levels or copies of level original created by others.

The editor, this is the place where the player is modify levels.

Figure 6.1: A series of screen shots from the menu and editor of Battle Angels.
6.2 Game design

(a) The player have just been spawned.
(b) The player flying around, using thrusters.
(c) The player turned around.
(d) The player continues to fly, still the same level.

Figure 6.2: Shows a series of screen captures taken while playing single-player, i.e. testing a level since *Battle Angels* do not really have a single-player mode other than for testing.
The left device is hosting a game, the right device is idle.

The left device have completed the initial hosting steps and have invited the right device. The right device have accepted the invite. The match can start.

Both devices shows approximately the same as the players are initially close.

Later in the same game, the player have flown away a bit, still the same level.

Figure 6.3: Shows a series of images taken while initiating and playing a multiplayer match.
It can be concluded that designing and implementing games, are in general, no easy task and using the tablet as a platform it becomes even more difficult, especially a real-time action game with multiplayer elements, as there are no "templates" for how to exploit the unique capabilities of the touch-screen. The lack of stable and reliable wireless Internet connections (when not using local Wifi-connections) also make the development of a multiplayer game a challenge. However the possibility for innovation is huge. The tablet platform is relative new compared to more established platforms such as the PC or consoles, which is probably the reason why it is difficult to find research papers and common guidelines for implementation. Therefore experimentation and exploration is needed in order to find suitable solutions that fits the application.

Multiplayer is not essential, but it is very important in games today, especially the social aspects. Adams [2009a] states that the flexibility and power of online multiplayer enables designers and programmers to produces entertainment experiences for users which simply can not be produced in other forms. This fact along with the fact that the market is huge and still growing makes the potential for real-time action games great. It can be concluded that trying to create a real-time action multiplayer game is worth the effort, as long as the unique capabilities and limitations of this platform are exploited and considered.

### 7.1 Future of Battle Angels

*Battle Angels* will hopefully make it to the App Store at some point, however in order to release it, it must be thoroughly tested and polished. It is very important that users have the impression that they are viewing a complete work and
not a work in progress.

Another important aspect of *Battle Angels* is that its content is created by users and in order to attract users there must exist interesting and good content. Therefore a lot of work have to be put into actually creating some initial good content that have the possibility to attract users.

### 7.2 Suggestions for future work

There are a lot of work which can be made in the area of smart devices however it is unclear which direct approach one should use. A possible enhancement for problems related to touchscreens e.g. controls would be to conduct user-testing in form of design research and brainstorming described by Saffer [2006]. Design research is the act of investigating a service potential which could be used to find one good solution among several alternatives.

Papers describing online multiplayer action games for statical environments can be quite easily found on the Internet however techniques for handling huge amounts of objects and/or physical-based environments are more difficult to find and are thus subject for potential research.
An Internet survey, called Game interaction for touchscreen-based devices, with a total of 76 participants, all answers were given anonymously. The survey was posted on Internet forums and social networks such as http://reddit.com.

The results are presented so that the question is shown first, then the different answers and how people answered. Some questions had more than one answer, which means that the percentage of a specific answer is how many chose that answer of the total people taking the survey. In some cases when the precipitants did not feel that any of the available alternatives suited them well the alternative “Other” could be chosen, which meant that he/she had to specify the answer as well. After every question and results there are table showing how many skipped the question.
A.1 Question 1

This question was non-skippable.

Have you ever played any games on touchscreen-based smartphones or tablets?

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Percentage</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>98.7 %</td>
<td>75</td>
</tr>
<tr>
<td>No</td>
<td>1.3 %</td>
<td>1</td>
</tr>
</tbody>
</table>

A.2 Question 2

If you have, how often do you play?

<table>
<thead>
<tr>
<th>Estimated hours per week:</th>
<th>Mean value</th>
<th>Total</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 338.69</td>
<td>100 402</td>
<td>75</td>
</tr>
</tbody>
</table>

One of the participants decided to answer that he/she played 100 000 hours per week, which is quite impossible, the answers were therefore deemed non-serious and therefore removed, which in turn gave the following results.

<table>
<thead>
<tr>
<th>Estimated hours per week:</th>
<th>Mean value</th>
<th>Total</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.432</td>
<td>402</td>
<td>74</td>
</tr>
</tbody>
</table>

Skipped 1 (2)
Answered 75(74)
A.3 Question 3

What kind of games do you play? (multiple answer question)

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Percentage</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform, &quot;Mario clones&quot;</td>
<td>36.0 %</td>
<td>27</td>
</tr>
<tr>
<td>Puzzle/Trivia, for example Wordfeud and Sudoku</td>
<td>58.7 %</td>
<td>44</td>
</tr>
<tr>
<td>Action puzzle, fast paced puzzle games such as Tetris</td>
<td>48.0 %</td>
<td>36</td>
</tr>
<tr>
<td>Strategy/Construction and management, for example Sim City</td>
<td>48.0 %</td>
<td>36</td>
</tr>
<tr>
<td>Tower defense</td>
<td>48.0 %</td>
<td>36</td>
</tr>
<tr>
<td>Racing</td>
<td>25.3 %</td>
<td>19</td>
</tr>
<tr>
<td>Adventure, for example Zelda</td>
<td>38.5 %</td>
<td>29</td>
</tr>
<tr>
<td>Other</td>
<td>24.0 %</td>
<td>18</td>
</tr>
</tbody>
</table>

A.3.1 Other answers

- Shooters
- Board Games. Catan, Carcassone, Bang, Ticket to Ride etc.
- I have at least 300 apps/games, comprising all types
- I program mobile apps.
- Card games, board games
- Action/arcade
- Physics
- Games that use the gyroscopes. Lots of small indie apps
- Infinite/Runner, Arcade/Shooter, Board games
- Whatever Jetpack Joyride is (also, it’s spelled ’puzzle’ ;))
- RPG
- RPG
- Temple run
- Card games like Magic
- RPG like Zenonia and Dungeon Hunter
- Some mmorpg
• First Person Shooters, Open world sandbox games (Minecraft)
• angry birds

Skipped 1
Answered 75

A.4 Question 4

Where do you usually play?

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Percentage</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>At home</td>
<td>42.7 %</td>
<td>32</td>
</tr>
<tr>
<td>On the bus/train/etc.</td>
<td>8.0 %</td>
<td>6</td>
</tr>
<tr>
<td>Anywhere, when waiting for something</td>
<td>45.3 %</td>
<td>34</td>
</tr>
<tr>
<td>Other</td>
<td>4.0 %</td>
<td>3</td>
</tr>
</tbody>
</table>

A.4.1 Other answers

• At work
• as a break from work
• At work when there’s nothing else going on.

Skipped 1
Answered 75
A.5 Question 5

What is the greatest difference, according to your opinion, between games for touchscreen-based devices and games for PC and consoles?

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Percentage</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
<td>2.7 %</td>
<td>2</td>
</tr>
<tr>
<td>Audio</td>
<td>0.0 %</td>
<td>0</td>
</tr>
<tr>
<td>Controls</td>
<td>76.0 %</td>
<td>57</td>
</tr>
<tr>
<td>Multiplayer</td>
<td>2.7 %</td>
<td>2</td>
</tr>
<tr>
<td>Other</td>
<td>18.7 %</td>
<td>14</td>
</tr>
</tbody>
</table>

A.5.1 Other answers

- Portability and ability to play in short segments at a time (e.g., five minutes)
- Convenience
- size of what you are play on
- simplicity. PC games usually immerse you, while iPhone are more casual
- Convenience of having it always on me
- Accessibility, they’re loaded in seconds on my iPad.
- Gameplay depth/complexity
- mobility, choice
- Portability
- smoothness
- mobility
- the feeling and the fun. It's more free on a PC. Quess u can say the multiplayer is better on PC.
- It drains my battery yo!
- mobility

Skipped 1
Answered 75

A.6 Question 6

Have you ever considered buying a physical control add-on to your phone/tablet or do you already own one?
A.7 Question 7

According to your experience what will ruin a otherwise fully functional game the most?

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Percentage</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad graphics</td>
<td>4.0 %</td>
<td>3</td>
</tr>
<tr>
<td>Missing/Bad audio</td>
<td>0.0 %</td>
<td>0</td>
</tr>
<tr>
<td>No physics</td>
<td>0.0 %</td>
<td>0</td>
</tr>
<tr>
<td>Bad Controls</td>
<td>85.3 %</td>
<td>64</td>
</tr>
<tr>
<td>Other</td>
<td>10.7 %</td>
<td>8</td>
</tr>
</tbody>
</table>

A.7.1 Other answers

- The futile effort to adapt classic gaming models (such as platforming) without reinventing them to adhere to the inherent limitations of the platform (i.e. pasting virtual controls as a solution to every function)
- Excessive reliance on in app purchases
- too many ads/in-app purchases
- Unintuitive gameplay combined with poor instructions
- Ingame ads and ingame option to buy crap
- drm
- smoothness
- the fun. I wouldn’t play a game if it wasn’t fun. And u have to understand the goal of the game.

Skipped 1
Answered 75

A.8 Question 8

What is the major improvement you would like to see in future touchscreen-based games?
A.8.1 Other answers

- More innovation to create games suited for the platform. See Battleheart, Trainyard, Galaxy on Fire, Soosiz, Let’s Create Pottery,

- 3dimensional Touchscreens ("Movescreens")

- Improved hardware in the device itself, enabling better graphics audio and controls.

- Improved multiplayer: it can be a nightmare to successfully get into games with my wife or brother even when we are right next to each other

- hm, storyline?

- smoothness

- all of the above

Skipped 1
Answered 75

A.9 Question 9

Your final opinions (not covered in the survey so far) on games for touchscreen-based devices.

Good. I wish my fingers didn’t get in the way.

Phone/tablet games will never truly replace handheld consoles like DS and PSP until physical controls became a standard built-in feature that all devs can rely on to be available on all devices; either that or touch screens that can produce tactile buttons on their surface at will.

When it comes to phone games, You just feel less invested in continuing the game, and it’s really ideal for short bursts of playing as opposed to playing on a computer/console where you have to setup your computer/console and start the games up. These things usually demand much more attention, as opposed to handheld gaming where you can easily start and at anytime and easily come back to your game at anytime you like.
Mobile phones are not designed to be gaming systems. They are engineered for simplicity, using a minimum of buttons, in order to meet the demand and purposes of a completely different market. Gaming on them is incidental.

Nearly all those games are crap, which is no wonder as most are developed by one or a small couple of persons and the devices are smaller than the hand which uses it - which makes every game crap because its either seeing or using...

It’s a goldmine for developers, but it’s hard for console and PC gamers to convert to touch gaming, but easy for FB gamers and the like.

Pick up and play vs sit down for hours.

Love them, but it will take some time before developers really master the differences. Tight and easy to use controls are critical... A single touchscreen is very different than a classic gaming controller and throwing a bunch of virtual buttons onscreen almost always results in failure. Also, games are trending towards the casual and part of that is designing games that can be played across many brief episodes but still maintain their fun.

If you’re making a game especially for iOS you need to have good graphics, good audio and good controls. All this on top of the great gameplay. it also helps if the controls are intuitive or innovative.

Spectacular! But the lack of a well-functioning control experience is really holding back the platforms ability to really start hosting well made, and larger games.

Most touchscreen games seem to be "casual" games. I’d like to see more in-depth "hardcore" games, especially strategy and RPGs.

If I wanted to use a controller, I would. Games should utilize the unique capabilities of touch for interaction.

Not a huge touch based gaming fan if I’m really going to get into a game. Something like Angry Birds for taking a dump is great, but like a racing game where you have to pay attention for a bit.. I’d rather play Battlefield or Minecraft.

It’s important to me that they’re made to be played on a screen instead of emulating buttons and controllers.

Most bad control schemes are ported d-pads done badly. I would like to see games take advantage of the touchscreen fully (you can interact with anything and in any way you can touch on-screen) and make games less dependent on fine, twitchy controls.
"Bad controls" really hit home. If a game isn’t responsive enough it’s pointless

It’s hard to separate out the touch screen nature of gaming on my iPhone & iPad from the fact that I carry my phone around *everywhere* (and it has a permanent Internet connection) and I have my iPad with me almost as much (except when the wife grabs it).

I prefer playing on a dedicated gaming-system. Preferably a PC.

I only play games on my iPhone that would be considered time waster type of games. It’s hard for me to play a game that takes up lots of time and has lots of levels. When I play a smart phone game I want something that will keep my occupied for a few mins until I find something else do do.

It has potential but the lack of physical controls will keep becoming a serious contender in the mobile market. But its not that bad for casual gaming.

good luck!

It works really well for games designed for the device. However ports don’t always work as well if the games aren’t redesigned for a touch interface.

I prefer games like Drop7 that have no timer; stuff I can play standing in line and not worry about dying when I look away for a second. I think on-screen gamepad controls suck, and it’s better when designers use the screen as intended, like with fruit ninja or DoodleJump. I made it through this whole survey without saying Angry Birds!

Don’t ruin the game by adding ingame ads and ingame mandatory stuff that cost money.

Usually good.

It’s amazing to have so many games with you wherever you go

I would like to see more games with dual device control. Ie steering a game on your ipad with your iphone

Games on smartphones should be different from console and computers, they should have their own catch and different controls so it can be played better on the phone and shouldn’t be a ‘ported’ version of the console/computer version of the game.

It’s hard to be serious about them.
Fix the controls and make the games smooth

I really like them, but it would be better if the control would be better. And Thx for the post on reddit :).

They are good to pass time but are no means a replacement for PC or consoles.

If a non-slide Android phone with dedicated game buttons existed, I'd buy it in a heartbeat. Nothing is as terrible as playing on a screen with poor multitouch.

None

a-ok

futur is spelled wrong in question 8. Should be future.

Games should have a soul

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Skipped</td>
<td>38</td>
</tr>
<tr>
<td>Answered</td>
<td>38</td>
</tr>
</tbody>
</table>
B.1 Introduction

This document explains the choice of platform and game engine, framework or library to be used to create a 2D game to base testing on. This evaluation is based upon information from the official websites as well as the experience of the project group.

B.2 Game SDKs

There are many good game engines, frameworks and libraries to choose from. A evaluation among the most common were made to make the choice easier. This evaluation is not platform dependent, as it may influence which platform to be used. Note that all engines, frameworks and libraries in this, rather small, evaluation were made (or modified) for mobile touch-screen-based devices, that exists on the market, today.

B.2.1 Requirements

- Free.
- Easy to install and use.
- Compatible with one or more physics engines.
- Compatible with other APIs, frameworks etc. for that specific platform.
- Fairly mature and stable.
- The project cannot be completely dead.
B.2.2 Cocos2D

Cocos2D\(^1\) is a game framework mainly for iOS devices, it is free and open source under the MIT License\(^\text{MIT}\). There exist ports to Android and a 3D version (Cocos3D). It is fairly mature and works well together with physics engines such as Box2D and Chipmunk.

B.2.3 Ooolong Engine

Ooolong engine\(^2\) is written in C++ and Objective-C for iOS devices. It is a 3D engine which is free and open source. It supports the physics engine, Bullet. It seems fairly mature.

B.2.4 Ogre3D

Ogre3D\(^3\) is not a game engine and really not framework either, it is a 3D rendering engine with many third party modules. It is mainly used to create games and similar application to computers but there exists SDKs for iOS. It is free and open source and have a rather large community around it.

B.2.5 libGDX

libGDX\(^4\) is a game library developed for cross-platform for desktop (Windows, Linux and OSX) and Android written in Java, it seems very mature and features easy integration with other Java projects such as physics engines.

B.2.6 andengine

andengine\(^5\) is 2D game engine, that is free and open source. It is (as the name suggests) an engine for Android. The engine features things like split-screen, multiplayer over network, multi-touch and a physics engine (Box2D).

B.2.7 Others

There exists a lot of promising game engines and frameworks, but many are non-free, which is also the reason why they were not considered. Some of them can be seen below:

- Epic Unreal Development Kit (UDK)\(^6\)
- Unity3D\(^7\)
- Corona\(^8\)

\(^1\)http://code.google.com/p/cocos2d-iphone/
\(^2\)http://code.google.com/p/oolongengine/
\(^3\)http://www.ogre3d.org/
\(^4\)http://code.google.com/p/libgdx/
\(^5\)http://www.andengine.org/
\(^6\)http://udk.com/
\(^7\)http://unity3d.com/
\(^8\)http://www.anscamobile.com/
B.3 Platform

As the game shall be in 2D all engines/frameworks in 3D are excluded, which means that the choice is between Cocos2D, libGDX and andengine.

B.3.1 Requirements

The chosen platform must fulfill the following requirements.

- Used for games.
- Fairly modern.
- Accessible.
- At least one engine or framework with the focus of game development.

It is probably a good thing if the chosen platform is known and used by regular consumers.

B.3.2 Android

As of today Android can be seen as the "under-dog" when it comes to computer tablets, but there exists a number of devices that is good enough and fast enough for games. As there are at least two engines, frameworks or library to chose from (libGDX and andengine) when developing games for it and everything that has with Android development is cross-platform, free and easily accessible, Android is definitely a very good choice. However testing and making the game compatible on many different devices will be tough as there are so many with different hardware.

B.3.3 iOS

iOS holds the majority of the tablet market, with their iPads, and there are only two kinds of iPads which will, probably, make the development easier as not much time has to be put into make the game compatible on different devices.

---

9 http://www.garagegames.com/products/torque-2d/iphone
10 http://www.sio2interactive.com/
will also make the testing of the game easier. As of this moment the project have access to the iPad and the iPad 2. The negative side of iOS development is the tools, SDK and similar as a computer running OSX, with Xcode and the iOS SDK installed, which is not free, has to be used. However project already have access to these tools, which makes this a non-issue.

B.3.4 Results

The choice of platform is very hard. Android would probably win if it was not for the fact that the project have access to the tools needed for development and testing for iOS, which means that the development will be done using Xcode for the iPad and iPad 2.

B.4 Conclusion

As the iPad is the platform of choice the choice of game engine, framework or library becomes quite easy, as Coscos2D seams to be the most suitable for the task.
Bibliography


Daniel Wigdor and Dennis Wixon. No touch left behind: Feedback is essential.


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Johan Lindström and Tobias Malm