Kingston University London

Faculty of Science Engineering and Computing

School of Computing and Information Systems

Student Name: Lawrence Schmid

Course Title: BSc Computer Science

Supervisor: Dr. Darrel Greenhill

Plagiarism Declaration

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computing and Information Science. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety I have made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as his or her own work.

Date: 29/4/12

…………………………….

Signature: Lawrence Schmid

KINGSTON UNIVERSITY

Faculty of Science, Engineering and Computing

School of Computing and Information Systems

CI3330 – Individual Project

Project Proposal – Dragon Island: Java 2D Side Scrolling Platform Game

Lawrence Schmid

Issue 1: 19th April 2012

Table of Contents

Introduction

This aim of this project is to develop a 2D side scrolling platform game using the Java programming language which is designed to be played on mobile devices. It will be similar in gameplay to the Super Mario Bros. games developed by Nintendo.

In a 2D side scrolling platform game the player runs and jumps to and from platforms while avoiding enemies and collecting power-ups. The level area can be greater than the size of the screen or viewport as the camera will follow the current position of the player. The main objective of the game is to avoid or defeat enemies and get to the end of the level before the time limit runs out while trying to set a high score. There are collectable items such as coins and power-ups which the player can collect to earn points and gain new abilities. (Brackeen, 2003, p. 221)

Project Objectives

The main objective of this project is to create an exciting action packed game which is easy to pick up and play without having to learn difficult controls. It should provide an enjoyable game playing experience which challenges the client or game player to complete a task.

It will be possible to determine the progress of the project as each new feature will affect the game play dynamics, user interface, graphics or performance. The success of the project can be measured by the opinion of the client game player and during development by the programmer. The project will be aimed at all age groups and use colourful cartoon style graphics which make the game more appealing to a younger audience.

The project will be implemented and tested using the Java Standard Edition programming language with the Eclipse development environment and documented with the JavaDoc utility. The Java software development kit (JDK) includes all the functions and libraries required for displaying 2D images and playing music and sound while updating game events in a thread. Java applications use a virtual machine so the finished game will run on any hardware which supports the JVM (Java Virtual Machine) without having to maintain separate code bases. During the implementation and evaluation stages detailed documentation will be created consisting of UML diagrams and JavaDoc comments. This documentation should be useful to other developers and academic students who want to develop their own 2D games. (Pankrashchenko, 2010, pp. 39, 45, 47)

The initial objective of this project was to display a user controllable character on the screen, but as this was easily accomplished it will be possible to develop more advanced game play features. The personal objectives of this project are to learn more about developing 2D side scrolling games, user controllable characters, level maps, collectable items, enemies (non-playable characters), displaying graphics and playing sounds. If the project is success and all of the requirements have been completed a separate version of the game will be created for Android smartphone / tablet devices and the GP2X or PSP handheld consoles.

Initial Requirement Specification

Functional Requirements:

- User controlled player who can walk, run, jump, crouch and throw fireballs
- Enemies with AI which the player will have to avoid or kill to gain points
- User interface allowing the player to start, pause and select game options
- Collectables items such as coins and power-ups which change the players state
- Collision detection between player, enemies and level objects
- Player, enemy and background graphics with animation
- Level editor which allow the player to create new levels for the game
- At least one fully playable level which can be edited
- Score counter
- Level time limit
- Player lives counter
- Music and sound effects

Non-functional Requirements:

- The implementation will take into account future growth and extensions
- The game will be released under an open source licencing agreement
- The source code should be comprehensively documented
- The player will not experience game play slow down or performance issues

Paper Prototypes

The following paper prototypes give an early indication of what the implementation of the project will look like when it is complete.

Main Game

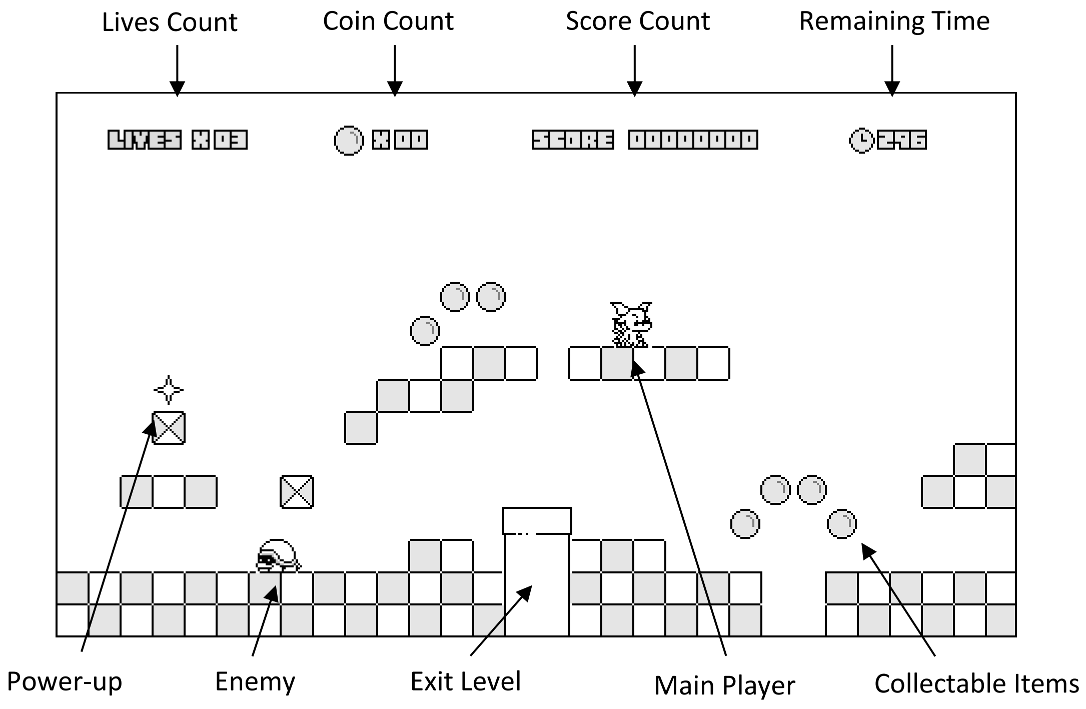


Figure 1.1 Level with enemies, collectable items and level exit

Title Screen / Main Menu

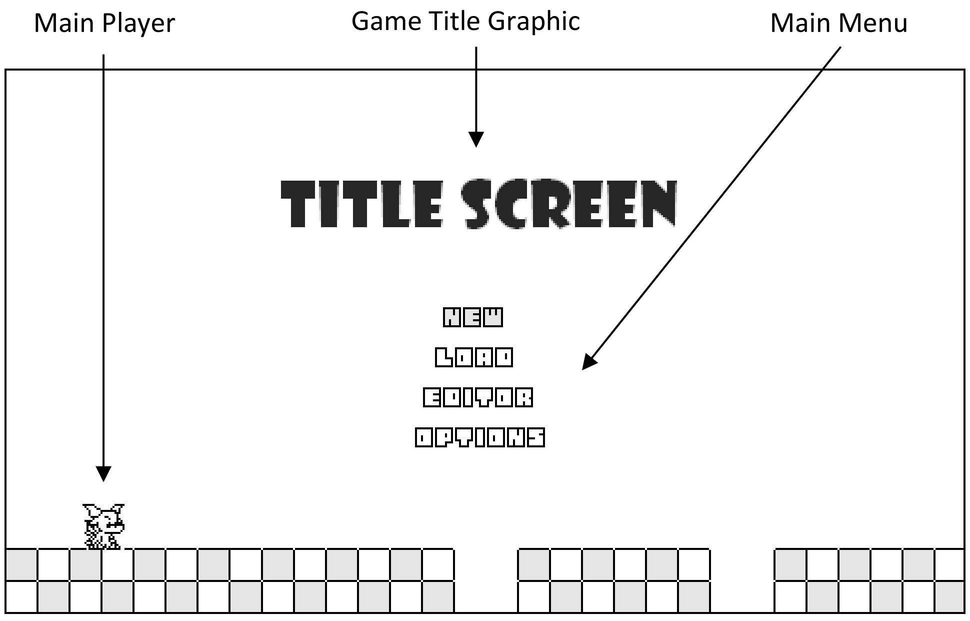


Figure 1.2 Title screen with main menu

Project Scope

The hardware scope of this project is PC and mobile devices such as Android, GP2X, PlayStation Portable or iPhone. The Java programming language follows syntax similar to C++ and the finished game should be easily ported to other hardware. The Java Development Kit (JDK) includes all the necessary software functions to develop games and applications for the target operating systems.

The graphics and sound resources where possible will be copyright free and available under the public domain or creative commons license scheme. Screenshots, references and URL locations of the original graphic or sound resources will be kept along with any modifications to the original graphics.

The screen size or viewport is an important consideration as one of the initial requirements of the game play is that the camera will follow the player around the level which extends well beyond the limits of the gaming pane. This is determined by the size of the drawing canvas inside of the application window or the screen size of the hardware device. (Davison, 2005, p. 298)

The game will let the player explore levels by walking, running and jumping between platforms. There will be different level themes with multiple backgrounds and tilesets used to display the graphics. The game will include a range of different enemies which have different behaviours such as flying, jumping and falling. The player will be user controlled and have different movements with corresponding animations. The player will be able to collect items placed in the map and hidden in blocks by jumping and hitting the block containing the item. The collectable items include coins, mushroom (increase size), fire flower (fire weapon) and extra lives. The main aim of game will be to avoid the enemies and try to reach the end of the level before the time limit runs out. There will be warp pipes which the player can enter to move to the next level or enter a bonus area. The player will be able to pause and save the progress of the game and resume it at another time. The game will feature a level editor which allows the player to create their own levels using the graphics from the game. The levels will provide a challenge for the user to complete and include features such as collectable items and enemies. The level data will be tile based and contain objects, items and enemies defined in a map array which is read from a file using a proprietary file system. The level data should contain additional header information which includes level variables such as a time limit and player start and end and coordinates.

There will be music and sound effects which are played back using midi or mp3 libraries such as jLayer for Java and SDL for C++. The game will use the same screen size resolution as the PlayStation Portable 480x272 pixels making it easier to port the game to mobile devices. The game will not include a multiplayer feature but this could be added later with two players cooperating to complete the level. The game does not include internet connectivity but it would be possible to introduce features such as an online high score table, downloadable level content in-app purchase and paid advertising.

Project Management

The project plan and implementation will follow a project management methodology known as Dynamic Systems Development Method (DSDM) which will incorporate ideas from Rapid Application Development (RAD).

DSDM is a project management methodology which is commonly used while developing computer software. It gives the project manager the chance to plan and set individual targets using set time scales called sprints with the overall aim of completing the project within time and budget constraints. (Wikipedia, 2010)

There are many advantages to using project management methodologies:

- A better overall finished implementation of the end product
- An easier development process which is less error prone
- Keeping the project on schedule and within the set timescale
- A consistent standardised API with well documented classes

The disadvantages include:

- Difficult to learn and time consuming to maintain documents
- Errors in project management could lead to problems with the implementation

Work Breakdown Structure

This work breakdown structure diagram shows the structure of some of the most important tasks which must be completed in the analysis, design, implementation and evaluation of the project lifecycle for the project to be a success.

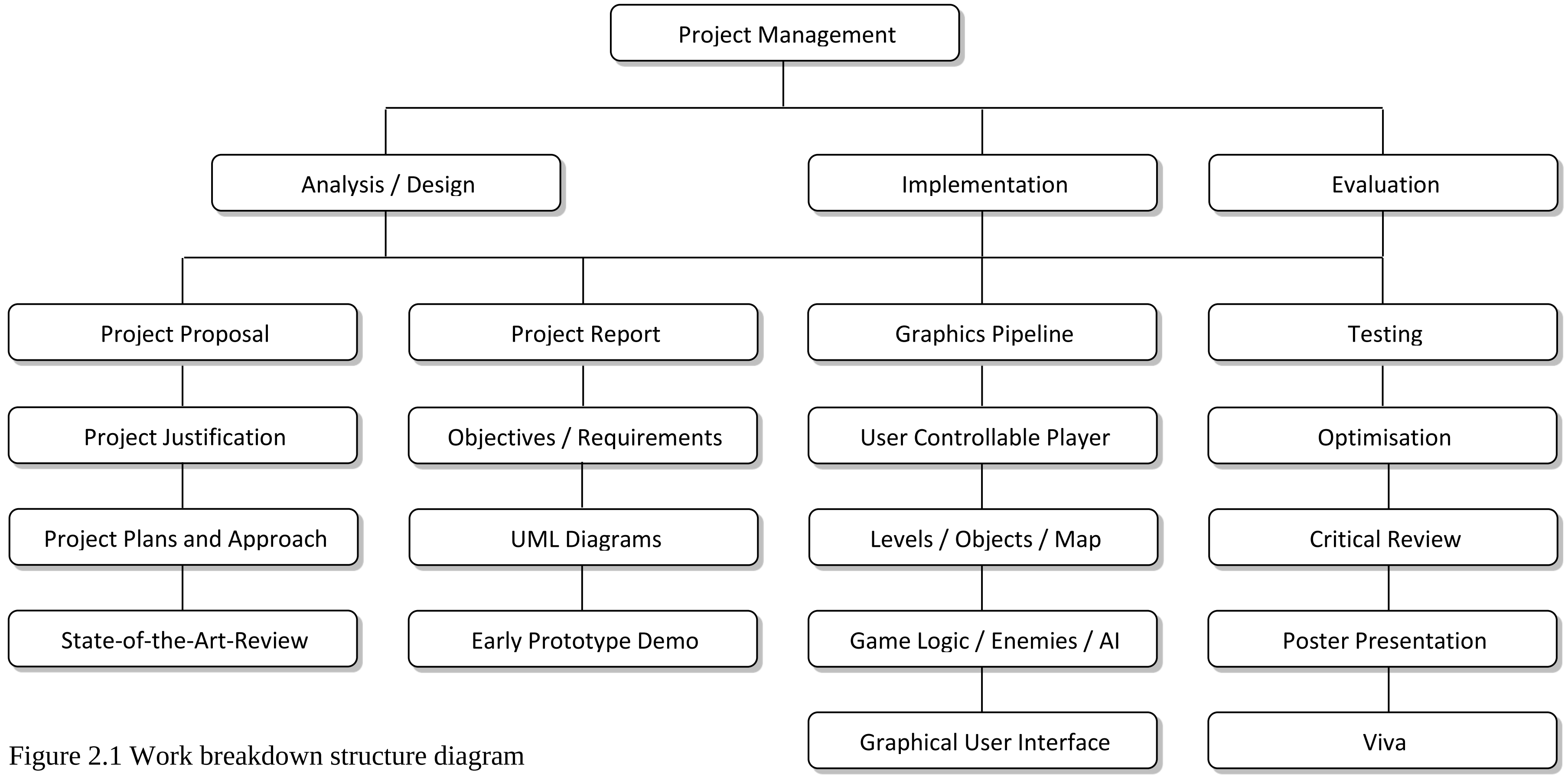


Figure 2.1 Work breakdown structure diagram

Gantt Chart

This Gantt chart shows how much time will be spent on completing each task in the project lifecycle. If there is a problem completing a task that will affect the project completion a contingency plan will be put into place.

ID	Tasks	Start	End	Duration (Days)	% Complete	Weekdays Days	Days Complete	Days Remaining
1	**Project Proposal**	26/9/11	10/11/11	46	100%	34	46	0
2	**State-of-the-Art-Report**	10/11/11	31/1/12	83	100%	59	83	0
3	**1st Prototype Demonstration**	31/1/12	6/2/12	7	100%	5	7	0
4	**Report**	6/2/12	3/5/12	88	100%	64	88	0
5	**Analysis**	6/2/12	25/2/12	20	100%	15	20	0
5.1	Approach	6/2/12	25/2/12	20	100%	15	20	0
5.2	Platform Overview	6/2/12	25/2/12	20	100%	15	20	0
5.3	Questionaire	6/2/12	25/2/12	20	100%	15	20	0
5.4	Stakeholder Analysis	6/2/12	25/2/12	20	100%	15	20	0
5.5	SWOT Analysis	6/2/12	25/2/12	20	100%	15	20	0
5.6	Risk and Competition Analysis	6/2/12	25/2/12	20	100%	15	20	0
5.7	Requirements	6/2/12	25/2/12	20	100%	15	20	0
5.8	Benefits for Clients	6/2/12	25/2/12	20	100%	15	20	0
6	**Design**	25/2/12	15/3/12	20	100%	14	20	0
6.1	Use Case Diagram	25/2/12	15/3/12	20	100%	14	20	0
6.2	Class Diagram	25/2/12	15/3/12	20	100%	14	20	0
6.3	Sequence Diagram	25/2/12	15/3/12	20	100%	14	20	0
6.4	Activity Diagram	25/2/12	15/3/12	20	100%	14	20	0
6.5	Package Diagram	25/2/12	15/3/12	20	100%	14	20	0
6.6	User Interface Prototype	25/2/12	15/3/12	20	100%	14	20	0
6.7	Game Design	25/2/12	15/3/12	20	100%	14	20	0
7	**Implementation**	15/3/12	3/4/12	20	100%	14	20	0
7.1	Game Framework	15/3/12	3/4/12	20	100%	14	20	0
7.2	Graphics	15/3/12	3/4/12	20	100%	14	20	0
7.3	Level Map and Editor	15/3/12	3/4/12	20	100%	14	20	0
7.4	Load / Save Game	15/3/12	3/4/12	20	100%	14	20	0
7.5	Audio	15/3/12	3/4/12	20	100%	14	20	0
7.6	Player and Enemies	15/3/12	3/4/12	20	100%	14	20	0
8	**Evaluation**	3/4/12	22/4/12	20	100%	14	20	0
8.1	Testing	3/4/12	22/4/12	20	100%	14	20	0
8.2	Non-Programming Issues	3/4/12	22/4/12	20	100%	14	20	0
8.3	Programming Issues	3/4/12	22/4/12	20	100%	14	20	0
8.4	MoSCoW Evaluation	3/4/12	22/4/12	20	100%	14	20	0
8.5	Requirements Evaluation	3/4/12	22/4/12	20	100%	14	20	0
8.6	Critical Review	16/4/12	5/5/12	20	100%	15	20	0
9	**Poster Presentation**	25/4/12	9/5/12	15	100%	11	15	0
10	**Viva**	9/5/12	23/5/12	15	100%	11	15	0

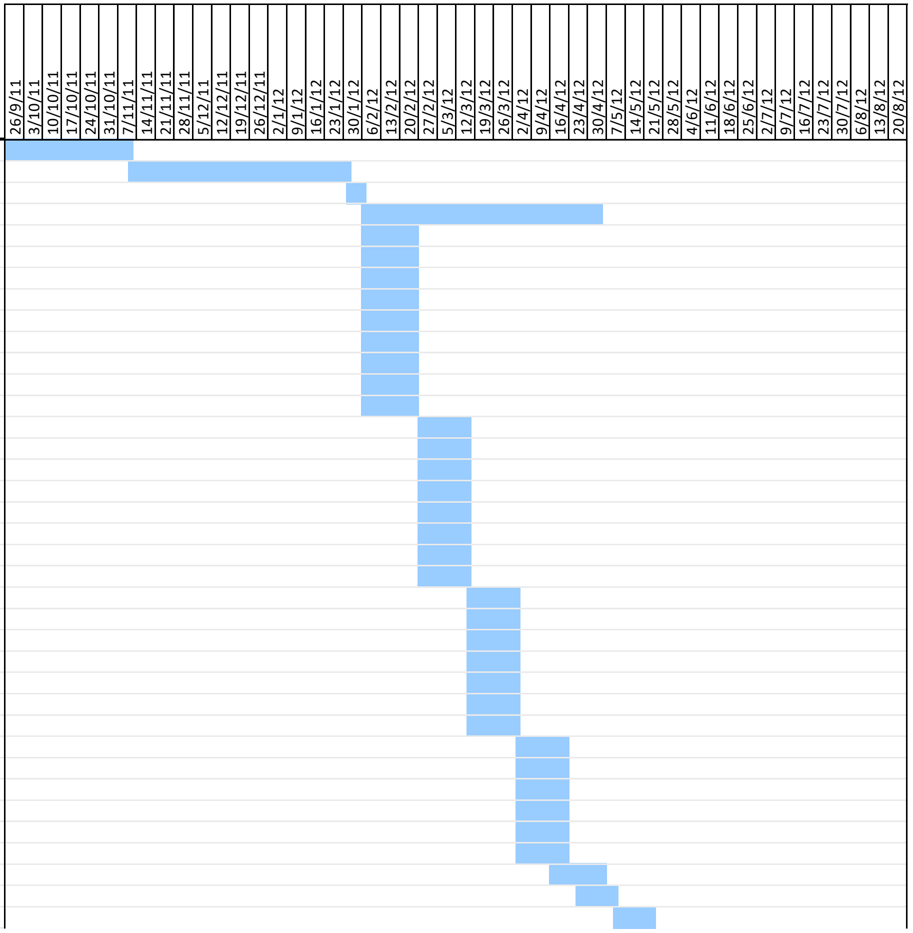


Figure 3.1 Gantt chart (Visible-Goal, 2008)

Financial Budget

There is no set budget for the project as it is difficult to calculate the cost of acquiring graphics and sound resources at this stage of the development.

The main development costs in this type of project would normally be hiring software developers, graphics artist and musicians to write the code and create graphics and sound resources.

The implementation of the project will be created using the Java development kit with the Eclipse integrated development environment. The development software is free and available as open source software under the GNU license agreement. This means the development costs will be relatively low and the game can be released as free software with no cost to the client.

If additional graphics and sound resources are required or the game needs to be thoroughly tested money has been allocated as shown in the financial budget below:

Name	**Description**	**Estimated Cost**
Development Costs	Development Software (SDK's) Programmers	£0 £0
Graphic / Audio Resources	Artists Musicians	£0 – up to £200 £0 – up to £200
Operational Costs	Game Testers Website Hosting Provider ESRB / PEGI Rating	£0 – up to £100 £0 – up to £50 £0- £250
	Total	£0 – up to £800

Figure 4.1 Project budget including development and operational costs

Project Organisation and Stakeholders

I will be the main stakeholder as the programmer and will communicate with other stakeholders who have an interest in the project. The project supervisor is Darrel Greenhill who is the module leader for Games Programming at Kingston University.

I will contact with him though emails and meetings where we can discuss the progress of the project. There will also be a second marker from Kingston University who oversees the final report and implementation to ensure a consistent and fair marking scheme. The end users or clients will help decide if the finished game is enjoyable to play and provides entertainment value and lasting appeal. The content providers are considered stakeholders and include artists, musicians and level designers who provide resources distributed with the finished game.

Stakeholders Diagram

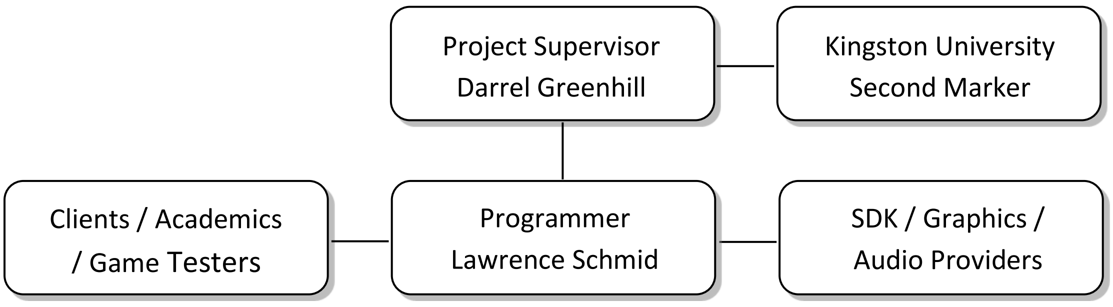
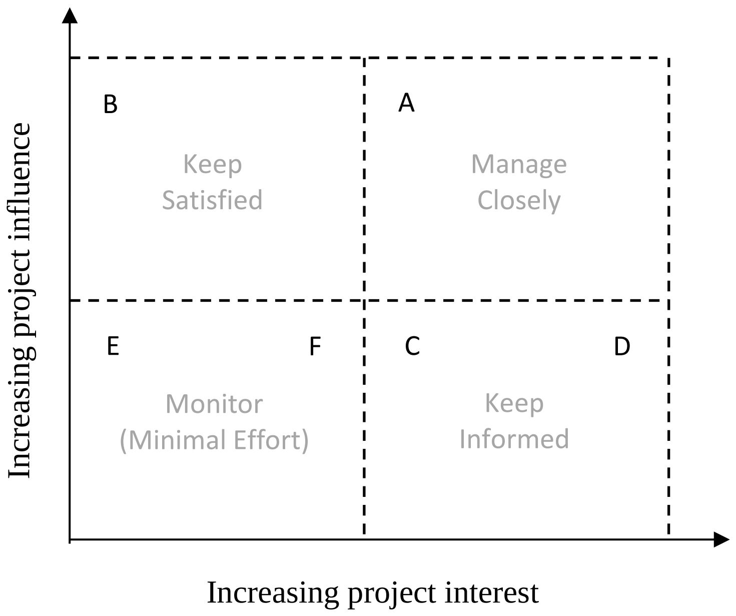


Figure 5.1 These are the stakeholders with an interest in the project.

Stakeholders Influence

The following diagram shows the influence of each stakeholder on the project.



A. Project Supervisor (Darrel Greenhill)
B. SDK Providers (Java / Google)
C. Artists / Musicians
D. Programmer (Lawrence Schmid)
E. End Users
F. Academic staff / Game testers

Figure 5.2 Power/Interest Grid for Stakeholder Prioritization (Thompson, 2011)

SWOT Analysis

Identifying the strengths, weaknesses, opportunities and threats associated with a project is a useful way of building on strengths to make the project a success. It can help identify potential obstacles to success, as well as flaws in the plan. (Knowledge, 2011)

Strengths:

- Action packed game with lasting entertainment value
- Level Editor providing long-term value
- Designed to be playable on mobile computing devices

Weaknesses:

- Difficult to find copyright free graphics and sound resources
- Lack of support for Java SE on mobile device hardware

Opportunities:

- The finished game could be made available to download for free
- The game could be sold online Android Market / PSN / Apple App store
- The source code could be made open source to help other students

Threats:

- Similar games with open source code which are already available
- Other types of game with advanced 3D graphics and improved game play
- Commercial games companies with teams of developers and more money

SWOT Analysis Table

SWOT identifies the Strengths, Opportunities, Weaknesses and Threats of the project.

Strengths	**Opportunities**
Action game with lasting appeal Continuous testing and evaluation Runs on Mobile/ Windows / Linux Dynamic iterative development	Sell game on Google Play Use advertising to generate revenue Open source code could be made available help other students
Weaknesses	**Threats**
Difficult to find free resource Development time constraints Lack of people willing to buy game SDK license restrictions	Performance issues / late delivery Software errors and bugs SDK documentation issues Commercial games companies

Figure 6.1 SWOT Analysis Table

MoSCoW Prioritisation

The following game play features have been identified in the analysis and design and must, should, could or would be completed in the project implementation.

Must:

- 2D side scrolling platform game with more than one level
- User controllable player with animation
- Collectable items which increase score and player abilities
- Enemies with collision detection

Should:

- Sound effects and music
- Scrolling parallax backgrounds

Could:

- Level editor

- Bonus levels
- Load / save game features

Would:

- Storyline / introduction
- Options screen

MoSCoW Overview Table

The requirements will be prioritised and classified using MoSCoW analysis. Each function will be assigned a letter which determines its priority.

MoSCoW stands for Must, Should, Could and Would and is used to determine which requirements must be implemented first and which must come later or will not be implemented at all. (Haughey, 2011)

Must Have	**Should Have**	**Could Have**	**Would Have**
User Controllable Character	User Interface	Sound Effects and Music	Animated Intro and Ending Sequences
Enemies	Collectable Items	Load / Save	Multiplayer
Collision Detection	Level Themes	Open Source	Response Time
Level Map	Level Endings	Game Options	
Level Editor	Backup	Advertising	
Accessibility	Documentation		
Legal Compliance	Extensibility		
Usability	Maintainability		
Score System	Portability		
Time Limit	Price		
Performance	Reliability		

Figure 7.1 MoSCoW Overview Table

Report Weightings

The project should be marked with the following weightings which reflect the amount of work put into each stage of the project lifecycle. The analysis and design should be worth 30 marks each and the implementation worth 60 marks as it will be more time consuming and will contain the documentation of the source code.

Risk Register and Contingency Plan

Risk management identifies potential risks or problems which could be encountered while completing the tasks and activities in the project lifecycle. The risks are labelled with id, name and descriptions. The mitigation explains how the problem could be avoided and a contingency plan describes the action which can be taken to minimize the effect. The probability and impact of the risk are measured on a scale of 1-3 where a higher number indicates a greater probability or risk. (Patel, 2007, p. 6)

Appendix

Screenshots of Platform Games

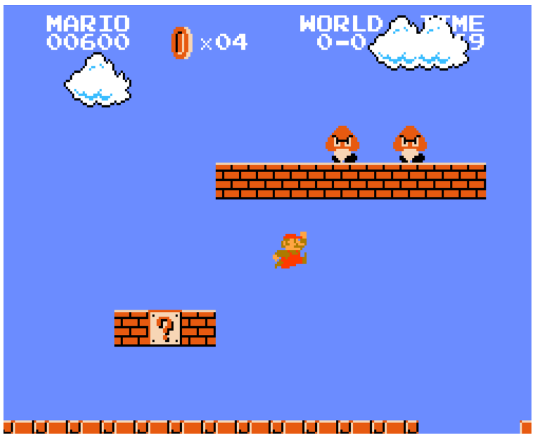


Figure 1.1 Super Mario Bros. (NES, 1983)
[http://1.bp.blogspot.com/-Ojd4iijuBqM/TmWESQwdKMI/AAAAAAAABzE/KbrvY1r9DUE/s1600/Super-Mario-Brothers.gif]

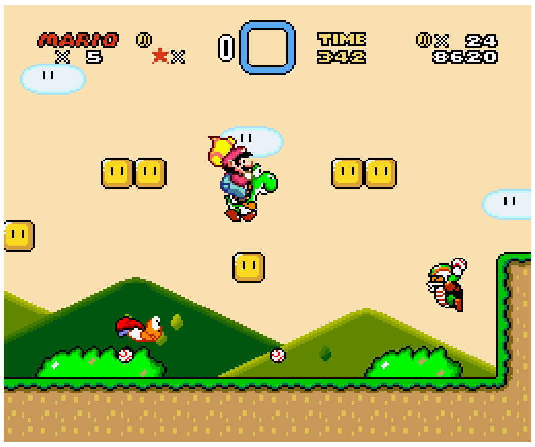


Figure 1.2 Super Mario World (SNES, 1990)
[http://www.mariowiki.com/images/1/1a/Donut_Plains_1.PNG]

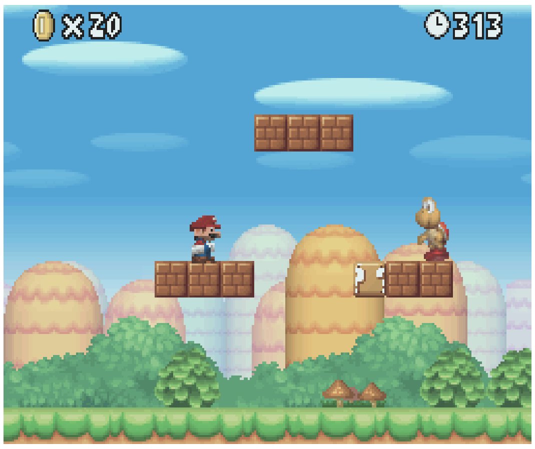


Figure 1.3 New Super Mario Bros. (Nintendo DS, 2006)
[Screenshot taken with No$GBA Emulator]

KINGSTON UNIVERSITY

Faculty of Science, Engineering and Computing

School of Computing and Information Systems

CI3330 – Individual Project

Dragon Island:
Java 2D Side Scrolling Platform Game

Lawrence Schmid

Issue 1: 19th April 2012

Table of Contents

Introduction

This report will document the design, implementation, testing and evaluation of Dragon Island a 2D side scrolling platform game project developed using the Java programming language. The finished game will resemble the Super Mario Bros. games by Nintendo. The objective of the project is to provide an exciting action game which is enjoyable to play. In a 2D side scrolling platform game the player walks, runs and jumps to and from platforms. The main objective of the game is to try and set a high score by getting to the end of the levels before the time runs out while avoiding enemies and collecting power-ups and coins.

The Java SE programming language will be used to develop the game it is an object orientated programming language which is open source and runs on multiple hardware platforms. There are many code examples available on the internet and in books which demonstrate using Java to display hardware accelerated graphics and play sound and music. The Java programming language can also be used to develop games and applications for Android and this means it will be possible to port the game to mobile devices without having to make too many changes to the code. The Java language has the advantage of web applet technology which means the finished game can be embedded on a webpage and made playable on the internet. If there is sufficient time available a C++ version of the game will be developed for the GP2X console using the SDL (Simple DirectMedia Layer) library.

The game will feature a level editor which allows the user to create their own levels using graphics which can be edited. The level editor will support multiple background graphics and tilesets used to display terrain and blocks and scenery. It will allow the player to specify options stored in the level header which can be used to specify additional level information such as start, bonus and end coordinates.

The graphics and sound resources for the game will be obtained from websites which provide images and sounds available under licensing schemes such as Creative Commons which permit their use for free and commercial projects.

The target audience of the game will be anyone who enjoys playing computer games. The game features cartoon style animated graphics which should appeal to children and adults. The level design will ensure the levels are challenging but not impossible to complete. There should be enough levels and game content to provide the player with an enjoyable game playing experience which they can return to over time.

Computer games have made a lot of progress from the early days of simple graphics and basic sound effects and this ever evolving progress of software and hardware makes it a challenge to develop games which run on multiple hardware platforms. To compete with other free and commercial games the finished project must provide an enjoyable game playing experience and try to offer something different to other games in the same genre.

This document has been divided into four main sections which contain the analysis, design, implementation, testing and evaluation of the project. The report weightings are 30 marks each for the analysis and design and 60 marks for the implementation. The analysis and design introduced in the project proposal has been expanded on to provide a complete introduction to the project.

This analysis section of this document will give an in-depth look at the approach, planning and management of the project. It will include market research and analysis of risks, benefits, stakeholders, functional and non-functional requirements.

The design section will examine the level design and conceptual diagrams produced during the software development lifecycle. It will include use case, class, sequence, activity, package diagrams, prototypes of the title screen, menu system and a discussion of the level design and game design.

The implementation section will describe the process of creating the game and will be an in-depth look at the actual source code with information on how the sound and graphics libraries are implemented in each version of the game. Techniques such as displaying parallax scrolling backgrounds, player and enemy sprites, level maps and collision detection will be discussed in detail along with code examples.

The evaluation section will contain the results of the testing compared with the requirements of the project and a review of the gameplay and performance. The success of the project depends on creating a game with meaningful play which provides entertainment value and an enjoyable experience for the end user. The overall success of the project will be evaluated by game testers reactions and the number of downloads.

State-of-the-Art-Review

Introduction

This state-of-the-art-review will examine the key topics and areas of interest which will be applicable to the implementation of the 2D side scrolling game project. Platforms games have been a popular genre throughout the history of computer games and have sold millions of copies worldwide. There are many concepts relating to platform games which may not be immediately obvious. The automatic generation of levels using procedural level design techniques can be optimized for player experience using rhythm based patterns and by measuring the player's anxiety. Platform games are often used in machine learning and A.I. research with autonomous agents being developed to complete the game in an optimal manner. Recent research has shown there are many more female social game players than males, and that female are often portrayed in a stereotypical manner which should be avoided in the implementation of this project. The narrative between the player and other characters is important as it contributes to the overall storyline of the game and will ensure the player does not become bored and lose interest. Scientific studies have shown that computer game players often exhibit improved behavioural, visual attention and motor skills. Experiments could be conducted to test player's reactions and anxiety levels and the results then used to procedurally generate optimal levels designs.

2D Platform Games

The aim of this project is to create a playable 2D side scrolling platform game similar to the Super Mario Bros. games by Nintendo. In this type of game the level is viewed from a side profile perspective and the player controls a character that can move, and run and jump to and from platforms, collecting coins and power-ups and avoiding obstacles to reach an end point within a set time limit while trying to set a high score.

This type of game has existed since the 1980's and has introduced some of the most widely recognizable computer game characters found in games today. The arcade game Donkey Kong is one of earliest examples of a 2D platform game. It was released by Nintendo in 1981 before the Super Mario Bros. series and does not have the side scrolling view port found in later games. In Donkey Kong the player must avid obstacles, climb ladders and jump across gaps to reach the top of the screen before the time limit runs out.

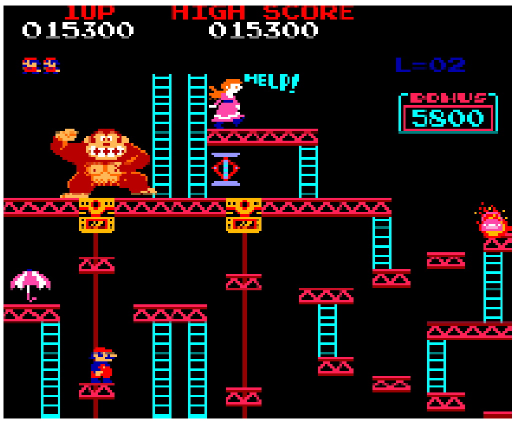


Figure 1.1 Donkey Kong (1981)

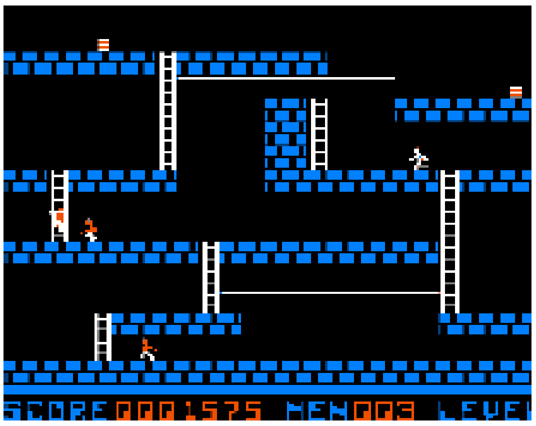


Figure 1.2 Lode Runner (1983)

Another early platform game is Lode Runner it is an interesting example as it was one of the first games to use a side scrolling view port allowing the player to navigate a larger level area. This game also included a level editor allowing the user to create their own levels which is a functional requirement of this project.

In 1985 Nintendo released Super Mario Bros. on the Nintendo Entertainment System which sold over 40 million copies worldwide and has helped define many of the characteristics of the 2D side scrolling platform game. The player controls the character around a level and can walk, run and jump to and from platforms avoiding obstacles and collecting coins and power up items. The game is separated into worlds containing levels with varied graphics, enemies and level backgrounds. The aim of each level in the game is to reach an end point which is usually a flag pole and castle or warp pipe in addition to this there are bonus levels and shortcuts through the game which can be accessed by entering a warp pipe or by climbing a vine. If the player reaches the end of a world they face a boss that is more difficult to defeat than normal enemies.

The success of Super Mario Bros. series has led to many other games being developed with similar game play mechanics which fall under the category 2D side scrolling platform game such as Sonic the Hedgehog released by Sega in 1991 in which the player controls a hedgehog that can run and jump around levels at high speed defeating enemies and collecting power up items. Advancements in computer hardware during the 1990’s meant developers could create games with improved graphics, sound and level content. The graphics and sound resources which will be used in the implementation of this project will be similar to the quality of these second generation platform games.

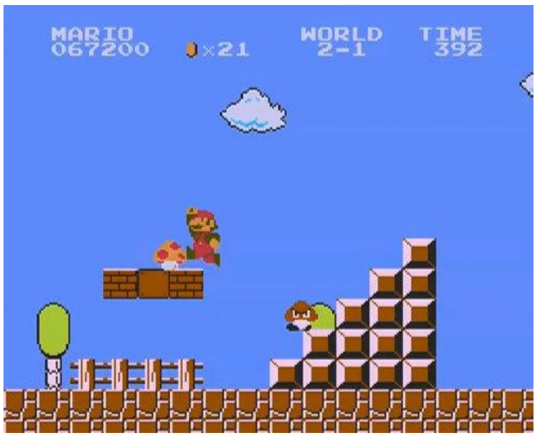


Figure 1.3 Super Mario Bros. (1985)

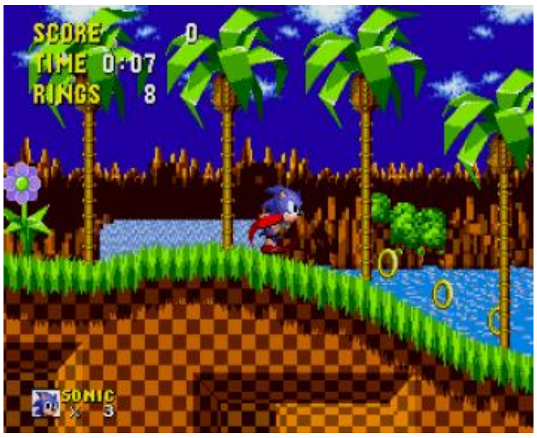


Figure 1.4 Sonic 1 (1991)

In 2006 the idea of combining 2D and 3D graphics was demonstrated in the game New Super Mario Bros for the Nintendo DS and Wii. More recently in 2011 Super Mario 3D Land for the Nintendo 3DS made use of a full 3D environment and was described by Mario creator Shigeru Miyamoto as a “3D Mario that plays as a 2D Mario game”.

Automatic Level Generation

Creating levels for computer games is a time consuming, repetitive and expensive task which results in static levels that have to be edited manually. Since the 1980’s games have used procedural level generation to automatically generate level content. Rogue is a roleplaying game with ASCII graphics. One of its main features was the ability to randomly generate an unlimited amount of levels. Other commercial games including Diablo II and Civilization feature procedural level generation to some extent. Despite this there has never been to this date a commercially available platform game that uses automatic level generation techniques. (Mateas, 2006, p. 1)

Infinite Mario Bros. is a free public domain open source Java clone of the original Nintendo platformer Super Mario Bros. It was developed in 2006 by Markus Perrson who also developed the successful Minecraft game. Automatic level generation is used to generate a new level each time the game is started. This makes it popular choice as a test bed platform for research into automatic level generation. MineCraft involves building your own levels and using blocks of terrain in a 3D environment. The level editor included in this project will aim to offer similar functionality allowing the user to design their own levels using 2D tiles. (Noor Shaker, 2010, p. 2)

Figure 2.1 Infinite Mario Bros.

Figure 2.2 MineCraft

ORE Algorithm (Occupancy-regulated extension)

Procedural level generation algorithms can be used as a substitute for human game designer saving time, money and resources. Processes which emulate a human designer are implemented using algorithms which by their definition produce playable levels. Extra processes are used to add features such as enemies and collectable items to the level. Occupancy-regulated extension (ORE) is a technique which is used to generate levels using pre-authored chunks of levels as the resources. The ORE algorithm uses positions that the player might occupy during play to anchor each chunk. This process limits the amount of levels that can be created by rejecting poorly designed levels which are unplayable.

The ORE algorithm consists of three main steps:

1) First a context is selected which is used to expand the current context.
2) Then a chunk of level is selected from the library which matches the context.
3) Finally the chunk of level is integrated with the current context.

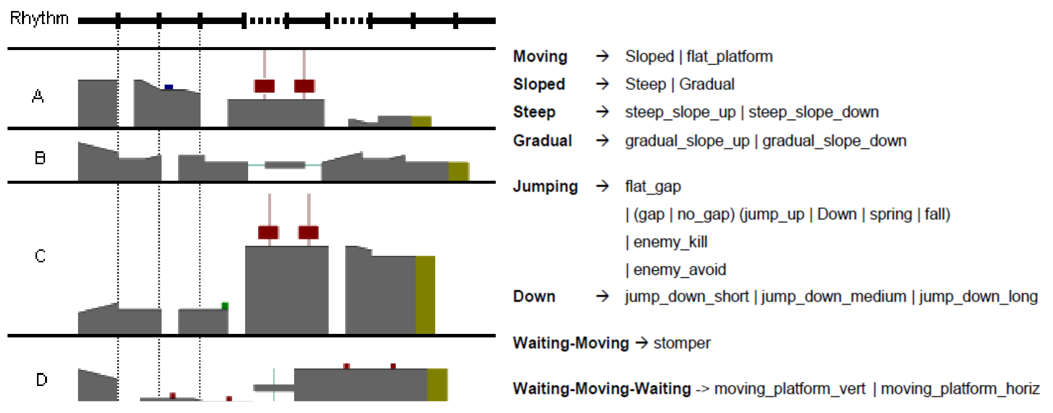
The pre-authored chunk library for the ORE algorithm should consist of at least 40 different random level sections of the size 10x10 tiles.

An advantage of the ORE techniques is that it combines the human designed templates with the automatically generated level content. It would be possible to provide a "mixed-initiative" system where a human creates part of the level and asks the ORE algorithm to complete the design. (Mawhorter & Mateas, 2010, pp. 1, 2 and 6)

The use of automatic level generation is not a functional requirement of this project but it would be interesting to do further research in to this area. This project will feature a tile based level editor that could be used to design the templates for an ORE algorithm. A procedural level generation algorithm could be developed which uses the occupancy-regulated extension (ORE) technique. The procedural level generator would use an ORE algorithm which splits the level into chunks and designs each chunk using a separate simpler algorithm. The algorithm would then iteratively generate sections of level to a random length based on five simple templates: 'straight', 'hills', 'valleys', 'jumps' and 'warp-pipes' with some random variation.

Optimizing Level Design for Player Experience

One of the key underlying features of 2D platform game design is idea of rhythm and the timing. It is therefore possible to automatically generate level geometry from a rhythm pattern. Rhythms groups are short non-overlapping chunks of level which incorporate a sense of rhythm and timing. The rhythm groups are constructed using sets of verb definitions for the players movements for example move or jump with start and end times. The geometry generator then takes the rhythm groups and creates the potential interpretation for them. (Gillian Smith, 2009)



```
Moving               → Sloped | flat_platform
Sloped               → Steep | Gradual
Steep                → steep_slope_up | steep_slope_down
Gradual              → gradual_slope_up | gradual_slope_down

Jumping              → flat_gap
                       | (gap | no_gap) (jump_up | Down | spring | fall)
                       | enemy_kill
                       | enemy_avoid
Down                 → jump_down_short | jump_down_medium | jump_down_long

Waiting-Moving → stomper

Waiting-Moving-Waiting -> moving_platform_vert | moving_platform_horiz
```

Figure 3.1 Geometry interpretations and generation grammar of a rhythm

The player's anxiety can be measured over time while completing a level and can be analysed with the described challenge metric to produce a characteristic anxiety curve. The resulting curve will show an increasing slope where for example there is a series of carefully timed jumps over a short period of time and decrease when there is less of a challenge. Rhythm groups can be identified in the anxiety curves, and a characteristic dramatic arc can be seen in all the levels. (Nathan Sorenson, 2010, pp. 1, 5, 8 and 9)

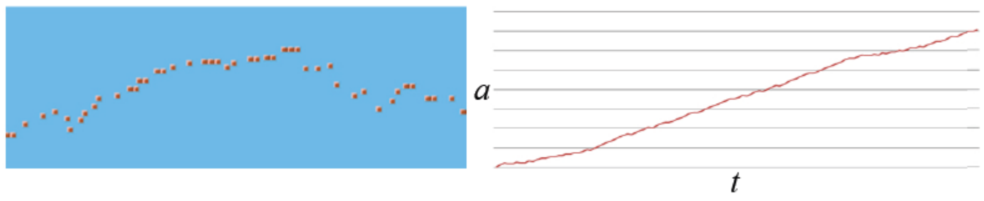


Figure 3.2 Generated level map data and its corresponding anxiety curve.

The player's anxiety can be measured over a period of time to generate optimal level designs which are not too easy and not too difficult to complete. The problem with this approach is that it would be a time consuming task as it would have to be repeated a number of times to generate enough data to create the anxiety curves. Although it is an interesting concept there is not enough evidence of it being successfully implemented to use it as a technique of creating levels for this game.

The use of rhythm based methods to create levels for a 2D platform game is a complicated and difficult process outside of the scope of this project. If there was enough time available it would be interesting to try and capture a rhythm which is enjoyable and non-repetitive from a group of game testers. It could potentially provide a level design with a strong sense of pacing flow. (Gillian Smith, 2009)

The rhythm based method could be combined with anxiety curve data, human designed and procedurally generated level content in a mixed initiative system to produce the most interesting and varied level designs.

Game Controllers

Computer games have been traditionally controlled using a gamepad or joystick to control what happens on the screen. Recently computer game controllers have evolved with the introduction of more immersive technologies such as motion sensing and touchscreens.

The Sony PlayStation Eye and Microsoft Xbox Kinect cameras make use of computer vision algorithms to recognize gestures and control the game. The Xbox Kinect system does not require a controller and instead relies on depth imaging and an infrared laser projector to provide three dimensional tracking. It is also capable of tracking user body parts such as hands and heads by implementing body model and gesture analysis algorithms. (Cummings, 2007, p. 1)

Motion control interfaces are another recent development in controller design. The Nintendo Wii combines accelerometer and infrared positioning sensors to provide a gesture-based control system. The PlayStation Move system relies on a controller with a LED orb which changes colour dynamically to make it recognizable by the PlayStation Eye camera. The PlayStation Move.Me is an application which resides on the PlayStation 3 console and communicates with the PC using software development libraries for C and C#. It enables the PlayStation 3 console, Eye camera and Move controller to be used as a motion control input device to supply sensor data to the PC application. The PlayStation Eye camera allows the Move controller to function in three dimensions by determining the distance of the controller from the camera via the size of the image corresponding to the orb. The PlayStation DualShock 3 controller features an accelerometer to track the player's movements and haptic technology to provide vibration feedback based on events programmed into the game that the user is playing. (Matt Benatan, 2011, pp. 213-214)

Haptic or force feedback technology is technology using interfaces with computers to produce the sense of touch by applying different forces. (Boduch, 2010, p. 1) Flight simulators often use haptic technology to generate forces and vibrations that simulate a realistic sense of touch and feel through the joystick allowing the user to experience what it is like to fly a real plane. (Kelly, 2011)

Touch screen technology is becoming increasingly popular and by 2012, 40 percent of mobile phones will likely utilize touch-screen technology. There are two types of touchscreen predominantly used in phones - resistive and capacitive. Resistive touch screens work by sensing pressure using a top layers which flexes under pressure and "effectively completes a circuit, telling the phone which part of the screen is being pressed". Capacitive technology uses electrodes to sense the conductive properties of object. Most modern mobile phones such as the iPhone use capacitive technology as it is more responsive and doesn't need much contact. (Koprowski, 2006)

For the implementation of this project camera based motion detection would not be suitable as a platform game requires precision and timing to jump between platforms and destroy enemies. The player would have to be constantly moving around and making different gestures and it would probably be easier to use a standard gamepad to control the game. This project will make use of touchscreen, keyboard, phone buttons and console gamepad technology. It should be possible to include haptic force feedback technology using the mobile phones vibration feature. The phone could vibrate during the intro sequence when the large tank is moving towards the player, or to alert the player when they are hit by an enemy.

The Android smartphones which the game will be developed for include an accelerometer which can detect the direction the phone is being moved in. It would be possible to tilt the phone left or right to move the player and forward to jump between platforms. This has been demonstrated in a game called MarioFit which uses an accelerometer to monitor the player's movement such as walking, turning or jumping and uses this to control the actions of the game. It was designed as an application to help players keep fit and use physical activity to combat obesity.

Autonomous Agents

Computer games can be used as test beds for research in Artificial Intelligence and Machine learning. A programmer called Markus Alexej Persson also known as 'Notch' invented a game called Infinite Super Mario Bros. This game includes an autonomous agent feature which automates playing the game using a user agent to reason and learn at several levels; from modelling sensory-motor primitives to path-planning and devising strategies to deal with various components of the environment.

The Soar (State, Operator and Result) architecture is a symbolic cognitive architecture that has been extensively used in designing intelligent agents across various domains. The main aim of the Soar architecture is develop systems that exhibit intelligent cognitive behaviour.

Soar aims to capture the behaviour of humans and animals using "trial and error" to develop strategies to deal with the environment through experimentation and exploration with a "positive / negative reward for reinforcement". (Shiwali Mohan, 2009, pp. 2-5, 7-8)

Reinforcement Learning (RL) is an approach which aims to solve a problem without specifying all of the details incorporated in the implementation. Hierarchical Reinforced Learning (HRL) is an approach to upscale RL to solve more complex problems. HRL involves splitting the task into a series of simpler subtasks that can be solved independently. Infinite Super Mario Bros includes a standard RL-Glue interface which allows the game to connect to reinforcement learning agents.

The RL-Glue interface allows the user agent to provide and set information in the current visual scene through arrays. The games drawing area is stored in a two-dimensional array of tiles which the user agent has access to and can respond to, for example collecting coins or power-ups by altering the array values.

The RL function takes into account the user agent reward which can be either negative or positive for completing or failing part of the level. The q value represents the reward associated with a particular state. Operator abstraction defines two types of operator. KLO (Key Level Operators) are the action that can be performed using a keyboard or joystick such as moving or jumping. FLO (Functional Level Operators) represent a collection of keystroke level operators that when performed in succession, perform a specific task. (Brian Tanner, 2009)

The complexity of Mario games requires abstractions to facilitate the learning process a user agent executes the operator steps which affect the q state variable representing the player’s reward.

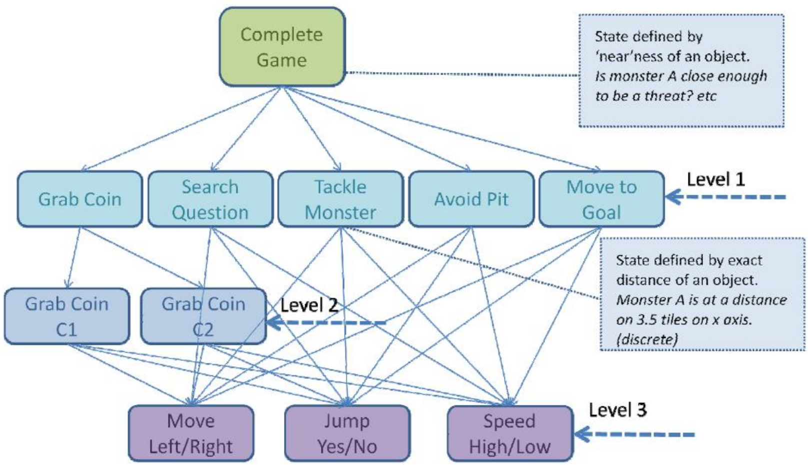

Complete Game
State defined by 'near'ness of an object. Is monster A close enough to be a threat? etc
Grab Coin
Search Question
Tackle Monster
Avoid Pit
Move to Goal
Level 1
State defined by exact distance of an object. Monster A is at a distance on 3.5 tiles on x axis. (discrete)
Grab Coin C1
Grab Coin C2
Level 2
Move Left/Right
Jump Yes/No
Speed High/Low
Level 3

Figure 4.1 Operator Hierarchy in Infinite Mario Task (Shiwali Mohan, 2009, pp. 2-5, 7-8)

The idea of state abstraction moves from tile based descriptions to a view of objects and their relationship with the agent and each other. The relative distances between the user agent and the objects can be calculated and used to determine whether the object is close enough to initiate an action. The object detection algorithms can incorporate rules to avoid obstacles in the level such as if there is a pit ahead, then jump while moving right. (Shiwali Mohan, 2009, pp. 2-5, 7-8)

The use of autonomous agents is not a functional requirement of this project. If there is enough time available it would be interesting to do further research on developing an autonomous agent that could complete the level in an optimal manner. This could be developed into a feature of the game where the user has to race the computer player to the end of level. The algorithm would involve using the Soar architecture to design and intelligent agent that can reason and learn and devise strategies to jump between platforms. Reinforcement Learning (RL) would take into account amount of user agent reward for completing or failing part of the level. The algorithm should be implemented using and RL-Glue interface allowing the game to connect to RL agents to maintain portability. If an algorithm was successfully developed the game could be entered into Mario themed A.I. competitions and used as research made available to other students. (Competition, 2009, p. 1)

Portrayal of Gender in Computer Games

The computer game market remains a medium that is dominated by primarily male interests with male players in 2003 spending as much as 400% more time playing than female players. (Grimes, 2003, pp. 3,6)

The third person action adventure game Tomb Raider (1996) features a leading female character called Lara Croft who the player must guide series of massive real-time 3D environments. Tomb Raiders creator Toby Gard is quoted as saying that “Lara was designed to be a tough, self-reliant intelligent woman.” The Tomb Raider developers aimed to balance the traits that would make Croft an attractive role model for game playing girls and a sexually attractive figure for their core male market. (Justine Cassell, 1998)

The survival horror game Resident Evil (2002) features a choice of male or female playable characters. The female character called Jill Valentine shows a number of characteristics that put her at a disadvantage to the male characters around her, such as being easily harmed and equipped with limited resources. In certain parts of the game she is placed in situations where she is being rescued by the male character. This is one example of a game which could provoke debate on sexism as the female character is often portrayed as weaker in comparison to her male counterparts. (Grimes, 2003, pp. 3,6)

In a study by the NPD Group sales data for a 133 games during the calendar year 2005-2006 were examined by game testers. It was found that males were over-represented and females were under-represented and that when females did appear they were more likely to play a supporting role.

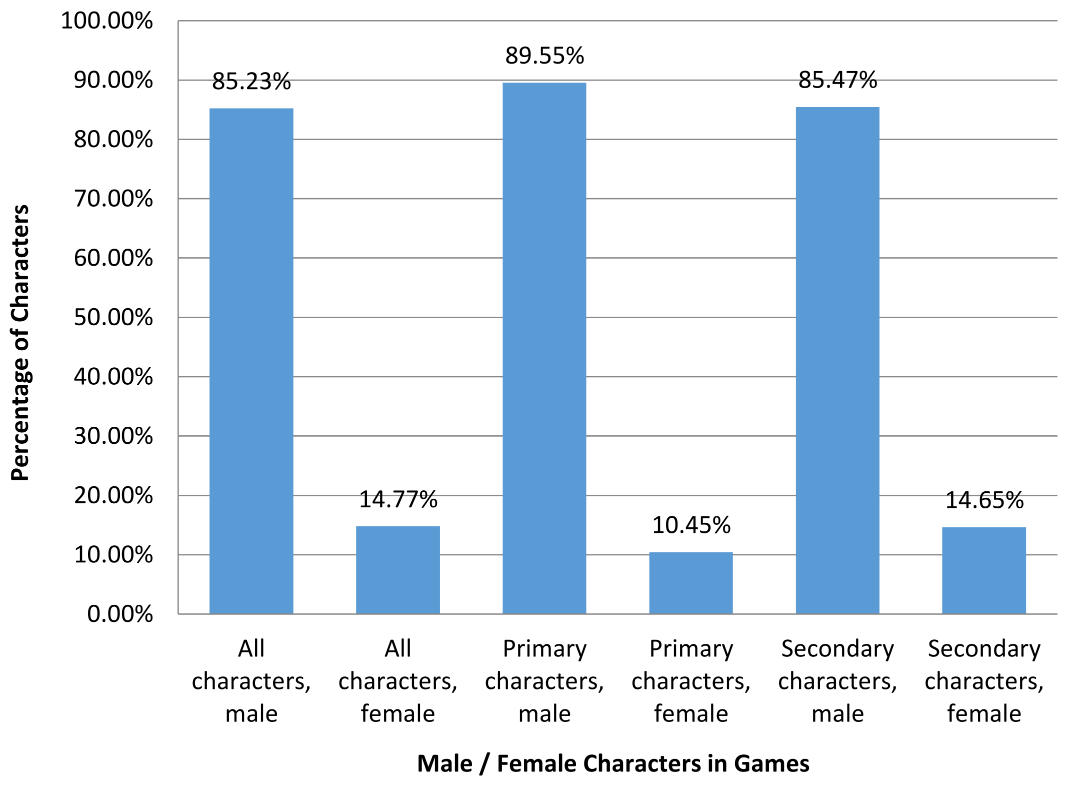


Figure 5.1 Percentage of male and female characters in games (Dmitri Williams, 2009, pp. 822 and 824-825)

In another study two hundred and twenty five computer game covers obtained from online retail sites were examined for portrayals of men and women. It was reported that male characters were featured four times more frequently than female characters and were given significantly more game relevant action. (Melinda Burgess, 2007)

There have already been some successful commercial games which feature female characters including Ms Pac-Man, The Great Giana Sisters, Metroid and Tomb Raider. The Great Giana Sisters is a clone of the Super Mario Bros. games similar to this project which has recently been rereleased in 2011 on the Nintendo DS. It proves there is currently a market on mobile devices for platform games with female characters. Surprisingly research conducted in 2010 shows that the average age of a social media game player is a 43 - year-old woman. In fact in the UK more women 58% compared with 42% of males play social games on Facebook such as Farmville, Mafia Wars and Happy Aquarium. (Ingram, 2010)

It would be a good idea to include a female character in this project to appeal to the growing market of female players. The game could be modified to load a second character and to let the player choose their character on the title screen. The majority of computer games do not include strong female playable character with some exceptions such as Tomb Raider. It would be important to avoid stereotypes such as the female character being portrayed as weak and helpless in comparison to her male counterpart. If a female was included in this project she must be a playable character with the similar actions and storyline to the male character.

Behavioural and Motor Skills

It is interesting to examine the effects of game playing on human behaviour and movement or motor skills. There are many examples of perceptual learning in specific tasks which results in increased performance as a result practice and experience. (Gibson, 1969, p. 3)

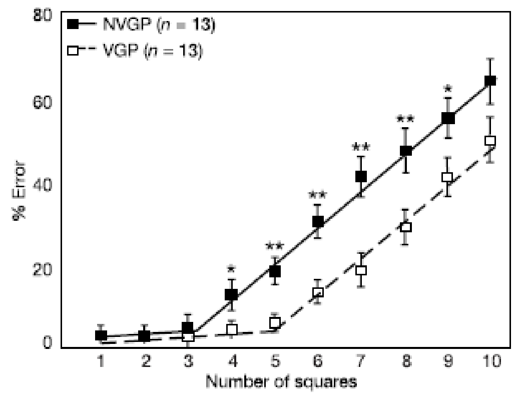


Figure 6.1 The percentage of error at detecting the number of squares briefly displayed on the screen in game players and non-game players. (C. Shawn Green, 2003)

In experiments it has been proved that the amount of time it takes to respond to two targets is significantly improved in computer game players.

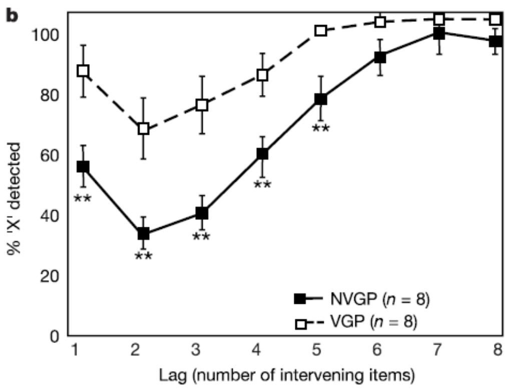


Figure 6.2 Graph showing the percentage of detected squares briefly flashed on the screen in game players and non-game players. (C. Shawn Green, 2003, pp. 534-546)

It has been proved that the visual attention of computer game players is better than non-computer game players. In one experiment the user is asked how many squares are displayed on the screen. The results show that computer game players are significantly more accurate with a detection rate of 78%. Computer games often push the limits of more than three visual attention skills simultaneously by juggling a number of varied tasks. This can include detecting enemies and tracking existing enemies amongst other tasks.

It would be possible to setup a series of experiments to test the range of visual skills of video game players and non-video game player. This might include timing the player's reactions to avoiding enemies and jumping between platforms. The data collected from the experiments could be used with procedural level generation algorithms, rhythm based systems and anxiety curve data to create level designs based on player experience which are optimized for playability.

Game Story and Narrative

Computer games can be compared in some ways to traditional stories found in books, comics and feature films. The relationship between computer games and stories is a subject which is open to debate between the uses of two different approaches known as ludology and narratology. The narratology approach implies computer games should be viewed as a variation of the narrative story form whereas ludology suggests games are a completely different type of narrative. (Løvlie, 2005, p. 1)

Stories can be described as a series of facts in time sequenced order that suggest a cause and effect relationship. Computer games on the other hand are interactive and present facts in a branching tree of sequences allowing the player to create their own story driven by their actions in the game. (Crawford, 2000)

Early arcade games such as Pac-Man and Space Invaders did not introduce much of a story element and did not have a set end point instead relying on the fact the player would keep inserting coins to continue. Console games began using a narrative element as an incentive to persuade people to buy the next game. (Kücklich, 2011, p. 1)

This project will focus more the ludologists view and the gameplay mechanics rather than the story telling aspects. There will be a short introduction sequence before the game begins. It will introduce the main character and the last boss and the two characters will talk to each other. This aim of this introduction sequence will be to introduce the player to the storyline and the characters they will encounter while playing the game. If there is enough time available it would be interesting to include shortcuts and bosses at the end of the level with different storylines, narrative and introductions.

Conclusion

The state-of-the-art-review has provided a valuable insight into areas of interest which are applicable to this project. The review of existing 2D games will help define and prioritize the functional and non-functional requirements. The research into game controller methods will be useful during the implementation of the Android version of the game as it will make use of touchscreen and haptic technology.

If there is enough time available the automatic level generation algorithms could be used to generate a large number of unique levels to compete with other commercial games. The study of autonomous agents is an interesting topic for further research and could be used to develop a player with A.I. that can complete levels in an optimal manner and built into a competition mode to race the computer opponent to the end of the level.

Analysis

Introduction

The analysis section of this report will explore the approach, development platforms and the methodologies used in the software development process. It will also discuss aspects of the games industry including market research and competition analysis.

The main objective of this project is to create a fun playable and challenging 2D side scrolling platform game similar to the Super Mario Bros. Games by Nintendo.

The target audience is younger people who play computer games. The difficulty of the game should be challenging and test the player's reactions and reflexes while jumping to and from platforms, avoiding enemies and collecting items.

Computer games have become increasingly popular on mobile devices such as the Nintendo DS, PSP and mobile phones. With the development of the mobile phone technology more and more people are playing computer games. Advanced 3D graphics have become popular but there are still many games developed using 2D graphics such as Angry Birds. Rovio Mobile the makers of this game recently announced that across all platforms (including Angry Birds Seasons and Rio) it has reached 500 million downloads.

New Super Mario Bros. is a 2D side scrolling platform game for the Nintendo DS, since its release in 2006 it has sold over 26 million copies worldwide. This project will aim to emulate the success of the New Super Mario Bros. game and will make use of similar graphics, sounds, enemies and gameplay mechanics.

The games industry spends enormous amounts of money conducting market research to determine people's opinions on what is a fun game playing experience. To help judge people's views on a new mobile phone game existing market research was examined and the results used to identify how the game could be improved.

Approach

The project planning and approach will follow the DSDM (Dynamic System Development Method) Atern Framework and will also incorporate features of the RAD (Rapid Application Development) methodology.

The DSDM approach to project management is an agile project management and delivery framework that aims to deliver the right solution at the right time.

The features provided by the DSDM Atern Framework enable developers to integrate prototyping and testing into the software development lifecycle. It focuses on the businesses needs and should be delivered on time, collaboratively, while never compromising on quality.

The DSDM Atern lifecycle diagram shows the project goes through a controlled start to a point where the understanding of the project is good enough to start building the solution iteratively and incrementally.

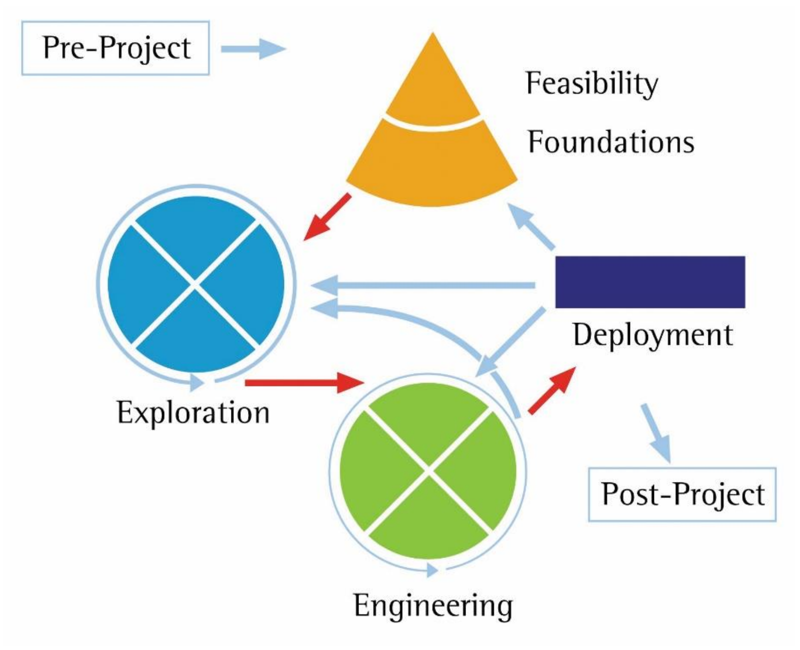


Figure 1.1 DSDM Atern Lifecycle Diagram (Richards, 2011, p. 3)

The first phase of the DSDM approach was to identify the prototype. Prototyping helps clients and developers gather an understanding of what the proposed product is, even if the requirements are uncertain.

The functional and non-functional requirements of a 2D side scrolling platform game were decided on and noted in the project proposal document. The functional requirements included a user controllable character, enemies, collision detection and collectable items. The requirements were then prioritized using MoSCoW analysis.

The second phase involved agreeing the prototyping plan with the project supervisor and setting time limits using a Gantt chart and Work Breakdown Structure diagram to prevent the project overrunning the schedule.

The third phase involved creating low and hi fi prototypes of the game. The low-fi paper prototypes consisted of a business prototype demonstrating the game play, and a usability prototype demonstrating the user interface for the game. The hi-fi prototype was a demo of the game developed using the Java programming language.

The fourth phase was to review the prototype with the project supervisor. He was satisfied that the project was viable and could be completed within the set time limit. The prototype review ensured that the project was on track and met the requirements.

The techniques which will be employed during the project lifecycle are listed in the project proposal document and include use case analysis, requirements gathering, MoSCoW prioritisation, risk registers, UML, OOP, testing and evaluation.

Platform Overview

Java2 Standard Edition (J2SE) is an object orientated programming language with a syntax derived from C++.

It is a widely used platform for developing games and applications and includes libraries and functions for displaying 2D and 3D graphics and playing sound effects and music.

The J2SE software bundle includes the JDK (Java Development Kit) and JRE (Java Runtime Environment) plus compilers and debugging tools for developing web applets and applications. (Oracle, 2011)

The Java source code is compiled to byte code which is run on a virtual machine. The JVM (Java Virtual Machine) is available on many operating systems including Windows, Linux and Mac.

The Java language includes a GC (Garbage Collector) which destroys objects which are no longer being referenced in the program.

Java Virtual Machine

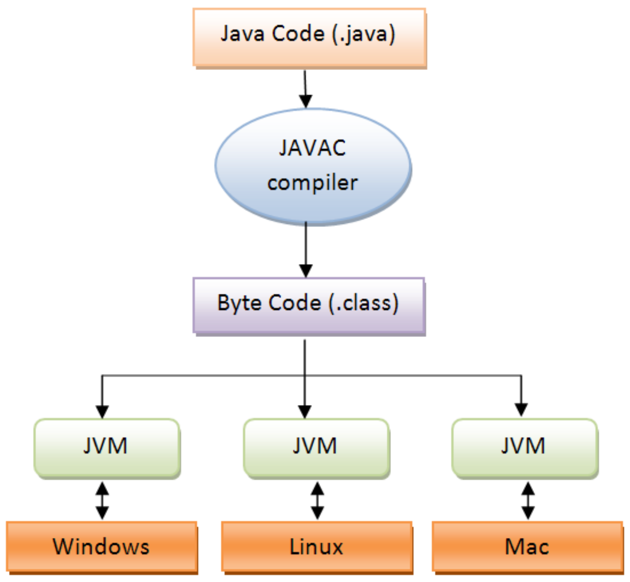


Figure 2.1 Java Virtual Machine Diagram (Patel, 2011)

Android is a Linux based open source software stack for mobile devices such as smartphones and tablet computers. It includes an operating system, software development kit and key applications. Google purchased start-up company Android Inc. and the Android software in 2005 and developed it in collaboration with the Open Handheld Alliance into a successful operating system. There is a large developer community for Android devices and in 2011 there were over 319,000 apps available for download on the Android app store Google Play.

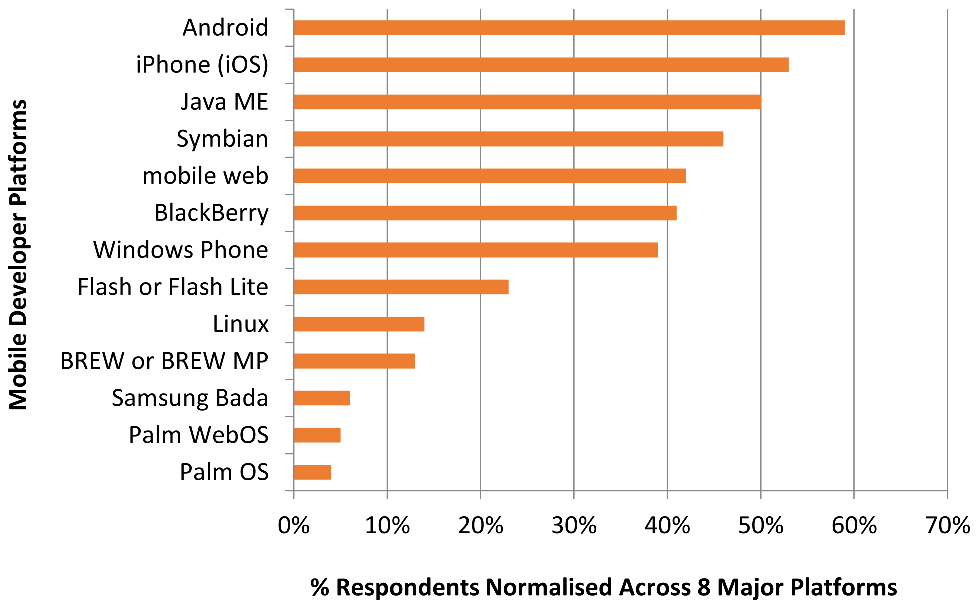


Figure 2.2 Platforms most used by mobile developers in 2010

The Android SDK (Software Development Kit) consists of a Java API with other libraries written in C++. It provides libraries to draw on a 2D canvas similar to J2SE and can be extended using the OpenGL ES framework to provide hardware accelerated 2D/3D graphics which will be implemented in this project. (Group, 2011)

The GP2X console created by GamePark Holdings is a mobile device which runs the Linux operating system. (Boyes, 2006) It has a C++ SDK which allows for cross-compiling for operating systems including Windows and Linux. The GP2X has an open source SDK games can be released without any specific licensing conditions. The hardware is capable of displaying 2D and 3D graphics using C++ with the OpenGL or SDL graphics libraries. SDL (Simple Direct Media Layer) is a cross platform open source multimedia library available for the GP2X and other hardware. It provides low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video frame buffer. (Layer, 2011)

The iPhone from Apple runs the iOS operating system with applications and games developed using the objective C language and additional libraries such as OpenGL ES for hardware accelerated graphics. Apps and games developed for this platform can be sold on the popular Apple iStore.

The PSP or (PlayStation Portable) is a closed source SDK and development platform with games and applications written in C++ with Visual Studio and the Pro DG tools that run on the Sony PSP hardware tool. The official PSP SDK does not create applications that will run on a production PSP console as the code needs to be electronically signed by Sony before being released on the PlayStation Network.

Computer Games Industry

Computer games have been around since the late 1970s with the introduction of the first home versions of Pong. They became increasingly popular in the 1980’s with the introduction of home video consoles like the Atari 2600, Intellivision, and Colecovision.

The computer game industry crashed in 1983, they vanished almost as quickly as they came. During the late 1980’s a new generation of consoles such as Nintendo NES and Sega Master System helped rejuvenate the market. (Feldman, 2005, p. 13)

It should be considered a constantly evolving market that demands ever more spectacular game graphics and more meaningful methods of interaction.

The computer game industry spends millions of pounds every year conducting market research to find out what types of games people like. While designing a computer game the developer should try to understand all of the formal, social, and cultural factors. It is important for example to understand how the culture at large perceives and regards games and how new audiences might be brought to your games. (Zimmerman, 2003, p. 23)

ESRB (Entertainment Software Rating Board)

The Entertainment Software Rating Board (ESRB) is a non-profit organization that assigns computer and video game content ratings, enforces industry-adopted advertising guidelines and helps ensure responsible online privacy practices.

It was established in 1994 in the United States following a trend of violent computer games such as Night Trap, Mortal Kombat, Lethal Enforcers, and Doom. The main objective of the ESRB is to provide age and content ratings which help consumers in particular parents to make informed decisions about the computer games they choose to purchase for their families. (Wikipedia, 2011)

The ESRB age ratings range from EC (Early Childhood) to AO (Adults Only) there are six different age ratings in total covering the various age and content groups.

The European equivalent of ESRB is PEGI it was introduced in April 2003 and has rated over 16,000 games as of August 2010. The PEGI system has five age categories ranging from 3 - suitable for everyone to 18 - suitable for age 18 and over. (Wikipedia, 2011)

If Dragon Island was submitted for review by the ESRB or PEGI it would probably receive an E or 3 rating with no additional category restrictions. This is because it is portrays minimal cartoon violence and most of the games in the Super Mario Bros. series similar to this game have already received this rating. The ESRB ratings system is voluntary but nowadays most games tend to receive a rating which is enforced by the shops which sell games.

The E and 3 for everyone categories saw nearly half of all game sales in 2010. If this project is kept within the guidelines for the E or 3 rating it is likely to receive a higher percentage of sales and will be played by more people.

The following graph shows sales for 2009 in the US for the four main ESRB categories where 67% of households play video games.

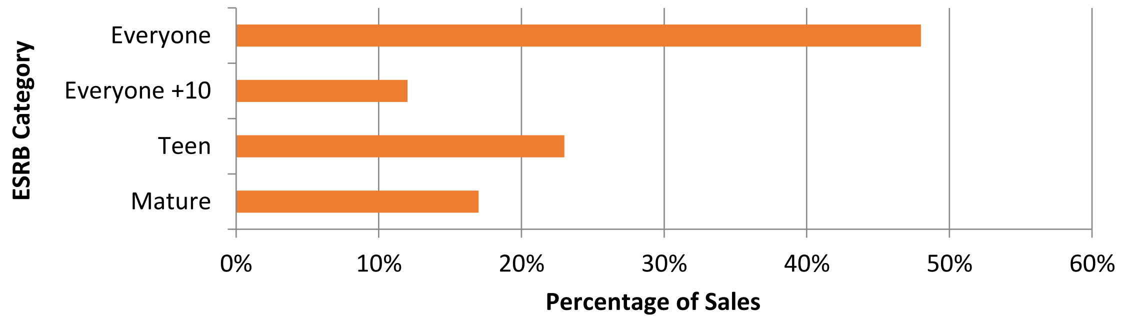


Figure 3.1 Computer and Video Game Sales by ESRB Rating (ESRB, 2009)

The computer game age rating system differs between countries presenting legal difficulties where a game may be reclassified in a different age group depending on the country it is released in. Computer games often have to be modified to provide localization for audiences of different cultural backgrounds. The challenges include language, graphics, music, hardware, software, legal and cultural issues. Computer games are an international industry and games companies depend on their ability to adapt to different cultures and languages.

Market Research

The Information Solutions Group conducted a mobile phone gaming web research survey for the PopCap Games Company in 2011. The objective of the research was to determine the popularity of mobile phone games. The audience was US and UK users that have played a mobile phone game in the last month.

There are currently three main categories of mobile phone type standard mobile phone, web-enabled smartphone and smartphone. The Pop Cap Games survey asked 2,425 people what type of phone they own and use. It found the in the UK 32% of people have a standard mobile phone, 31% have a web-enabled smartphone and 37% have a smartphone. It was interesting to note that smartphones have a larger market share in the UK [37%] than in the US [24%]. (Games, 2011)

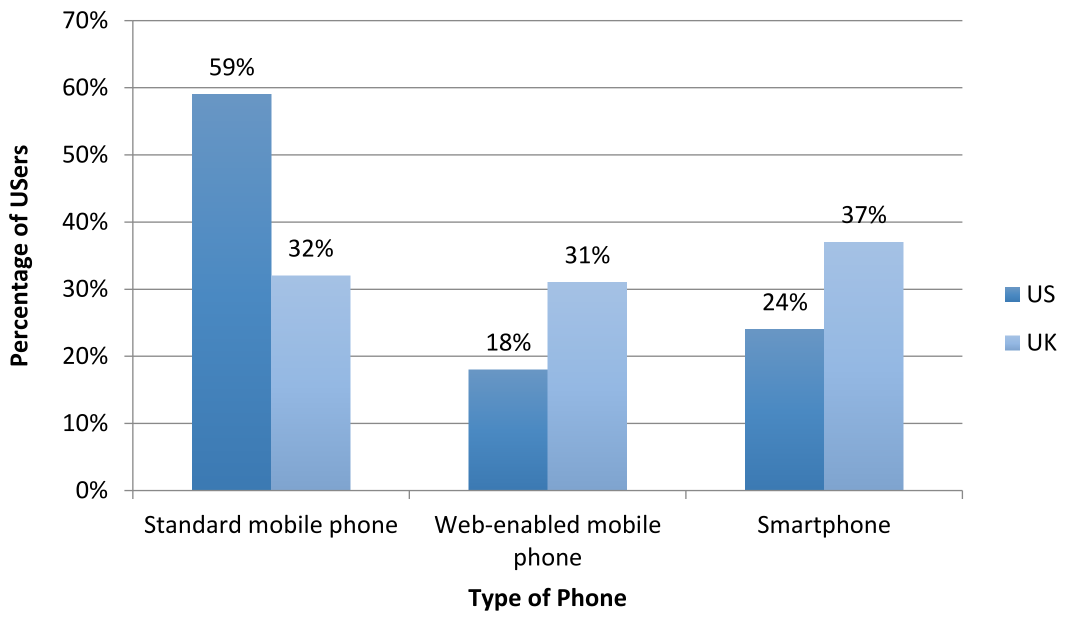


Figure 4.1 Mobile Phone Ownership. (Games, 2011)

In another survey conducted in 2011 by Business Insider over 2000 people were asked “what kind of smartphone do you use?" It found that the Android is the most popular platform with 51.4% compared with the iPhone at 33% and Blackberry at 8.3%. Other platforms such as Microsoft WP7 and Palm were less than 3% of the total ownership. (Goldman, 2011)

What kind of smartphone do you use?

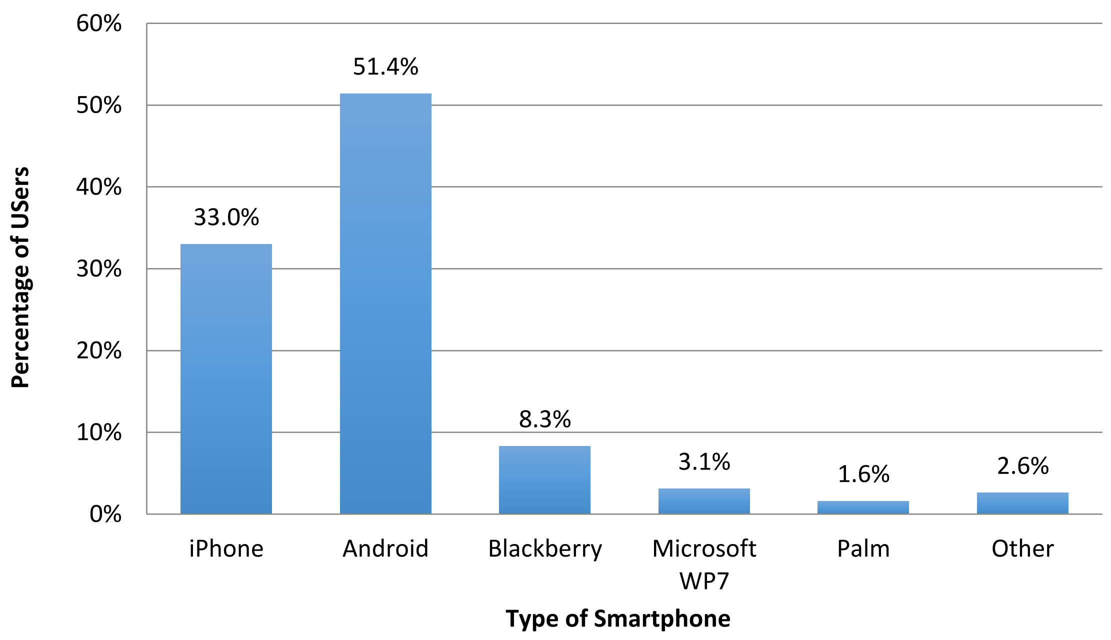


Figure 4.2 What kind of smartphone do you use? (Goldman, 2011)

The survey results confirm that Android is a good choice of development platform for this project. Over half of the people already use Android smartphone and 54.4% plan to buy one in the future.

What smartphone (platform) do you think you will buy?

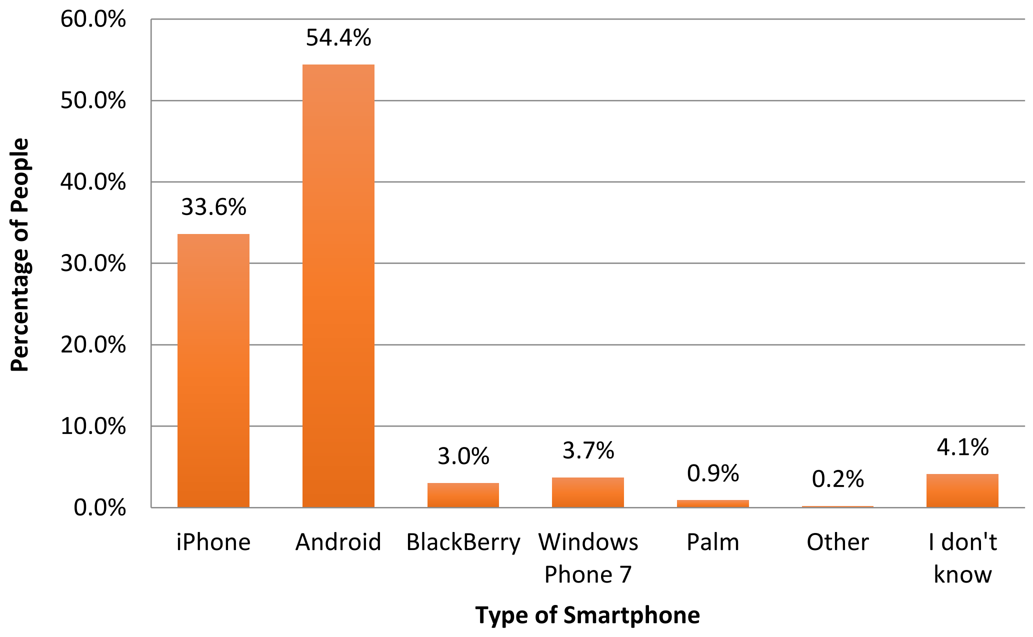


Figure 4.3 What smartphone do you think you will buy (Goldman, 2011)

As smartphones become more popular there has been an increase in the number of users who have purchased or plan to purchase one or more mobile phone games. Web-enabled smartphones are gaining popularity in the UK with 31% of people surveyed owning one, this will make it possible to include advertising, online high score tables and downloadable content. (Goldman, 2011)

The mobile ad service AdMob by Google will be used to display banner ads in this project. Sometimes people accidently click on these ads and research conducted in 2011 found that 47% of ad clicks were mistakes. (MarketingProfs, 2010) Advertising other products will generate revenue which can be used to improve the game with new graphic, sound and resources and game play features.

It is important to consider the characteristics which can lead people to recommend a game to others. The PopCap Games survey found that of the people surveyed the most important characteristics were fun to play 85%, easy to learn 49% and is the game challenging enough 45%. These characteristics will important to consider throughout the development process.

Game Characteristics Influencing Recommendations

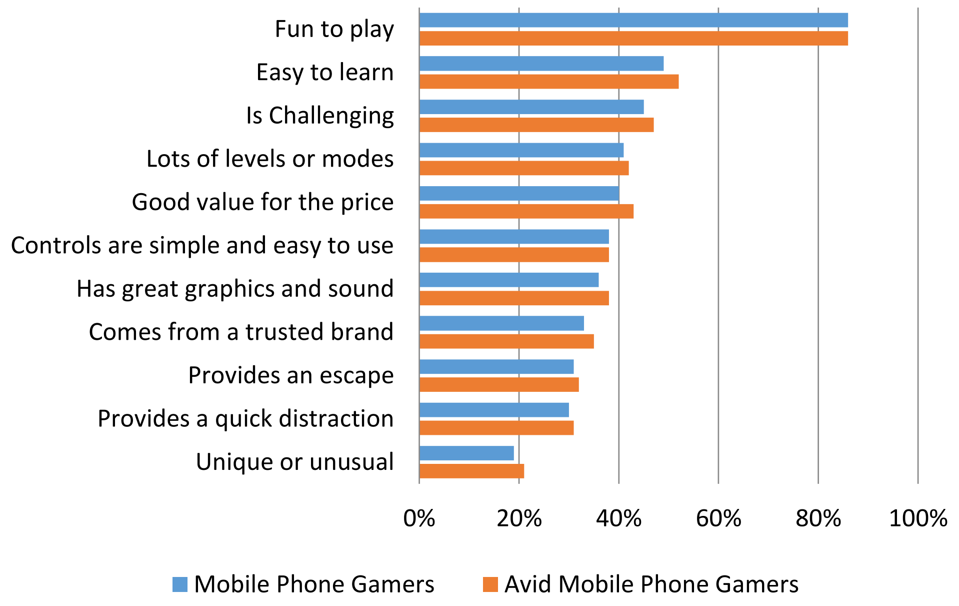


Figure 4.4 Characteristics Influencing Recommendations (Games, 2011, p. 5)

If the game is a success it is then important to consider whether people will be willing to purchase downloadable content. The PopCap Games survey shows that one-third (34%) of smartphone owners purchased content for a mobile phone game they originally obtained for free.

Content Purchase (Power-ups, New Levels / Modes)

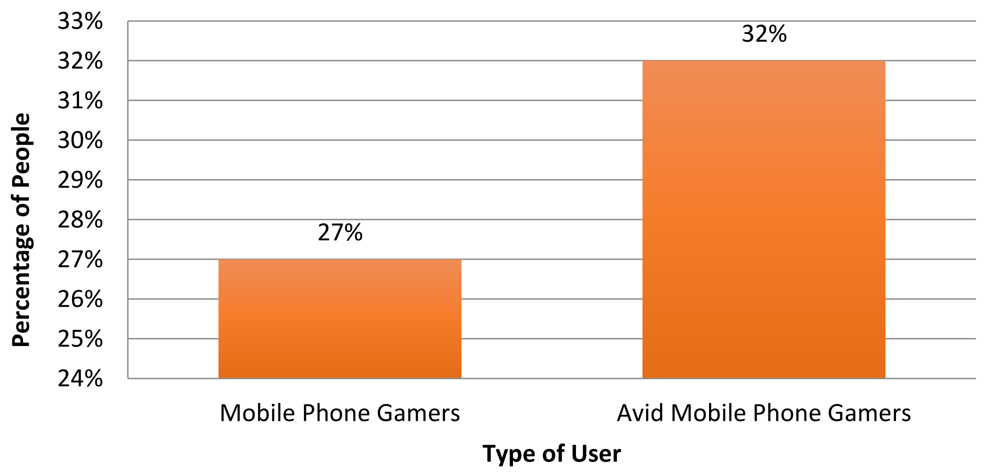


Figure 4.5 Content Purchase (Power-ups, Levels and Modes) (Games, 2011, p. 5)

Another important aspect to consider is social networking as it is popular with mobile gamers with almost two-thirds (63%) accessing sites such as Facebook, MySpace or Bebo. Social gaming could be integrated into this project using a service such as OpenFeint to provide online high score tables.

Risk Analysis

The following risk management table identifies the potential risk and problems which could occur while completing the tasks and activities in the project lifecycle.

Risk	**Implications**	**Mitigation**
Project overruns schedule and is delivered late	Project fails or is delivered late and suffers penalty	Schedule tasks and identify priorities Maximize remaining time by rescheduling tasks
Function is too difficult to implement in the remaining time	The schedule of tasks is disrupted	Review the function Drop the function Find an alternative approach
Software bug or game crash	Effects stability or gameplay	Resolve software error Modify source code Test modified code
Third party SDK or development tool documentation issue	The schedule of tasks is disrupted	Ask for help with documentation Check forums and mailing lists for solutions
Confusing requirement	The functional requirement is overcomplicated or is not understood completely	Simplify the function Drop the function Find an alternative approach and test that it operates similarly

Unexpected time delay due to sickness	Effects development time	Reschedule the project Adapt project plan Simplify requirements
Data loss or hardware failure	Project is deleted, lost or stolen	Backup work regularly Recover lost data Adapt the project plan

Figure 5.1 Risk Analysis Table

Competition Analysis

Computer game development is a competitive market which attracts large amounts of business interest and investment. The initial market has varying entry requirements ranging from free for the Android SDK to the PSP and Nintendo DS which have official hardware only available to academic and commercial scale projects.

The mobile game market consists of many companies developing 2D/3D games and has expanded during the years 2008-11 generating over $2.4 billion dollars in 2010. The mobile application analytics service provided by Flurry tracks over 12 billion anonymous user sessions per month and reports that in 2010 iPhone and Android devices represent 34% of the portable game market.

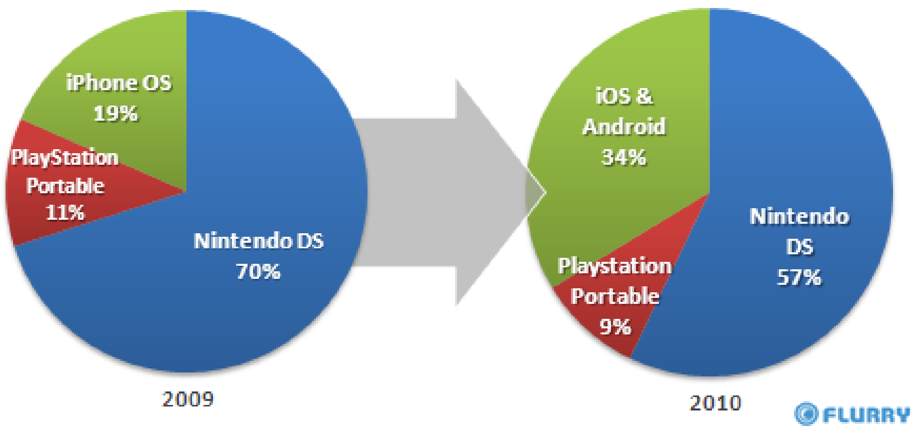


Figure 2.4 U.S Portable Game Software by Revenue (Farago, 2011)

The iPhone developer platform costs £60 to join and requires an Apple Mac OSX computer. There are many examples of homemade and commercial games already available including Angry Birds, Cut the Rope, and Fruit Ninja. Applications are sold on the App store with an average price of $1.44 as of 2011. (Lowensohn, 2011)

The Google Android platform cost £15.00 to join and is an open source platform with a large community of developers creating games and applications to be sold and downloaded for free on the Android app store Google Play. Replica Island is a notable game example as it is a completely open source platform game very similar to this project. (HNYD, 2011)

The Nintendo DS has a large proportion of the software sales but requires an official Nintendo SDK which is only distributed to commercial games developers and will not be available to this project.

Project Requirements

The project requirements describe the functions of a software system and must be established before the development can begin.

The functional requirements are clearly defined functions or tasks which the system must perform and capture the intended behaviour of the system. (Bredemeyer, 2011)

The non-functional or quality attributes - describe how well a system performs its function, these could be constraints or added features which are not vital to the project functioning. (Stellman, 2010)

Overview of Functional Requirements

- User Controllable Character
- Enemies
- User Interface
- Collision Detection
- Collectable Items
- Level Map
- Level Editor
- Level Themes
- Sound Effects and Music
- Load / Save Game Feature

- Score System
- Time Limit

Overview of Non-Functional Requirements

- Accessibility
- Documentation
- Maintainability
- Portability
- Usability
- Reliability
- Performance

Benefits for Clients

This project will be aimed at anyone who wishes to play a fun platform game on their mobile device. It will be released as a free demo on the Google Play app store and will use advertising to generate revenue. The main benefit for clients will be an exciting game which provides lasting entertainment value. This will include multiple levels with different themes, sound and music, enemies, collectable items, bonus levels, level editor and load /save game features.

Hardware Requirements

The following are recommended hardware requirements for this project:

- 320 x 240 (2.8" screen)
- Touch screen or keyboard control input
- Minimum storage of 4mb

Design

The design section of this report will use UML (Unified Modelling Language) diagrams to visualize the software development process.

The user interface and game play features will be demonstrated using paper prototypes. The process of computer game and level design will be discussed in relation to how it applies to this project.

Unified Modelling Diagrams (UML)

UML, short for Unified Modelling Language, is a standard modelling language which uses a set of diagrams, developed to help system and software developers specify, visualize, and document the different parts of a software systems.

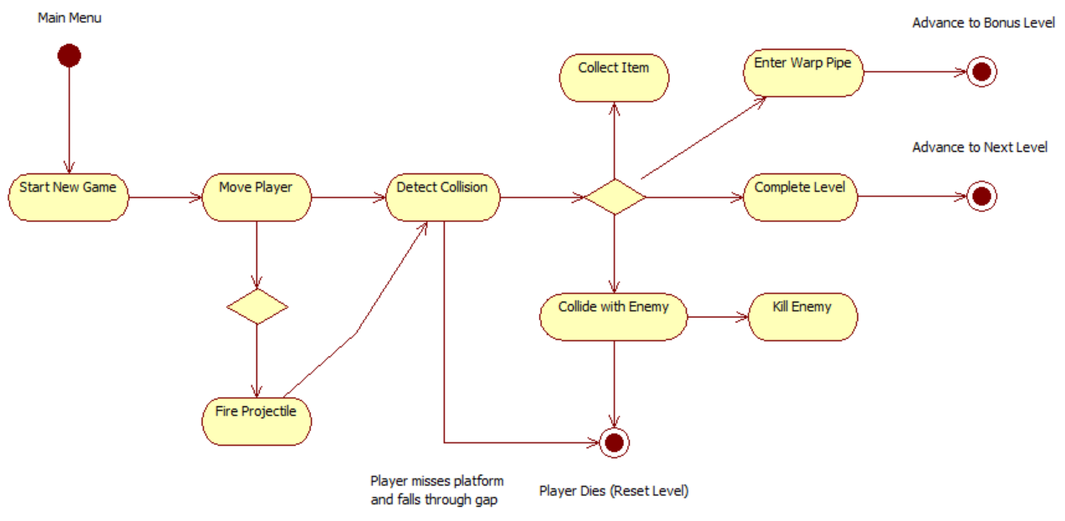
The UML is an important part of developing object oriented software and the software development process. Using the UML helps project teams communicate, explore potential designs, and express the architectural design of the software. (Paradigm, 2011)

Activity Diagram

The activity diagram "shows activities and actions to describe workflows" (Bennett, McRob and Farmer, 2006) of the program and the decisions made by the player.

The diamond represents a decision and occurs when the player decides whether to fire a projectile, collect an item, kill or avoid an enemy or exit the level.

The circle represents a final state where the game ends, resets or advances the level.



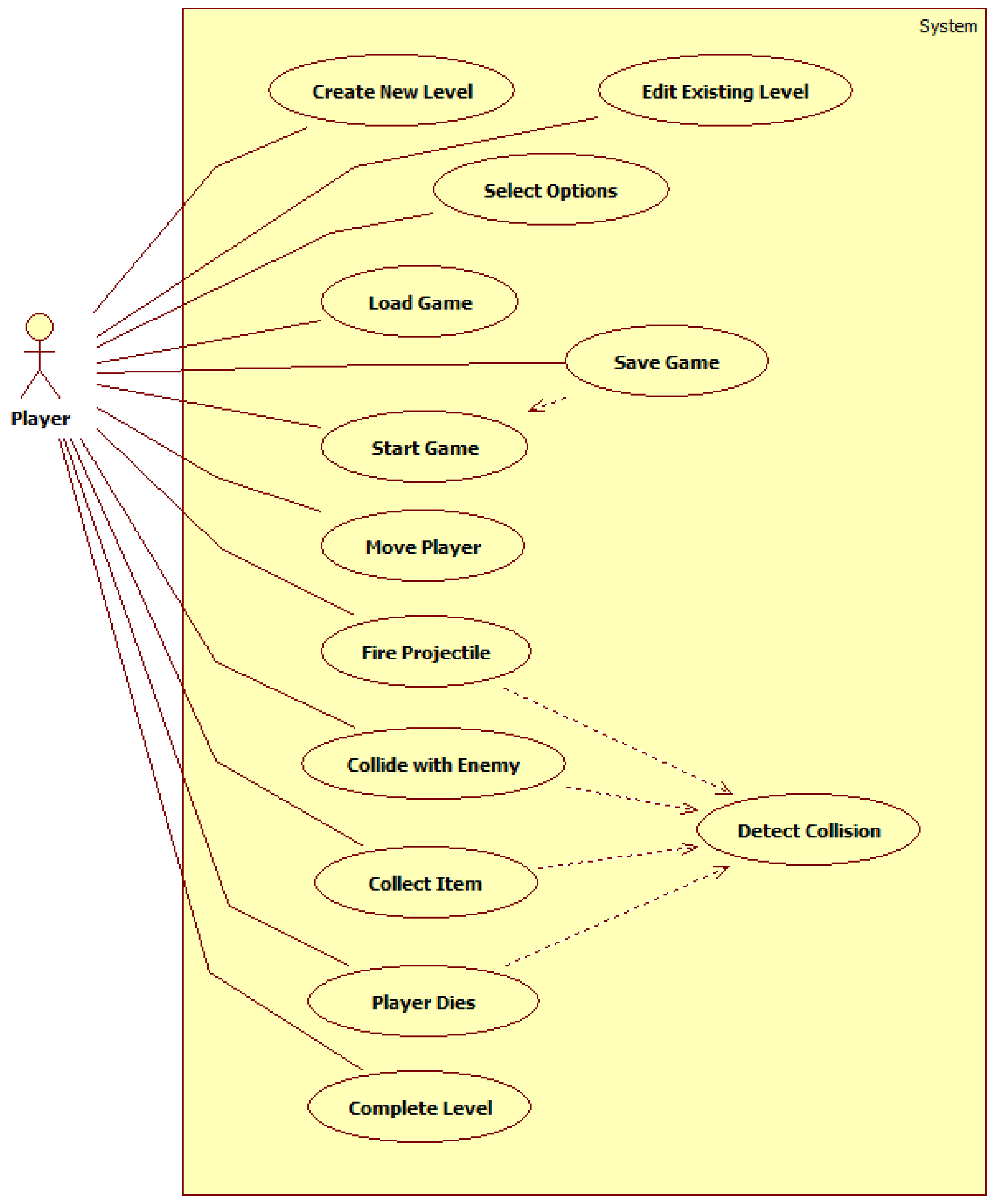
Key: → : Transition ◉ : Final State

Figure 1.1 Activity Diagram

Use Case Diagram

The use case diagram illustrates the interaction of the actors with the system and the dependencies of the use cases.

The game is designed to be played by one player so there is only one actor in the diagram although more could be added later.

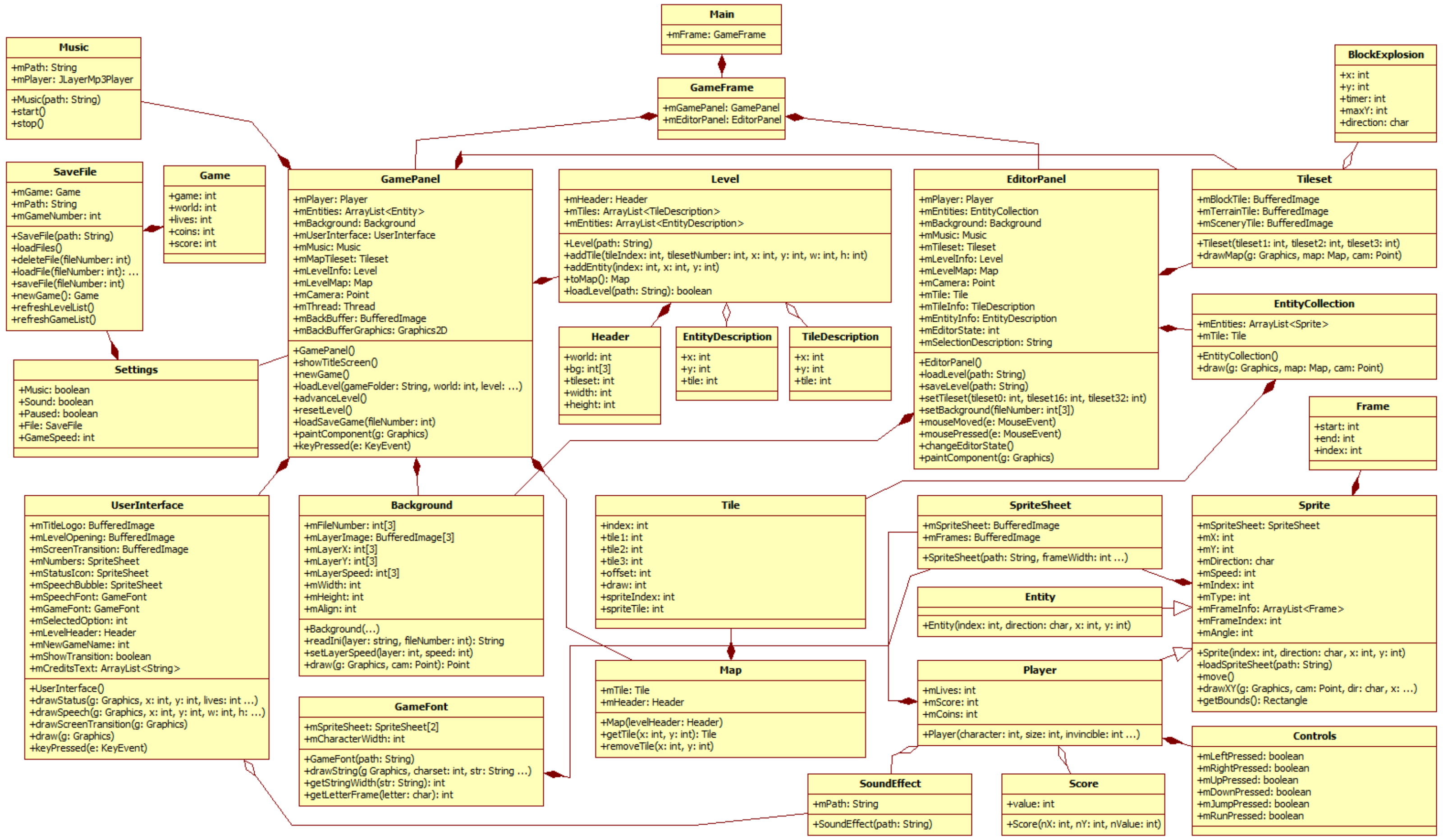


Key:

——	:	Association
- - -▶	:	Dependency

Figure 1.2 Use Case Diagram

Class Diagram - The following class diagram is and overview of the structure of the Java SE version of the project.


Main
+mFrame: GameFrame
GameFrame
+mGamePanel: GamePanel
+mEditorPanel: EditorPanel
Music
+mPath: String
+mPlayer: JLayerMp3Player
+Music(path: String)
+start()
+stop()
BlockExplosion
+x: int
+y: int
+timer: int
+maxY: int
+direction: char
SaveFile
+mGame: Game
+mPath: String
+mGameNumber: int
+SaveFile(path: String)
+loadFiles()
+deleteFile(fileNumber: int)
+loadFile(fileNumber: int): ...
+saveFile(fileNumber: int)
+newGame(): Game
+refreshLevelList()
+refreshGameList()
Game
+game: int
+world: int
+lives: int
+coins: int
+score: int
GamePanel
+mPlayer: Player
+mEntities: ArrayList<Entity>
+mBackground: Background
+mUserInterface: UserInterface
+mMusic: Music
+mMapTileset: Tileset
+mLevelInfo: Level
+mLevelMap: Map
+mCamera: Point
+mThread: Thread
+mBackBuffer: BufferedImage
+mBackBufferGraphics: Graphics2D
+GamePanel()
+showTitleScreen()
+newGame()
+loadLevel(gameFolder: String, world: int, level: ...)
+advanceLevel()
+resetLevel()
+loadSaveGame(fileNumber: int)
+paintComponent(g: Graphics)
+keyPressed(e: KeyEvent)
Level
+mHeader: Header
+mTiles: ArrayList<TileDescription>
+mEntities: ArrayList<EntityDescription>
+Level(path: String)
+addTile(tileIndex: int, tilesetNumber: int, x: int, y: int, w: int, h: int)
+addEntity(index: int, x: int, y: int)
+toMap(): Map
+loadLevel(path: String): boolean
EditorPanel
+mPlayer: Player
+mEntities: EntityCollection
+mBackground: Background
+mMusic: Music
+mTileset: Tileset
+mLevelInfo: Level
+mLevelMap: Map
+mCamera: Point
+mTile: Tile
+mTileInfo: TileDescription
+mEntityInfo: EntityDescription
+mEditorState: int
+mSelectionDescription: String
+EditorPanel()
+loadLevel(path: String)
+saveLevel(path: String)
+setTileset(tileset0: int, tileset16: int, tileset32: int)
+setBackground(fileNumber: int[3])
+mouseMoved(e: MouseEvent)
+mousePressed(e: MouseEvent)
+changeEditorState()
+paintComponent(g: Graphics)
Tileset
+mBlockTile: BufferedImage
+mTerrainTile: BufferedImage
+mSceneryTile: BufferedImage
+Tileset(tileset1: int, tileset2: int, tileset3: int)
+drawMap(g: Graphics, map: Map, cam: Point)
EntityCollection
+mEntities: ArrayList<Sprite>
+mTile: Tile
+EntityCollection()
+draw(g: Graphics, map: Map, cam: Point)
Settings
+Music: boolean
+Sound: boolean
+Paused: boolean
+File: SaveFile
+GameSpeed: int
Header
+world: int
+bg: int[3]
+tileset: int
+width: int
+height: int
EntityDescription
+x: int
+y: int
+tile: int
TileDescription
+x: int
+y: int
+tile: int
Frame
+start: int
+end: int
+index: int
UserInterface
+mTitleLogo: BufferedImage
+mLevelOpening: BufferedImage
+mScreenTransition: BufferedImage
+mNumbers: SpriteSheet
+mStatusIcon: SpriteSheet
+mSpeechBubble: SpriteSheet
+mSpeechFont: GameFont
+mGameFont: GameFont
+mSelectedOption: int
+mLevelHeader: Header
+mNewGameName: int
+mShowTransition: boolean
+mCreditsText: ArrayList<String>
+UserInterface()
+drawStatus(g: Graphics, x: int, y: int, lives: int ...)
+drawSpeech(g: Graphics, x: int, y: int, w: int, h: ...)
+drawScreenTransition(g: Graphics)
+draw(g: Graphics)
+keyPressed(e: KeyEvent)
Background
+mFileNumber: int[3]
+mLayerImage: BufferedImage[3]
+mLayerX: int[3]
+mLayerY: int[3]
+mLayerSpeed: int[3]
+mWidth: int
+mHeight: int
+mAlign: int
+Background(...)
+readIni(layer: string, fileNumber: int): String
+setLayerSpeed(layer: int, speed: int)
+draw(g: Graphics, cam: Point): Point
Tile
+index: int
+tile1: int
+tile2: int
+tile3: int
+offset: int
+draw: int
+spriteIndex: int
+spriteTile: int
SpriteSheet
+mSpriteSheet: BufferedImage
+mFrames: BufferedImage
+SpriteSheet(path: String, frameWidth: int ...)
Entity
+Entity(index: int, direction: char, x: int, y: int)
Sprite
+mSpriteSheet: SpriteSheet
+mX: int
+mY: int
+mDirection: char
+mSpeed: int
+mIndex: int
+mType: int
+mFrameInfo: ArrayList<Frame>
+mFrameIndex: int
+mAngle: int
+Sprite(index: int, direction: char, x: int, y: int)
+loadSpriteSheet(path: String)
+move()
+drawXY(g: Graphics, cam: Point, dir: char, x: ...)
+getBounds(): Rectangle
GameFont
+mSpriteSheet: SpriteSheet[2]
+mCharacterWidth: int
+GameFont(path: String)
+drawString(g Graphics, charset: int, str: String ...)
+getStringWidth(str: String): int
+getLetterFrame(letter: char): int
Map
+mTile: Tile
+mHeader: Header
+Map(levelHeader: Header)
+getTile(x: int, y: int): Tile
+removeTile(x: int, y: int)
Player
+mLives: int
+mScore: int
+mCoins: int
+Player(character: int, size: int, invincible: int ...)
Controls
+mLeftPressed: boolean
+mRightPressed: boolean
+mUpPressed: boolean
+mDownPressed: boolean
+mJumpPressed: boolean
+mRunPressed: boolean
SoundEffect
+mPath: String
+SoundEffect(path: String)
Score
+value: int
+Score(nX: int, nY: int, nValue: int)

Key:

—◇ : Aggregation

—◆ : Composition

—▷ : Generalization

Figure 1.3 Class Diagram.

Level Design - The following screenshots show the design of the first level created using the level editor.

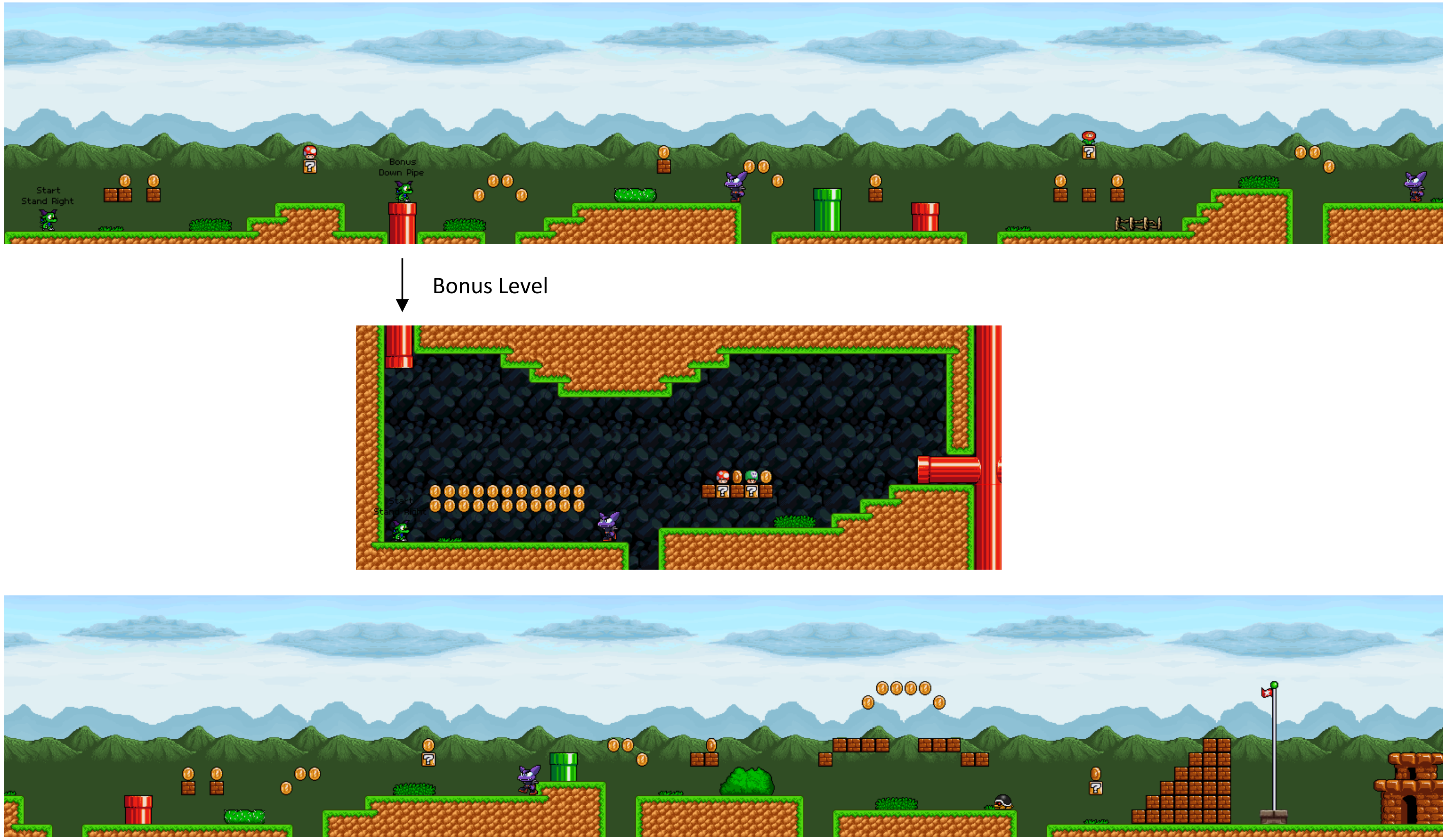


Figure 2.1 Level 1-1-1 Screenshots from the level editor

User Interface – The user interface prototype demonstrates the main game menu.

The game will begin with a menu system allowing the player to choose to start a new game, load a game, edit levels or select options. The following diagram outlines the options that will be displayed to the player on selecting each menu item.

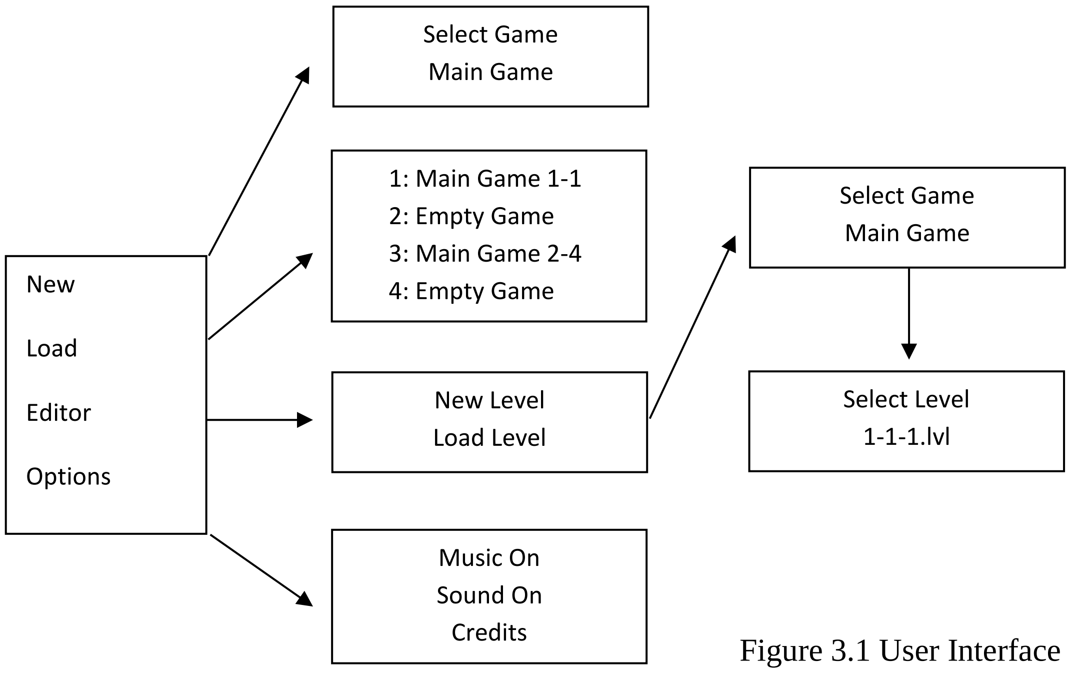


Figure 3.1 User Interface

Paper Prototype – The business prototype demonstrates the gameplay features.

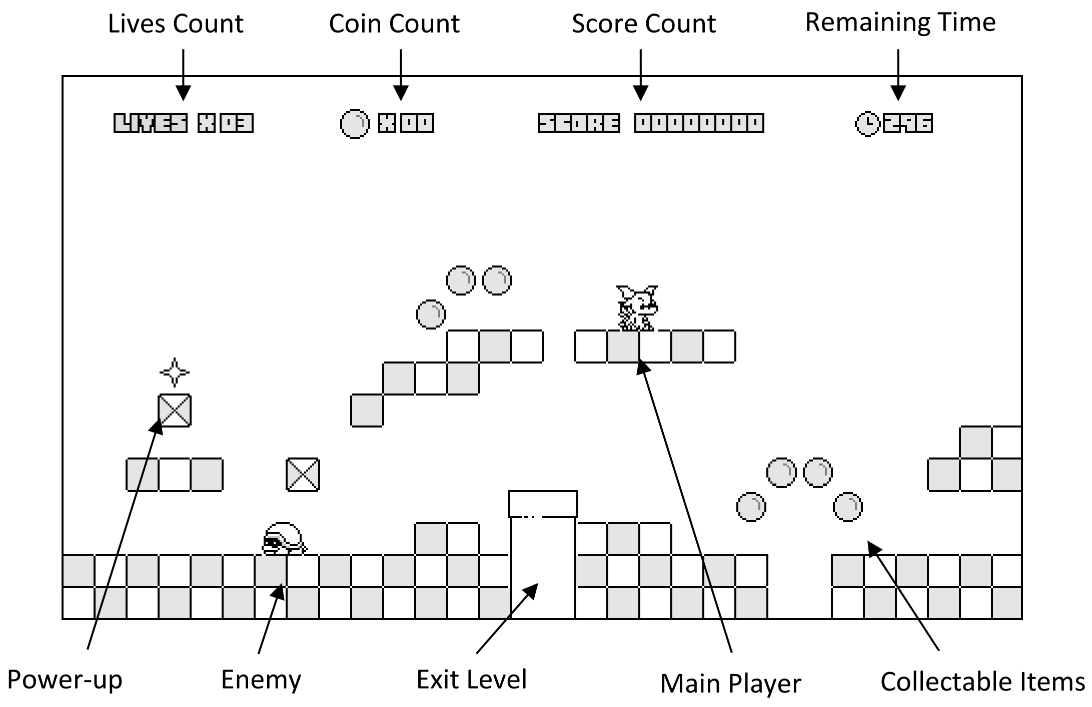


Figure 3.2 Paper Prototype

Computer Game Design

The general definition of design is the process by which a designer creates a context to be encountered by a participant, from which meaning emerges. Computer game designers use the concept of meaning and making sense of a situation and to form systems of interaction. (Zimmerman, 2003, p. 23)

Level designers are generally not involved in creating textures, characters or the source code for a game but instead arrange the different game elements to create a fun game playing experience. (John Feil, 2005, pp. 2-4, 9-10) This project will involve combining various different elements including diagrams, game artwork, prototypes sounds, music, and level designs into an enjoyable game. (Darby, 2007, p. 2)

Research should be conducted to find the audience of the game and to identify design ideas and features which will provide a fun game playing experience. This should include using internet forums, market research, game review sites and magazines. It is important to consider while conducting research that some people may be looking at the game with a biased opinion. (John Feil, 2005, pp. 2-4, 9-10)

Computer game players expect games to be enjoyable, to provide entertainment value and be a source of enjoyment and distraction from real life. Meaningful play is one of the goals of successful game design and it can be defined as the process by which a player takes action within the designed system of a game and the system responds to the action. (Zimmerman, 2003, p. 23)

The game genre defines what people expect your game to do and can be a deciding factor in whether it will be a success or a failure. If a game can meet the player's expectations, or even exceed them it will almost certainly be a success. Certain game genres such as FPS (First Person Shooters) have predefined control standards for example using the W, A, S, D keys to move around. Another example is PC games which have typically incorporated save anywhere features whereas console games more often use pre-defined save points. This game will make use of easy to use keyboard and touch screen control systems and a save anywhere feature. (John Feil, 2005, pp. 2-4, 9-10)

DSDM Atern is an agile and iterative design framework where the initial prototype should be developed as early in the design process as possible. Iterative design is an important part of game design and this this should include a play-based design process. The prototype will be evaluated by the developer and game testers during the software development process and used to make decisions on what should be included in future versions of the game. (Zimmerman, 2003, p. 23)

Storyboarding is an effective method of planning a computer game. It has many advantages including saving development time and costs, explaining game sequences, communicating concepts and working out problems which could arise. Paper prototypes and storyboards will be used to create a consistent vision of the game reducing the amount of wasted effort. (Pardew, 2004, p. 55)

Game theory is a field of mathematics that studies decision making in formal models based on optimal strategies. It could be used to help explain the logic behind the decisions the player makes. (Koster, 2013, p. 225)

While designing games it is important to provide a challenge this should include objectives and obstacles. The standard challenges which could be included are time, dexterity endurance, memory, logic and resources. (John Feil, 2005, pp. 2-4, 9-10)

Semiotics can be defined as the study of signs and communication through drawings, words, sounds and body language. (Chandler, 2000) The general definition of a sign is something that stands for something to somebody, in some respect or capacity. Computer games often use signs to denote the elements of the game world. Computer game players interpret signs and react to them. For example in this project mushrooms, fire flowers and stars will be used to represent power-ups which the player can collect. (Zimmerman, 2003, p. 23)

Level design can sometimes produce unintentional game play elements. This is known as emergent gameplay, as new systems emerge from the ways the old ones combine with each other. This could occur in any game where the player can move objects around through interaction. The player could place an object where it could be used to to overcome a challenge that you intended to be much more difficult. In this project the level design will aim to stop the player using these unintentional gameplay elements and implement an alternative path through the level. (John Feil, 2005, pp. 2-4, 9-10)

Implementation

The implementation will discuss the process of creating the Java, Android and GP2X versions of the game. To make it easier to read the different topics they have been grouped into four main sections. The game framework will discuss the methods to create a game window or graphics surface and the main game logic. The graphics section will explain the libraries and functions used to load draw images. The load / save game topic covers the classes used to store and load the player's progress. The audio section describes the different audio libraries used to play mp3 music and wav sound effects. The players and enemies section explains how the player interacts with the map and the use of collision detection algorithms.

Game Framework

Main

The main class contains the main method which is used as the entry point of the application. In the Java programming language every application must contain a main method which can be passed command line parameters. Starting the game from the command prompt with the parameter "debug" will enable debug mode. The main application window is created in the Main class constructor using the GameFrame class which extends the parent Java Swing class JFrame.

MainActivity (Android)

Activities are a key concept in Android development and represent a single screen with a user interface. The onCreate(Bundle) method initializes the activity. The layout of the UI is defined using the setContentView(View) method. For a normal application which requires UI components to be positioned on the screen a XML layout resource file stored in R.layout. As this project is a game and requires hardware accelerated graphics the OpenGL ES library will be used instead. (Google, 2011)

The GLSurfaceView API class included since the 1.5 of the Android SDK and can be passed as an argument for the setContentView() method. It is defined as a member variable of the MainActivity class and using polymorphism is constructed with the GLSurfaceViewEvent class to provide control input events. (Google, 2009)

Main (GP2X)

The C++ version of the game does not include the GameFrame class as it does not require a game window. The graphics are drawn to the screen using something known as the SDL surface. The main class creates the drawing surface and includes member variables for the GamePanel and EditorPanel and the key and gamepad event listeners.

GLSurfaceViewRenderer (Android)

The GLSurfaceViewRenderer class implements the GLSurfaceView.Renderer interface and contains the code required to initialize and render the GLSurface and create the GamePanel.

GLSurfaceViewEvent (Android)

The GLSurfaceViewEvent class subclasses the GLSurfaceView class to provide key and touch screen events and set the custom GLSurfaceView renderer. It is constructed using the MainActivity application context which is then passed on to the GLSurfaceView parent class constructor. The touch screen and key events are processed using the onTouchEvent(), onKeyDown() and onKeyUp() methods.

GameFrame

The GameFrame class creates the application window, and contains the member variables GamePanel and EditorPanel. It extends the javax.swing.JFrame top level container class which can have other JPanel components added to it using the add() method.

GamePanel

The GamePanel class contains the main game thread which draws the games graphics and handles game logic such as collision detection between the player, map objects and entities. In the Java implementation it inherits the Jpanel class which is a lightweight container class that can be added to GameFrame.

The Java keyboard and mouse input is provided through the implementation of the KeyListener, MouseListener, MouseMotionListener and MouseWheelListener interfaces. The class must define certain methods for each listener interface such as keyPressed, keyReleased and keyTyped for the KeyListener.

The class implements the Runnable interface which is used by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

The GamePanel constructor creates a back buffer or "double buffer" to display the graphics on. This helps to prevent the games graphics flickering between screen updates. First the graphics are drawn to the back buffer when this is complete the back buffer is drawn to the graphics surface.

The loadLevel() function loads the level header, level map, background, tileset and entities and returns a Boolean value indicating if the level is loaded successfully.

The toMap() function is used to initialize the Map object with the object and entity tile data. The entity objects are added to the level using a for loop which iterates through the descriptions, constructs Entity objects and then adds them to an array list.

The newGame() method tries to open the .ls (level script) and "0.0.1.lvl" files to display the intro sequence. The loadScript() method loads a level script file containing commands which run in a time sequence. The .ls (level script) file is opened using the RandomAccessFile class and the readLine() function reads commands from the file. The ScriptCommand class is declared as an inner class of GamePanel and stores the variables time, command, text, x, y, width, height, direction and timer.

The advanceLevel() method advances to the end level specified in the level header. The loadSaveGame() method loads a save game. The save game information is retrieved from the Settings File object using the getGame() function.

The run() method contains a while loop which calls the repaint() method measuring the time between frame updates known as delta time to apply to all moving graphics to keep the frame rate consistent.

The repaint() method invokes paintComponent() which is passed a graphics context.

The camera coordinates are initialized using the player position and passed to the background class along with the graphics context:

```
mCamera.x = mPlayer.getLeft() - Settings.LimitCameraX;
mCamera.y = mPlayer.getDown() - Settings.LimitCameraY;
mCamera = mBackground.draw(back_buffer_graphics, mCamera);
```

The Tileset object is used to draw the blocks, terrain and scenery defined in the level:

```
mMapTileset.drawMap(back_buffer_graphics, mLevelMap, mCamera);
```

The collisions between the entities and map tiles are detected and the entities are drawn on the graphics surface:

```
for (int i = 0; i < mEntities.size(); i++) {
   mEntities.get(i).detectMapCollision(mLevelMap);
   mEntities.get(i).draw(back_buffer_graphics, mCamera);
```

The collisions between the player, map tiles and entitles are detected:

```
mPlayer.detectEntityCollision(mEntities);
mPlayer.detectMapCollision(mLevelMap);
```

The player sprite is drawn at the camera position:

```
mPlayer.draw(back_buffer_graphics, mCamera);
```

The keyPressed() method is invoked when a key has been pressed. The player setControl() method is passed the key event:

```
if (mPlayer != null)
   mPlayer.setControl(e, true);
```

If the player's top y coordinate is greater than the level height the player has fallen off the map and loses a life. If the player has remaining lives the level is reloaded using the resetLevel() method. If the players lives count is equal to zero the game ends.

If the Settings game state equals "game" the players status information including the lives, coins, time and score is displayed at the top of the screen:

```
mUserInterface.drawStatus(back_buffer_graphics, 50, 15,
mPlayer.getLives(),mPlayer.getCoins(),mPlayer.getScore(),
mPlayer.getTime());
```

The back buffer containing the games graphics is drawn to the screen using the drawImage() function in the following code:

```
g.drawImage(back_buffer, 0, 0, getWidth(),getHeight(), this);
```

UserInterface

The UserInterface class handles key events and draws the screen overlays for the title, menu, pause, options, status, level opening, level editor settings, credits, load and save game screens. The constructor loads the game fonts, images and sprite sheets required to display the user interface graphics.

The drawLives() function draw the number of lives the player has remaining at the top of the screen while the game is in progress.

The drawCoins(), drawScore() and drawTime() methods are similar to the drawLives() method but use different icons to display the amount of coins, score or time. The drawStatus() method combines the all of the above functions to draw the players status information.

The drawSpeech() method draw speech bubbles for the games intro sequence. It accepts the graphics context, x, y, width, height, text and direction parameters.

The drawScreenTransition() method draws a scrolling screen overlay either horizontally or vertically and is displayed at the start of each level.

The UserInterface draw() method draws screen overlays such as the game menu and level editor settings. It decides which screen to draw using an if statement to select the Settings game state variable.

The keyPressed() method is invoked when a key has been pressed on the GamePanel and is used to select menu options and edit game settings.

Settings

The Settings class stores global setting variables for the game. Storing all the settings in one place simplifies the code and makes it easier to edit game settings.

The Settings class includes variables to set the following options sound, music, debug mode, cheats, pause, save file information and game speed.

The settings are declared as static member and can be accessed without having to construct the class.

Graphics

Image

The Image class loads a loads a bitmap image with a set transparent colour and creates a BuffferedImage image with an alpha channel.

GLSprite (Android)

The GLSprite class is similar to the Image class found in the Java version. It loads a PNG image with an alpha channel that is mapped to a quad constructed of vertices and can be drawn on the GLSurface. The textured quad is made of a pair of triangles and is defined by 4 sets of coordinates as show in the following diagram:

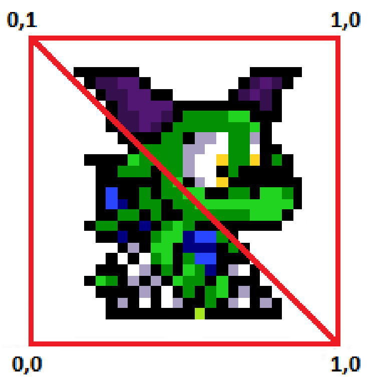


Figure 1.1 – Dragon Sprite as a textured quad (PJ Cabrera, 2010, p. 270)

The GLSprite class extends the Renderable class It also implements the cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class. This method is used in the SpriteSheet class to make a copy which frames of animations can be extracted from a copy of the sprite sheet.

Image (GP2X)

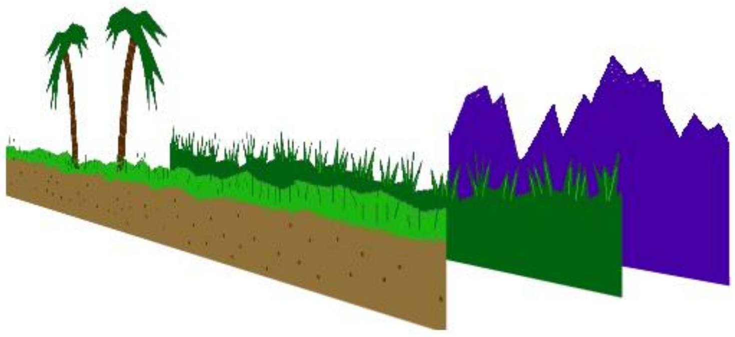
The C++ version of the game uses the SDL library to load the PNG images which can contain an alpha transparency channel.

The image class is constructed with a char pointer to the file path of the image file. The SDL IMG_Load() function loads the image and the SDL_BlitSurface() is used to draw the image to the screen surface.

Background

The background class draws a parallax scrolling background with three image layers on the graphics surface. Parallax scrolling is a pseudo-3D technique where the background layers move at different speeds to create the illusion of depth in a 2D game. (Wikipedia, 2011)

The following illustration shows how the three layers are staked to create the effect:



Layer 0 (Near) - Ground

Layer 1 (Middle) - Grass

Layer 2 (Far) - Mountains

Figure 1.2 –The three background layers stacked (Martins, 2009)

The Background constructor accepts the parameters file number, width, height, layer speed and alignment. The file number parameter is an array of three numbers which specify the file numbers of the background images.

The width and height parameters specify the total width and height of the level map. It is important to know the dimensions of the level so the camera which follows the player does not leave the level area. The height is used to calculate they position of the layer images which should be anchored to the bottom of the screen.

The layer speed parameter is an array of three numbers which specify the scrolling speed of each background layer and is used in the draw() function to calculate the scrolling position of the background layers.

Finally the alignment parameter specifies if the background should be tiled horizontally or vertically. If the alignment is set to 1 -vertical only the far layer is drawn on the graphics surface vertically.

The draw() function draws the background on the graphics surface at the camera position and returns the updated camera coordinates which are limited to the width and height of the level. The areas of the background which are not covered by the layer images are filled with the top-left pixel colour of the far layer.

The x, y drawing coordinates of the background layers are calculated in relation to camera coordinates using the background layer image width 512 pixels:

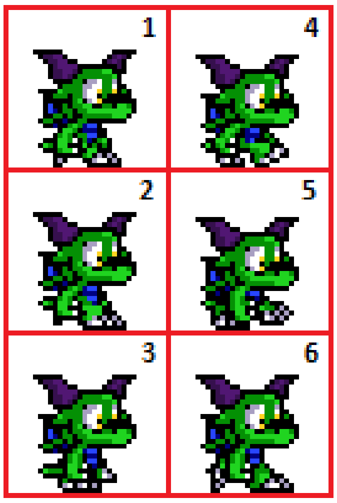
```
int bgX = cam.x - (cam.x / 512 * 512);
int bgY = cam.y - (cam.y / 512 * 512);
```

If the Settings variable animation is equal to true, the background layers will scroll along the x axis.

SpriteSheet

The SpriteSheet class loads a bitmap image from a resource file using the Image class and the getImage() function to retrieve an image with alpha channel. If the file path is valid the image is stored and the sheet width and height is initialized.

The sprite sheet image is divided vertically into rows and columns as shown in the image below:



The sprite sheet width = 67 and height = 100.
The frame width = 32 and height = 32.

The sprite sheet has been divided vertically as it makes it easier to align the frames of animation.

Figure 1.3 – Dragon sprite sheet with frame numbers (NO-Body-The-Dragon, 2012)

The frames array is initialized with the array size equal to the frame count parameter and the getFrameFromSheet() function is used to get the subimages containing frames of animation from the sprite sheet image and stores them in the array.

The getFrame() function is passed the frame number returns the buffered frame of animation. It is overloaded with the parameters frame number and frame direction passed as a char value either 'l' – left or 'r – right'. The frames of animation in the sprite sheet are all facing left and need to be flipped horizontally when facing right.

GameFont

The GameFont class loads two sprite sheets for a font character set such as upper and lower case and draws strings of text on the graphics surface. The constructor is passed a folder parameter which specifies the folder inside the "res/fnt/" resource folder that contains the sprite sheets. The width of each character is stored in an .ini file which is read using the BufferedReader.

The drawstring() method draws a string of text at the x, y coordinates detecting upper and lower case letters. It loops through the length of the string, gets the frame number, selects the character set and draws the frame from the SpriteSheet.

Tileset

The Tileset class loads and draws the block, terrain and scenery map tiles. The constructor is passed the folder number for each tile set and loads the images containing the tiles.

The image below shows the "res/obj/block/1.png" tileset. This contains the end of level castle, blocks and warp pipes tiles.

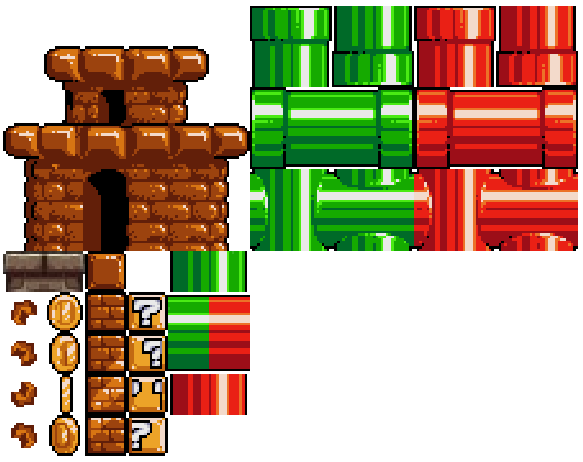


Figure 1.4 Block tileset

The position of the tiles in the tileset image are defined in a tileset.ini file for example the coin is defined as: "10,0,Coin,1,1,0,16,112".

The initDefinitions() function opens the tileset.ini file for each tileset using a BufferedReader object to read the lines of csv (comma separated values) into an array list. The getTileDescription() function returns a string array which contains [0]-index [1]-tileset [3]-name [4]-width [5]-height [6]-collision flag [7]-tileX [8]-tileY.

The getFrame() function returns a tile from the specified tileset. It is passed the tile, tileset and tiled integer parameters. The tile is the index of the tile, tileset is the type of tile and tiled specifies a piece of the four parts of the tiled terrain.

The draw() method draws a tile on the graphics surface. It is passed the graphics context, x, y, camera coordinates and tile information. The x, y coordinates of the tile are calculated using the camera coordinates and tile offset values. The tile offset increases if the tile state equals one and has been hit by the player jumping.

If an empty block is hit as the large or fire power-up player it will explode into four pieces. The setBlockExplode() method adds a piece of a block explosion to an array list of BlockExplosion objects.

The drawMap() method draws the tiles defined in the map within the camera area. It is passed the graphics context, map and camera coordinates.

Level Map and Editor

Map

The map class stores header and tile information. The tiles include the terrain, blocks and scenery and entities are stored in a 2D array.

The class is constructed with the level Header information and creates a 2D array of tiles to the dimensions of the level:

```
w = mHeader.width / 16;
h = mHeader.height / 16;
mTile = new Tile[w][h];

for (int a = 0; a < w; a++){
   for (int b = 0; b < h; b++) {
      mTile[a][b] = new Tile();
}
```

The setObjectTile() method is passed a tile index and tile description which specifies the properties of the tile. It loops through the width and height of the tile setting the index, tile, tileset, repeat, count, collision, state and offset variables.

The setEntityTile() method follows the same process and sets the sprite index, tile, sprite and draw variables.

Tile

The Tile class is defined as an inner class of the Map class. It stores tile information which is read from the .lvl file using the Level class. This includes the index, tile, tileset, repeat, count, collision, offset, draw, entity index, entity tile and entity draw.

Level

The Level class opens and saves .lvl files containing level header, map tile and entity tile information. The class is constructed with a .lvl file path for example "res/lvl/Main Game/0.0.0.lvl" for the title screen.

The loadLevel() function reads the header and tile information from the file using the BufferedReader. The Header class is defined as an inner class of the Level class and stores the header information. If the Header information has been read successfully, a while loop runs until the object count is reached. It reads the tile descriptions using the BufferedReader readLine() function, which are stored as lines of csv (comma separated values).

The TileDescription and EntityDescription classes store map tile and entity information including the name, x, y, tile width, tile height, tile, tileset, type and collision variables.

The toMap() function creates a Map object by looping through the tile and entity descriptions and adding the tiles to the map.

The initDefinitions() function reads the tile and entity descriptions stored in the tileset.ini and sprite.ini initialization files. This includes the name, tile width, tile height and collision variables which can be retrieved using the getTileDescription() and geEntityDescription() functions.

The addTile() function adds a block, terrain or scenery tile to the level. It accepts the parameters index, tileset, x, y, width and height which are used to initialize the tile information with the information returned from the getTileStringDescription() function.

The addEntity() function adds an entity such as a power-up, enemy, growing vine, or end of level sprite. It accepts the parameters index and the x, y coordinates and follows the same process as the addTile() function creating, initializing and adding a EntityDescription to the array list of entities.

The saveLevel() function is passed the file path as a parameter and writes the level header, map tile and entity tile information to a .lvl file.

EditorPanel

The EditorPanel class provides a GUI (Graphical User Interface) for creating new and editing existing levels. It inherits the Jpanel class and can be added to GameFrame.

The mouse wheel is used to scroll through the map and entity tiles which can be added to the map. The middle mouse button changes the editor state from 0-select object, 1-add terrain, 2-add blocks, 3-add scenery, 4-add sprite. Pressing the left mouse button adds the tile to the map, holding the mouse button down and dragging the mouse adds more than one tile to the map at a time. Pressing the right button removes the tile from the map.

The Editor Panel constructor initializes the level header, map, camera, tileset, background music and then the entities using the EntityCollection. The levels are opened for editing using the loadLevel() method which is passed a .lvl file path. The level object is constructed using the path. If the file does not exist a new level is created.

The saveLevel() method saves the level that is being edited in the level editor. The Level class saveLevel() method is passed the file path, world, level, area and the .lvl file extension and saves the level header and map to a file.

The setHeader() method sets the level header and is passed a header object as a parameter. The getHeader() function returns the level header object.

The setTileset() method sets the tilesets for the blocks, terrain and scenery. It accepts three integer numbers specifying the tilesets. The level header tileset information is updated and the tileset is constructed.

The setBackground() method sets the background image layers. It accepts an array of three integer numbers specifying the background layer image file numbers. The level header background information is updated and the background is constructed. The EditorPanel class also includes methods to set the other background properties.

The paintComponent() method draws the level editor background, map tiles, entity tiles, mouse coordinates, background grid and tile descriptions on the screen.

The Background object draw() method draws the background and is passed the graphics context and camera coordinates:

```
mBackground.draw(g, mCamera);
```

The Tileset object drawEditorMap() method draws the blocks, terrain and scenery and the power-up items contained in blocks defined in the level map:

```
mTileset.drawEditorMap(g, mLevelMap, mCamera);
```

The EntityCollection draws entities defined in the level map within the camera area:

```
mEntities.draw(g, mLevelMap, mCamera);
```

The animation state, mouse coordinates, level editor instructions and selected tile information is drawn above the player sprite using the console font.

Load/Save Game

Game

The Game class stores integer values which represent the players saved progress. The world, level and area variables store the level the player has reached. The coins and score variables store the players current coin and score counter. The size stores the current power-up.

SaveFile

The SaveFile class reads and writes a .sav files containing game progress data and stores this in the Game object array. It is declared as a member variable of the Settings class and initialized when the game is started.

The loadFile() function accepts a file number parameter and opens the .sav file path appended with the file number as a RandomAccessFile. The RandomAccessFile is more secure than a normal text file and makes it more difficult for users to cheat in the game. The saveFile() function writes the game progress data to a random access file.

Audio

SoundEffect

The SoundEffect class loads and plays wav sound effects, it is constructed with a file path and includes a constructor override accepting a loop parameter.

The File class opens the wav file at the specified file path which is then stored in an AudioInputStream, opened using the AudioSystem getAudioInputStream() function which accepts a file as a parameter. The getClip() function provided by the AudioSystem obtains an audio clip that can be used for play back. The clip is passed the AudioInputStream and opened for playback and the loop parameter determines how many times to loop the sound.

The SoundEffect class makes it easy to load and play a sound in a single line of code as shown in the following example which plays the jump sound effect:

```java
new SoundEffect("res/sfx/jump.wav").start();
```

Music

The Music class loads an mp3 file and plays it in a loop. Java SE does not directly support the loading and playback of mp3 files but there are many other libraries available. jLayer was chosen as it is an open source mp3 decoder library which is easy to install.

The play() function called in the constructor creates an instance of the JLayerMp3Player class and sets the playback listener:

```java
player = new JLayerMp3Player(is);
player.setPlayBackListener(listener);
```

A threading mechanism is implemented for loading and playing mp3 files concurrently. First the thread is created and then the run() method is invoked which starts the playback of the mp3 using the play() method from JLayerMp3Playback class with the start and end frame parameters.

The JLayerMp3Players also includes the playbackFinished() event listener to detect when the playback finishes.

The Music class includes methods to start, stop and close the playback of the music using the JLayerMp3Player pause() and close() methods and it is easy to load and play an mp3 file using a single line of code:

```
mMusic = new Music(System.getProperty("user.dir")+ "/res/snd/1.mp3");
```

Player and Enemies

Sprite

The Sprite class loads a sprite sheet and an .ini file which specifies sprite initialization information such as animation start and end frame, jump speed, frame width and animation speed. It includes methods to move the sprite, change animation state and draw the sprite on the graphics surface. The Player and Entity classes inherit and extend the functionality of the Sprite class to add specific features.

The constructor accepts the parameters index, direction, x and y. The index specifies the sprite sheet contained in the "res/spr/" folder to use to display the graphics. The direction is a character value either 'l' – left or 'r' – right. The x and y values specify the coordinates of the sprite.

The loadSpriteSheet() method takes a file path parameter and reads the .ini file, sets the default values, frame information and loads the sprite sheet image. The Frame class is declared as an inner class of Sprite and stores the frame information.

The setAnimationState() function accepts a string parameter specifying the animation e.g. "stand" and sets the animation state returning a Boolean true value if successful. The function iterates through the array list containing frame objects, if the animation state value equals the value passed to the function the frame information is retrieved.

The advanceFrame() method advances the animation frame until it reaches the end of the animation and is reset to the start frame.

The draw() method draws the sprite at the x,y coordinates with the current animation state. It accepts the graphics context and camera coordinates as parameters. The animation frame is retrieved using the SpriteSheet getFrame() function which is passed the frame index and direction.

```
g.drawImage(mSpriteSheet.getFrame(mFrameIndex, mDirection), mX -
cam.x, mY - cam.y, null);
```

The Sprite class stores a rotate and flicker variable. If the flicker variable equals true the sprite is drawn every five frames to create a flickering effect.

If rotate is equal true the sprite will rotate around the centre axis. The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates. It is used to apply a rotation transformation to the frame of animation.

The move() method moves the sprite in the direction it is facing with the set speed. The moveLeft(), moveRight(), moveUp**()** and moveDown() methods manually move the sprite in the specified direction and accept a speed parameter which controls the speed of the movement.

The Sprite class also provides methods to set and get the min and max coordinates. The setMinX() and setMaxX() methods restrict the sprite movement in the horizontal axis and the setMinY() and setMaxY() movement in the vertical axis.

The getLeft(), getRight(), getUp() and getDown() functions return the exact coordinates of the bounding box around the sprite and can be used to calculate collisions with other objects.

Control

The Control class stores keyboard and joy pad control variables. The direction held variable stores the amount of time a direction has been held and the jump released variable equals true when the jump button is released.

Player

The Player class draws a user controllable sprite, stores the coin, time-limit and score count variables, and detects collisions between the player, map and entity tiles. It inherits the Sprite class which provides functions to load the sprite sheet, draw, move and animate the sprite.

The loadSpriteSheet() function accepts a file path and loads the sprite sheet for the index and size of the character which is passed to the constructor. The size parameter can take the values 0-small, 1-large, 2-fire.

```
super.loadSpriteSheet("res/chr/" + getIndex() + "/" + size);
```

The Controls class stores the keyboard and joy pad state variables. If a key is pressed or released the GamePanel class invokes the setControls() method which sets the players control variables based on the key event.

The getSize() function returns the player size and the setSize() function sets the size

The addLife() methods adds a life and the removeLife() method removes a life.

The addCoin() adds a coin and plays a sound effect. If the coin count reaches one hundred it adds an extra life and one hundred points.

The drawPoins() method draws points on the screen when the player kills enemies or collects a power-up.

The draw() method draws the player sprite on the graphics surface. It accepts the graphics context and camera parameters which are passed to the Sprite parent class:

```
super.draw(g, cam);
```

The move() method move the player in the specified direction. The player can exit the level through warp pipes or by reaching the end of level castle. If the player is colliding with a bonus or exit coordinate the end state variable is set to 7-exit level or 8-exit bonus.

The setMovement() function moves the player and sets the animation state based on the key control variables. If the left or right direction key variables equal true and the player is not already facing the new direction the player's direction is set and the direction held variable is increased. The players speed is set using the Sprite setSpeed() method based on the amount of time the direction is held. If the player is not jumping, climbing or crouching, the animation state is also set based on the direction held value.

If the player collects the fire power-up pressing the run button will throw fire balls which bounce along the screen and destroy enemies. The fireball is added to the GamePanel entity collection and a sound effect is played.

```
new SoundEffect("res/sfx/fire.wav").start();
if (getDirection() == 'r') {
   Main.mFrame.mGamePanel.addEntity(1, 'r', getCenter().x,
   getCenter().y);
```

If the jump button is pressed the min y height is set to the down y coordinate minus the jump height, the animation state is set to "jump" and a sound effect is played.

```
setJump(true);
setMinY(getDown() - Settings.JumpHeight);
setAnimationState("jump");
new SoundEffect("res/sfx/jump.wav").start();
```

If the jump key is released or the player has reached the jump height the player begins to fall and the jump state and animation state are set to fall:

```
setJump(false);
setAnimationState("fall");
```

The detectEntityCollision() function detects collision between the player and entity objects and returns a Boolean true value if a collision occurs. It is passed an array list containing Entity objects initialized in the GamePanel loadLevel() function. A for loop iterates through the entity objects in the array list and the Rectangle intersect method is used to test for intersections:

```
 if (getBounds().intersects(e.getBounds()))
```

If the player collides with the end of level flagpole, an inner for loop iterates through the entities finding the flag and setting the animation state to "fall". The player's animation is set to "flag" and x,y coordinates aligned to the flag pole. When the players y coordinate reaches the flag pole base the flag state is set to two indicating the player should jump from the flag pole. The flag state variable determines the animation and movement of the player and with the following values 1-flag pole, 2-jump off pole, 3-stand while time is counted, 4-walk to castle.

If the player is invincible and collides with an enemy the enemy is killed and spins off the screen by setting the entity jump and rotate variables.

If the player collides with a mushroom power-up and is the small state the player is enlarged and can be hit by an enemy without losing a life:

```
if (e.getType() == 10)
   if (mSize == 0) setSize(1);
```

The same code is repeated for the fire, star and extra life power-ups. The fire power-up sets the player size to two and gives the player the ability to throw fireballs. The star power-up gives the player temporary invincibility.

There are certain enemies in the game which kill the player if a collision occurs for example the "spikey beetle", these enemies must be avoided or killed with a fireball. The enemyCollision() set the player size if the player collides with an enemy without killing it. If the player has collected the fire power-up the size is reset to large, and then small if hit again. If the player is small and collides with an enemy a life is lost and the level restarts.

If the player collides with an enemy that can be killed by jumping on its head. The following code detects the collision, makes the player bounce off the enemy and sets the entity state to "die":

```
if (e.getType() == 31 || e.getType() == 35) {
   if (getDown() < e.getUp() + 5) {
      new SoundEffect("res/sfx/kill.wav").start();
      entities.get(a).setAnimationState("die");
      bounce();
```

There are also enemies which fly and will be temporarily stunned or turn into a shell when hit by the player, using the same code as above but with the animation state set to "hit". If the player collides with an enemy that is in a shell the animation state is set to "kick" and it is kicked across the screen.

The final type of enemy is the falling enemy which falls quickly when the player is near. If the player is within five pixels of the enemy the animation state is set to "fall".

```
if (e.getType() == 37) {
   if (getRight()+5 >= e.getLeft() || getLeft()-5 <= e.getRight()) {
      entities.get(a).setAnimationState("fall");
```

The detectMapCollisions() function detects collision between the player and map tiles such as blocks and terrain and returns a Boolean true value if a collision occurs. The getTile() function return the tile information and the x, y coordinates. If the map tile positioned below the player has a collision value equal to one the player can stand on the platform and the max y coordinate is set:

```
if ((map.getTile((getRight()-5) / 16, getDown() / 16).collision == 1
|| map.getTile((getLeft()+5) / 16, getDown() / 16).collision == 1))
   setMaxY(getDown() / 16 * 16);
```

If the map tile position above the player when jumping has a collision value equal to one the coordinates of the tile are stored in a x,y Point class and passed to the Map getTile() function which returns the tile information. The Map functions can be used to test for empty, coin, enlarge, fire, invincible, extra life and growing vine blocks.

Entity

The Entity class draws a NPC (non-playable character), such as a power-up or enemy and detects collisions between the entity and map tiles. It inherits the Sprite class which provides functions to load the sprite sheet, draw, move and animate the sprite. In the constructor the speed, jump, min x, min y, start x, start y values of the Entity are initialized depending on the type of Entity. The super keyword is used to invoke the parent class Sprite constructor with the sprite index, direction, x and y coordinates:

```
super(index, 'r', x, y);
```

The loadSpriteSheet() function accepts a file path and loads the sprite sheet for the index of the Entity:

```
super.loadSpriteSheet("res/spr/" + getIndex());
```

The draw() method draws and moves the entity sprite on the graphics surface. It accepts the graphics context and camera coordinate parameters which are passed to the parent Sprite class draw() method.

If the entity is a power-up it will appear to jump from the block it is contained in, when the power-up entity reaches the jump height the jump state is set to false and the speed set to two making the entity move. The star power-up and the flying enemies jump up and down the y axis. When the entity reaches the min y coordinate the jump variable is set to false. If the y coordinate is greater than the max y coordinate the jump variable is set to true. The piranha plant that grows from the warp pipes is animated and when it reaches the end of the animation frame count for the current state a new animation is changed from normal, down to up.

The detectMapCollision() function detects collisions between the entity and map tiles such as blocks and terrain and returns a Boolean true value if a collision occurs.

If the entity is rotating it has been hit by the player with the invincible power-up and the collision detection is skipped.

If the map tile positioned below the entity has a collision value equal to one. The entity can stand on the platform and the max y coordinate is set

```
if (map.getTile(getCenter().x / 16, getDown() / 16).collision == 1) {
   setMaxY(getDown() / 16 * 16);
```

If the entity is a fire ball the min y is set to the down coordinate minus twenty-two. If the entity is a star power-up the min y is set to the down coordinate minus fifty.

If the map tile position above the entity has a collision value equal to one the min y coordinate is set to the current up coordinate:

```
if (map.getTile(getCenter().x / 16, getUp() / 16).collision == 1) {
   setMinY(getUp() / 16 * 16);
```

The left and right collisions prevent the entity walking through blocks, or falling off edges. If the map tile at the left or right and up or down coordinates collision variable is equal to one the entity turns around.

```
if (map.getTile(getLeft() / 16, getUp() / 16).collision == 1 ||
  map.getTile(getLeft() / 16, getCenter().y / 16).collision == 1)
  setDirection('r');
```

Testing

Software testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. Testing helps identify errors and makes sure the program conforms to requirements while giving an indication of performance and quality.

The objective of test case design is to derive a set of tests that have the highest likelihood of uncovering errors in software" with a minimum amount of effort. (Pressman, 2009, p. 508) The following table is an overview of the tests cases conducted on the Java, Android and GP2X versions of the game.

Test No.	Test Description	Expected Result	Outcome
1	Direction arrow key is pressed	Player moves left and right, enters pipe or climbs growing vine	Pass

2	Run key (q or shift) is pressed	Player runs and moves faster while pressing left or right direction key	Pass
3	Jump key (w or space) is pressed	Players jumps and falls when button is released	Pass
4	Players jumps and hits block containing item	The item contained in the block is added to the level map or coin count	Pass
5	Player collects a coin	Player coin count increases and an extra life is awarded at 100 coins	Pass
6	Player collects a power-up	Player receives points and increased power up state	Pass
7	Player collides with level map object	Collision detection prevents the player moving through the object	Pass
8	Player collides with enemy without killing it	Player loses a life or power up state	Pass
9	Player jumps on enemies head	Enemy is removed and score is increased	Pass
10	Player enters a warp pipe on normal level	Player exits level to next level or bonus level	Pass
11	New Game is selected from the menu	The intro is displayed and the game begins	Pass
12	Player completes the last level of the game	End credits displayed and game returns to title screen	Pass

Figure 1.1 Test Case Overview.

This project was thoroughly tested for errors to confirm that it conforms to the requirements and to ensure that it is quality piece of software. Black-box testing was conducted by independent testers. White-box testing was conducted during the implementation phase using test cases, unit testing, integration testing, validation testing and system testing to identify any errors.

One of the unit tests involved testing the load / save game feature. The save file class stores, reads and writes the save game information containing the player's progress in the game. A driver module was constructed which passes a file to the stub save file module which will attempt to open the file and print out the results.

Integration testing was conducted on the game during the implementation stage. The top-down approach was used to test modules in one branch of the game framework; this included the main, game frame, game panel, level, map, player and background classes. This approach tested the functionality, performance and reliability of displaying the user controllable character in a level with a background and level map with collectable objects and power-up objects.

While developing a computer game it is particularly important to pay attention to the usability of the game. This can be achieved by conducting play testing to uncover potential design problems. Heuristic evaluations can be conducted on the game using a set of usability guidelines known as heuristics. (David Pinelle, 2008, p. 1)

The play testing involved getting feedback from five people who had played the game for a ten minute session. The first version of the game did not include touchscreen controls and instead relied on the built in keyboard of the test phone. Most users commented that the game should include touch screen controls to control the player and select menu items. Two users additionally commented that the on-screen controls should move to the position where the screen is touched.

Two of the users found the player moved too fast and the game was difficult to play. One user found there were too many enemies and the levels were too difficult. Three users commented that the loading time while restarting a level was slightly annoying.

All five of the users gave positive feedback and said they would consider downloading and paying for the full game if it included more levels and features.

Evaluation

Functional Requirements

The function requirements describe the function of the software system. The finished game implements all of the functional requirements and does not contain any errors or defects which affect the gameplay. This section will describe how the most important functional requirements were implemented.

There were sixteen functional requirements identified in the analysis section. The first requirement was a user controllable character that could walk, run, crouch and jump depending on the keyboard or touchscreen input. This feature was implemented using key and touch event listeners to receive user input. The key and touch events were stored as separate variables in the Control class which made it handle multiple keys pressed at the same time. The Player class sets the direction, speed and animation of the player and the Sprite class moves the sprite along the x, y axis.

The second functional requirement was enemies which the player must avoid or kill to gain points. The finished game includes nine unique types of enemy including walking, flying, turtle shells, bullets and piranha plants. The Entity class handles the loading and drawing of the enemies and power-up objects.

The fourth functional requirement was collision detection between the player and blocks, terrain and entities. The collision detection uses a bounding box intersection test and then takes the appropriate action depending on the type of collision

The sixth functional requirement was a level map which is read from a file. The Level class opens and saves .lvl files containing header, map tile and entity information. The loadLevel() function reads the level header, map tile and entity data.. The map tile and entity data is then used to construct a 2D array representing the level map.

The seventh functional requirement was a game environment with a background, tile set and music which suits the theme of the level. The game features six unique worlds including mountains, beach, desert, city scape, Aztec and space. The background and tileset images were found on the DeviantArt PixeJoint and MFGG websites. The t*i*tlesets include blocks, question blocks, warp pipes and background scenery.

The eighth functional requirement was a moving camera view which follows the player around the level map. The camera is implemented using a Point variable which stores the x, y coordinates of the camera. The cameras position is updated in the GamePanel paintComponent() method using the players x, y position

The tenth functional requirement was the level editor which can be used to create new or edit existing levels. The level editor is user friendly and is easy to use. It loads and saves level data using the Level class. The mouse can be used to select, move and add tiles to the level map, the arrow keys move the camera around the level map.

The eleventh functional requirement was sound effects played when the player performs an action. The game has full support for wav sound effects played simultaneously in a thread using the SoundEffect class. The sound effects include jumping, hitting blocks, collecting coins, killing enemies, entering warp pipes, completing a level, collecting power ups, throwing fireballs and losing a life.

The twelfth functional requirement was introduction and ending sequence displayed at the beginning and end of the game. The intro sequence presents the main character running through a level and encountering the last boss. The intro brings a story element and meaningful play to the game and is a reason for the player to complete the levels. The ending sequence was not implemented due to time constraints but instead the end credits are displayed and the game returns to the title screen.

The thirteenth functional requirement was a time limit where the player must try to complete the level before the time runs out. The time limit is displayed at the top of the screen during the game, if the time limit reaches zero the player loses a life.

The fourteenth functional requirement was a score system where the player must collect points to show how good they are at the game. The points can be accumulated for killing enemies, collecting power-ups, collecting coins and for the remaining time after completing a level. The points system could be improved by storing the high scores in an online database or using an online service such as OpenFeint.

The fifteenth functional requirement was load and save game features allowing the player to save and resume their progress in the game. The save game features allows the player to save their progress and continue the game at a later time. It supports up to four separate save games across multiple game folders.

Non-Functional Requirements

Non-functional requirements are added features which are not vital to the project success; these include cost, reliability or overall improvements. This section will describe how the most important non-functional requirements were implemented.

There were fourteen non-functional requirements identified in the analysis section. The first non-functional requirement was accessibility with the game content complying with any age, accessibility or cultural constraints. The games content has been kept within the guidelines for an ESRB E or PEGI 3 rating as these categories receive the highest percentage of sales according to statistics. The game should appeal to children as it has a simple storyline, is fun to play and uses colourful cartoon style graphics. There are no cultural issues with the storyline or the characters from the game. If the game becomes popular in non-English speaking countries the text displayed in the games menus and options screen game could require translation.

The second non-functional requirement was that a regular backup of the source code would be kept on an external media. Backups of the source code were regularly stored on CD-R and USB flash drives in rar archives named with the project title and backup date. The backups were useful when changes to the source code resulted in the game not functioning correctly and needed to be reversed.

The third non-functional requirement was legal compliance with SDK providers, graphics and sound resource providers license agreements. The project complies with the Java2 SE, Android and GP2X C++ and SDL licensing agreements.

The fourth non-functional requirement was documentation of the classes, variables and methods. The Java SE, Java Applet and Android versions of the game have been documented using the Javadoc tool. The Javadoc tool creates HTML documentation using specially formatted comments proceeding class, field and method declarations. The C++ version of the game was documented using Doxygen which produces HTML documentation using similar code commenting syntax to Javadoc.

The fifth non-functional requirement was extensibility where the system takes into account future growth and can be expanded without having to make major changes to the code. The level editor provides a method to expand the system by creating new levels in separate folders to the main games. The level data is stored in the "res/lvl" folder followed by the game folder name. The game supports modding and the resources can be edited or replaced to change the look and feel.

The seventh non-functional requirement was that the source code should be released under an open source license agreement. This requirement has not been met as it was decided that it would make more commercial sense to keep the game closed source.

The eighth non-functional requirement was platform compatibility and the game should be fully playable on the Windows, Linux, Android and GP2X platforms. This requirement has only been partially met with the game tested and operating correctly under Windows, Linux and Android. The Android version has been tested on the Samsung Galaxy 551 and S2 smartphone and operates correctly with no errors or defects which affect gameplay. The GP2X version has been tested in Windows but will require slight changes to the drawing functions for it to work on the various screen sizes of the different GP2X devices.

The ninth non-functional requirement was that the game should be competitively priced or available to download for free. The game was released as a demo of the first four levels on Google play and includes advertisements to generate revenue. If the game is a success it will be possible to charge a small amount such as £0.59 for a full version which has more levels does not include advertisements.

The fourteenth non-functional requirement was acceptable game performance which meant the player would not experience any slowdown or performance issues during game play. The results of testing the game on Windows and Android devices show that it plays at an acceptable speed under most conditions.

The Android version performed well in basic level map tests. Level maps which contained more than forty tiles simultaneously displayed on the screen began to slow down on slower phones and made the game more difficult to play as the controls were unresponsive. The level loading time for the Android version is slower than the Java SE version this is due to the processor speed of the mobile devices.

The C++ version of the game has been tested in Windows. It performs faster than the Java SE version. This was expected as C++ applications generally run faster than Java due to it using a Virtual Machine. The SDL library provided an easy to use fast method of drawing images on a hardware accelerated drawing surface.

Project Review and Conclusion

This project was a test of my skills and knowledge of computer games programming using Java and C++. The finished game implements nearly all of the requirements and is an enjoyable side scrolling platform game with lasting entertainment value.

During the implementation of the project there were problems involving graphics and sound resources which were not related to the actual programming of the source code. It was difficult to find graphics and sounds distributed under free licenses such as the Creative Commons. To overcome this problem I searched the internet using Google for free graphics and sounds. Using search terms such as “mario sprites”, “free pixel art” and “digital art” returned a number of website such as Deviant Art and Pixel Joint offering custom drawn copyright free graphics.

Midi music was included in the first version of the game as it is already supported by the Java Audio API and does not require any additional libraries. The problem with Midi music is it only supports a small range of emulated instruments which are used to compose the sound file. In the final version of the game I decided to use mp3 music as it is higher quality sound format and would improve the overall presentation. The Incompetech website found using the search term “royalty free music” provided a library of free mp3 music composed by a musician called Kevin Mcleod.

The sound effects were difficult to acquire and the best solution was to use and application called SFXR found by searching free audio forum discussions. It programmatically produces unique wav sound effects similar to those found in the Mario Bros. games, which are copyright free and could be included in the game.

The Android platform runs on multiple hardware platforms such as mobile phones and tablet computers. This made it difficult to test the game at different screen resolutions to avoid blurred or pixelated graphics due to stretching the screen size to a higher resolution than it was intended for.

Throughout the implementation of the project there were various issues relating to the programming of the game on the different hardware platforms. The problems were generally resolved by researching functions and source code examples on the internet using forums, newsgroups and websites.

For the Android version of the game there was a choice of different methods to display the graphics including the Canvas, OpenGL ES and RenderScript. The canvas method is a 2D custom graphics library with a software renderer. It is suitable for creating games with animation but is not as fast as the hardware solutions. RenderScript is designed for high performance mathematical calculations and low level 3D and was not suitable for this application. OpenGL ES is a subset of the industry standard graphics library OpenGL. It was selected for as the graphics library as it provides cross platform 2D and 3D graphics for embedded systems used in mobile phones. The OpenGL ES method of drawing graphics involves applying a texture containing the image to a quad constructed of vertices.

The Android version of the game would not run on the software emulator as it does not support OpenGL ES hardware acceleration. This meant the game had to be copied to and loaded on an actual Android phone before it could be tested which took up valuable development time.

The drawing speed of the Android version of the game was significantly slower than the Java SE version of the game. This meant the amount of movement applied to the player and entities sprites when moving left, right, jumping or falling had to be increased to compensate for the reduced frame rate.

The font used to draw the menus in the game is not a standard true type font type and instead the font is loaded from a sprite sheet. This required creating a class to load the font and draw the text and provided complete control over the display of the font including the character spacing. It also made it easier to display the font on other hardware platforms in the same way using the existing functions.

During the course of the project I learnt a lot about the SDK's and libraries which are used to develop computer games. It was a challenge to learn and use new API's and libraries during the implementation. The Java version taught me about graphics, sounds, threads and applet technology. The Android and GP2X versions of the game gave me the chance to look at mobile game development. The Android version taught me about the OpenGL ES library and these skills could be transferred to other projects for the iPhone or desktop 3D applications. The C++ version taught me about the SDL library and the differences in program syntax between Java and C++.

Future Work

Market research has shown that players can become bored and lose interest in games if there are not enough levels. If there was more time available it would be interesting to do further research into automatic level generation. This would provide a method of generating a large number of unique levels to compete with other commercial games.

The game could be improved by including a second character and even a two-player mode with each player competing for a high score. The second character could be female to appeal to the ever growing market of female social gamers.

The Android version suffers from some loading time delays at the beginning of the game and while loading levels. The texture loading functions could be optimized by using compressed texture formats to reduce the loading times.

The high score feature in the Java applet version does not save the current high score and it is reset every time the page is reloaded. This could be improved by storing and retrieving the high score information in an online database.

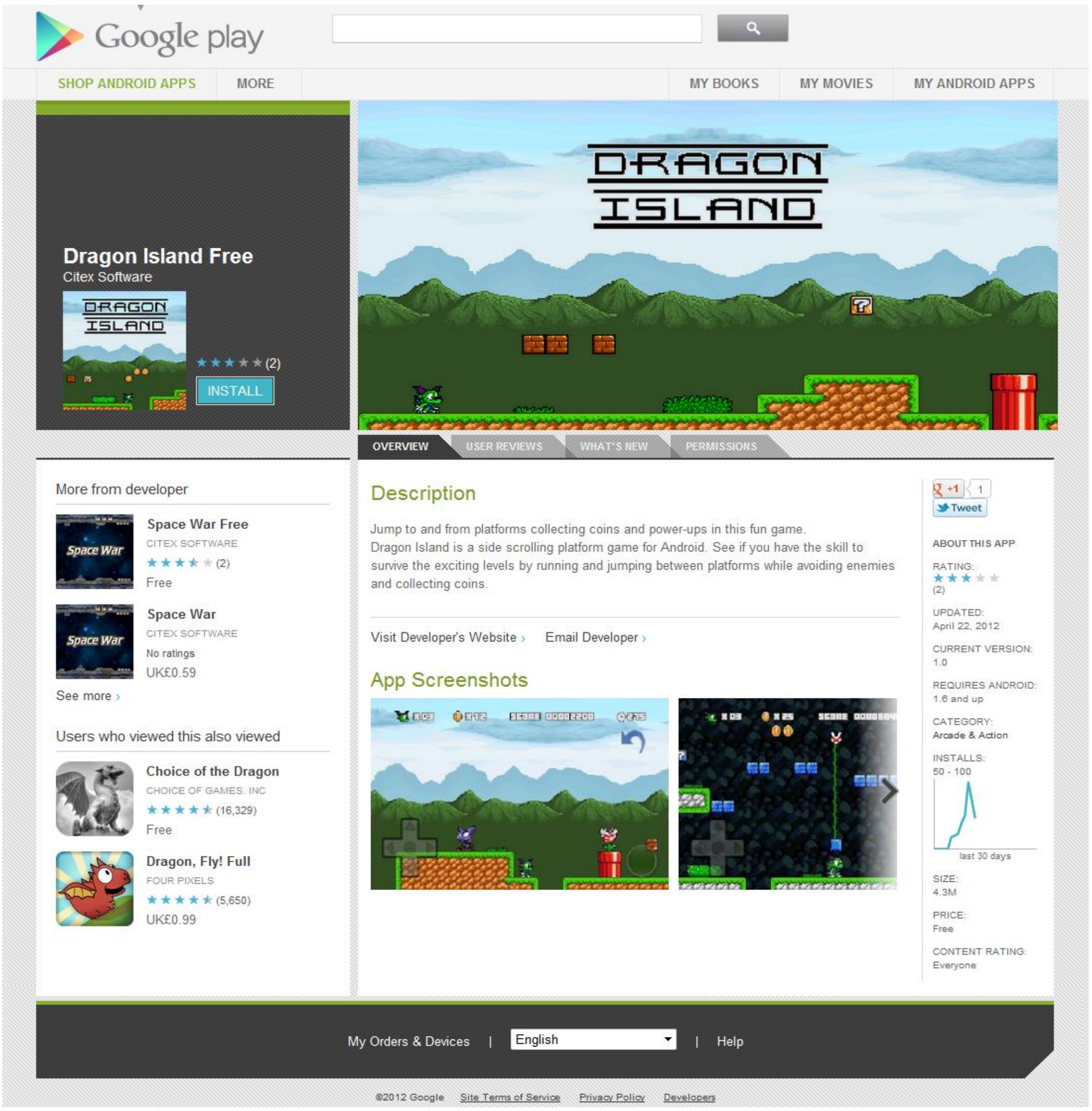
Social networking is popular with mobile games. The game could be improved by integrating links to Facebook and Twitter. There are a number of services which can be used to connect to social gaming networks such as Papaya, PlayPhone and OpenFeint. OpenFeint is a free service available for the Android platform and provides a global online high score leader board, unlockable achievements and the social networking aspect of meeting other players who enjoy similar games.

The Android version of the game uses touchscreen controls. The d-pad and buttons are overlaid on the game window in a static position. This method works well but could be easily improved by implementing a d-pad which moves to the position where the player touches the screen. This would avoid the problem of the d-pad covering the player and obscuring the view of the level. In addition haptic control technology could be used to alert the player by vibrating the phone when they collide with an enemy.

The AdMob service was used to display banner advertisements in the free version of the game. This service only displays ads from one ad provider network. It could be improved by using the free AdWhirl service which connects to multiple ad networks and selects the best performing network for the application to maximize the revenue.

Appendix

Dragon Island on Google Play



Google play

SHOP ANDROID APPS | MORE | MY BOOKS | MY MOVIES | MY ANDROID APPS

Dragon Island Free
Citex Software
★★★★★ (2)
INSTALL

OVERVIEW | USER REVIEWS | WHAT'S NEW | PERMISSIONS

More from developer

Space War Free
CITEX SOFTWARE
★★★★★ (2)
Free

Space War
CITEX SOFTWARE
No ratings
UK£0.59

See more ›

Users who viewed this also viewed

Choice of the Dragon
CHOICE OF GAMES. INC
★★★★★ (16,329)
Free

Dragon, Fly! Full
FOUR PIXELS
★★★★★ (5,650)
UK£0.99

Description

Jump to and from platforms collecting coins and power-ups in this fun game.
Dragon Island is a side scrolling platform game for Android. See if you have the skill to survive the exciting levels by running and jumping between platforms while avoiding enemies and collecting coins.

Visit Developer's Website › Email Developer ›

App Screenshots

+1 1
Tweet

ABOUT THIS APP

RATING:
★★★★★
(2)

UPDATED:
April 22, 2012

CURRENT VERSION:
1.0

REQUIRES ANDROID:
1.6 and up

CATEGORY:
Arcade & Action

INSTALLS:
50 - 100
last 30 days

SIZE:
4.3M

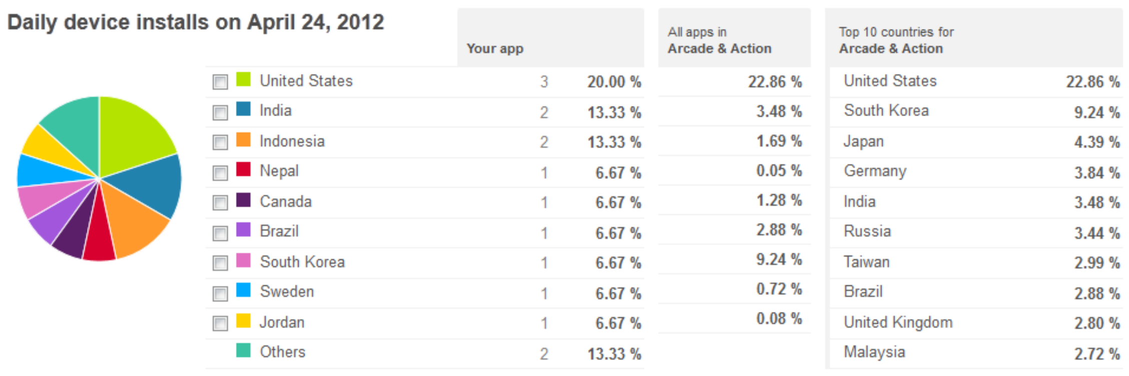
PRICE:
Free

CONTENT RATING:
Everyone

My Orders & Devices | English | Help

©2012 Google Site Terms of Service Privacy Policy Developers

Figure 4.1 Screenshot of Dragon Island on Google Play



Daily device installs on April 24, 2012

	Your app		All apps in Arcade & Action	Top 10 countries for Arcade & Action	
🟩 United States	3	20.00 %	22.86 %	United States	22.86 %
🟦 India	2	13.33 %	3.48 %	South Korea	9.24 %
🟧 Indonesia	2	13.33 %	1.69 %	Japan	4.39 %
🟥 Nepal	1	6.67 %	0.05 %	Germany	3.84 %
🟪 Canada	1	6.67 %	1.28 %	India	3.48 %
🟪 Brazil	1	6.67 %	2.88 %	Russia	3.44 %
🟪 South Korea	1	6.67 %	9.24 %	Taiwan	2.99 %
🟦 Sweden	1	6.67 %	0.72 %	Brazil	2.88 %
🟨 Jordan	1	6.67 %	0.08 %	United Kingdom	2.80 %
🟩 Others	2	13.33 %		Malaysia	2.72 %

Figure 4.2 Daily Device Installs for Dragon Island

Log File Form

Meeting No: 1	**Date:** 5/10/2011	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the first draft of the project proposal. This included improving the Gantt Chart and removing the techniques explained section from the report.		
Action/Outcome: Remove Techniques Explained from appendix Remove Project Proposal task detail from Gantt chart Email Ruediger regarding left margin and font size		
Next Meeting:	PRSB116	
Date: 12/10/2011		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 2	**Date:** 12/10/2011	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the feedback and changes which could be made to the first draft of the project proposal. This included improving the Project Scope section to make clear what the project will do and won't do.		
Action/Outcome: Make changes to report based on comments Arranged to borrow Android smartphone to begin implementation Start work on first draft of the State-of-art Review		
Next Meeting:	PRSB116	
Date: 26/10/2011		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 3	**Date:** 26/10/2011	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the final draft of the project proposal. This included editing the Project Objectives section to clarify personal objectives.		
Action/Outcome: Make changes to project objectives to clarify personal objectives Remove development software from the first draft of the SOAR Search Google scholar for articles containing the phrase "Super Mario Games"		
Next Meeting:	PRSB116	
Date: 16/11/2011		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 4	**Date:** 16/11/2011	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the progress of the state-of-the-art-review. This included agreeing on the contents page of the report and discussing the articles relating to "Super Mario Games" found on Google scholar.		
Action/Outcome: Write about PlayStation Move Me controller and haptic / force feedback Remove phrase "game learning" and replace with "autonomous agents" Add bibliography references for diagrams in appendix		
Next Meeting:	PRSB116	
Date: 7/12/2011		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 5	**Date:** 7/12/2011	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the new topics added to the state-of-the-art-review these included optimizing level design for player experience, autonomous agents and behavioural and motor skills.		
Action/Outcome: Access articles on IEEE and other websites using Athens login Write a critique for each topic		
Next Meeting:	PRSB116	
Date: 13/1/2012		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 6	**Date:** 13/1/2012	**Time:** 11:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the new topics added to the state-of-the-art-review these included portrayal of gender in computer games and game story and narrative. It was also a chance to discuss what else needed to be included in the report.		
Action/Outcome: Write about what has been learnt from the research and how it can be applied to the project Too much background with not enough information on how to apply it. Research the fact that the fastest growing sector of video game purchases is surprisingly older females. Write critiques for all of the remaining topics.		
Next Meeting:	PRSB116	
Date: 25/1/2012		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 7	**Date:** 25/1/2012	**Time:** 11:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the final draft of the state-of-the-art-review and identify any changes which needed to be made before submitting it for assessment.		
Action/Outcome: Make slight changes to grammar of Autonomous Agents topic Research Sony commercial game checklist for final report Write about how the game is improved by adapting it for female players		
Next Meeting:	PRSB116	
Date: 15/2/2012		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 8	**Date:** 15/2/2012	**Time:** 11:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to present the 1st prototype demonstration. This included demonstrating the Java SE, Java Applet and Android versions of the game. The features of the game were explained and future changes were discussed.		
Action/Outcome: Modify Android version to use frame rate independent movement Create A1 poster template in Photoshop		
Next Meeting:	PRSB116	
Date: 29/2/2012		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 9	**Date:** 29/2/2012	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the progress of the project report and the poster design presentation design. It was also a chance to demonstrate the online mobile gaming questionnaire for the analysis stage of the report.		
Action/Outcome: Modify the questionnaire How much would you pay for a mobile game? Include Free option and increase price range up to £5.00 Write more for the conclusion of the report		
Next Meeting:	PRSB116	
Date: 14/3/2012		**Time:** 11:00am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 10	**Date:** 14/3/2012	**Time:** 11:00am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to demonstrate the Space War game which was recently released on Google Play. This game includes many features such as touch screen, advertising, application signing and publishing on Google play.		
Action/Outcome: Change font size of report to 12pt Times New Roman and set margins Update bibliography and contents page Consider releasing free demo level of Space War with advertising		
Next Meeting:	PRSB116	
Date: 27/3/2012		**Time:** 10:30am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 11	**Date**: 27/3/2012	**Time:** 10:30am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss the draft version of the Project Report and how it could be shortened and improved.		
Action/Outcome: Add reference after every quote and update project proposal and soar Shorten the implementation to under 20 pages Add page numbers		
Next Meeting:	PRSB116	
Date: 3/4/2012		**Time:** 11:00am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Meeting No: 12	**Date:** 3/4/2012	**Time:** 11:00am
Attendance: Dr Darrel Greenhill, Lawrence Schmid		
Agenda: The agenda of this meeting was to discuss how to improve the Project Report.		
Action/Outcome: Write more about future work in conclusion Talk about usability testing Redo graphs in Excel Shorten appendix to under 40 pages		
Next Meeting:	PRSB116	
Date: 17/4/2012		**Time:** 11:00am
Supervisor **Name:** Dr Darrel Greenhill	**Signature:**	

Bibliography

Boduch, R., 2010. *Haptic technology, ELE 282.* [Online]
Available at: http://www.ele.uri.edu/courses/ele282/F06/Rebecca_2.pdf
[Accessed 8 12 2011].

Brackeen, D. B. B. &. V. L., 2003. *Developing Games in Java.* s.l.:New Riders Publishing (Pearson Education).

Bredemeyer, R. M. a. D., 2011. *Functional Requirements and Use Cases.* [Online]
Available at: https://www.bredemeyer.com/pdf_files/functreq.pdf
[Accessed 17 3 2012].

Brian Tanner, A. W., 2009. *RL-Glue: Language-Independent Software for Reinforcement-Learning Experiments.* [Online]
Available at: https://www.jmlr.org/papers/volume10/tanner09a/tanner09a.pdf
[Accessed 18 12 2011].

C. Shawn Green, D. B., 2003. *Action video game modifies visual selective attention.* [Online]
Available at: https://www.nature.com/articles/nature01647
[Accessed 4 1 2012].

Chandler, D., 2000. *Semiotics for Beginners.* s.l.:University of Wales.

Competition, T. 2. M. A., 2009. *Julian Togelius, S. Karakovskiy, Robin Baumgarten.* [Online]
Available at:
https://www.researchgate.net/publication/224177833_The_2009_Mario_AI_Competition
[Accessed 22 12 2011].

Crawford, C., 2000. *The Art of Computer Game Design.* [Online]
Available at:
https://www.digitpress.com/library/books/book_art_of_computer_game_design.pdf
[Accessed 17 1 2012].

Cummings, A. H., 2007. *The Evolution of Game Controllers and Control Schemesand their Effect on their games.* [Online]
Available at:
https://www.academia.edu/828181/The_evolution_of_game_controllers_and_control_schemes_and_their_effect_on_their_games
[Accessed 26 11 2011].

Darby, J., 2007. *Awesome Game Creation: No Programming Required.* s.l.:Cengage Learning, Inc.

David Pinelle, N. W., 2008. *Heuristic evaluation for games: usability principles for video game design.* [Online]
Available at: https://www.semanticscholar.org/paper/Heuristic-evaluation-for-games%3A-usability-for-video-Pinelle-Wong/60b1063b7bec0a5a90213fcb4b17e0e855797a1c
[Accessed 14 11 2011].

Davison, A., 2005. *Killer Game Programming in Java.* s.l.:O'Reilly Media, Inc..

Dmitri Williams, N. M. M. C. J. D. I., 2009. *The virtual census: representations of gender, race and age in video games.* [Online]
Available at: https://journals.sagepub.com/doi/10.1177/1461444809105354
[Accessed 9 1 2012].

ESRB, 2009. *Video Game Statistics. (2009). ESRB..* [Online]
Available at: http://www.esrb.org/about/images/vidGames04.png

Farago, P., 2011. *Apple & Google Capture U.S. Video Game Market Share in 2010.* [Online]
Available at: https://www.flurry.com/blog/apple-and-google-capture-u-s-video-game-market/

Feldman, A., 2005. *Designing Arcade Computer Game Graphics (Wordware Game Developer's Library).* 1 ed. s.l.:Wordware Publishing Inc..

Games, P., 2011. *2011 PopCap Games Mobile Phone Gaming Research.* [Online]
Available at: https://www.scribd.com/document/75749987/2011-PopCap-Mobile-Phone-Games-Presentation

Gibson, E. (., 1969. *Principles of perceptual learning and development.* s.l.:Century psychology series.

Gillian Smith, M. T. J. W. M. M., 2009. *Rhythm-based level generation for 2D platformers.* [Online]
Available at: https://dl.acm.org/doi/10.1145/1536513.1536548
[Accessed 24 11 2011].

Goldman, H. B. a. L., 2011. *The Truth abpit Smartphones: Our Exclusive Survey On iPhone vs Android.* [Online]
Available at: https://www.businessinsider.com/smartphone-survey-results-2011-4?op=1&r=US&IR=T
[Accessed 19 3 2012].

Google, 2009. *Introducing GLSurfaceView.* [Online]
Available at: https://android-developers.googleblog.com/2009/04/introducing-glsurfaceview.html
[Accessed 28 3 2012].

Google, 2011. *Application fundamentals.* [Online]
Available at: https://developer.android.com/guide/components/fundamentals
[Accessed 28 3 2012].

Grimes, S. M., 2003. *"You Shoot Like A Girl!": The Female Protagonist in Action-Adventure Video Games.* [Online]
Available at: http://www.digra.org/wp-content/uploads/digital-library/05150.01496.pdf
[Accessed 7 1 12].

Group, K., 2011. *OpenGL ES Overview.* [Online]
Available at: https://www.khronos.org/opengles/

Haughey, D., 2011. *MoSCoW Method.* [Online]
Available at: https://www.projectsmart.co.uk/tools/moscow-method.php#google_vignette
[Accessed 17 3 2012].

HNYD, 2011. *Replica Island.* [Online]
Available at: https://www.amazon.com.au/HNYD-Replica-Island/dp/B00AP20ZAI
[Accessed 17 3 2012].

Information, P. E. G., 2001. *Pegi Public Site.* [Online]
Available at: https://pegi.info/

Ingram, M., 2010. *Average Social Gamer Is a 43-Year-Old Woman.* [Online]
Available at:
https://archive.nytimes.com/www.nytimes.com/external/gigaom/2010/02/17/17gigaom-average-social-gamer-is-a-43-year-old-woman-19735.html
[Accessed 15 1 2012].

John Feil, M. S., 2005. *Beginning Game Level Design.* s.l.:Thomson Course Technology.

Justine Cassell, H. J., 1998. *From Barbie® to Mortal Kombat: Gender and Computer Games.* 1 ed. s.l.:The MIT Press.

Kelly, E., 2011. *Tactile technology provides reality boost for flight simulator.* [Online]
Available at: https://www.flightglobal.com/tactile-technology-provides-reality-boost-for-flight-simulator/103333.article
[Accessed 12 8 2011].

Knowledge, P. M., 2011. *Strengths, Weaknesses, Opportunities and Threats (SWOT) Analysis.* [Online]
Available at: https://project-management-knowledge.com/definitions/s/strengths-weaknesses-opportunities-and-threats-swot-analysis/
[Accessed 18 10 2010].

Koprowski, G. J., 2006. *Interest in Touch-Screen Technology for Mobile Phones Growing.* [Online]
Available at: https://www.technewsworld.com/story/interest-in-touch-screen-technology-for-mobile-phones-growing-51710.html
[Accessed 8 12 2011].

Koster, R., 2013. *Theory of Fun for Game Design.* s.l.:O'Reilly.

Kücklich, J., 2011. *Insert Credit to Continue. Narrative and Commodity Form in Video Games.* [Online]
Available at:
https://www.researchgate.net/publication/266582375_Insert_Credit_to_Continue_Narrativ

e_and_Commodity_Form_in_Video_Games
[Accessed 28 1 2012].

Layer, S. D. M., 2011. *SDL.* [Online]
Available at: https://www.libsdl.org/

Løvlie, A. S., 2005. *End of story? Quest, narrative and enactment in computer games.* [Online]
Available at:
https://www.researchgate.net/publication/221217561_End_of_story_Quest_narrative_and_enactment_in_computer_games
[Accessed 22 1 2012].

Lowensohn, J., 2011. *Study: Prices of top iPhone games keep dropping.* [Online]
Available at: https://www.cnet.com/culture/study-prices-of-top-iphone-games-keep-dropping/?adTargeting_campaign=msfttagpage
[Accessed 17 3 2012].

MarketingProfs, 2010. *Mobile App Users Click on Ads Mostly by Mistake.* [Online]
Available at: https://www.marketingprofs.com/charts/2011/4365/mobile-app-users-click-on-ads-mostly-by-mistake
[Accessed 19 3 2012].

Mateas, K. C. a. M., 2006. *Procedural Level Design for Platform Games.* [Online]
Available at: https://ojs.aaai.org/index.php/AIIDE/article/view/18755
[Accessed 11 11 2011].

Matt Benatan, I. S. a. K. N., 2011. *Mobile Motion: Multimodal Device.* [Online]
Available at: https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/EVA2011.42
[Accessed 28 11 2011].

Mawhorter, P. & Mateas, M., 2010. *Procedural level generation using occupancy-regulated extension.* [Online]
Available at: https://ieeexplore.ieee.org/document/5593333
[Accessed 24 11 2011].

Melinda Burgess, S. P. S. S. R. B., 2007. *Sex, Lies, and Video Games: The Portrayal of Male and Female Characters on Video Game Covers.* [Online]
Available at:
https://www.researchgate.net/publication/226396946_Sex_Lies_and_Video_Games_The_Portrayal_of_Male_and_Female_Characters_on_Video_Game_Covers
[Accessed 9 1 2012].

Nathan Sorenson, P. P., 2010. *The Evolution of Fun: Automatic Level Design Through Challenge Modeling.* [Online]
Available at: https://philippepasquier.com/publications
[Accessed 11 11 2011].

NO-Body-The-Dragon, 2012. *Deviant Art - NO-Body-The-Dragon.* [Online]
Available at: https://www.deviantart.com/no-body-the-dragon
[Accessed 16 3 2012].

Noor Shaker, G. Y. a. J. T., 2010. *Towards Automatic Personalized Content Generation for Platform Games.* [Online]
Available at: https://ojs.aaai.org/index.php/AIIDE/article/view/12399
[Accessed 11 11 2011].

Oracle, 2011. *Java SE 6 Documentation.* [Online]
Available at: https://docs.oracle.com/javase/6/docs/index.html

Pankrashchenko, L., 2010. s.l.: Kingston University.

Paradigm, V., 2011. *What is Unified Modeling Language (UML)?.* [Online]
Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Pardew, L., 2004. *Beginning Illustration and Storyboarding for Games.* s.l.:Premier Press Game Development.

Patel, A., 2007. *J2ME memory game for mobile devices,* s.l.: Kingston University.

Patel, V., 2011. *Java Virtual Machine, An inside story!!.* [Online]
Available at: https://www.viralpatel.net/java-virtual-machine-an-inside-story/

PJ Cabrera, P. B. I. M. B. S. E. W. S. P. S. M. R. S., 2010. *Beginning iPhone Games Development.* s.l.:Books for Professionals by Professionals.

Pressman, R., 2009. *Software Engineering: A Practitioner's Approach.* 7 ed. s.l.:McGraw-Hill Education / Asia.

Project-Management-Knowledge, 2010. *Project Management Knowledge - SWOT Analysis.* [Online]
Available at: http://project-management-knowledge.com/definitions/s/strengths-weaknesses-opportunities-and-threats-swot-analysis
[Accessed 18 10 10].

Richards, K., 2011. *Agile project management:Integrating DSDM into an existing PRINCE2® environment.* [Online]
Available at: https://docplayer.net/16692580-Agile-project-management-integrating-dsdm-into-an-existing-prince2-environment.html
[Accessed 16 3 2012].

Shiwali Mohan, J. E. L., 2009. *Learning to Play Mario.* [Online]
Available at:
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3047abd4b5ed5dab914162ef7b200194c233301c
[Accessed 18 12 2011].

Stellman, A., 2010. *Nonfunctional Requirements Q&A.* [Online]
Available at: https://www.stellman-greene.com/2010/02/17/nonfunctional-requirements-qa/
[Accessed 17 3 2012].

Thompson, R., 2011. *Stakeholder Analysis.* [Online]
Available at: https://www.mindtools.com/aol0rms/stakeholder-analysis
[Accessed 24 3 2012].

Visible-Goal, 2008. *MS Excel format Gantt chart.* [Online]
Available at: http://www.visiblegoal.com/VG%20Project%20Gantt%202008.xls

Wikipedia, 2010. *DSDM (Dynamic Systems Development Method).* [Online]
Available at: http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method
[Accessed 22 10 10].

Wikipedia, 2011. *Entertainment Software Rating Board.* [Online]
Available at: https://en.wikipedia.org/wiki/Entertainment_Software_Rating_Board
[Accessed 10 11 2011].

Wikipedia, 2011. *Parallax scrolling.* [Online]
Available at: https://en.wikipedia.org/wiki/Parallax_scrolling
[Accessed 31 3 2012].

Wikipedia, 2011. *PEGI.* [Online]
Available at: https://en.wikipedia.org/wiki/PEGI
[Accessed 16 3 2012].

Zimmerman, K. S. T. a. E., 2003. *Rules of Play Game Design Fundamentals.* s.l.:The MIT Press.