

University of Glasgow | School of Computing Science

A Platformer Game in Flash
Self Defined Project

Matthew Roszak

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — May 15, 2012

Abstract

The online game industry has been growing rapidly in the last few years, and many new technologies are emerging to support the demand for web applications.

This project involves developing an action-platformer game for the Flash platform. The user controls a character who must progress through levels, while avoiding obstacles, collecting items and fighting many types of enemies. Role-playing game elements such as equipment and upgrades support user customization and varied battle strategies.

Developing a successful Flash game involves several factors. Flash games must be simple to get into, yet rewarding to play in both short and long sessions. Flash games must be accessible to a wide audience; both in terms of user preferences and technical limitations. Finally, Flash games must offer something unique to stand out from the crowd, in a market where hundreds of free games are published every month. This report covers the research, design and implementation done to achieve these requirements, in terms of game mechanics, interface, level design, visual design, accessibility options, and replay value.

Technical challenges include building, testing and optimizing a game engine and interface from scratch, balancing the game mechanics and difficulty, and structuring the whole development process in a way that enables easy creation of new content.

In addition to game design and development, this project also deals with the business aspects of developing online games; how games generate revenue, how they are marketed and distributed, and developing trends in the consumer market.

Project success in different areas has been evaluated through usability studies, user ratings and reviews, and vast quantities of usage and distribution statistics. Overall, the project has been a success in terms of user reception and generated revenue, and the final section of this report includes plans for a second game, utilizing and building upon the same game engine and mechanics.

Contents

1 Glossary **6**

1.1 Video Game Terminology 6

1.1.1 Game Genres 6

1.1.2 General Terms 6

1.1.3 RPG Terms 7

1.2 Flash Terminology 7

1.2.1 MovieClips and the Animation Metaphor 8

1.2.2 Tools and Applications 8

1.2.3 Filetypes 9

2 Introduction **10**

2.1 Report Overview 10

2.2 Motivation 10

2.3 Goals and Challenges 11

2.4 Report Contents 11

3 Analysis and Requirements **13**

3.1 Introduction 13

3.2 Proposed Game Overview 13

3.3 Non-Functional Requirements 14

3.4 Functional Requirements 15
3.4.1 Game Controls and Mechanics 16
3.4.2 Level Design 17
3.4.3 Menus and GUI 17
3.4.4 Miscellaneous 18
3.5 Game Design Heuristics 19
3.6 Development Approach 21
3.7 Why use Flash? 22
3.8 Alternatives to Flash 23
3.8.1 HTML5 23
3.8.2 Unity 24
3.8.3 Java 24
3.9 Market Background 25
3.10 Personal Background 27
3.11 Measuring Project Success 28

4 Design and Architecture **30**

4.1 Introduction 30
4.2 Structure Overview 30
4.3 Instance Classes 31
4.4 Static Classes 40
4.5 Graphical User Interface 46
4.6 Gameplay Visuals and Audio 50
4.7 Game Design and Creating Game Content 53
4.7.1 Level Design 53
4.7.2 Enemy Design 55

4.7.3 Equipment and Spell Design . . . 56
4.7.4 Tutorial Level . . . 57
4.8 External APIs . . . 58

5 Implementation and Testing **60**

5.1 Introduction . . . 60
5.2 Development Tools and Resources . . . 61
5.2.1 Graphics and Animation . . . 61
5.2.2 Programming . . . 61
5.2.3 Recycled Assets . . . 62
5.2.4 Sound Effects and Music . . . 62
5.3 Final Product Overview . . . 62
5.3.1 Menu Screens . . . 63
5.3.2 Gameplay . . . 68
5.4 Level Editting . . . 70
5.5 Collision Detection . . . 73
5.6 Testing Strategy . . . 75
5.7 Play Testing . . . 75
5.8 Demos and Prototypes . . . 77
5.9 Functional Testing . . . 77
5.10 Stress Testing and Resource Usage . . . 79
5.11 Optimizing Performance . . . 81
5.11.1 Vector and Bitmapped Graphics . . . 81
5.11.2 Optimizing Vector Graphics . . . 83

6 Evaluation and Future Work **86**

6.1 Introduction . . . 86

6.2 Evaluation Stages and Design Changes 86
6.2.1 Prototyping 87
6.2.2 Pluralistic Walkthroughs 88
6.2.3 Mochi Analytics API 90
6.2.4 User Comments 91
6.2.5 User Ratings 91
6.3 Distribution and Reception 92
6.3.1 User Ratings 92
6.3.2 Distribution 92
6.4 Generated Traffic and Revenue 92
6.4.1 Sponsored Links 93
6.4.2 3rd Party Advertisements 94
6.5 Non-Functional Requirements 94
6.5.1 Addictiveness 95
6.5.2 Difficulty 96
6.5.3 Length 97
6.5.4 Novelty and Originality 98
6.5.5 Visuals and Audio 99
6.5.6 Robustness 99
6.5.7 Accessibility 100
6.5.8 Performance 101
6.6 Specific Criticisms 102
6.7 Future Work 104
6.7.1 Direct Sequel 104
6.7.2 Developing for Other Platforms 107
6.8 Conclusion 108

Bibliography **109**

A Full Class Diagrams **110**

B Qualitative Feedback **112**

B.1 User Comments 112

B.2 Videos 115

Chapter 1

Glossary

1.1 Video Game Terminology

1.1.1 Game Genres

Platform Game (Platformer): Games characterized by requiring the player to jump between chunks of land, or platforms[1]. The jumping mechanics must be flexible and must give the player lots of control over his trajectory. Popular examples of platformers include the Super Mario series[2].

Action Game: A broad range of games that emphasize physical challenges, including handeye coordination and reaction-time.

Role Playing Game (RPG): Games where the player assumes the role of a character or group of characters, and which are driven by storyline, exploration, strategic battles and quests. Characters typically grow in power and gain new abilities. A good example is the Final Fantasy series[3].

1.1.2 General Terms

Player: The user playing the game. Often used interchangeably with ”player character”.

Player Character/Playable Character: The in-game character(s) which the player controls.

Learning Curve/Difficulty Curve: The rate at which new challenges or concepts are presented to the player. Should increase steadily without any spikes.

Boss: A large and/or powerful enemy that must be defeated, usually at the end of a level, or group of levels.

[1] http://www.youtube.com/watch?v=w2NjUDfOp2o
[2] http://en.wikipedia.org/wiki/Mario_%28series%29
[3] http://en.wikipedia.org/wiki/Final_Fantasy

Spawning: The automatic and often randomized creation of an object such as an enemy or item. Respawning refers to objects reappearing at a different location after being destroyed or leaving the map.

Environmental Damage: Damage taken from the level terrain itself, rather than from enemies. Common examples include: spikes, lava and bottomless pits.

Button-Mashing: This is when the player rapidly presses keys or buttons without much thought into what actions his character will take as a result. This is often encountered when a new player is thrown into a complicated action game and has little knowledge about how to play. This also occurs when the player is forced to use the same action over and over again, due to poor game design or low player skill. Button mashing should be avoided in most games.

1.1.3 RPG Terms

Health Points (HP): Represents the amount of life a player or enemy has.

Magic Points (MP): Represents the number of times that magic or special abilities can be used by the player or an enemy.

Experience Points (EXP): The player collects these to Level Up.

Level Up: Improving the player's attribute(s), such as HP and MP, by accomplishing common tasks such as killing enemies, and collecting EXP, until a threshold is reached. The number of experience points needed to "Level Up" tends to increase exponentially with each Level Up.

Grinding: A repetitive task, such as hunting enemies, done purely to gain EXP or other resources.

Item drop: An item that has been left behind by a defeated enemy.

Equips: Short for "equipment". Refers to items such as armor and weapons that come in many varieties and must be "equipped" before use. Equips usually alter the player's attributes or add new abilities.

1.2 Flash Terminology

Flash was originally a platform for vector-based animation, and is still built around this animation metaphor today. It is important to know about the animation concepts in Flash, and how ActionScript's programming style differs from more general purpose languages.

1.2.1 MovieClips and the Animation Metaphor

MovieClip: Most visible objects in Flash extend the built-in MovieClip class. Every MovieClip has a corresponding library symbol, which contains a timeline of animation frames. Each frame can contain graphics, sounds, labels, and code. MovieClips have many built in display properties such as scale, x and y coordinates, alpha (transparency), rotation and many others. MovieClips also have many methods for controlling their timeline, including play(), stop(), nextFrame(), and so on.

Class code is kept in a separate ActionScript files, and not within timeline frames. This may contain a constructor, variables, and additional methods. Classes may not require an ActionScript file if they simply extend the MovieClip class without adding new functionality.

MovieClips are nested within other MovieClips to form a tree, starting at the ”root” MovieClip. Each MovieClip contains an array of ”child” MovieClips, and these children refer to it as their ”parent”. The order of children in this array determines their layering when graphics are being displayed. For example, the child with the highest index will appear in front of the rest when graphics are rendered.

MovieClips can be created by calling their constructor through code, and then assigning them to a parent, or by placing them on the stage manually through Flash Pro's drawing tools. The second method is not allowed if their constructor requires additional arguments. MovieClips are deleted by the garbage collector if no references to them exist.

Child & Parent: A child is a MovieClip (or other displayable object) nested within a parent MovieClip.

root: The root MovieClip of the MovieClip tree. All MovieClips are children or ancestors of this.

Stage: A class that represents a Flash project's drawing area. It contains the project's resolution, framerate, and other settings.

Timeline: Each MovieClip has a timeline which consists of a series of animation frames, which can also contain graphics, labels, sounds and code.

Playhead: A pointer to a MovieClip's current timeline frame. Can be manipulated using methods such as stop(), play(), goto(), and nextFrame().

Framerate: The rate at which timelines progress and frames are displayed. This is measured in frames per second (FPS), and is usually set to around 30 for Flash games (including this project).

Library & Symbols: A list of all the re-usable assets or ”symbols” within a Flash project, including MovieClips, graphics and sounds. Contains information on which assets are associated with which class files, and various compiling properties.

1.2.2 Tools and Applications

Flash Player: The browser plug-in or stand-alone player which runs Flash content.

Flash Professional: Adobe's official Flash authoring environment which includes drawing and animation tools. Can also be used for simple scripting, but is not usually used for programming larger projects. Flash CS3 (Flash Pro 9) is used for this project.

FlashDevelop: A freeware integrated development environment for ActionScript, similar to Java's Eclipse. It cannot compile projects and must be used together with Flash Pro.

ActionScript: Flash's series of programming languages, which is quite similar to Java and Javascript in syntax and features. The newest version, ActionScript 3.0 is used for this project.

1.2.3 Filetypes

.swf: The filetype of compiled Flash projects. Can be opened by most browsers and some media players.

.fla: The source filetype of Flash projects. Contains all project assets except class code and can be opened by Flash Pro.

.as: The source filetype of ActionScript classes, which can be viewed by any text editor. Note that subclasses do not need to have a file associated with them, as this can be defined within the project library.

.sol: .swf files use .sol files, or shared objects, to save data locally. These are similar to cookies.

Chapter 2

Introduction

2.1 Report Overview

This report is a comprehensive account of the Level 4 project ”A Platformer Game in Flash”. The aim of this project was to develop an online Flash game, entitled ”Adventure Story”, and monitor it’s distribution through the web.

A platformer, or platform game, is a game that makes use of jumping between platforms as it’s central mechanic. This project will mix traditional platformer mechanics with elements of RPG and action games. These include melee combat, player customization through equipment, and an upgrade system.

This report looks at the background of the Flash game industry, the design and development of this game, and finally assesses this project’s overall success after publishing.

Adventure Story was published on the 27th of December, 2011. The final product can be played here[1], but is also fully described later in this report.

2.2 Motivation

As explained in more detail in the next chapter, the online gaming market is a rapidly growing and potentially very profitable one. It is no longer unusual for high quality Flash games to generate $10,000 or more in revenue, with development times usually being a few months for a single developer or a small groups of developers. Additionally, many of the skills applied in developing Flash games are also transferable to creating games for mobile platforms such as iPhone or Android, or casual social games like those found on Facebook, which are also rapidly growing in popularity. Video games in general are already a huge industry, but one which is still opening to more and more demographics

[1] `http://www.kongregate.com/games/kupo707/adventure-story`

as technology evolves and becomes more accessible[2], with the Nintendo Wii[3] and Xbox 360 Kinect[4] being good examples of recent innovations.

The environment for Flash games in particular is an exciting and challenging one. Flash games are generally not advertised or supported by large publishers, and are usually free to play. They are distributed only by word-of-mouth and by being featured on gaming websites, and funded mainly by in-game advertisements. Flash games are rated by the players, and featured by site-owners, so their distribution depends on simply appealing to as many players as possible, rather than technical excellence in areas such as graphics.

I have many years of experience creating Flash animations, and more recently have begun developing Flash games as well. I have seen the Flash games scene develop from a niche hobby to a serious industry, and wish to continue being a part of it.

2.3 Goals and Challenges

The central aim of this project was to create a platformer that is fun and accessible to as large an audience as possible. This means the game must be simple for a player to pick up, instantly engaging, and yet rewarding to play for an extended period of time. The game must not rely on powerful hardware, nor alienate users who may have foreign keyboards or not speak English at all. The game must also be unique and memorable, by incorporating RPG and action game elements, as to separate if from the countless other free games which are available online.

The game engine and content will be created from scratch, using Adobe's Flash Professional development environment and the Actionscript 3 language. This type of game is different from any that I have previously worked on, and includes many new challenges, such as creating accurate and intuitive physics, optimizing performance, balancing the game elements and difficulty, creating a level editor which is convenient to use, and structuring the whole project to minimize overhead when adding new content.

The data shows that the project was successful in most of these areas; the game received good ratings from users and exceeded expectations in terms of distribution and revenue. There were no major issues reported since the game was published, confirming that the game was well designed, and the implementation was robust.

2.4 Report Contents

The rest of this report is divided into the following chapters:

[2] http://en.wikipedia.org/wiki/Video_game_industry
[3] http://en.wikipedia.org/wiki/Wii
[4] http://en.wikipedia.org/wiki/Kinect

- **Chapter 3: Analysis and Requirements** defines the functional and non-functional requirements of this project. These are justified by looking at existing games and game development heuristics. This chapter also looks at the current market for Flash games and online games in general, compares different platforms and technologies, and outlines how project development will proceed.

- **Chapter 4: Design and Architecture** first describes the low-level structure of the project, breaking down the functionality into components, and explaining how these components interact. The design of higher level aspects is then explained, including the user interface, visuals, audio, and the process of creating game content.

- **Chapter 5: Implementation and Testing** explains the tools and resources used for development, overviews the final product, and explains in detail how more complicated features were implemented. It explains how components were tested, both internally and externally, and how performance was measured and optimized.

- **Chapter 6: Evaluation and Future Work** explains the different stages of evaluation done during development, and the design changes that were made as a result. Statistics and reviews collected post-publishing are then analyzed to determine how well the project achieved it's non-functional requirements and distribution goals. Finally, the chapter concludes with the discussion of improvements and new features which could be included in future work.

Chapter 3

Analysis and Requirements

3.1 Introduction

This chapter describes and justifies the project requirements, and examines why entering the online game market is worthwhile. It explains why Flash has been used as opposed to other competing platforms, and gives an overview of it's strengths and weaknesses.

3.2 Proposed Game Overview

This project will in many respects be a typical platformer game; the player will travel through medium-length levels (around 5-10 minutes each), while fighting enemies and collecting items. Levels are made up of chunks of land, or platforms, and background scenery. A level select menu will act as the game's central hub, linking levels and other menus together.

To make the game more unique, it will include many action and role-playing game (RPG) elements. These include:

- In depth melee combat with a variety of short-ranged attacks. In most platformers, enemies are defeated simply by jumping on them or are avoided altogether.
- Leveling up by defeating enemies and collecting experience points, which improves the player's general combat proficiency and attributes.
- Collecting new weapons and armor from treasure chests within levels, which customize the player's attributes. Equipment will specialize the player in different areas, such as speed, melee attacks or magic, and the most useful equipment will depend on the user's playing style.
- A unique magic system, where the player has a number of powerful, limited-use abilities with various special effects. These will have to be used strategically, and in combination with other

attacks. Magic will be bought with coins collected in levels, and the player will have control over which abilities he receives.

The finished game will have 22 levels, including 4 boss levels, and take at least 2-3 hours for the average player to complete. The aim is to include enough secrets and customization for many players to play through the game more than once. Levels will be designed in such a way that they can be completed quickly if the player is not interested in challenges or exploration, but will have many hidden areas and optional path branches for those who have more time to be thorough.

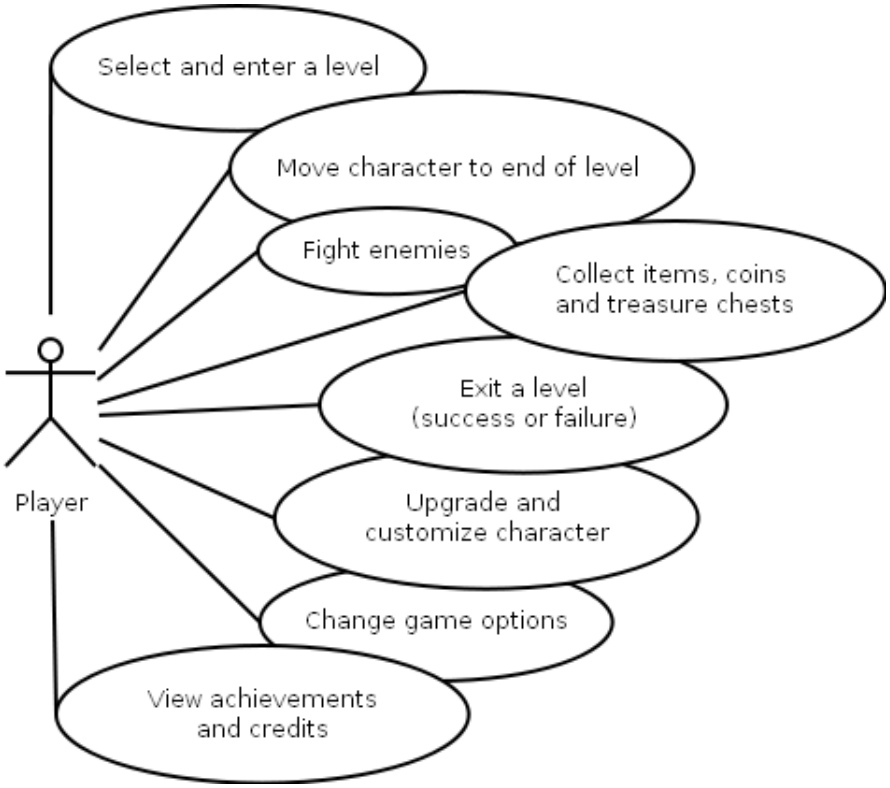


Figure 3.1: Use case diagram showing the main activities performed by the player.

3.3 Non-Functional Requirements

Below are the general non-functional requirements for this project. These can not be measured reliably, but should be used as guidelines during development:

- **Originality:** The game must have a unique mix of platformer, action and RPG elements. This is the game's main selling point, and should be implemented in a way that sets it apart from other similar platformers. More specific details in the next section.
- **Accessibility:** Care must be taken to make the game accessible to non-English speakers and children. This could include reconfigurable keys for foreign keyboards, and the use of easily

recognizable icons instead of text wherever possible. A familiar and clear interface is key, and testing will be done at many stages of development to ensure the interface and other mechanics are intuitive and easy to learn.

- **Addictiveness:** Gameplay should be rewarding and addictive, even if progress is not being made, through the use of achievements, power ups and exploration. More details on this in the next section.
- **Visuals and Audio:** The graphics and music should be reasonably appealing by Flash standards. Flash games are not known for good graphics or music, and players don't particularly expect them to be impressive. However, cartoony graphics can be appealing without requiring much effort, and they certainly add to the experience.
- **Clarity and Consistency:** The graphics and game mechanics must be clear and consistent. Platforms should be easily distinguishable from background scenery. Enemies that look similar should behave similarly, and so on.
- **Length:** The game should have at least 2-3 hours of gameplay for the average user. The exact amount will vary widely depending on the user's level of skill, and the amount of optional content they wish to complete. The important point is that the game should be long enough for the player to become attached to it, which will increase the chance of him showing it to a friend, and also increase the amount of ads that can be shown within the game.
- **Difficulty:** The difficulty or learning curve must be balanced. This ensures the player does not become bored if the game is too easy, or frustrated if the game is too hard. The difficulty should naturally get harder as the user progresses through the levels, and this can only be confirmed through extensive play testing involving a large group of participants.
- **Robustness:** The game mechanics must be robust and without any major bugs. A serious bug that crashes the game, and potentially the user's browser, will most likely result in the user never playing the game again, especially if this happens early on. Minor bugs or programming quirks are acceptable, but they should be in the latter levels of the game, by which point the user will be more addicted and forgiving.
- **Performance:** Performance must be optimized for low-end or single-CPU machines. There should be very little user feedback indicating that the game runs slowly or "lags" for a prolonged period of time (longer than 2 seconds). Brief lag during magic usage and large special effects is acceptable. A laggy game will deter many users. Many users play Flash games because they are under the (often false) impression that a powerful computer is not needed to do so.

3.4 Functional Requirements

Specific functional requirements are listed below. Each section has it's requirements listed roughly by priority. "Must" requirements are essential, while "Should" requirements may be dropped if it is not feasible to include them.

3.4.1 Game Controls and Mechanics

- The player must use keyboard input to control his in-game character. Keyboard controls are the standard for PC platformers, as they are quite similar to console gamepads[1], and similar functionality could not be achieved with a mouse alone.
- The player must be able to jump, walk, and crawl. These actions provide basic transportation through levels. They should be very responsive and refined. Testers should feel comfortable with these actions within a minute or two of gameplay.
- The player must have a variety of short-ranged melee attacks, which are effective in different situations. For example, attacks should vary in range, damage or knockback power. This is to discourage "button mashing", where a user constantly taps the same key over and over again during combat, leading to a repetitive experience. Testers should be seen using all of these attacks during combat.
- The player must have a variety of powerful limited-use magic attacks, which have special effects such as freezing or burning enemies. These will have to be used strategically, and should be balanced so that none is overall more useful than the others. Magic rewards the player for collecting coins, and adds variety and strategy to the game. I am unsure as to how exactly testers will use magic, but I would like to see testers using it against large groups of enemies or boss monsters.
- The controls and collision detection must be intuitive, responsive and accurate. Users must never feel "cheated" by the game mechanics. If the player is killed, the player must feel that it was his own fault, and not the fault of the game. Instant feedback should be generated when a key is pressed, in the form of a sound or animation, and if the input was ignored, it should be clear why. Instant feedback should be generated when collisions are made; when the player takes damage, when an enemy is hit, or when an item is collected. Collisions should be forgiving in favor of the user. Failing this requirement will lead to a frustrating experience.
- The basic controls must be fully explained by the end of the first level, but presented gradually as to not bore the player. The level must force the user to learn the controls, one by one, before it can be completed. For example, hurdles will be presented that require the player to learn to jump. This ensures the player will actually know how to play. Most users do not read game instructions if they are presented to them in the form of a long chunk of text.
- The player must have HP (Health Points) and MP (Magic Points). These represent the player's life, and how many times he can use magic. These must be displayed to the player at all times, very clearly.
- The player must gain EXP (Experience Points) by defeating enemies, and "Level Up" when a certain threshold is reached. Leveling up will increase the amount of damage a player deals to enemies.
- The player must have access to different equipment (hats, armor, and weapons) that modify his attack power, magic power, movement speed, and HP. This allows different play styles based on the player's preferences, and also increases replayability.

[1] `http://en.wikipedia.org/wiki/Gamepad`

- The player should stay in the center of the screen as he moves around, with the level scrolling around him. The field of view should include any point that the player can reach within a single jump.

3.4.2 Level Design

- Levels must contain platforms (land) for the player to stand on. All platformers have platforms. Special platforms with unique effects should also exist to add challenge and variety. For example, ice that causes the player to slide while walking.
- Levels must contain enemies to fight with. Enemies must have a variety of behaviors, to increase the variety of combat and prevent the game from becoming too easy or familiar. Enemy behaviors should include jumping, flying, projectile attacks, stealth attacks, teleportation and so on.
- Levels must have many items to collect, to motivate exploration and progress, and to upgrade the Player's abilities. Items include:
  - Coins, which are used to purchase and upgrade magic.
  - Treasure Chests, which may contain coins, HP and MP upgrades, and equipment.
  - HP and MP item drops from enemies, to keep the player alive.
- Levels must not become repetitive, and each level should introduce at least one new type of enemy or obstacle. Platform layouts should be reasonably unique in each level. Background scenery should also change after every 5 levels.
- Levels should include optional path branches, which offer bonus challenges for more thorough players.
- After every 5 levels there should be a boss battle against a powerful enemy. This is to contrast with the usual level structure, which is focused more on exploration rather than combat, and to challenge the player.
- Levels should be divided into areas separated by portals (doors), to reduce computer resource usage, and to add structure to the level layout. Areas should be labeled so that the player does not get lost.
- Levels should include signs near new objects that may require explaining. Signs should show text or images to the user when approached.
- Progress within a level (items collected) should be saved even if the level is not completed, to avoid excessive repetition if the player dies, and to allow the game to be played in short sessions or to be interrupted.

3.4.3 Menus and GUI

- The game must include the following menus:

  - A loading screen and preloader (loading bar). This will be the first thing the player sees, and some credits or instructions can be displayed here if needed.
  - A title screen and main menu, which gives the player a choice of getting started, or changing some options first.
  - A level selection menu, which shows all unlocked levels, and the progress made in them. This will be the central hub of the game, which links to most other menus and to each level.
  - An equipment menu, where all available equipment is displayed, and can be selected. This menu should make comparison between the different equips easy, and show how they modify the player's stats.
  - A magic menu, where magic is available for purchase using coins the player has collected. The menu should offer descriptions of each item, and give clear feedback when an item is purchased.
  - An options menu, where the player can change difficulty, quality, key and sound settings.
  - A pause menu which can be opened during gameplay, which is used to exit a level, and change a subset of the settings in the options menu.

- Each item of equipment, each magic ability, each achievement, and each level should have a unique icon. The use of icons makes collecting and unlocking these items more rewarding, and makes it easier for the user to select a specific one in a large menu.

- Menu items should be represented using text and icons. Icons make it easier for children and non-English speakers to understand the menus.

- Menus should also support mouse selection, as well as keyboard selection. Mouse controls are more widely used in Flash games, and many users will expect at least some support for mouse input.

3.4.4 Miscellaneous

- Gameplay must be instantly engaging. A rule of thumb is to have the user playing the game after only 3 mouse clicks once the game is loaded, and I shall aim to achieve this. This is essential for attracting players and sponsors, and ensuring that the game receives good distribution.

- The final .swf file must be under 15MB in size. This is around the size that most websites will accept by default. If possible, the size should even be under 8-10MB, so players with slow internet connections are not waiting too long to play the game.

- The game must save level progress, player attributes, settings, achievements, and all other variables between sessions.

- There must be an in-game achievement system, where the user earns "medals" for completing various tasks such as killing 100 enemies. This is now a standard feature in most video games, which rewards players and increases replayability. Many websites that host Flash games have APIs to link achievements to a user's account.

- The game must have appropriate background music and sound effects. Music should change after every 5 levels to avoid becoming repetitive, but care should be taken not to use too much, as music greatly adds to a Flash game's file size. Sound effects should be used to indicate collisions and reinforce animations.
- There must be an option to mute sound effects and music. Many online gamers listen to their own music while playing.
- There should be an option to change the overall difficulty level of the game. This is to help balance the difficulty curve.
- There should be an option to customize the key settings. This supports different keyboard layouts and left-handed users. Additionally, there should be on option to reset these settings once they are changed.
- If time allows, there should be 2 or 3 animated cutscenes that present a simple story to the player, and provide another incentive for playing the game.

3.5 Game Design Heuristics

The goal of any free web based game is to be addictive and make the player feel as if he is having fun. The exact definition of "fun" in video games is debatable (many games require the exact same actions to be done for hours, but are still fun), but it should in any case make the user want to continue playing. If a user did not pay for a game, they feel little motivation to finish it unless they are having fun. There are many tried and tested heuristics which can be followed to lure players into a game, and keep them playing. This section identifies good game design and justifies the project requirements.

Game Design - Theory & Practice (Rouse, 2005) has an interesting chapter on "What Players Expect", which lists general advice on designing fun games:

- The game world should be consistent. Actions should lead to predictable results, which the player can understand, based on his experience so far. If something completely unexpected happens due to the player's actions, this can be very frustrating or confusing. Additionally, the scope of the game should be clear; what is possible and what is not. This project aims to have the player fully grasp the game mechanics after 2 or 3 levels.
- The goal of the game should be clear, and the player should be directed towards it. Small goals, such as defeating enemies or collecting items can show the player that he is making progress towards a larger goal, such as completing a level. Maps and other hints can also help guide the player. The requirements I've listed for level design cover this.
- Real-world logic and physics should still apply wherever possible. Enemies should be knocked back when hit. Bigger weapons should be slower and deal more damage. Coins should be used for buying items, and so on.
- Challenges should not be repeated unless they are actually fun or are modified between each occurrence. For example, fighting the same enemies could get boring quickly, but can stay

interesting if the terrain is changed, or if there is more of them each time. The player should not be forced to replay too many completed challenges if he is defeated later on. Checkpoints should be reasonably frequent. I apply this point in my level designs, in which partial progress is still saved, and new types of enemies are constantly introduced.

- Games should aim to immerse the player. The player should begin to forget he is playing a game. Game bugs or stylistic inconsistency can damage immersion. The menu-based game hub I have chosen may break this rule (I feel it is more suitable for Flash games however), but it should still hold within levels.

- ”Gamers do not know what they want, but they know what is missing”. The best way to collect suggestions and ideas for improvement is to let people try playing a game, and finding what they dislike about it. Just describing the concepts to them and asking open ended questions will not produce good ideas. This advice is relied on heavily during the development and evaluation of this project, where play testing is done frequently.

Game Architecture and Design (Rollings & Morris, 2003) has an interesting section about ”Game Balance”, stating many common pitfalls that developers should avoid, and some advice for making a rewarding game:

- Just learning how to play the game should be rewarding. When a player learns how to use a new ability and applies it, it should open up new possibilities in the game. For example, a new ability could offer a more convenient method of transport or make a known enemy much easier to defeat.

- A balanced level of difficulty is vital, as players can easily get bored if they are not challenged, or frustrated if they are challenged too much. There are several strategies for this. Difficult, optional levels could be added, which can be ignored by more casual players, while offer challenges for those more experienced. Failing a level should never result in losing more than a few minutes of progress, as this could be frustrating. Finally, giving the option to change the level of difficulty at any time will allow players to solve the problem themselves.

- ”Eye candy” is another motivating factor. This is the introduction of new special effects or animations to excite the user. These can come in the form of colorful magic attacks, massive boss monsters, or cutscenes that tie into the game’s story, and I hope to include all of those.

- Challenges in the game should be passable on the player’s first attempt, given his previous knowledge and experience of the game mechanics. Many games force the player to advance through trial and error, where error may lead to a frustrating death and loss of progress. Unfair tricks should not be used to add difficulty.

- Competing game elements such as different player abilities and equipment should have a ”rock paper scissors” relationship with one another. This ensures that each element is useful in different situations, and none are overall superior to the rest. This is important so that the user does not end up always using the same elements, with the others being a waste of time for the developers. The only way to enforce this is through a lot of play testing, as predicting how these elements will be used in practice is difficult.

Flash games in particular put a lot of emphasis on the rewards a player receives. Widely used methods include achievements, upgrades, high-scores, and a leveling up system. The user can often feel as if he is making progress in the game even if he isn't completing any levels, and won't become demoralized or frustrated. These can also add a lot of replayability without actually adding any content.

To summarize; a game should be reasonably challenging without feeling unfair, have a consistent and rational game world, and always reward the player for making progress.

3.6 Development Approach

Long before the start of this academic year, a primitive and experimental prototype was developed. This prototype consisted of the core platforming and battle concepts which carried on to this project. It was implemented using the now outdated ActionScript 2 language, and gave me a picture of the scope and challenges involved in developing a full game of this type. The current requirements are roughly based on this prototype, though it also experimented with other features that were deemed unfeasible and dropped. No code or other assets from this prototype will be used in the final product. The prototype can be played here[2].

Development on the actual product will begin by first programming the core game mechanics. These include the scrolling camera, controlling the player (walking, jumping, attacking) and a simple test level. Basically, these are the minimum requirements of a platformer game. Once this is done, it will be released as a demo online, to be tested and for some early opinions. It is essential to make sure these mechanics are refined and robust before creating actual levels and other content.

Next, I'll begin programming some actual enemies to fight, collectible items, portals, the magic system, the equipment system, the levelling up system, and the rest of the main game mechanics. The online demo will be updated for testing whenever a significant number of new features or content has been added. Emphasis is on removing bugs early on, and collecting opinions on which game elements are fun, and which need to be modified. This will also be a good time for performance testing, and to study the limitations of the game engine. For example; testing how many enemies can be on screen at once, or how large levels can be before experiencing a drop in performance. It is important to know these limits before designing levels, as it would be very inconvenient to discover them late on in development.

Once all of the game mechanics and level elements are finished, I'll create the level selection, equipment, magic, and pause menus, which are required to move between levels, and select different game options. The menus do not have to be finalized yet, as long as they serve their basic functions. Once the menus are done, I'll start working on the actual levels, which will consume a large portion of the development time.

Working on the graphics will not require much time as the majority of them have already been prepared prior to the start of the academic year, or have been otherwise recycled from my previous projects. Similarly, sound effects will also be recycled and are also easily obtainable through many

[2] http://kupo707.deviantart.com/art/Action-Platformer-engine-136020719

free sources. The background music will be composed by a friend of mine, so I don't need to concern myself with that either.

Once all of the game mechanics, menus, and around half of the actual levels have been finished, more formal evaluation will begin. This will be to ensure the difficulty curve is balanced, and that the game has long-lasting appeal, beyond the first few levels. The evaluation will involve observing users and studying how they would play the game under realistic conditions.

A final demo of the game will be released online once all of the game content is finished. This will be mainly for extensive testing and debugging, as making changes to the game mechanics at this point would be risky. During this testing I will be finalizing the appearance of menus, balancing the game difficulty, and doing various tweaking that will be unlikely to produce new bugs.

When the game is completely finished and has been played for several days by hundreds of testers, it will be published on Kongregate.com, shortly followed by other websites. Distribution data and user ratings will be collected from then onwards. I aim to reach this point before the start of the second semester. This will generate massive amounts of user feedback, in the form of comments, ratings and usage statistics. I will be reading all user comments to create a list of common complaints and suggestions, which will be used to consider future work.

Overall, the development process will be very transparent with frequent play testing and incremental demo releases. It will begin with the most essential features and build on them until most or all requirements are met.

3.7 Why use Flash?

Flash is now a very well established platform, and most internet users will have it installed on their PC, or even on their mobile devices. An estimated 96% of all PCs have shipped with, or already have installed, the Flash plug-in[3]. Nearly anyone with an internet connection can play Flash content without installing any additional software, and as Flash files are also very compressed, the internet connection need not be a fast one.

There are many websites which feature exclusively user-submitted Flash games and animations. Some examples, which are referred to often in this report, include Newgrounds.com, Kongregate.com and ArmorGames.com. Many of these sites have millions of regular visitors, and a dedicated user base. Content on these sites is mainly rated by users, and higher rated content receives more attention. Additionally, Facebook and other social networks host some extremely popular Flash games, such as FarmVille, which has over 80 million players[4].

Thus distributing Flash games is very simple; a developer simply has to upload them to a few large websites, and many other websites will then upload the games themselves. There are no distribution costs to the developer, and Flash games can easily attract millions of players provided they are of high quality or novelty.

[3] `http://www.statowl.com/plugin_overview.php`

[4] `http://www.joystiq.com/2010/02/20/farmville-community-surpasses-80-million-players/`

Flash is also relatively fast and easy to develop in, compared to other platforms, for reasons described later. The ActionScript 3 language is very similar to Java, as are the available integrated development environments. The most popular IDE for Flash, and the one I will be using, is FlashDevelop, and it is freeware. Flash Pro's development environment also includes a framework for creating graphics and animations, as this was Flash's original purpose, so no other applications are needed. Flash is ideal for rapid prototyping, as starting a project requires no boilerplate code, and projects are compiled very quickly, usually in only a few seconds, even for large games.

Flash's main disadvantage is that it runs very slowly. Flash games perform extremely poorly compared to games written in C++ for example. However, C++ is more difficult to use for smaller projects, and C++ games are not as easy to distribute over the web. Many older PCs will struggle to run Flash without much slow down, as will many mobile devices. Although this particular project does not target mobile devices, it aims to be playable on PCs which may still be using a single-core CPU, and thus testing will be done on a variety of machines. CPU, memory and bandwidth usage will be monitored and optimized. Regardless, low performance is not unique to Flash, and many web platforms such as JavaScript also suffer due to the requirements of security and ease of use.

3.8 Alternatives to Flash

There are several alternative platforms for online games, which already, or will soon, compete with Flash. I looked at the advantages of some of the more promising ones; HTML5, Unity and Java.

3.8.1 HTML5

HTML5 is a powerful new web technology that many, including Steve Jobs[5], claim will replace Flash. However, it is currently in its infancy and is not widely supported. Tests show various degrees of compatibility with different browsers. Internet Explorer 9 has the lowest compatibility of all popular modern browsers[6], and it is also worth noting that many users, especially those who are still using Windows XP, will not even have the latest version. Recent statistics show that Internet Explorer is still the world's most popular web browser, holding an estimated 40% of the browser market[7]. Therefore, incompatibility with Internet Explorer would be very damaging to a game's potential distribution. In time, when all popular browsers support HTML5, there will still be minor differences in the way each browser runs it. This may make debugging large games quite tedious. Flash on the other hand is not browser dependent.

Another major advantage Flash has is that the development environment is much more suited to creating vector graphics and animations, which is particularly useful when creating games. Flash Pro has well established drawing and animation tools, which have changed very little since the early versions. HTML5 does not currently have the tool-set required to make these activities as simple as they are in Flash[8].

[5] http://www.apple.com/hotnews/thoughts-on-flash/
[6] http://html5test.com/results.html
[7] http://en.wikipedia.org/wiki/Usage_share_of_web_browsers
[8] http://thenextweb.com/apple/2010/05/14/html5-vs-flash-what-you-havent-heard/

HTML5's source code is freely available to view, requiring more effort from the developer if he wishes to hide it. Flash files on the other hand must be de-compiled, and even this will not retrieve any useful code if the file has been obfuscated.

Even in terms of performance, HTML5 may not offer a definitive improvement. Some benchmark tests have shown that Flash still out-performs HTML5[9], although these are by no means comprehensive, and results will vary in practice.

Overall, HTML5 may yet be a powerful platform for web game development in the future, but compatibility issues and a lack of animation tools give Flash the advantage.

3.8.2 Unity

Unity is a new game platform which sells itself on providing hardware-accelerated 3D graphics and support for many operating systems and games consoles[10]. These include Xbox 360, PlayStation 3, Wii, iPad, iPhone and Android. It does not yet support Linux, but Flash does not run well on Linux either.

Unity's biggest disadvantage is that it requires a plug-in, and only around 1-3% of PCs currently have it installed. It is then unlikely that a Unity game will receive a similar level of attention to a Flash game. This may change in the future, as Unity aims to run within the Flash plug-in too. Unity games are already supported by some online game websites, such as Kongregate.com, alongside Flash games, but have not yet seen the same success.

Another disadvantage is that 3D games generally require greater development time than 2D games, especially if they aim to compete with the quality of graphics found on modern console games. It is unlikely that I would be able to create a reasonably complex 3D game in the time available to me.

3.8.3 Java

Java is another well established plug-in, which is now installed on around 78% of all PCs. However, many browsers see it as being more dangerous than Flash, and may display warnings about it to the user. Flash therefore still has greater distribution potential.

Java supports more complex programming features such as multi-threading and generics, while Flash's ActionScript 3 does not. Java features are also very well documented. For these reasons, to an experienced programmer working on a large project, Java may be more suitable than Flash. Though the project would have to attract enough attention to motivate users to install the plug-in. This factor combined with Flash's superior animation tools and ease of development, means that Flash is still a safer choice for most low budget games.

[9] `http://themaninblue.com/writing/perspective/2010/03/22/`
[10] `http://en.wikipedia.org/wiki/Unity_%28game_engine%29`

3.9 Market Background

The online game market is a new but rapidly expanding one. Traditionally, players would have to pay for a hard copy of a game, but now, internet users have become accustomed to getting online services for free. As a result, many new marketing strategies are appearing[11]. One strategy is to make games free to play but including advertisements in them, which generate revenue as the game is distributed. A similar strategy is to have the game sponsored by a particular gaming website, and concentrate on moving traffic to that website. Several high quality games even include extra content which must be paid for by the user, while the majority of the game's content is still free to play. In these ways, developers can profit without limiting the distribution of their games.

This project will be sponsored by Kongregate.com, meaning that it will contain several links to various areas of their website, and integrate with their high-score and achievements API. These links typically appear at the bottom of menus or in the corner of the screen, and are fairly non-invasive. The game will also contain a 3rd party advertisement during the preloader, which is implemented by using an API provided by CPMstar.com. All revenue will therefore be purely performance-based, and the game will remain 100% free to play.

Flash games are becoming a profitable market, and the ease of digital distribution makes it possible to develop them without any funding. However, as the majority of developers are still young, individual hobbyists, the average quality of games is not so high[12]. This is evident if one looks at the user scores of newly submitted games at any website which rates content. There are now tens of thousands of free games out there, so games must really stand out from the crowd to be successful.

I personally communicated with Greg McClanahan, a staff member at Kongregate.com and my sponsor on a few occasions, about what qualities he looks for when sponsoring Flash games. The major one was a game's ability to go viral, and spread across many websites. An important factor here was to make the game accessible and instantly fun, right from the start. The first few minutes are vital for grabbing a player's attention. Games which are instantly appealing and easy to pick up have a much greater ability to attract players, and are more valuable in the market. Greg also mentioned that a game that is original, with some interesting themes or mechanics, will receive better distribution through word-of-mouth and will be more memorable to users.

I also asked followers of my personal blog[13] about what sort of features they felt were lacking in Flash games, and in platformers specifically. A common complaint was that Flash games were in general too short, and lacked depth, often only focusing on a few core mechanics and being under 1 or 2 hours in length. They wanted more freedom in how to play a game; different ways of defeating enemies, more strategic boss battles and a wider range of abilities. They also wanted to see more open and varied environments, leading to more exploration. Finally, many also mentioned that Flash games often have gimmicky and intuitive controls, due to the limitations of mice and keyboards compared to console game-pads. Keyboard input involving more than 3 to 4 keys plus the arrow keys is often difficult to become familiar with, especially in fast paced action games, and some keyboards only support 3 or 4 simultaneous key presses. When keyboard and mouse inputs are used together, controls become especially difficult on laptops. Modern console game pads have upwards of 10 buttons, plus

[11] `http://www.adobe.com/newsletters/inspire/march2011/articles/article1/`
[12] `http://wiki.mochimedia.com/w/page/15156195/Flash%20Games%20Market%20Survey`
[13] `http://kupo707.deviantart.com/journal/44914325/`

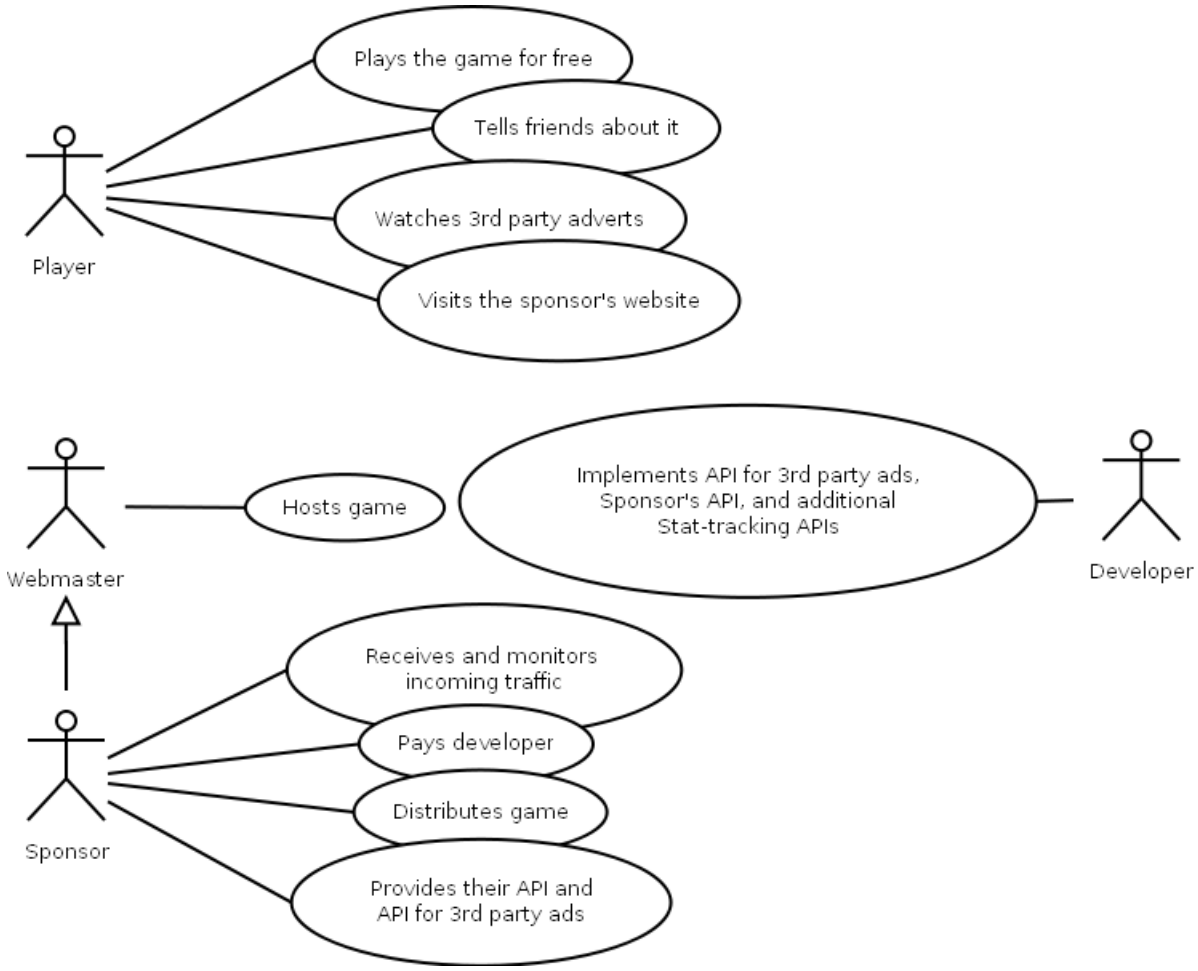


Figure 3.2: Use case diagram showing the business context for this project.

two or three directional inputs, and are more comfortable to hold, leading to less stress on a player's hands compared to a keyboard.

Many Flash platformers already exist, but the majority of them are very short and simple, limited only to a few core mechanics and lacking in depth.

Edmund McMillen's games, such as Meat Boy[14], Spewer[15] and Time Fcuk[16] are great examples of what can be done with the Flash platform. His games mix platforming action with puzzle elements, which leads to some interesting level designs. His unique but creepy art style and humor are also very memorable. For example, in Spewer, the central game mechanic involves vomiting, which can be used to propel your character into the air, or solve puzzles in various ways. Meat Boy even evolved into a console and PC game, Super Meat Boy, which was incredibly well received, due to it's excellent controls and level design. However, Spewer and Time Fcuk did have some flaws. Spewer's fluid dynamics and Time Fkuc's visual effects required some intense processing, and many players com-

[14] `http://www.newgrounds.com/portal/view/463241`
[15] `http://www.newgrounds.com/portal/view/494129`
[16] `http://www.newgrounds.com/portal/view/511754`

mented that the games did not run smoothly, even on the lowest quality settings. Also, the shortness of the levels and linear progression between them was criticized. One simply completed level after level, and there was little exploration or other activities to be done.

Platformers are generally harder to program than other genres, as they include more complicated interactions between many objects and very specific bugs can be difficult to detect and recreate, due to the nearly unlimited amount of execution paths. I say this from experience and the opinions of other Flash programmers. Many otherwise successful platformers have launched with game breaking bugs[17][18] (such as the player occasionally falling through walls or getting stuck), and it is rare to find a Flash platformer that has a truly robust engine. Controls and physics also require a lot of fine-tuning. For example, the player's aerial movement speed should not be too fast for the user to control his trajectory.

3.10 Personal Background

I have around 8 years of experience with the Flash platform. The first 5 years only involved animation with some very basic scripting. At this time Flash was strictly a passive hobby and I did not make a great effort to learn any programming.

Over the last 3 years I have moved on to developing games in Flash, as the Java programming and software engineering practices we have been taught on this course have been very helpful in this area. Quite conveniently, demand for browser games has increased during roughly the same time. Back when I started animating in Flash, browser based games were very crude and not as mainstream as they are today.

I have completed a total of 7 projects which I would consider to be fully functioning games, although the first 4 could be considered interactive movies rather than games. Only the most recent 2 were made with a reasonable amount of programming knowledge. All of these games are quite different from this project, most of them being turn-based strategy games.

6 of these projects were written in the ActionScript 2 language, and only the most recent was written in ActionScript 3, so I am still learning many things about the language.

My most recent project was a shooting game called "Bullet Heaven"[19]. In terms of gameplay and programming it was very different from this project, but several elements from Bullet Heaven were used to base designs on, most notably the GUI. A lot of the usability feedback from Bullet Heaven was relevant to this project, in particular how menus and tutorials should be laid out.

My current testing strategy has been shaped as a result of what has worked for me in the past. It has also been made possible due to the large number of followers my blog[20] has accumulated over time; mainly users who are looking forward to testing new games. I have also made many connections with website owners over time, which makes finding a sponsor much easier than it otherwise would be.

[17] http://www.newgrounds.com/portal/view/569281
[18] http://www.newgrounds.com/portal/view/566862
[19] http://www.kongregate.com/games/kupo707/bullet-heaven
[20] http://kupo707.deviantart.com/

In future, I may move on to developing games for mobile devices or platforms such as Steam[21] or Xbox Live Arcade[22]. Many popular Flash games, such as Crush the Castle[23] do get ported to these platforms, as they are potentially much more profitable, but they are also more difficult to develop for. However, for me, Flash game development is already profitable enough to be full-time work, so the motivation to change platforms is not so great.

3.11 Measuring Project Success

This project's overall success will be determined by how many people ultimately play the game and for how long. It is impossible to accurately predict how many plays the game will receive, due to the unpredictable nature of viral internet content, but I personally would hope for at least 3 million sessions in total.

Traffic and distribution information will be collected using Mochi Media's and Kongregate's statistics APIs. Mochi will collect information about websites the game is hosted on, what countries users live in, how many levels the user completes, the number of game sessions, the length of an average session, and the quality and difficulty settings the user has chosen. Kongregate will specifically measure the amount of traffic directed from the game to various pages on their site. This information will be used for evaluating success in distribution and game addictiveness.

My previous Flash game project, Bullet Heaven, will be used to compare data. Bullet Heaven used the same APIs for collecting statistics, and was developed for the same sponsor.

Bullet Heaven received around 2 million sessions in it's first 2 months after publishing, and has been hosted on around 250 different websites. Bullet Heaven had a much shorter development time compared to this new project, so I am expecting statistics to be around 50% higher this time. It is naive to relate these statistics directly to development time, but such predictions may demonstrate whether it is more worthwhile to work on many small projects, or one large one.

The average user ratings on websites will indicate how players generally feel about the game. User ratings are not directly proportional to distribution, but there is usually a strong correlation. User ratings offer a good comparison to how much users enjoy a certain game compared to other games. I will consider this project well received by players if it achieves the following ratings:

- At least 4.27/5 on Kongregate.com. This will get the game into the first page of the adventure games section of the site.
- Around 4.43/5 on Newgrounds.com. Any higher rating will eventually tend to this, as users are allowed to rate a game once a day, and this is currently the most stable score.
- Around 8.2/10 or higher on ArmorGames.com is a respectable score.

[21] http://en.wikipedia.org/wiki/Steam_%28software%29
[22] http://en.wikipedia.org/wiki/XBLA
[23] http://en.wikipedia.org/wiki/Crush_the_Castle

The balance of the game difficulty will be evaluated based on what settings players use, and how many times they must attempt each level. If for example, a much greater proportion of players are playing the game on easy mode, rather than hard mode, the game may be considered to be too difficult.

Finally, qualitative user feedback will be received in the form of hundreds of comments on many websites, and will be used for evaluating success in all areas. Feedback will be considered more relevant if many users mention the same issues. The most common user suggestions will be used to consider future work.

Chapter 4

Design and Architecture

4.1 Introduction

This chapter deals with the game engine design, in terms of the high-level structure, the functions of each component, and how the components communicate. It also explains how game content is designed and created, and then run by the engine. This chapter is divided into the following sections:

- **Structure Overview:** High level description of the project architecture.
- **Instance Classes:** Explanation of the classes which extend MovieClip and are instantiated during gameplay.
- **Static Classes:** Explanation of static classes which are used to store data and provide utility methods.
- **User Interface:** Design and wireframes of the graphical user interface, and keyboard input.
- **Visuals and Audio:** Explanation of how the visual style was developed to support novelty and user feedback, and how audio supports this.
- **Game Design and Creating Content:** Overview of the process used to design and implement game content, including levels and enemies.
- **External APIs:** Explanation of the stat-tracking APIs used for evaluation and competition between players.

4.2 Structure Overview

The structure of the project is split into three main layers:

- **Top Layer:** This layer is that which is visible to the player; game menus, and objects during gameplay.
- **Middle Layer:** This layer consists of static classes (singletons and factories) which store global variables and provide utility methods, and are used to communicate between the different sections of the top layer, and the bottom layer.
- **Bottom Layer:** The bottom layer of the game contains persistent data which is saved to a file on the player's computer, and is loaded at the start of the next session so that the user may resume playing from where he stopped.

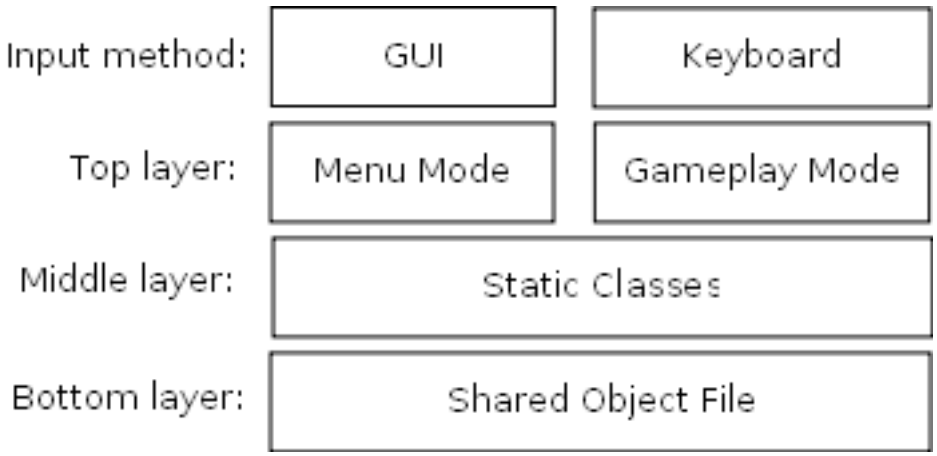


Figure 4.1: The main structural layers of the game.

The top layer of the project structure is divided into two main sections:

- **Gameplay Mode:** This is where the player is actually playing the game, and where objects such as platforms, enemies, items and other game elements exist in the form of MovieClips, and are controlled by a central loop of code. The contents of this mode are dynamic and largely created and controlled through code at runtime.
- **Menu Mode:** This is where the player is not playing, but rather browsing through menus, in order to select a level to play, select equipment, buy upgrades, view progress statistics and change game settings. The contents of this mode are mostly static, and the code is stored on frames rather than in classes, as it is tightly coupled to the interface.

4.3 Instance Classes

This subsection describes classes which extend the MovieClip class. A detailed description of Flash's MovieClip class can be found in the Glossary, and is required knowledge for this chapter and the next.

Figure 4.2 shows the MovieClip hierarchy and layering during gameplay mode. At the lowest layer is the level, which contains all objects which the player can interact with; platforms, enemies (foes),

treasure chests, signs, coins, portals and item drops. The Level MovieClip also contains background scenery which scrolls as the player/camera moves, but is otherwise purely aesthetic. The middle layer contains the player MovieClip. This is the player character which interacts with the level. Finally, the top layer contains the Head Up Display (HUD), which always displays the information that is needed during gameplay. It also holds miscellaneous game elements which must appear in front of everything else, including speech bubbles, special effects, and the pause and game over menus.

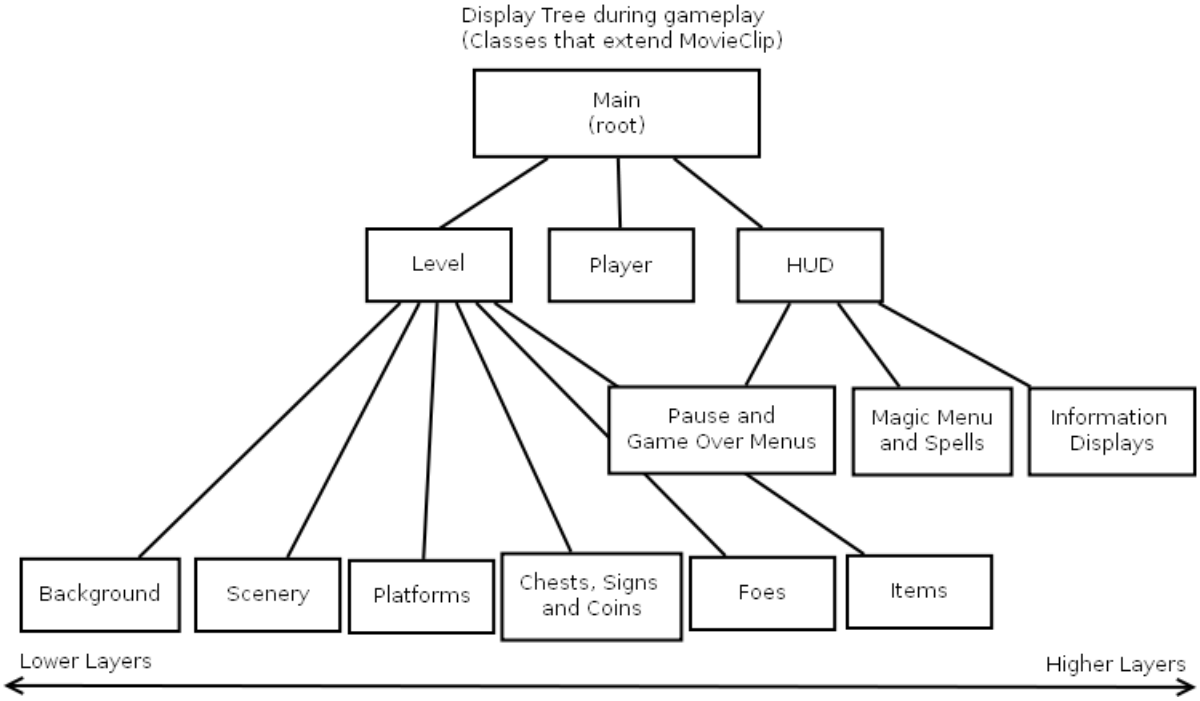


Figure 4.2: The MovieClip tree during gameplay, representing the visual layering of objects.

Main

The Main class is used as the root MovieClip, of which all other MovieClips are children. It initializes listeners for mouse and keyboard (key-down and key-up) events, and passes these events to the Key class, and creates the enter-frame listener, which is used by the Game class. It also calls methods to initialize data in static classes such as SFX and Equips, once the game is loaded.

This class supports the switching of game modes, between gameplay mode and menu mode, by calling methods to create or remove MovieClips and by setting any required parameters.

Player

The Player MovieClip is the character which the player controls. This class handles most of the user's input, and converts it into various actions depending on the Player's state. The Player's state variables include whether he is currently attacking, on the ground, ducking, interacting with a background

object, dead, his current speed, which direction he is facing and so on. The Player's state determines what actions he can perform. For example, to jump, the jump key must be pressed, the Player must be on the ground, and must not be interacting or attacking. Each action has its own requirements. When an action is performed, the corresponding animation is played (all animations are stored on a single timeline, with frame labels indicating the start of each), and the end of that action is usually signaled by the end of the animation. The appropriate state variables are then updated. Possible actions are listed below:

- **Walking**
  When the left or right keys are pressed (but not both), the Player will accelerate in that direction, and will reach maximum walking speed in around half a second. This brief acceleration makes small precise movements possible, and also feels more natural than instantly walking at the maximum speed. This acceleration still takes place while the Player is in the air (although it is a bit slower), giving the Player control while falling or jumping. Similarly, the Player can move slowly while using certain attacks, but not at all when using others.

- **Ducking and Crawling**
  When the down key is pressed while the Player is on the ground, the Player ducks instantly. Combined with the left or right keys, the Player will crawl slowly. This is for dodging certain enemy attacks.

- **Jumping and Falling**
  When the jump key is pressed and the Player is on the ground, the Player's vertical speed will increase instantly. The vertical speed will then decrease due to gravity until the Player begins to fall and lands back on the ground. While in the air, the player can move left and right, and use melee and magic attacks.

- **Melee Attacks**
  When the attack key is pressed, the Player will perform a short ranged melee attack. The exact type of attack depends on the Player's current state:

  - **While standing on the ground,** the Player will perform a simple forward sword slash. If the user presses the attack key again shortly after the first attack, the Player will perform a 2-hit combo. A third press of the key will perform a full 3-hit combo. The first two attacks have little knockback power, as to not push the enemy out of range, while the third attack pushes the enemy to a safe distance. Combos increase the damage done by the player and add variety to an otherwise basic attack.
  - **While on the ground and holding the up key,** the Player will perform a 180 degree, upward sword swing. This is more of a defensive attack to knock back enemies which are flying just above the Player.
  - **While ducking,** the Player performs a weak needle-like sword stab. This attack is to avoid leaving the Player completely helpless while in the ducking position.
  - **While in the air,** the Player will perform a 360 degree sword swing, which reaches almost equally well in all directions. If the Player hits an enemy with this attack, the Player will bounce higher into the air, allowing for the Player to perform more attacks or reach higher Platforms. To prevent abusing this and trivializing many enemies, the bounce is smaller after each attack, until there is no bounce at all.

  - **While in the air and holding the up key,** the Player performs an upward sword stab, which has greater vertical reach than the standard aerial attack. This is for attacking enemies far above the Player.

- **Interacting**
  When the up key is pressed while the Player is in contact with a Portal or Treasure Chest (these are described later), the Player will interact with that object. In the case of the Portal, he will be teleported to a different area. In the case of the Treasure Chest, he will play a short animation and receive an item. During both of these interactions, the Player cannot receive damage, and all Foes are temporarily stopped from moving, as this would be an annoyance.

- **Magic**
  When the magic key is pressed, the magic menu pops up, and the game is paused. The player can then view and select available spells. Each spell costs 1 MP to use, and their effects vary widely. Each spell is described in more detail later in this chapter. Spells can be assigned to three hotkeys, to bypass the magic menu and to not slow down gameplay.

It is difficult to predict all of the state variables and other properties that will be required to fine-tune the controls and the above actions. Chapter 6 covers these in more detail, and how they were updated to feel more intuitive.

The Player's main control method is called on every frame by the game loop (explained later), and this method maps the inputs to actions and performs collision detection with platforms and enemies.

The Player class also contains attributes such as HP and attack power. These are modified as the user progresses through the game, and are used in combat calculations. Attributes implement a large portion of the RPG element of the game, as they allow the player to grow stronger over time. The growth rate of the attributes must be balanced so that each increment is noticeable by the player, but also not large enough to quickly make a level trivial to complete. The Player's attributes and their uses are as follows:

- **HP:** This represents the player's health, and the player dies when this reaches zero. HP decreases when foes attack the player, or when the player takes environmental damage. HP is recovered when the heart item is picked up, but only up to a maximum value. This maximum value is increased when the player finds HP-ups in treasure chests. Maximum HP begins at 20 points, and tends to 60 points as HP-ups are collected.

- **MP:** This represents how many times the player can use magic. Similarly to to HP, this is recovered when MP orbs are collected, and the maximum value is increased when MP-ups are collected. Maximum MP starts at 3 points, and tends to 7 points.

- **Attack Power:** This is used in damage calculations when melee attacks are used. Attack Power increases as the player levels up, and is modified by the equipment currently being used.

- **Magic Power:** Same as attack power, but used for magic calculations.

- **EXP level:** Determines attack and magic power. Levels are gained by collecting EXP from enemies and treasure chests.

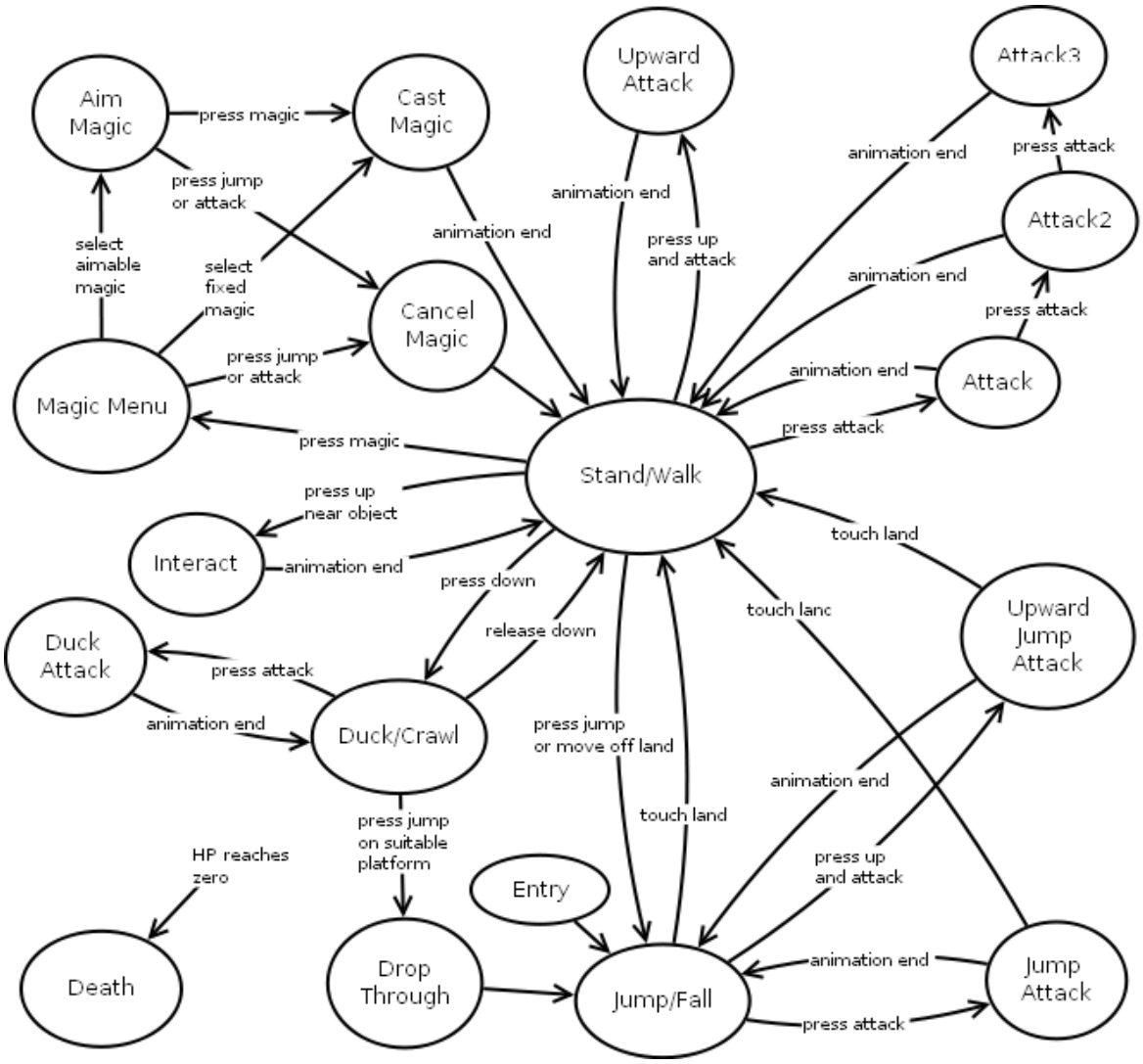


Figure 4.3: Diagram showing the possible transitions between Player states.

- **Equipment:** Includes hats, armor, and weapons. Equipment modifies the above attributes, the Player's physical appearance, and sometimes adds additional effects.

Foe

The abstract Foe class defines all the basic behaviours of enemies, and each enemy type extends this class.

Foes share several attributes with the Player class, including HP and attack power, but these are scaled based on the size of the foe, rather than experience level. The foe's size is determined during level editing (described later).

The Foe class contains many behavior variables which are to be set by subclasses. These include the

Foe's maximum walking speed, the Foe's acceleration, whether the Foe can move between platforms, whether the foe can move at all, whether the Foe can be knocked back, whether the Foe can fly, and so on. Most subclasses do not actually implement any new methods, but simply change these default parameters to make the Foe's behavior seem unique. For example, "Slimes" have a lower acceleration variable, making them appear to slide around. "Idols" have a 100% probability of jumping per frame, giving them unique behavior even though most other Foes are also capable of jumping.

However, some Foe-specific methods are defined in subclasses, such as using projectile attacks, or moving in an unusual way. These features are rarely shared with other Foes. Bosses in particular have much more complex behaviors. Bosses have 4-6 different attacks, and choose from a different subset of those based on their remaining HP. The lower their HP, the more powerful the attacks they use. Bosses also spawn other Foes frequently.

There are some very fundamental behaviors which most Foes share:

- Most Foes do not have any specialized attacks, and damage the Player simply by making body contact with him.
- Foes that do have specialized attacks usually use them when they are within attacking range of the player.
- Foes all have a field of view, represented by a circular, invisible MovieClip. Foes remain inactive until the Player moves within this field of view, or attacks them.
- Foes that have been attacked by the Player will become permanently aggressive and begin attacking the Player, even if he is not within their field of view.
- Foes which can move freely always accelerate in the direction of the Player, unless this is overrided. Horizontal and vertical acceleration is calculated from the ratio of the horizontal distance to the vertical distance, between the Foe and the Player.
- Most Foes are knocked back when they are hit by an attack from the Player. The direction of the knock back is the exact opposite of the attraction just mentioned above. The speed of the knock back depends on the type of attack used by the Player.
- All Foes which do not fly and are not stationary, are effected by gravity and collide with Platforms in the same way the Player does.
- Foes which do fly do not interact with Platforms at all, and can freely pass through them.
- Foes can also be damaged by the environment, in the same way the Player can. Foes can even hurt each other with certain projectile attacks. This gives the player some battle strategies which do not involve directly attacking the Foes.
- Foe projectiles are also Foes, although they are always created dynamically, unlike regular foes, which are usually created during level editing.

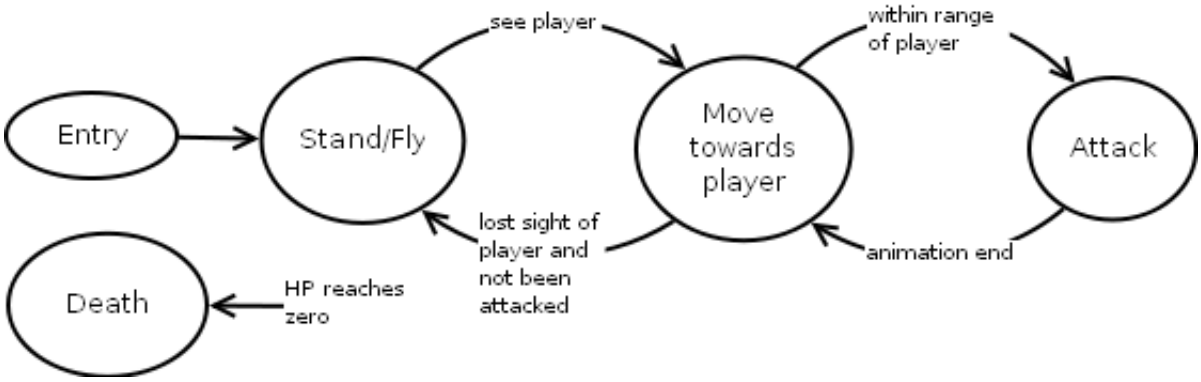


Figure 4.4: Diagram showing the state transitions of a basic foe.

Platforms and Landforms

Platforms and Landforms are chunks of land which the Player and non-flying Foes can stand on, and they make up the majority of objects in any level. These are never created through code, but only in Flash Pro's drawing environment. Collision with Platforms and level editing is described in detail in the next chapter.

The only difference between Platforms and Landforms is that the Player can pass through Platforms horizontally, and vertically too (if jumping while the down key pressed), while Landforms are completely solid. For these reasons Landforms are placed near the bottom of Levels, as there is no need for the Player to move below that point. Platforms and Landforms are both referred to as Platforms in the rest of this report, unless otherwise noted.

Some Platforms also have a special effect defined in their subclasses. These effects are called when contact is made with the Platform, and sometimes also trigger an animation. These effects may differ slightly between the Player, the Foes, and Items, and there is a different method for each. Some examples include:

- **Mushroom:** Bounces the player high into the air upon contact. Allows the player to reach higher platforms than normally possible through jumping.
- **Cloud:** A regular platform that disappears if the Player stands on it for 2 seconds, and then reappears a few seconds later.
- **Ice:** Decreases the player's horizontal acceleration and deceleration.
- **Lava:** Damages the player if he is in contact with it for 2 seconds.

Treasure Chests, Coins, Signs and Portals

These are objects which contain very little code, and are mainly used as visual placeholders for specific events, which are triggered when the player makes contact with them:

- **Treasure Chests:** These are activated when the player presses the up key while standing near them. They play a brief opening animation and then an item is obtained, which is displayed in a small speech bubble. The following items can be found in a treasure chest:

  - Coins, which are later used to buy spells.
  - Equips, which alter the player's attributes.
  - EXP, which level ups the player.
  - HP-ups, which improve maximum HP, and MP-ups, which improve maximum MP.
  - HP or MP can be fully recovered.

  Treasure chests which recover HP and MP can be opened again if the a level is revisited, but all other treasure chests stay opened permanently. The status of each treasure chest is stored in the SaveData class.

- **Coins:** These are collected when the player makes contact with them. Coins are accumulated across many levels, and are later used to buy magic spells in the Shop Menu. There is 100 coins in each level. Coins which have already been collected reappear when the level is revisited, but become silver instead of gold. Silver coins do nothing, merely indicating the original state of the level. The status of each coin is stored in the SaveData class.

- **Signs:** When the player is near a sign, a speech bubble containing text appears within the HUD. Signs are mainly used for giving tips and information to the player in the first few levels.

- **Portals:** When the player presses the up key near a Portal, a screen-transition animation is triggered, and the player is teleported to another area of the level.

- **End Portal:** This is similar to a regular Portal, but triggers a successful Game Over instead of an area transition. This is the end of a level.

Item

The Item class defines items that Foes drop when they are killed, and are picked up by the Player when contact is made with them. Items are a reward for defeating enemies and allow the player to recover his starting resources and continue playing. They simply fall until they land on a platform, then disappear after a few seconds if untouched.

Items come in two varieties:

- **Hearts:** These recover the player's HP by a small amount. Hearts are a good metaphor for health.

- **Green Orbs:** These recover the player's MP by 1. Green is also used in the HUD to represent MP.

Level

The Level class holds all of the MovieClips mentioned above that a player can interact with; platforms, foes, treasure chests, coins, signs and portals. It also contains non-functional decorative MovieClips such as the large background graphics, and small objects like trees and rocks.

The Level class is responsible for initializing all of these objects. When each object is constructed, it calls the appropriate method in its parent, an instance of the Level class. The objects are put into arrays of foes, platforms etc. The game loop can then iterate through these arrays and call methods on these objects. Creating individual levels should require very little coding, instead relying on placing objects onto the stage within Flash Pro's drawing environment. How this has been implemented is described in detail in the next chapter.

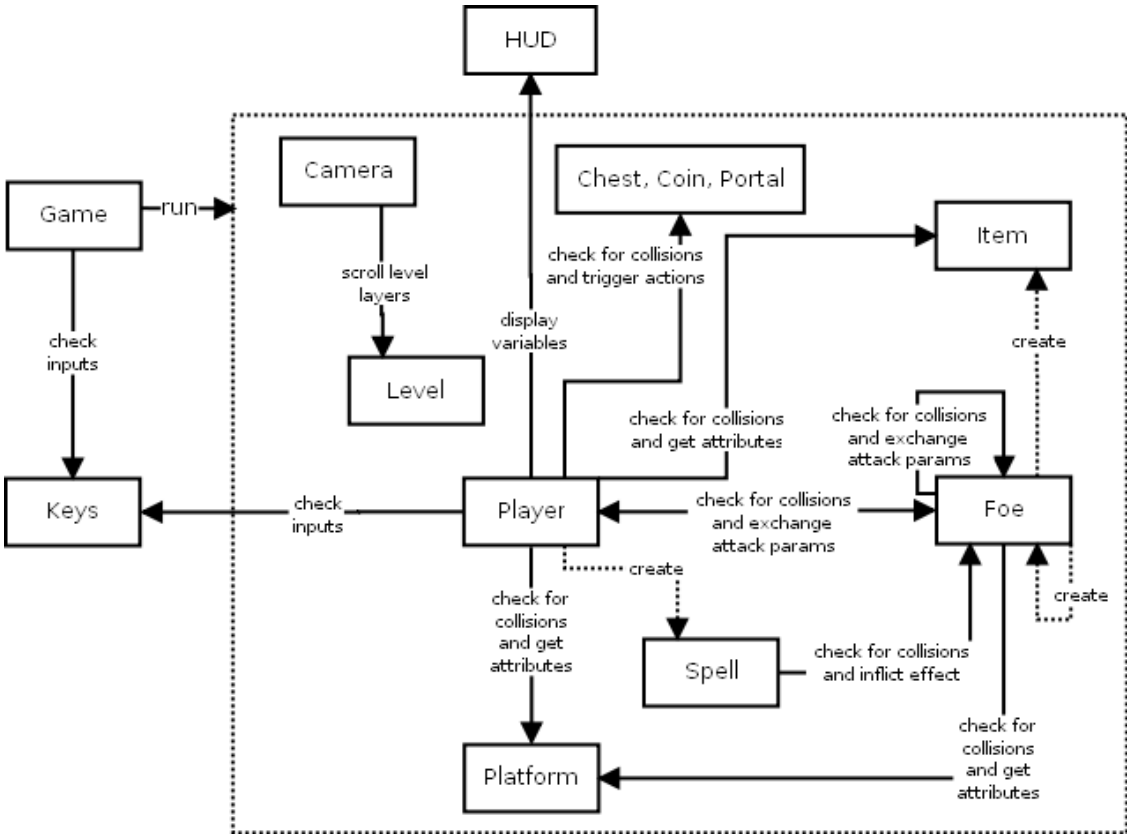


Figure 4.5: Diagram showing what messages classes constantly exchange during gameplay.

HUD (Head Up Display)

This MovieClip occupies the top layer in the MovieClip tree, therefore always being completely visible. It displays the Player's HP, MP, EXP, number of coins and chests collected, hotkeys and the current level, and allows the user to keep track of these variables as they are essential to many gameplay elements. The Player and Game classes call the appropriate methods to update these displays whenever variables change.

The pause and magic menus and the Spell instances which the player creates are also nested within this MovieClip. Additional effects such as speech bubbles and the transition animation between areas are also included here, and are made visible when needed.

Basically, this class is a container for any MovieClip that must appear in front of the Level and Player, and for any functions that they require.

4.4 Static Classes

Static classes implement the singleton and factory patterns and are used to group together global variables and utility methods, which are used by many other classes. Chapter 19 of Game Architecture and Design[11] emphasizes the importance of these particular patterns when developing games.

Game

The Game class contains the main code loop, which is executed on every frame during gameplay, but not during menus. I will refer to this as the ”game loop”. The game loop calls methods on all MovieClips that have to constantly perform some action or be updated. Pausing the game simply involves pausing this loop.

The game loop iterates through all the MovieClips in the current Level instance, calls methods to update them, and also triggers checks for collisions with the Player. Most MovieClips also have their playback stopped by default, and depend on the Game loop to advance their playhead using nextFrame().

The game loop checks for user input that would open a menu (pause or magic), and then redirects future inputs to that menu, instead of to the Player class, which handles the majority of inputs.

The Game class also contains references to MovieClips which are accessed from many other classes, and should effectively be seen as global objects. These include the root timeline, the level, the player and the HUD. This reduces the need for reference duplication, and makes deleting these objects at the end of a level much easier, reducing the chance of a memory leak.

Keys

The Keys class holds the current key configuration and handles raw key events. Reassigning game keys simply involves changing the keycodes which are stored. Keycodes are unique identifiers of each key on a keyboard, but their positions may vary depending on the type of keyboard.

When a key is pressed or released, an array of booleans is updated, with the key’s keycode being used as the index number. The Player class can then use this array to check if a certain key is being pressed during a given frame.

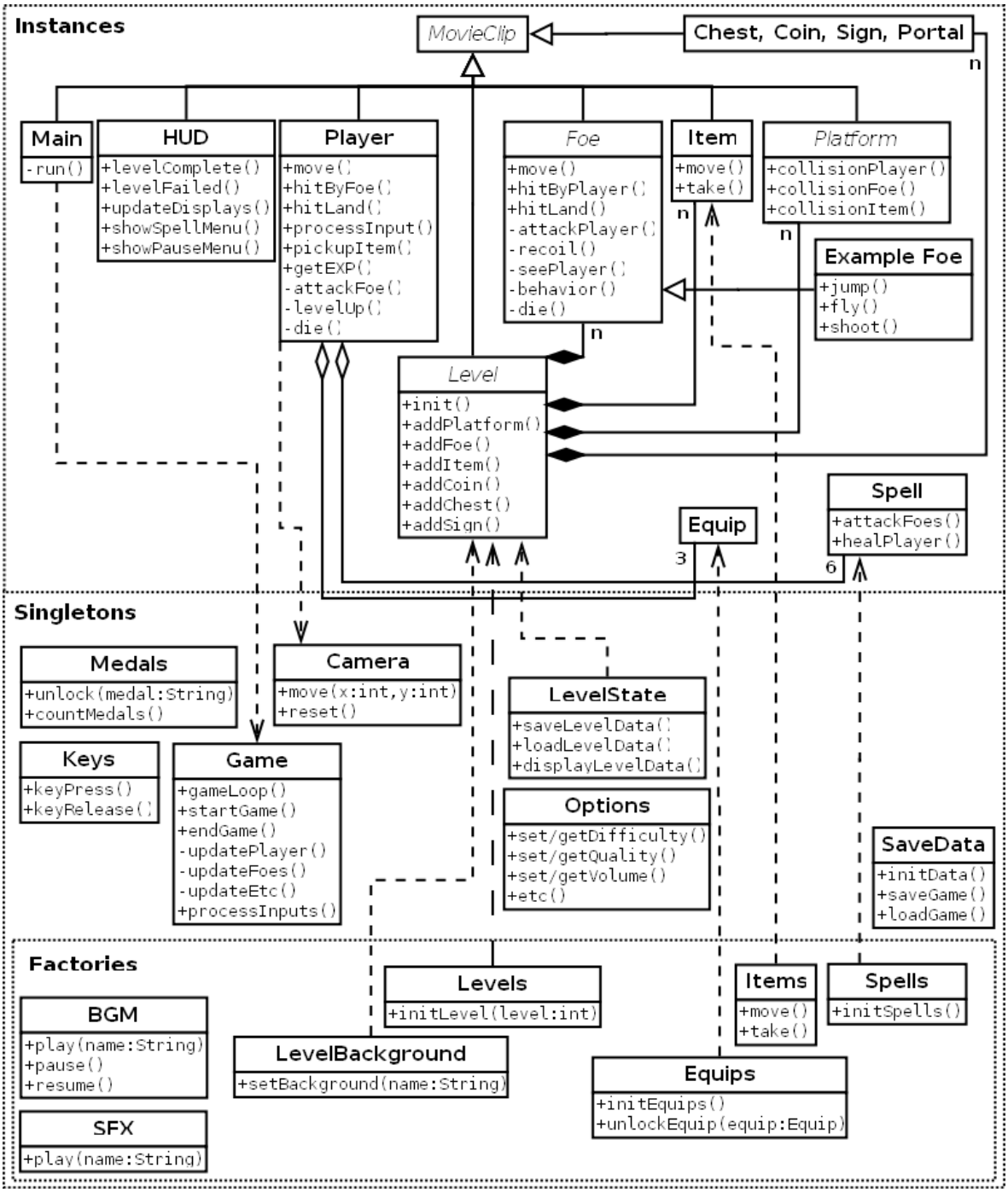


Figure 4.6: Diagram showing all game classes and how they are related. Only strong dependencies are shown.

Additionally, key press events are passed along to the Game class during gameplay, which uses them to open menus and navigate the pause and magic menus. Similarly, when the game is in Menu mode, the key events are sent to and handled by the appropriate menus.

During gameplay, the active keys and their associated actions are as follows. Refer back to the Player class for descriptions of some of these actions.

- **Movement:** 4 directional keys used to navigate menus and control the movement of the player character, and perform a few other actions. These are set to the arrow keys by default.
- **Jump:** Used to jump with the player character. Set to A by default.
- **Attack:** Used in combination with other keys to perform melee attacks with the player character. Set to S by default.
- **Magic:** Used to open the magic menu and to select a spell during gameplay. Set to D by default.
- **Magic hotkeys:** 3 keys used to bypass the magic menu and to use 3 different spells. Set to Q, W and E by default.
- **Pause:** Used to open and close the pause menu during gameplay. Set to Escape and P. Cannot be reconfigured.
- **Cancel:** Used to close menus, or return to a previous menu screen. Set to Escape. Cannot be reconfigured.
- **Accept:** Used to select menu items. Set to Enter and Space. Cannot be reconfigured. The attack and jump keys can also be used in the same way during menus.

These keys have been chosen as QWE and ASD are safe choices for foreign keyboards, while other keys (such as X) are sometimes found in different locations. WASD and the surrounding keys are commonly used in many PC games. P and Escape for pause are also standard.

Levels

The Levels class is a factory that holds a list of information required to initialize the contents of each level. This includes the backgrounds to be used, the links between level areas through portals, the text used on signs, and the contents of treasure chests. Each game level extends the Level class, but does not have an associated class file, as only the above information is used to differentiate levels in terms of code, so I feel this method makes the data more maintainable.

LevelBackground

LevelBackground is another factory related to initializing Level instances. It holds information about the backgrounds to be used in levels. It takes in a string such as ”forest day”, and sets the parameters needed to construct that background, reducing the amount of information to be stored in the Levels class. The parameters include which graphic to display for each layer of the background. Starting with the furthest from the Camera, the background layers are:

- The sky, which is a very tall graphic, and only scrolls vertically.
- The distant background, which is only visible while the Player is in the lower parts of a Level.
- The near background, which is the same as the distant background but closer to the Camera, and is therefore scrolled faster.

The scrolling of the Level and background layers is described in the Camera class section.

LevelState

The LevelState class holds the current state of all levels, even once their instances have been removed. Information about which coins have been taken, which enemies have been killed, and which treasure chests have been opened, is stored in multidimensional arrays. This information is loaded when a level is instantiated and initialized, or when displaying level data on the Level Select menu, and is saved whenever the Player exits or completes a level.

SaveData and Options

SaveData and Options, along with LevelState, hold all game data that is to be saved between sessions. SaveData deals with storing overall game progress, collected power ups, unlocked Equips and other miscellaneous variables. Options stores any settings the user has changed, and these are described in the Options Menu section.

Saving is done using Flash's Shared Object files (.sol). Data is saved whenever a Level is exited by the user, or whenever a notable update has been made, such as obtaining a new Equip. Data is loaded only once; when the game is fully loaded.

Figure 4.7 shows what data is saved, and how it is modified.

Camera

The Camera class controls the vertical and horizontal scrolling of the Level, and therefor all objects within it, as the Player moves. The camera is usually exactly centered on the player, except briefly when a player is respawning at another location, in which case the camera scrolls to that location.

The background layers are scrolled at different speeds to give the impression of depth; the lower down a layer is, the slower it scrolls. The sky and cloud layers do not scroll horizontally at all.

Object coordinates are calculated from the Player's offset from his starting position, rather than moving objects proportionally to the Player's movement. Platforms and scenery also have their coordinates rounded to the nearest pixel. These are to avoid rounding errors building up, and to avoid minor jittering.

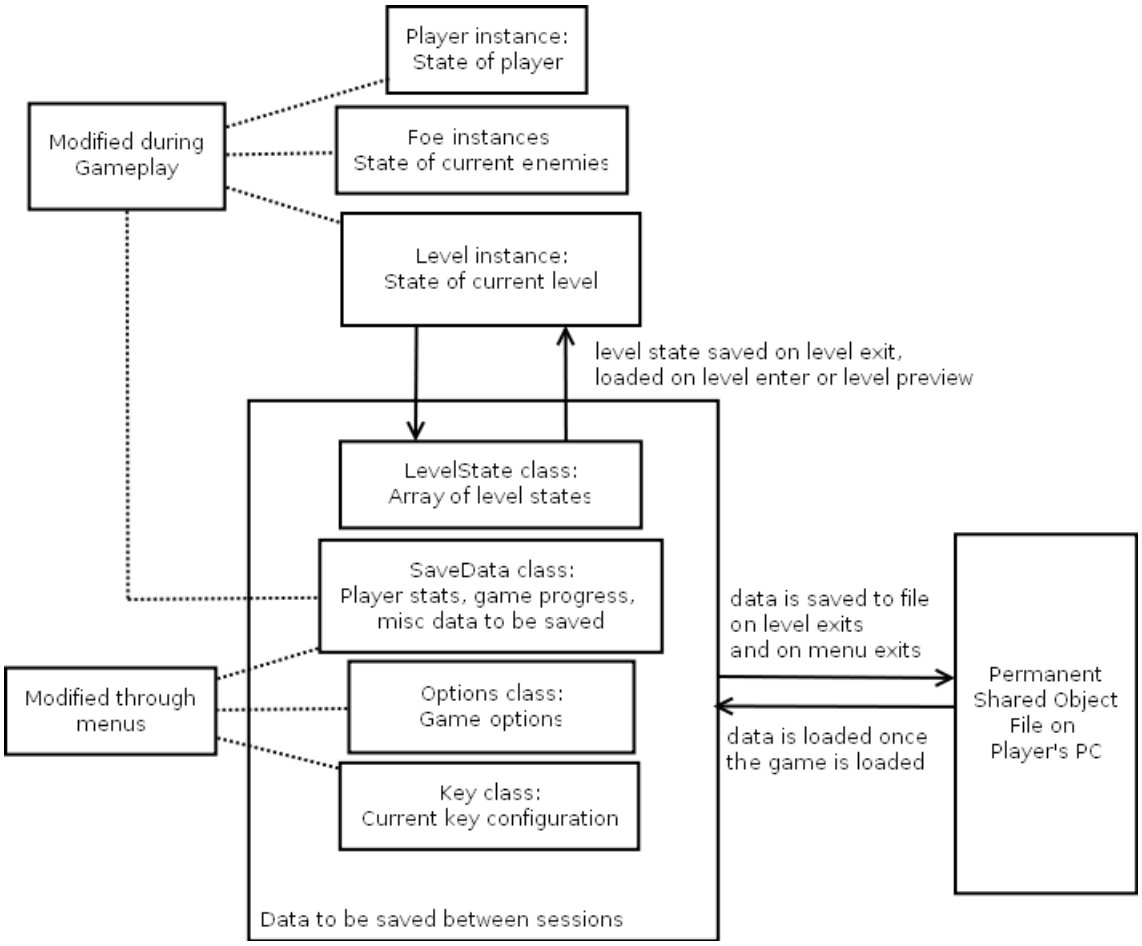


Figure 4.7: Diagram showing when data is modified and saved.

This class also moves the Player back to the Level entrance if he falls beyond the bounds of the Level.

Equip and Equips

The Equip class defines an item of equipment that the Player can use. An Equip has a name, description, icon identifier, attack and magic modifiers, and sometimes an additional property. The Equip class is the only class which is instantiated but is not a MovieClip, as its physical appearance is stored within the Player class.

The Equips class initializes all 15 equips, giving them stats and descriptions. It also sorts them into an array so they can be iterated through when being displayed in a menu.

The Player class uses the attributes from the 3 equips currently equipped (a hat, armor, and weapon), and these are used in damage or movement calculations, and also specify which graphics to display.

Spell and Spells

Similar to the above, Spell defines a spell that the Player can use, while Spells is a factory for initializing all of the spells. Spell attributes include a name, description, cost, and special effects.

When spell animations are instantiated, they are stored in the HUD, and check for collisions with Foes.

Items

The Items class is a factory that creates Item instances at given coordinates, given a probability of occurrence.

SFX and BGM

The SFX class controls the playback of sound effects. Sounds are played by calling the play(”soundName”) method. This is mainly used by the Player, Spell and Foe classes, and all menus. This class limits the amount of sounds that can be produced to 3 per frame, to avoid excessive overlap of sounds which can lead to uncomfortable noise. The sounds are all stored in an instance of a single invisible MovieClip, with a different sound on each frame. The playhead of this MovieClip is moved to the requested sound to play it. I felt this was easier and cleaner to implement than giving each sound a class name and calling their constructors, although it may less efficient.

The BGM class is similar to the SFX class but instead it controls background music, which is constantly playing on loop. In addition to play(”musicName”), BGM includes pause() and resume() methods for when the music must be paused. Transitions between music should not be instant, and this class therefor slowly fades one track to another. This is done by lowering or increasing the volume slightly on each frame until the maximum or minimum is reached, and this requires use of the Main class’s enter-frame listener.

Medals

The Medals class contains a list of all in-game achievements[1] that the player can earn. Each ”medal” has a name, a description of how to obtain it, and a unique icon. Usage of this class involves calling the unlock(”medalName”) method once the requirements to obtain it have been met within other classes. This displays a small pop-up window showing that the medal has been unlocked, and updates the list of unlocked medals in SaveData.

As mentioned in the previous chapter, achievements/medals are a form of reward for accomplishing tasks that often don’t relate to a game’s main goal, and have in the last few years become a standard feature on most gaming platforms. In this project, medals are also sometimes used to explain new

[1] http://en.wikipedia.org/wiki/Achievement_%28video_gaming%29

concepts in a subtle manner. For example, when the player encounters a new type of platform, they earn a medal that describes the platform's behavior in its description.

4.5 Graphical User Interface

The majority of the game's menu screens are contained within a single MovieClip, which is the only visible MovieClip during Menu mode. Each menu is on a different frame within this MovieClip, and the playhead is moved between them as necessary.

Menus which are not contained in this MovieClip are the Game Over, Pause and Magic menus, which instead appear within the HUD during Game mode.

Menus can be navigated using either the keyboard or the mouse, due to popular user request. The arrow keys are used to select choices, Enter and Space are used to choose items, and Escape is used to cancel an action or exit a menu. This is standard in most keyboard based games.

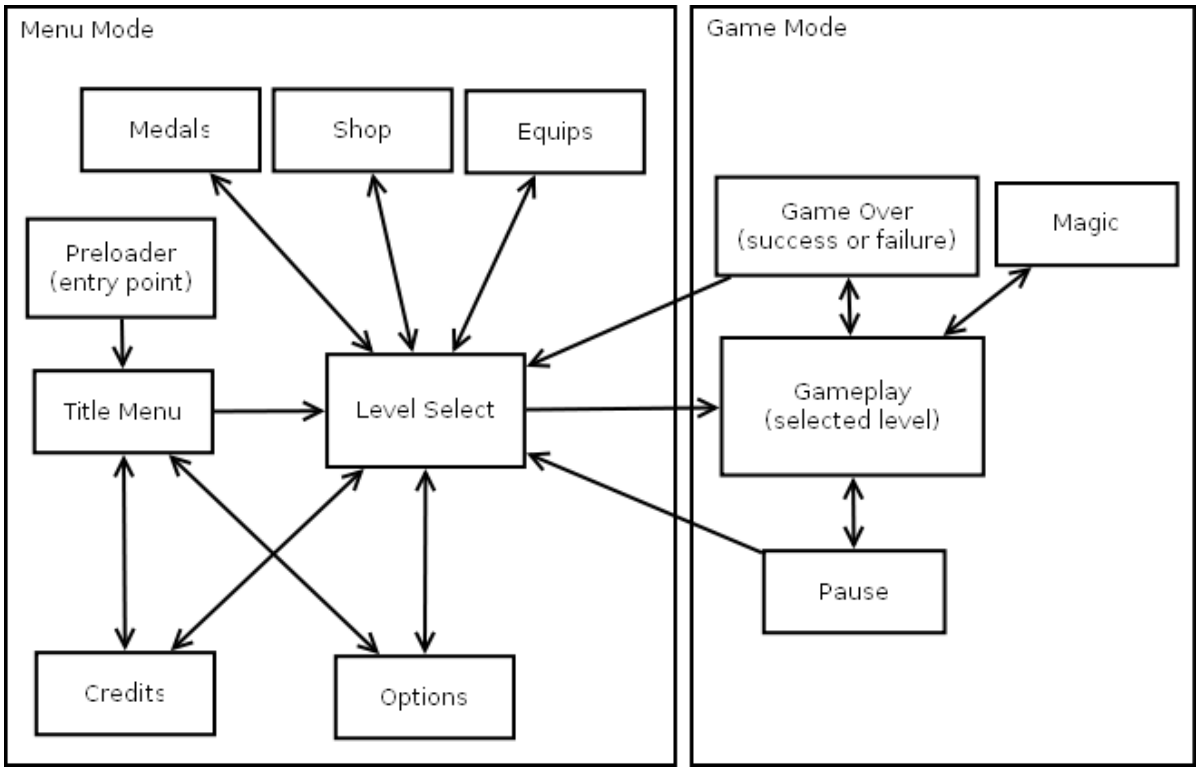


Figure 4.8: Connections between the game menus and gameplay, from the user's perspective.

Figure 4.8 shows how the menus connect to each other, and how a player progresses through them. Gameplay begins from the Level Select menu, and stops through the Game Over or Pause menus. The rest of this section explains the purpose of each menu.

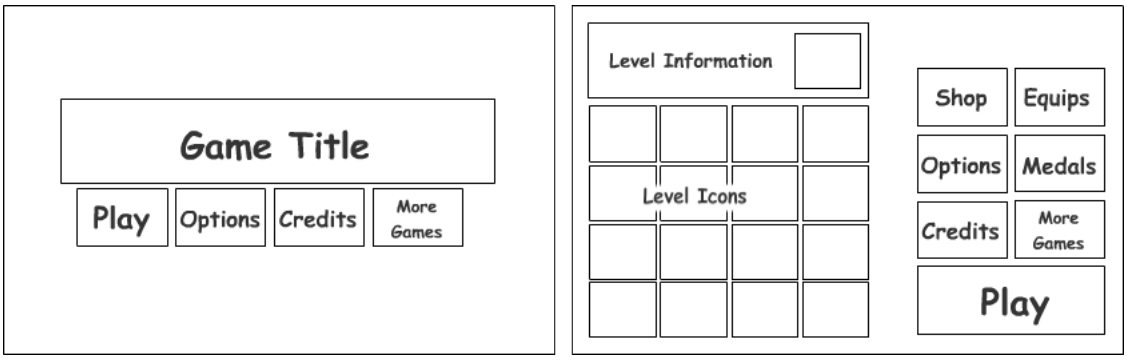


Figure 4.9: Wireframes of the title and level select screens.

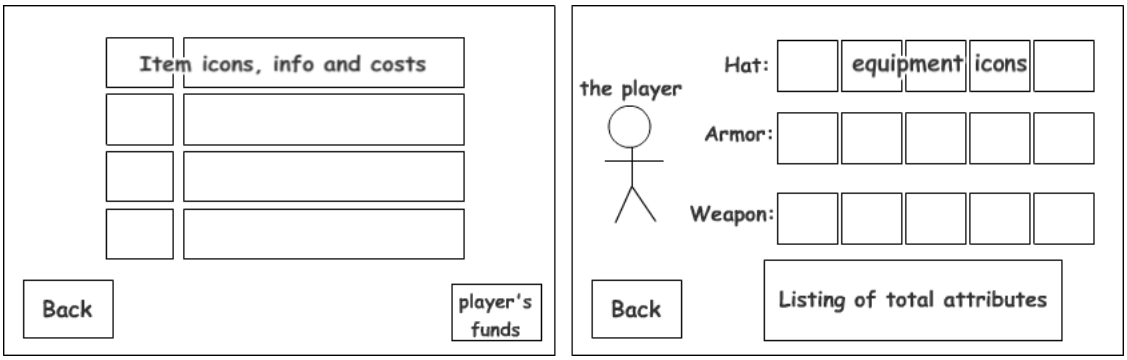


Figure 4.10: Wireframes of the shop and equipment screens.

Preloader Screen

The Preloader is the first screen that the player sees. It appears before the game is fully loaded, and displays a loading bar, the game title, a version number, and some advertisements to watch during loading. Once the game is loaded, a play button appears.

The preloader is a vital part of any Flash game, as without it, the player would see a blank or otherwise useless screen while the game loads. Once the game is loaded, a play button appears, which takes the player to the title screen, after a brief splash animation[2] from the sponsor.

Intro Animation

After the preloader and before the title screen appears, a short introduction animation plays which introduces the main characters and covers most of the story. This helps absorb the player and give them a goal to work towards. The animation is skippable by pressing any key, as players may not wish to watch it more than once.

Most Flash games do not have intro animations, but the ones that do stand out more as a result. Robot

[2] http://en.wikipedia.org/wiki/Splash_screen

Dinosaurs[3] is a good example.

Title Screen

The title screen is the first time the player is given control after the game has fully loaded. It displays the game title again and provides the initial options available to the player. The player may either start the game, or visit the options or credits screens. These features are all available on the level select screen, along with many others, so there is never a need for the player to revisit this menu. The advantage here is that the title screen's available options are limited and not as overwhelming to a new player; all features that are not useful at this point are not available.

Title screens are common to many media platforms; games, DVDs, webpages, and so on.

Level Select Screen

The level select screen is the central hub of the game, and is visited at least once in between game levels. Here the player can select any unlocked level to play, or just view the progress made in that level. Level progress includes the number of coins and treasure chests collected, the difficulty setting the level was completed on, and the fastest time taken to complete the level. A new level is unlocked every time an uncompleted level is completed. There are 22 levels in total, and each has a unique icon. Locked levels have a question mark as their icon. Level data is loaded from the LevelState class when needed.

The player can also access the options, credits, medals, shop and equips screens from here. Each of those screens has a "back" button, leading back to this screen.

Shop Screen

The shop screen is where the player can purchase new magic spells using collected coins. Each spell displays an icon, a name, a description and a price tag. The player's current funds are also displayed on screen. When a spell is purchased, a confirmation animation is played. Purchased spells are recorded in the SaveData class.

As the name suggests, this screen's interface metaphor is that of a shop where items are displayed and bought.

Equipment Screen

The equipment screen is where the player can choose which equipment to use. Equipment which has been unlocked shows a unique icon, while locked equipment show a question mark instead, similarly

[3] http://www.newgrounds.com/portal/view/490036

to how levels are displayed. Equipment is divided into three sections: hats, armors, and weapons. The attributes of each selected equip are shown next to them, while the bottom of the screen shows the total effect that all three equips have on the player's HP, attack and magic attributes. This screen also shows an enlarged graphic of the player character, as his appearance changes with each item of equipment. Selected and unlocked equips are saved to and loaded from the SaveData class.

The interface metaphor used for this screen is that of changing room. The user picks out clothing to wear, and leaves the rest behind until his next visit.

Options Screen

The options screen allows the player to change the following settings:

- **The key configuration:** When this is selected, the user is prompted several times to press which key he wants to use for each action. This feature allows users to adjust the controls to their preferences, especially if they are left handed, use a foreign keyboard, or want to use a gamepad.
- **Reset the key configuration:** This feature is important in case the player sets his keys to something even worse than the default settings.
- **The graphic quality (high/medium/low):** A standard Flash game feature, and built-in feature of the Flashplayer. This setting has a large effect on performance.
- **The background (on/off):** Disabling backgrounds improves performance slightly, and for some players, visibility.
- **The music (on/off):** This is a standard feature in Flash games as many users will wish to play their own music, or mute the audio altogether.
- **The sound effects (on/off):** Similar to the above, but it is important to separate these two to fit more preferences.
- **The game difficulty (easy/normal/hard/epic):** This is an important setting to make the game accessible for both experienced and casual players. It works by modifying the attributes of all enemies.
- **Delete all saved game data:** This is another standard feature for any game, but especially for those with only one save slot. The user cannot have two separate sets of saved data here, so this allows him to play from the beginning again. Manually deleting saved games (shared objects) in Flash is possible, but it is often difficult to find where the files are, so this option is helpful.

All of these settings are stored in the Options and Keys classes, and are accessed from many other classes when needed. For example, the sound settings are checked before playing sound effects, and the difficulty setting is checked when initializing enemies.

Medal Screen

The medals screen displays all of the medals which the player has unlocked, in effect acting as a trophy case. Each medal has a unique icon and a description.

Credits Screen

The credits screen simply displays information about the development team, the sponsor, and contains promotional links to my other games (also on the sponsor's website), the soundtrack (which is downloadable) and the team members' personal blogs. This allows the player to find more content or information related to the game, if they are interested, and helps trap users.

Gameover Screen

The gameover screen appears when the player fails or completes level, and offers the player the option to retry the same level, or return to the menu select screen. It also gives a link to the sponsor's website, as this is the point where the player is most likely to stop playing.

Pause Screen

The pause screen appears when the player presses Escape or P during gameplay, and offers the player the option to exit the level, and a subset of the options available in the options menu, as sound and other options may need to be changed frequently. While the pause screen is opened, the main game loop is paused, and music is muted. The pause screen allows the user to take a break at any time, which is especially important for Flash games as the user may be taking a short break from work or other tasks.

4.6 Gameplay Visuals and Audio

This section covers how visuals and audio have been used to maximize usability, by providing feedback, real-world metaphors and pleasing aesthetics to the user.

Feedback

Displaying understandable feedback to the user is key in any software application, but in games, it has the additional requirement that it must also not interrupt or slow down the gameplay. Many conventions have been developed over time, and I have attempted to follow as many as possible:

- If a player or enemy has been damaged by an attack, they will flash white (many games also use red) once to indicate this, and the player/enemy will also be physically knocked back. A number will appear near them to display the number of HP which have been deducted (typical in RPGs). A brief spark-like animation will also be displayed at the point of contact. Finally, a sound effect is also played. All of these are important to emphasize that a successful collision was made.
- If a player or enemy is temporarily invincible, they will flash repeatedly during this time.
- If the player is low on HP, he will slowly flash red (some games also use audio to indicate this, but this is often annoying).
- If the player has recovered HP or MP, he will flash, display numbers and play a sound effect similarly to when he is attacked, but this effect is different enough to be easily distinguishable.
- The colors used to represent different variables are consistant; red is used for HP, green for MP, and yellow for EXP and treasure. Red is universally used for health, and the other two aren't unusual.
- Object interactions all trigger unique animations and sound effects.

These examples are consistent in most platformers and RPGs, but the games I had in mind were the Megaman[4] series.

Real World Similarities

Most in-game objects have been designed to represent real-world objects and the associations people have with them, and this is a common theme in adventure games. No video game experience is needed to recognize their functions. For example, treasure chests contain treasure. Sign posts display information. Sharp objects and high cliffs are dangerous. Fire is hot, ice is slippery. Kids are brought up from a young age to become familiar with ideas like this, as these objects appear in toys, books, video games, and often in real life too.

A non-gamer can look at Adventure Story and have some idea of what is happening at first glance. This may sound like an obvious design decision, but many games are so visually stylized or otherwise unusual that it may be difficult for non-gamers to understand them.

Varied Environments

Environmental variety helps support the feeling of exploration and progress, even if the environments do not change very much functionally. Different backgrounds reduce repetitiveness and provide eye candy. Even something as simple as changing the color of the sky in each level helps remind the user that they have made progress.

[4] `http://en.wikipedia.org/wiki/Megaman`

Different environments also allow different types of enemies and obstacles (ice, lava etc) to be used, and are thus a convenient source of inspiration for content. The environments I have chosen and implemented are all radically different from each other. They are as follows:

- **Forest:** A very green and cheerful environment, covered in trees and vegetation. The enemies here are rather cute and harmless.
- **Desert:** A mostly yellow and brown environment, consisting of a rocky landscape with ancient ruins. Occupied by gunslingers, living cacti and living statues.
- **Tundra:** A mostly white a blue environment, with hidden enemies under the snow and icicle projectiles being used to attack the player.
- **Volcano:** A dark and gloomy environment, with fire based enemies and explosives, which fits the increased difficulty of the game by this point.

Varied and colorful environments are particularly apparent in the platformer and adventure genres, and games which do not include enough variety, such as Super Mario Sunshine[5], are often criticized.

Use of Music and Sound

Each environment has a unique background music track that loops constantly. In addition to this, different tracks also play during boss battles and on the menu screens, for a total of 6 tracks. The purpose of background music is to add atmosphere and support what is happening on screen, and each track used is quite different in style from the others.

Sounds effects are used whenever animations require them, and when feedback should be given to the player during menu selections, or performed actions.

The only difficulty in implementing sound effects is to make sure that they don't become too annoying or repetitive. For these reasons, I reduce the volume of sound effects that are played often, and make sure they are not particularly noticeable in how they sound. Where possible, it also helps to add some variation. For example, each sword slash has a slightly different variation of the same sound effect. Sound effects are also not used in some situations where it would be realistic to have them. For example, jumping and landing produces no sound effects, as this quickly became annoying.

Clear and Consistent Level Design

Clear level layout is key, as it shows the player the scope and boundaries of the game. The player should be able to tell what actions are possible within the game just by looking, rather than through trial and error.

[5] `http://en.wikipedia.org/wiki/Super_mario_sunshine`

To support this, even though level scenery may be aesthetically varied, platforms appear in limited varieties. For example, in the Forest environment, all regular platforms are either covered in grass, or are wooden planks. They also all have flat surfaces. Coins or chests are often placed on new platforms to indicate that the player can stand on them. These allow a user to quickly learn to distinguish between the background and the foreground, and this is vital in any game, but platformers in particular. Objects that look similar or identical should always behave in the same predictable ways.

The distant background is also blurred slightly and tinted to the color of the sky, to avoid attracting the user's attention. This allows it to be quite complicated without being distracting.

Cave Story 3D[6] for the Nintendo 3DS is a good example of a platformer where level clarity is strongly lacking due to complicated backgrounds, and inconsistent foregrounds. This is mentioned in several professional reviews[7].

4.7 Game Design and Creating Game Content

The previous parts of this chapter dealt with the game engine and menu screens. This section describes how game content is designed and the process of implementing it.

4.7.1 Level Design

Chapter 23 of Game Design - Theory and Practice[12] explains the basics of good level design. These include dividing gameplay into small chunks, evenly distributing different types of gameplay, allowing and rewarding exploration, defining clear goals which are easy to follow, and providing pleasing yet unobtrusive aesthetics. My level designs aim to support these points.

The book breaks the process of level design into the following steps, which are closely followed in this project:

- Sketch out a level concept, showing roughly what the level should look and feel like.
- Implement the land and obstacles/enemies which the player will interact with.
- Refine and balance these until they function as intended.
- Add aesthetics, such as background items.
- Allow potential users to test the level.

Levels are made up of between 3 to 6 areas, with each area extending the Level class. The level as a whole is not represented in any way internally, and is only defined by the connections between areas. The rest of this report uses "areas" and "levels" almost interchangeably for simplicity. Portals are used to travel between areas, and a special portal marks the end of a level.

[6] http://en.wikipedia.org/wiki/Cave_Story_3D
[7] http://uk.gamespot.com/cave-story-3d/reviews/cave-story-3d-review-6346795/

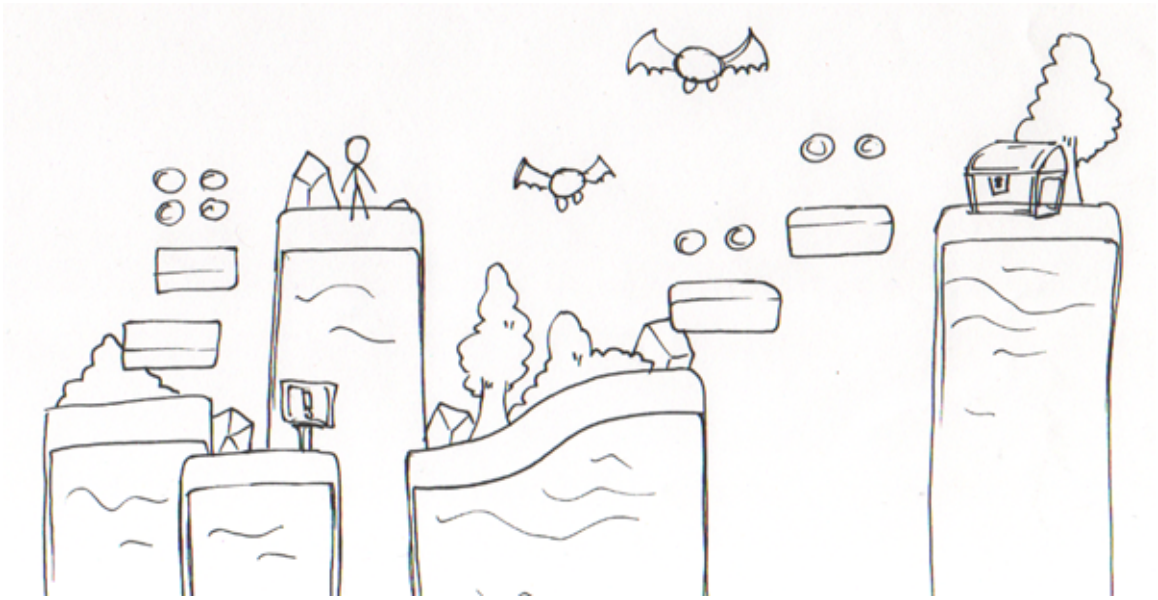


Figure 4.11: A layout sketch of a simple level.

The level layout is structured in such a way that allows for free exploration yet prevents the user from getting lost. To guide the player, the direction of the level ending is marked quite often using directional arrows. The main path through the level is fairly linear, but many branches exist. Dead ends are marked clearly so the player realizes he must go back. They are either marked with a reward which indicates the purpose of the branch, or with a skull to indicate that they are a challenging optional area. Sometimes a portal is included that takes the player back to the beginning of the branch, to save time and signal the completion of that branch. This design satisfies the requirements of both casual and more advanced players.

A level's challenge comes from the timing and control of the player's jumps. The player must jump between platforms to progress, and a failed jump will set the player back slightly. Platforms get smaller and farther apart as the game progresses, in order to increase the level of difficulty. The player must also avoid obstacles such as spikes, lava and bottomless pits which damage the player similarly to enemies.

It should take between 3 and 10 minutes to complete a level, depending on how many optional paths are taken and how many enemies are defeated. Deaths are not included in this time. If the player dies, it should be quite easy for them to return to their original position once they are familiar with the path and obstacles.

As levels make up the majority of the game's content and a large portion of the development time, it is important to simplify the process of creating them.

The aim is to create a framework that allows for what-you-see-is-what-you-get level editing. Levels would first be quickly sketched on paper to get a rough idea of the layout and contents. They would then be recreated using stock library objects within Flash Pro's drawing environment. Flash's "library" is a list of all game assets such as graphics and MovieClips which can be reused multiple times within a project. The library would be populated with self-contained objects; platforms, enemies, scenery,

and everything else that would become part of a level. These would then be placed on stage and arranged to create a playable level, extending the Level class, without any additional programming involved for most objects. The few objects that do need code to operate would have it listed in the Levels class and this would be loaded when the Level instance is initialized.

The framework requires that each interactive object sends a message to the Level instance upon it's creation at runtime, and that the Level then stores a reference to each object, for the game loop to use. Non interactive objects need not do anything at all. At runtime the level contents could also be sorted into layers based on their type (enemies would appear in front of platforms etc), however this may not be required as structured layering is already a common practice when drawing in Flash.



Figure 4.12: A level sketch showing more detail in the scenery.

4.7.2 Enemy Design

Enemies populate levels, and there are usually around 10 to 15 per area. Their purpose is to challenge the player and slow their progress through the level, and potentially force him to restart the level altogether if he is defeated by them.

Enemies are first sketched on paper, sometimes in several poses if they have more complex attack animations. They are then drawn and animated in Flash Pro, and hitboxes are added to allow for collision detection (explained in the next chapter). All enemies extend the Foe class.

The goal in enemy behavior is to make each enemy unique in some way, either in the way it moves or attacks, and to make this behavior easy for the player to memorize. Implementing new movement patterns should rely on changing the parameters inherited from the Foe superclass, which cover a wide range of properties. New specialized attacks however will require some additional methods for each enemy, but the Foe class will once again provide many utility methods to support these. This strategy keeps Foe subclasses very lightweight and simple to implement, easing the creation of new content.

In addition to behavior, all enemies will of course have unique graphics, though some will share animations with only their skins changed.

Normal enemies have up to two specialized attacks, while bosses have at least four. Attacks should force the player to react in different ways to avoid them; either by jumping in a certain way, ducking, or running away. They should be reasonably simple to predict and avoid once the player has seen them a few times. Enemies do not require any artificial intelligence, and exploiting their lack of intelligence is often a good strategy for defeating them; for example, waiting for an enemy to move into your attack range instead of moving towards it. This style of behavior is the trend in almost all action platformer games, including the Super Mario and Mega Man series.

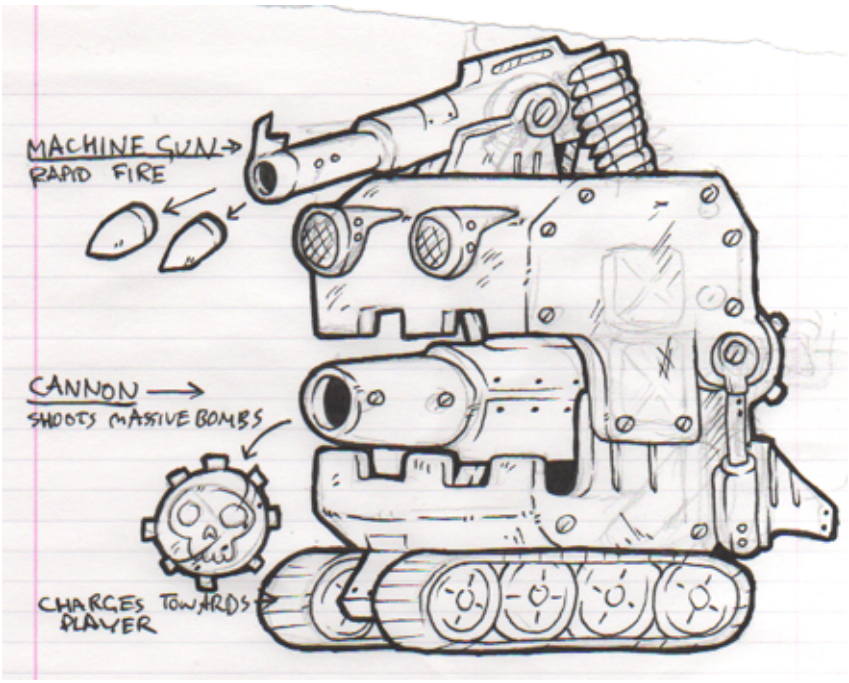


Figure 4.13: A sketch of a boss enemy and brief notes on it's behavior.

4.7.3 Equipment and Spell Design

As equipment and magic spells are the primary means of customizing the player, it is important that they are varied in their effects and are all more or less equally useful. This gives the player a sense of freedom and allows them to customize in a way that suits their playing style, making the game accessible to a wider audience and offering different battle strategies. Additionally, players wishing to replay the game will have more incentive to do so, as they can replay old levels using a different "build" (a set of customizations).

There are 15 items of equipment in total; 5 hats, 5 armors, and 5 weapons. Each group has an equip that specializes in one of the following areas: melee attack power, magic power, HP count, and movement, and a default equip with balanced attributes. By selecting a mix of these equips, the player can decide to what degree he wishes to specialize in different areas. If the player prefers slashing

enemies in close-range combat, he will use more attack-oriented equips. If he wishes to have an easier time with the platforming aspects of the game, he will use equips that improve his jumping and movement.

There are 6 spells in total, with 3 versions of each. Each spell has a unique animation and effect, and is used best in different situations, with different combinations of equipment. Each version of a spell is more powerful than the previous. Including different versions of spells allows a greater number of upgrades with little extra development cost, and also makes the effect of these upgrades easier for the player to understand. Eventually the player can acquire all spells, but the order in which he acquires them can make a significant difference. The spells and their optimum usage are as follows:

- **Heal:** Defensive spell that recovers the player's HP. More useful to inexperienced players or on harder difficulty settings.
- **Temper:** Increases the player's attack and defense attributes temporarily. More useful when damage is being exchanged frequently, such as when fighting bosses and large groups of enemies.
- **Seiken:** Fast and concentrated offensive magic. Does not pause the game in order to be aimed like the other offensive spells, and offers more fast-paced gameplay. Works best against large, slow-moving enemies.
- **Fire:** Offensive spell that lowers the targets' defense attribute, making following attacks more damaging. Works best for setting up combos with an attack-based build.
- **Ice:** Versatile spell that stops enemies from moving for a given time. Effective against fast-moving enemies, and works well with any build. Can be used for running away from enemies or making an opening to attack.
- **Bolt:** Offensive spell that deals massive damage to the enemies. Works best with a magic-based build and is particularly effective at killing hard-to-reach enemies quickly due to it's range.

4.7.4 Tutorial Level

The first level of the game serves as a tutorial that introduces concepts and guides the player through various obstacles. Experience has shown that the majority of players of Flash games (and the users of many other types of products) never read any instructions. Instructions are instead presented through pictures that display a graphic of the player performing an action, and the keys associated with that action, so any player can understand the game controls without having to do any reading. To progress through the level the player is forced to use each action once it is presented to them. For example, the player must learn to jump before overcoming the first obstacle in front of him. By the end of this first level the player will have experience of the basic game mechanics, and will be prepared for the actual challenges in the upcoming levels.

The second and third levels also include a few tutorial elements, but these are less central and were included later as to not overwhelm the player.

Many similar console games do not give the player any instructions, leaving the player to experiment and learn the controls on his own. This is not suitable here for several reasons. The first being that console games use gamepads which have a limited set of buttons, while the majority of keys on a keyboard are not used in PC games. The second reason, is that Flash games are designed to be played in short sessions, so the player will not be motivated to play a game which takes time and experimentation to get into.

4.8 External APIs

This project takes advantage of several different APIs.

The first is Kongregate's API, which is used for recording highscores and game progress, and these are publicly displayed on their website. It unfortunately does not offer any particularly useful views of this data, and is mainly used for competition between players, rather than for evaluation.

Although not technically part of the API, Kongregate also records traffic data from sponsored links by using query strings to identify the game. Kongregate records which page on their site is visited, what page the user came from, what country the user is located in, and the bounce rate. I do not have direct access to this data, but can receive it upon request.

The second API is Mochi Media's Analytics API, which is used for collecting general data on game sessions. The API automatically collects data on session length and the user's location, and allows developers to track custom events as well. Chapter 6 explains what stats I have collected and how they were used in detail.

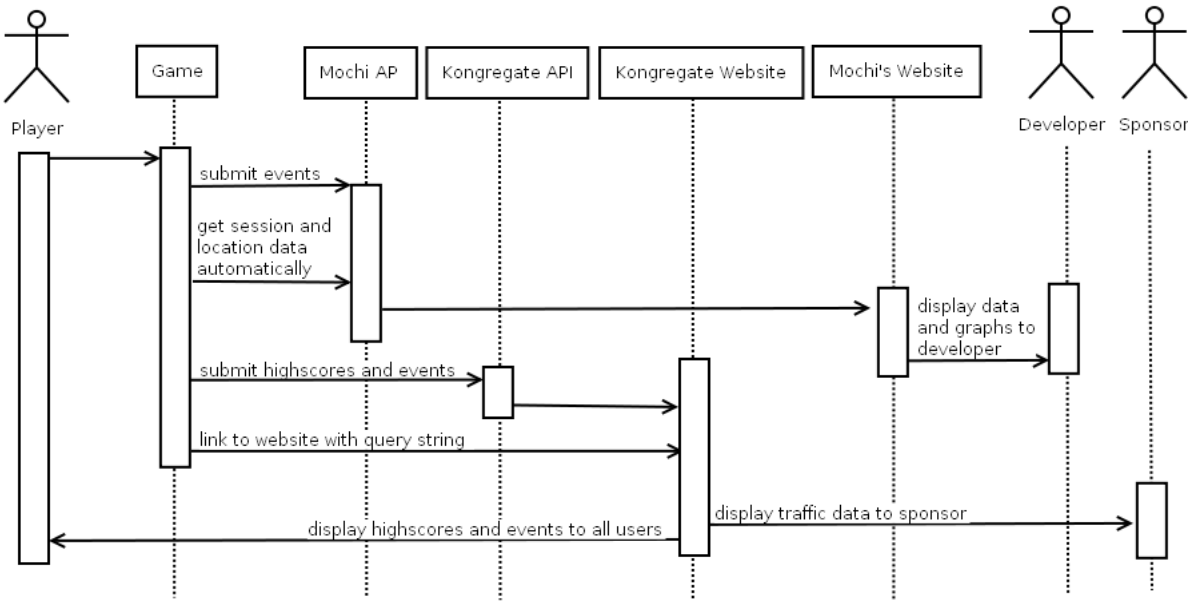


Figure 4.14: Diagram showing how data is being sent to external systems.

Usage of these APIs is simple, when the game is loaded it must first make calls to connect to them.

In Mochi's case, the API files are included in the source code, while Kongregate's API files are downloaded at runtime, and only when the game is hosted on Kongregate. To record events each API has a single method which takes in the name of the event and a value as the parameters, and these are called whenever an event of interest occurs.

In addition to the above, some other websites will also request their own APIs to be implemented for highscores.

Chapter 5

Implementation and Testing

5.1 Introduction

This chapter covers the more interesting elements of the project implementation, while ignoring any features that are already described fully in the previous chapter. It is divided into the following sections:

- **Development Tools and Resources:** A listing of tools and resources and how they are used within the context of this project.
- **Final Product Overview:** Demonstration of the final product using screenshots, and referring back to the design chapter.
- **Level Editing:** Explanation of how the level editor was implemented, and some of the possible alternatives.
- **Collision Detection:** Detailed explanation of how collision detection was implemented.
- **Testing Strategy:** Overview of the project implementation and testing cycle.
- **Play Testing:** Explanation of why play testing is essential in games, and how I have used it.
- **Demos and Prototypes:** Listing and justification of the project demos released for play testing.
- **Functional Testing:** Description of how new components and content are integrated and tested.
- **Stress Testing:** Testing done to determine the reasonable operational limits of the game engine.
- **Optimizing Performance:** Results of experiments done to determine development guidelines to improve performance.

5.2 Development Tools and Resources

This section credits tools, websites and people that have helped or been used during development.

5.2.1 Graphics and Animation

All graphics are done within Flash CS3[1] (Flash Pro 9), no other programs are used. Although this version is now several years out of date, it still does not lack any noteworthy features. The primary tools I use for drawing are a Wacom Bamboo graphics tablet[2] and Flash's basic Paint Brush tool.

Animations are mostly made using traditional motion tweening[3], there is very little frame-by-frame animation. The main advantage with motion tweening is that the MovieClip being tweened can have several graphics along it's timeline, and a line of code that selects which one to show, based on some external conditions. This is the basis for changing the player's appearance with equipment, and recycling animations for several enemies. In general, tweening is one of Flash's greatest strengths as it makes animation very quick and simple compared to other methods.

5.2.2 Programming

This entire project was programmed in Flash's ActionScript 3[4] language.

ActionScript 3 is an object oriented programming (OOP) language with similar syntax to Java. It should not be confused with ActionScript 2, which is a much slower scripting language, and is now rarely used for games.

All of the class code was done in FlashDevelop[5], while a lot of GUI related code was done on timelines in Flash Pro. Flash develop is a free ActionScript IDE which is similar to Java's Eclipse, and offers all of the features you would expect. Flash Pro is more suited for writing simple scripts to aid animation rather than writing proper OOP applications. FlashDevelop requires Flex or Flash Pro to compile projects.

Essential Actionscript 3.0 (Moock, 2007) has been a helpful guide to learning the ActionScript 3 language. However, much of the language will quickly become familiar to experienced Java users, once they learn about Flash's animation metaphor and a few other differences.

Adobe's official ActionScript 3 documentation[6] is often used to understand the more obscure features of the language. 8 Bit Rocket[7] is a good source for Flash game related tutorials, and Google searches

[1] http://www.adobe.com/products/flash.html
[2] http://www.wacom.eu/index2.asp?pid=9240
[3] http://en.wikipedia.org/wiki/Inbetweening
[4] http://en.wikipedia.org/wiki/ActionScript
[5] http://www.flashdevelop.org/
[6] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html?filter_flashplayer=11.1
[7] http://www.8bitrocket.com/

of specific issues often lead me to the forums of Kirupa [8].

5.2.3 Recycled Assets

Several classes used are modified versions of those used in Bullet Heaven. The SFX, BGM and Medals classes have been recycled and are almost unchanged, as their functions would be identical in almost any game.

Most non-functional background graphics (trees, rocks, etc) have been recycled from Epic Battle Fantasy 3[9], which is the project I worked on before Bullet Heaven. Several menu icons were also reused from the same game. All other graphics and animations are new and were made for this project.

5.2.4 Sound Effects and Music

The sound effects used in the game come from many sources, and are all free to use. I take no credit for creating any of them. The main websites that I have used sound effects from are FindSounds[10], SoundJay[11] and Flash Kit[12], but many others also exist. I often simply use Google to search for specific sounds.

The background music is all original and has been created specifically for this project, by the musician "HalcyonicFalconX"[13], who I have worked with on 3 previous projects. The way we work is that I first fully plan out the game, and decide the visual themes for the levels and the approximate filesize I am willing to allow for music. I send her this information and give her a deadline sometime before beta testing is estimated to begin. In return for the music I share a small portion of the game's revenue with her, and promote her music within the game.

5.3 Final Product Overview

Like the previous chapter, this section divides the project into two parts; menu mode and gameplay mode.

[8] http://www.kirupa.com/
[9] http://www.kongregate.com/games/kupo707/epic-battle-fantasy-3
[10] http://www.findsounds.com/
[11] http://www.soundjay.com/
[12] http://www.flashkit.com/
[13] www.facebook.com/HalcyonicFalconX?sk=app_178091127385

5.3.1 Menu Screens

All menus can be navigated both by mouse and by keyboard. The mouse is easier to use and is the standard for most PC games, but the keyboard is also essential as the gameplay input is keyboard only, and some users will feel uncomfortable switching frequently. Additionally, some users may be playing with a gamepad, in which case the entire game must be usable by keyboard alone.

Menus are implemented by storing references to buttons and their associated functions in an array or matrix, which represents their position on screen. The arrow keys on the keyboard are then used to traverse this data structure to make selections, and a highlight graphic appears over a selected item. Items are activated using Space or Enter. Mouse interactions are simpler and only use the properties of Flash's built-in button class.

All menus and GUI components have been implemented as described in the previous chapter, although the contents may have been rearranged slightly to improve usability, so full descriptions of each menu are not necessary.



Figure 5.1: The preloader screen, showing the title, a progress bar (the sword), a play button, the game version number, the sponsor, and a 3rd party advertisement. This is the first screen the user sees.



Figure 5.2: The intro animation, which introduces the game's story, and lasts for 17 seconds. It can be skipped with any key. This screenshot also shows the boss sketched in Figure 4.13 of the previous chapter.



Figure 5.3: The title screen, showing the game title and initial options available to the player.



Figure 5.4: The level select screen, showing the game levels and information about them on the left side, and links to the other menus on the right. Notifications pop up in the top right space.



Figure 5.5: The shop screen, showing a list of available spells, their effects, and their cost. The user's funds are shown in the bottom right. A "purchased" animation plays to indicate a successful purchase.



Figure 5.6: The equipment screen, showing the available equipment and it's corresponding attributes on the right, the player's total attributes under that, and the player's physical appearance on the left.



Figure 5.7: The options screen, showing a list of all changeable game options on the left, and some related advice on the right. Some options are toggled, while others lead to a confirmation screen.



Figure 5.8: The key config screen displays a slideshow of images, and requests that the user press a key to correspond to the shown action.



Figure 5.9: The medal screen, showing a list of achievements the player has earned. Medals can be selected to view descriptions.



Figure 5.10: The credits screen, showing the people involved in development on the left, and promoting various web pages on the right.

5.3.2 Gameplay

The next screenshots show some typical gameplay. The player character is always in the center of the screen, surrounded by the contents of the level.

The HUD at the top of the following screenshots shows, in order from left to right:

- A red heart-shaped bar and number representing the player's HP.
- A green bar shaped like a row of circles representing the player's MP.
- A plain yellow bar representing the player's EXP.
- The three numbers and icons show how many coins and treasure chests have been collected, and the number of the current level.
- The clock shows the time spent in the level, and the number just above it shows the average framerate.
- Finally, the three larger icons show what spells the player has assigned to the three hotkeys.

All of this information is quickly identifiable by it's unique shape and fixed location on screen. Visual impairments such as color blindness would have little effect on clarity. Additionally, the HUD is as transparent as possible as to not obscure the level.



Figure 5.11: Typical scene during gameplay; the HUD elements are at the very top, the player is in the center, some enemies are to the right of him, and they are all standing on platforms.



Figure 5.12: The game over screen, which allows the player to replay the level or go to the level select screen. It appears in two varieties, based on the outcome of the game (death or level completion).



Figure 5.13: The pause screen, opened with ”P” or ”Esc”, pauses all game activity and allows the player to exit the level, and change a subset of the options available in the options screen.



Figure 5.14: Interactive objects, from left to right: a treasure chest, a coin, a sign, a standard portal, and an end-of-level portal.

5.4 Level Editting

As explained in the previous chapter, the goal in level design was to create a what-you-see-is-what-you-get editor.

Each level is stored in a different MovieClip extending the Level class, and consists of a single frame containing platforms, enemies, scenery and so on. The Level class contains an array for each type of object, which can be iterated by the game loop to control these objects.



Figure 5.15: Around 40 different enemies have been implemented in total.

Each object within the level is a self-contained MovieClip which, with a few minor exceptions, requires no setup besides being placed on the drawing stage. At runtime, each MovieClip which requires communication with other components (mainly for collision detection) calls the level instance once it is constructed. This call adds the MovieClip to the level's arrays, essentially making the MovieClip fully accessible to other game components. In addition to this, all of these objects can be accessed by iterating through the level instance's children, and this is the only way of accessing the non-functional scenery objects.

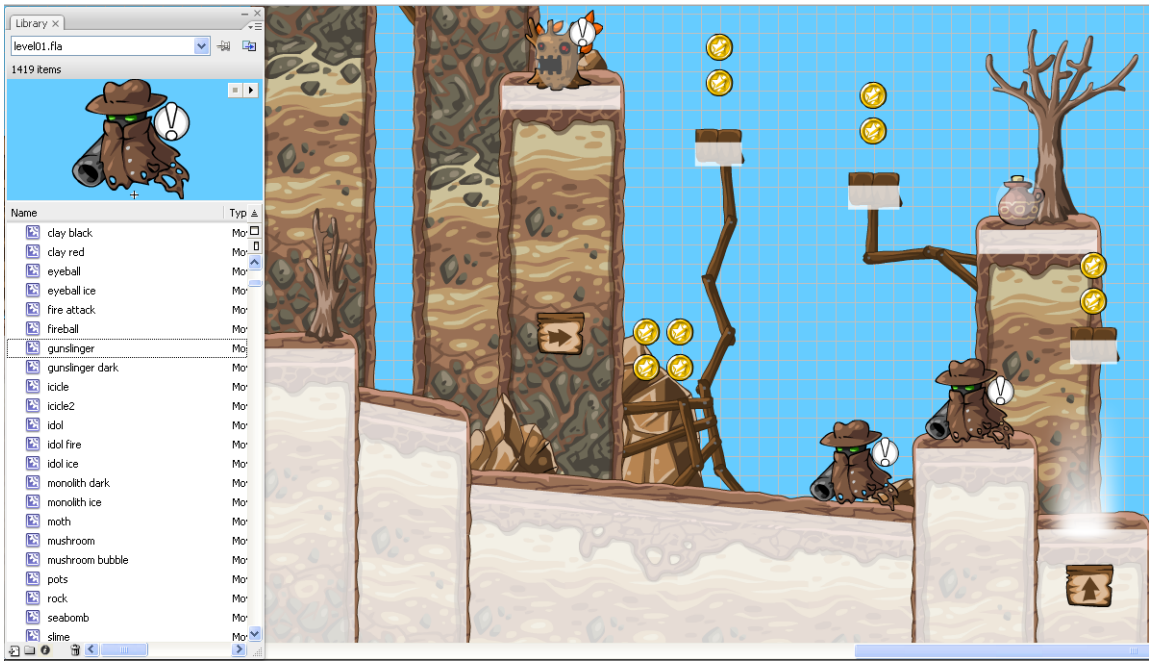


Figure 5.16: Screenshot showing level editing in Flash Pro, with library items on the left being placed on the stage to the right. Hitboxes are visible in white.

Figure 5.16 shows a level being created in Flash Pro. At runtime, the level will appear as it does here, except for some minor differences. The editor shows hitboxes in white (explained in the next section), which become invisible at runtime. The editor also does not show the level background, as this will be reused between several levels and is created separately. At runtime, all object coordinates are rounded to the nearest pixel, as most of them will be cached as bitmaps (explained later), and enemies and

portals are brought to the top layer. Enemies' attributes are scaled automatically if they are scaled on stage. For example, if an enemy is stretched to be larger than its normal size, its HP and attack power will also increase automatically at runtime.



Figure 5.17: Screenshot showing the same level as above, but at runtime.

The only code that is required for each level is a few variables and arrays that specify the contents of treasure chests, the text on signs, the links between level areas, and the background to be loaded at runtime. Each portal, sign and treasure chest must also be given an instance name, while all other objects are identified by their position in the level's arrays, which stays the same when a level is revisited.

The main disadvantage to this method is that if a level becomes too large and populated, Flash Pro slows down drastically, but luckily, the size of level that I was aiming for still performs reasonably well. Another disadvantage compared to more traditional tile-based[14] level editors is that it is sometimes difficult to judge distances between objects, as objects do not snap to tiles of fixed sizes. However, the current method is much more flexible and allows levels to appear much more natural.

One feature that is quite popular in many online games is including a level editor within the game that the average user can use to create his own levels, and then allow these levels to be shared with others online. Unfortunately, implementing such a tool would be difficult, and the future work section of Chapter 6 discusses the idea in more detail.

[14] http://en.wikipedia.org/wiki/Tile-based_video_game

5.5 Collision Detection

All interactions between objects require collision detection to be done first. Which objects collide with eachother is shown in Figure 4.5 in the previous chapter.

The most basic way of doing collision detection in Flash is using the built in hitTestPoint() and hitTestObject() methods. hitTestPoint checks if a given point intersects a graphic, and hitTestObject checks if two graphics' bounding rectangles intersect (these are the rectangles that outline the edges of a graphic). Both of these are quite restrictive. One cannot test if two irregular shapes intersect; you either test a point and a shape, or two rectangles. I ended up only using the hitTestPoint method.



Figure 5.18: The player, with his hitbox shown as a white ellipse, and his hitpoints marked with arrows.

The Player and Foes have a circular "hitbox" (an invisible MovieClip used for collision detection), and many "hitpoints" (represented as arrow-shaped MovieClips) on their attack animations. The Player's hitbox is tested against hitpoints related to enemy attacks, and vise versa. Each animation frame of an attack requires that a set of hitpoints be attached to it. The more points there are, the more accurate the collision detection, but the slower the performance. It is best to use just enough so that no enemy is small enough to fit in between them.

In addition to these, Foes have a "sightbox", which is used in a similar way to their hitbox, but instead represents a foe's field of vision, and is usually very large compared to the foe itself. As explained in the previous chapter, foes begin to attack the player once he enters their field of vision. Some foes have more than one sightbox, and perform different behavior depending on which the player is in.

The player and foes also have points on their feet and sides, which are used to test against Platforms. All Platforms have square hitboxes. A combination of tests is done to determine the shape of the Platform, and give the player and foes the appropriate actions and behaviors. For example, if the Platform is sloped, one foot will touch it, and the other won't, but additional tests will reveal it to be pretty close to the platform. This is important so that the Player knows to show a walking animation instead of a falling animation when moving down a sloped Platform, or so that the Player doesn't collide with a hill as if it were a wall.

When a collision with a platform is detected, and the appropriate conditions are met, the player is moved to the top of that platform. The conditions are that the player must be above the platform

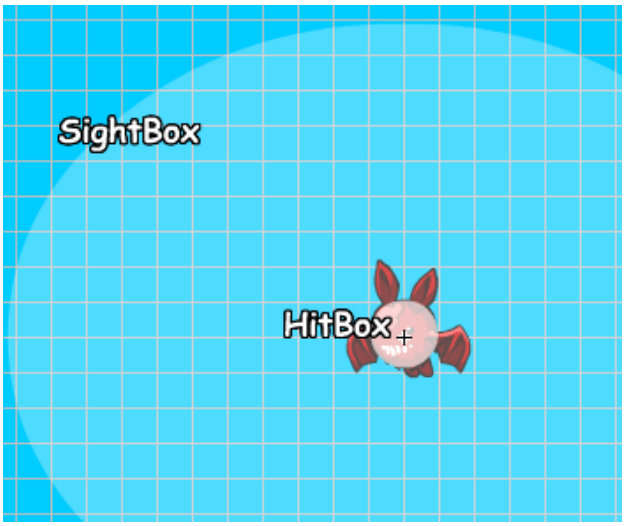


Figure 5.19: A regular flying foe, with hitbox and sightbox shown.

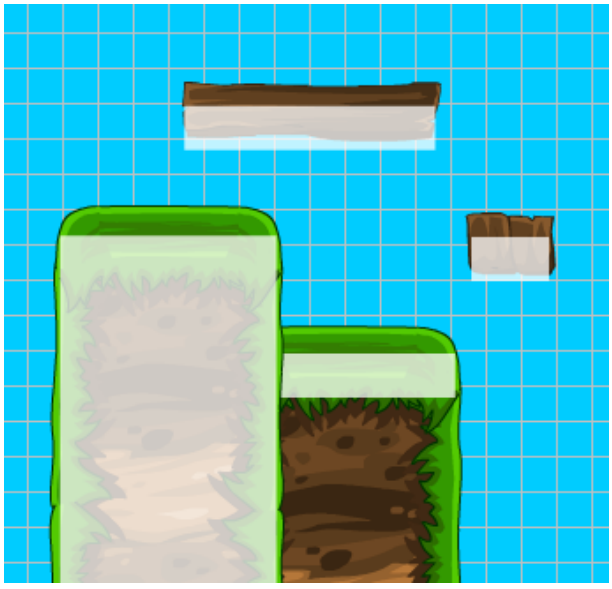


Figure 5.20: hitboxes of a landform (left) and platforms (right)

and moving downwards, to ensure that he ”cleared” the platform, and that the player has not just performed a ”drop through platform” action.

Some foes collide with platforms as the player does, others fly around freely without interacting with Platforms, and some have fixed positions and do not move at all. Foes which interact with Platforms may also be set not to move off of their current Platform. This simply involves limiting the foe’s x coordinate to that of the Platform’s edges.

Items also collide with platforms, but as they only move vertically, and do not play any animations based on their movement, the tests involved are very simplified.

5.6 Testing Strategy

The testing strategy used in this project is shown in Figure 5.21, and is explained as follows:

- A new component or feature is created and integrated. These updates are kept small to isolate any new bugs.
- The new feature is tested briefly for correct functionality, as specified in its design. Robustness and usability are not tested thoroughly at this stage.
- Once a significant number of features have been added, a public demo of the work done so far is released for play testing.
- If parts of the design are shown to be difficult to use or are unbalanced, the design is revised.
- Once all product features are implemented and usable, they are tested for robustness.

Testing is done in this order to minimize the time spent on design revisions. For example, a new feature will not be tested for robustness and fully debugged, until it has been determined to be fun and usable.

5.7 Play Testing

Chapter 25 of Game Design - Theory and Practice[12], and chapter 22 of Game Architecture and Design[11] explain the important of play testing. Play testing is firstly used to verify the design of a game; to check that level design, game usability and overall balance provides a fun gaming experience to actual potential users.

The second purpose of play testing is to identify bugs. As video games are very visual in nature, it does not matter that play testers do not understand the inner workings of a game; any bug will clearly be visible to the tester, in the form of strange events being displayed on screen. Testers will also be aware of the conditions that were present when a bug is encountered, and will be able to pass these onto the developers.

Many high budget games will have specialized testing teams that are trained to find and identify bugs. However this would not be feasible nor necessary for this project, due to the smaller scale. Instead I prefer to release public game demos on my DeviantArt page, where anyone who wishes to test the game may do so without any obligations. DeviantArt.com is a good website to use for this kind of testing, as it is basically a social network full of artists and developers, and provides a good framework for receiving feedback on one's work. Additionally, since it is not specifically an online game website, most sponsors do not see it as competition, and have no problems with their games being tested there.

There are drawbacks with public play testing however. If there are too many play testers, there may be less customers later; users who have already tested a game extensively may not feel the need to buy it once it is released. Many of these testers will also not have been trained to test games; they will

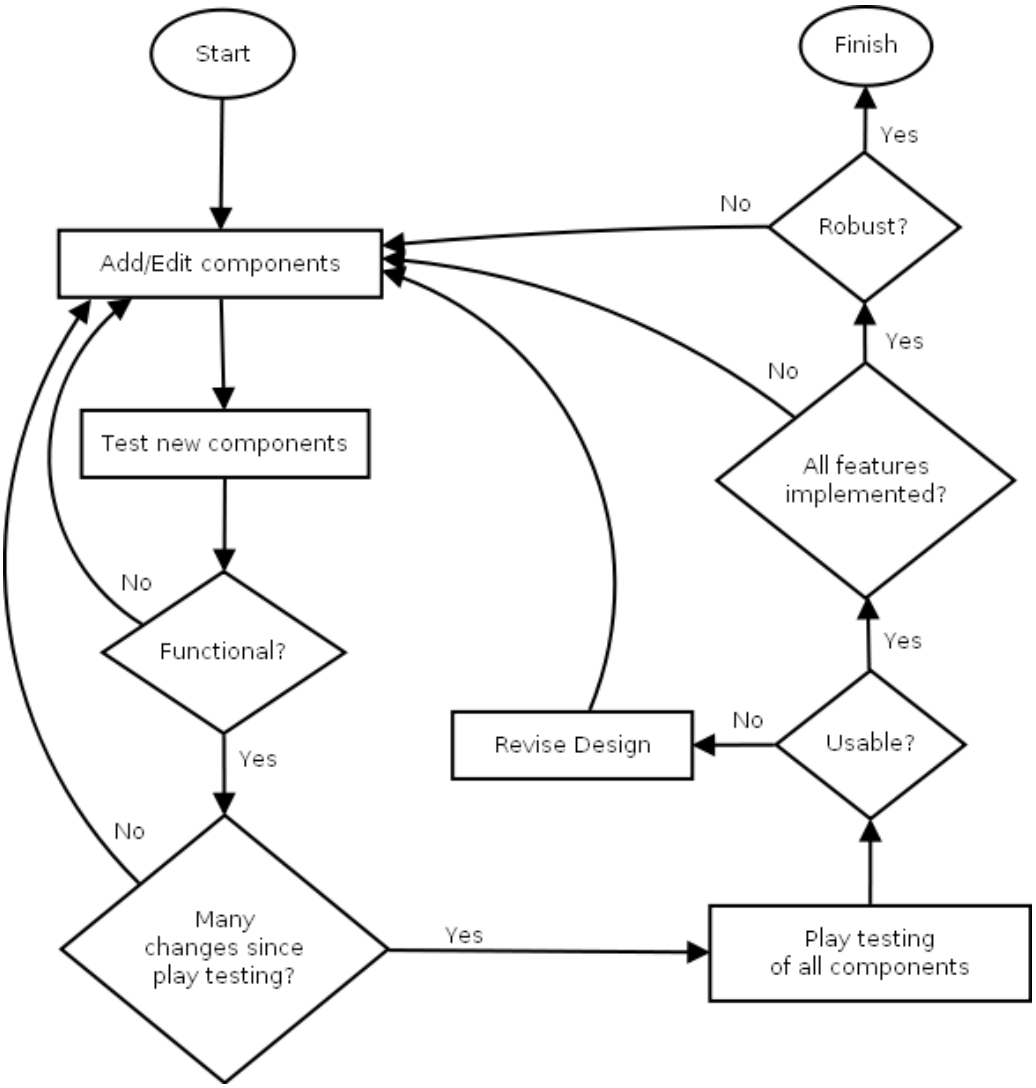


Figure 5.21: The stages of implementation, design and testing.

simply play the game normally instead of specifically trying to break it or identify bugs. So a large number of testers is required to increase the chances that bugs are found.

Fortunately, these problems are not significant in my case. My project generates revenue from advertisements rather than game sales, so customers are not lost through excessive play testing. I also have a large fan following due to previous projects, so acquiring several hundred testers is not difficult.

One final issue with play testing is that there is always a chance that the game could be leaked, modified and distributed without permission. There are a few measures that can be taken to help prevent this, which I have used. Flash games can be site-locked, meaning that they will only run on given websites. They can also be watermarked much like images, indicating that they are unfinished and should not be distributed. Of course, these measures will not prevent a motivated thief. Fortunately, most well known online game websites will take down games that have been shown to be stolen, so any leaked game is unlikely to get much attention.

5.8 Demos and Prototypes

The first demo released was a very early experimental prototype with no planning involved. This was done long before the start of the academic year, and no code or graphics from this prototype remain in the final product. Much of the requirements and design are based on this prototype, as it outlined the essential game mechanics. The aim of this prototype was mainly to assess feasibility, and although it was not very robust or usable, it was functional and only took roughly one week to develop. So the project was decided to be feasible, and was expanded into its current form.

The second demo was released once development on the actual product had begun. The basic game mechanics were implemented; moving, jumping, attacking, spell casting, the camera, the HUD, collision detection and a crude level to practice in. This demo was mainly to receive ideas from testers as the design had not been finalized at this point, and to receive feedback on the usability of the core mechanics.

The third demo was similar to the second, but the features demonstrated in it had been perfected based on feedback. Interactive objects and simple enemies had also been added, as well as the pause and options menus. The game mechanics changed very little from this version to the final one. This is when the majority of performance testing was done.

The fourth demo included the first 10 levels of the game, all of the spells and equipment, and all of the menus. This demo was intended to receive feedback on the level design and game performance, as the game was now functionally complete. At this point I also personally carried out a think-aloud walkthrough with several participants, to identify more subtle usability issues.

The fifth and final demo was released once all game content had been completed, and only debugging and minor tweaks remained to be done. Development remained at this stage until all potentially game-breaking problems had been fixed.

Development progressed to the schedule described in the Requirements and Analysis chapter, with no delays. More information about the feedback and design changes generated from these demos is given in the next chapter.

5.9 Functional Testing

Each new component's individual behavior is tested once it is implemented, to ensure it behaves as required. "Component" in this section refers to some functionality, which may not be represented by only one specific class. Unfortunately, it would be impractical to do formal Unit testing and test components completely independent of the rest of the system. This is because most components are tightly coupled due to their interactions with other components. For example, enemies can not demonstrate any useful behavior if they are tested without being creating within a level, or without the player to interact with.

Instead, the state and behavior of the other components is simply kept constant during testing of a new component, to ensure that there are no external influences. Only one component is developed at a

time. For example, when testing a new enemy, the code for the player and level classes is unchanged between tests. If the level or player code does need to be changed for any reason, they will then have to be tested again with a variety of enemies, to ensure there are no unintended consequences in earlier components.

At this stage of testing it is not critically important to remove all bugs, as this will happen during play testing. Instead, only the most common issues should be resolved, as to make better use of play testers' time later on in development.

The following sections describe how some different components were tested:

Menu Screens

Testing menu screens simply requires checking that all menu items function correctly and that any information displayed is displayed correctly regardless of the state of that information.

In the cases where a menu item modifies some data, such as in the shop, equip and options menus, this data is always displayed to the player, so any inconsistencies are easily recognizable. Each menu screen must also be refreshed to ensure the data is actually stored correctly internally, and not just displayed correctly once.

Different sets of data were created to test all of the possible states of the screens, and much of this test data remains in the code comments for later regression testing.

Player Controls

Player controls and physics are tested through use in a test level, and the results are visually observed. The content of the level is designed to emphasize the testing of certain mechanics, and additional debugging features are added, including the display of variables and the option to change some data values while playing. This sort of test harness is known as a debug room[15], and is used in the development of many games.

Due to the wide number of possible input and state combinations, and external objects to interact with, it is likely that many bugs will be undetected until play testing. One effective approach is to rapidly press all of the keys and see if anything goes wrong as a result. There have been very popular commercial games released that have game-breaking bugs caused by obscure input combinations, including the Super Smash Brothers series[16], showing that this is a difficult issue to debug.

[15] `http://en.wikipedia.org/wiki/Debug_room`
[16] `http://super-smash-bros.wikia.com/wiki/Glitch`

Enemies

Newly implemented enemies are also tested in a debug room, which reflects their actual environment and includes all objects that they may in practice interact with. The level is small to keep most of the action on-screen at all times. Large amounts of enemies are created, to increase the chances of bugs appearing, and their behavior and interactions are observed visually, or through variable printouts if needed.

All interactions between the enemy and the environment are tested. Collisions with platforms, walls, and the interactions with special platforms are observed. Some enemies are capable of interacting with other enemies, and in these cases, other enemies are also included.

All interactions between the player and the enemy are also tested. The enemy is hit with all of the player's attacks, and vice versa, and damage, knock back, collision detection and special effects are monitored to ensure they are as intended. Difficulty is balanced by testing that the enemy's attacks are all easily avoidable with some practice, and that the enemy behaves rationally and predictably. For example, some enemies had elements of their behavior randomized, making them too unpredictable and unfairly difficult to fight.

Levels

New levels don't require much testing as what you see during level editing is exactly what you get at runtime. The only testing required is a quick play-through to ensure level related data has been entered correctly. Balancing level difficulty is best done during beta testing, as the developer's opinion on this matter will be heavily biased.

However, saving the state of the levels did require some in-depth testing, as there were many opportunities for data inconsistencies to arise. For example, levels could be exited by the player, the player could die and have to restart the level, the game might be closed while in a level, the player could preview one level and then select a different one on the level select screen, and so on. It was important to decide at which events data should actually be saved and loaded, and to make sure that this operation was atomic. Every possible combination of events then had to be tested.

5.10 Stress Testing and Resource Usage

Stress testing was carried out to establish the operational limits of the engine, which are then used to guide level design.

The game's framerate is shown on screen at all times, even in the final version of the game, near the top left corner, and can be used by testers to identify at what speed the game is running at at any given moment. The framerate for this game is 30 frames per second, which is standard for Flash games and is occasionally used on other platforms too. Observing the framerate allows testers to see what in-game circumstances effect performance the most. For the purposes of stress testing, the framerate was increased to 60 fps.

The results are not surprising; the greatest slowdown is caused by large numbers of enemies on screen, and the usage of spells.

Spell animations and effects are brief, around 1 or 2 seconds, so slowdown for such a small period of time is not a problem, and is common even in many console games. More concerning is the longer periods of slowdown caused by large numbers of enemies. Tests were carried out to determine what a reasonable limit of enemies would be. These tests involved creating simple levels with large numbers of enemies in them, as shown in Figure 5.22, and then observing the framerate and memory usage.



Figure 5.22: A chaotic scene with many enemies on screen, showing how stress testing was performed.

On my personal machine, which is of rather mediocre specifications (3.5GB RAM, Dual-Core Athlon 7850, Windows XP), I could have around 15 to 20 enemies on screen at the same time, on the highest quality settings, before there was any significant slowdown. From this I decided that choosing around 10 enemies as the limit would be a safe choice, and this is still more than would ever be needed. The final game levels were then designed to avoid having any more than 10 enemies on screen at a given time. Figure 5.23 shows how performance drops as the number of enemies increases.

In addition to monitoring the framerate, memory usage was also monitored.

The frequent use of large bitmaps in this project means that overall memory consumption is slightly higher than in the average Flash game, which may use mostly vector graphics, however this trade off is essential for smooth performance. The size of levels has shown to be the main factor in memory usage.

Memory usage varies between around 80 to 200 MB on average, while occasionally going much higher when waiting for garbage collection to occur. Unfortunately, Flash gives me no control over garbage collection, so these spikes in memory usage cannot be avoided. My previous project, Bul-

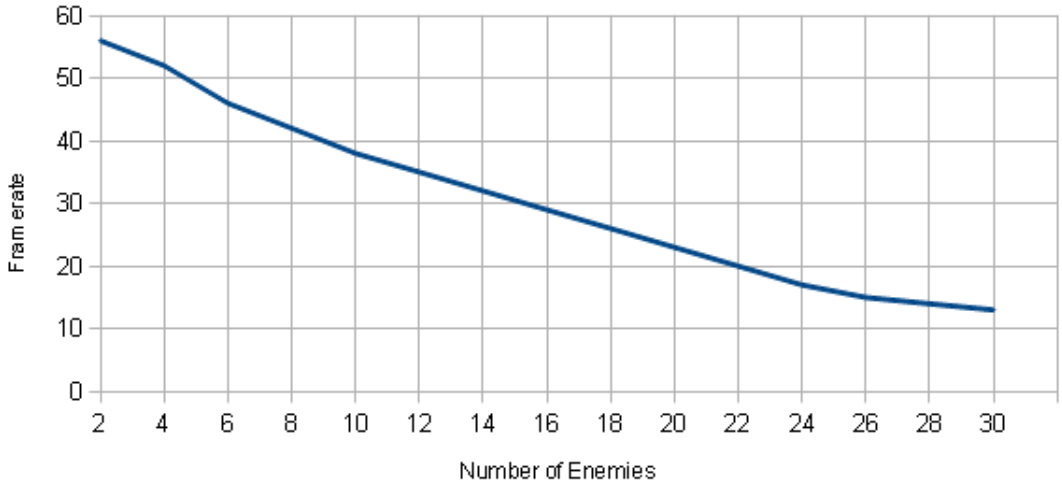


Figure 5.23: Graph showing the framerate with different numbers of enemies on screen.

let Heaven, had a much lower memory usage at around 100MB, mainly due to the much simpler backgrounds and smaller levels. To help remedy this, Adventure Story includes the option to disable several background layers, reducing memory usage by around 30 MB. Memory monitoring has also been helpful in spotting and resolving memory leaks, which are crippling to any application.

Figure 5.24 shows memory usage during different parts of the game. Peak memory is before garbage collections occurs, average memory is during normal gameplay, and reduced memory is with the performance settings lowered. It should also be noted that playing the game within a browser will use additional memory, as these tests have been done using a stand-alone Flash Player.

Along with memory usage, Flash Pro's authoring environment was another limit on the size of levels. Flash Pro slows down drastically when working with large images, so it was important to determine a suitable trade off here.

5.11 Optimizing Performance

Timing the execution of the main game loop shows that the time taken for code to execute is negligible; usually less than 1 millisecond per frame on my machine. This means that processing power is mostly consumed by displaying graphics, and this section deals with the various ways that they can optimized.

5.11.1 Vector and Bitmapped Graphics

Flash has been designed to use primarily vector graphics, which are stored as geometric shapes and mathematical equations, as opposed to bitmapped graphics, which store each individual pixel.

There are two main advantages of using vector graphics. Firstly, they have a much smaller file

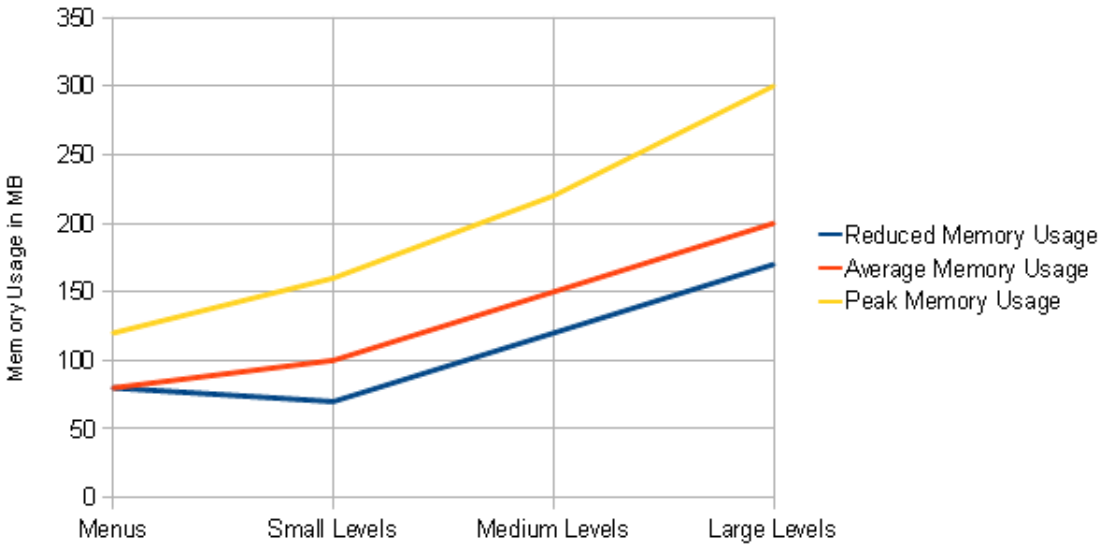


Figure 5.24: Graph showing the memory usage in different parts of the game.

size, which is important when distributing content over the internet. Secondly, they can be resized, stretched, rotated etc without loss of quality, which means they have much less restrictions in how they can be used, making development easier. The major disadvantage of vector graphics is that it takes a lot of processing to display them. This is usually fine in simple applications, but if one requires very complicated images to be animated, performance will suffer greatly.

Bitmaps on the other hand, require very little processing to be displayed, but instead require a much larger filesize and greater memory usage.

There are two methods I have considered for taking advantage of the best features of each format. Each method has it's own benefits and disadvantages.

Bitmap Caching

The simple method is to use Flash's bitmap caching option on selected vector graphics. When this is enabled, the vector is saved in memory as a bitmap once it has been rendered. This means the game file will be small as the graphic is saved as a vector, but performance will not suffer at runtime, as it becomes a bitmap. This method is very restrictive though, as the resulting image can only be moved around, and not cannot be animated, scaled or rotated, in which cases it would have to be cached again. There is also a large overhead when caching graphics, and they are cached only when they are needed for display, so it is best to cache all the required graphics during a loading screen, and never cache any during gameplay.

Therefor, this method is ideal for scrolling backgrounds, which is where I use it. When a level is entered, a brief loading screen appears while all the background layers are cached, and they will never be modified.

Detailed bitmap caching documentation can be found at Adobe's website. [17]

Blitting

The more complicated method is the use of sprite sheets, also known as blitting. In the case of vector graphics, this involves caching all of the frames in an animation as a single bitmap, or sprite sheet[18], and then using code to display sections of that bitmap when required.

This method requires a lot of extra coding as Flash does not have any features to help automate such operations. It is also best suited in games where the animations are quite simple and repeated very often. Blitting long or large animations would use a lot of memory, and consideration must be given to which animations will actually be needed in the near future. It also has the restriction that blitted graphics cannot be rotated or scaled. Basically, the uses are limited to how sprites were used in old 2D video games. However, where this method is appropriate, it often gives the best performance Flash can offer, allowing several hundred animated objects on screen.

Blitting would speed up the display of all animated objects in my game, but there are usually not many of those on screen at one time. The majority of the graphics belong to the scrolling background layers, where the much simpler bitmap caching method is sufficient. So the benefits of blitting would be outweighed by the difficulty in implementation, and the additional restrictions while developing. Due to these reasons, I did not use any blitting on this project, but it will be strongly considered for future work and where more time is available.

Blitting is described in detail here[19] and here.[20]

5.11.2 Optimizing Vector Graphics

Where using bitmaps is not appropriate, there are still many ways of optimizing vector graphics. I carried out a series of short tests to determine what kind of factors effect processing time. These tests involved the creation of a few simple vector based animations, putting many instances of them on stage, and then measuring how changes to them effect the frame rate during runtime. Measurements are in frames per second (fps). Each set of results follows from the previous set, and not from the first set.

Quality Setting

The biggest factor is Flash's built in quality setting. Unless this is disabled, Flashplayer allows the user to change this at any time. It has 3 settings, High, Medium and Low, which effect the smoothness of vector graphics. The Medium and High settings look quite similar, with medium only having a slight degradation of quality, while Low has a very notable drop in quality (jagged edges and rounded coordinates).

[17] http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=00001283.html
[18] http://en.wikipedia.org/wiki/Sprite_sheet
[19] http://www.8bitrocket.com/2008/7/2/Tutorial-AS3-The-basics-of-tile-sheet-animation-or-blitti
[20] http://www.8bitrocket.com/2007/12/23/Flash-AS3-Speed-Tests-Rendering-and-Update-Models/

The difference in framerate between these settings is massive. Here are the results:

High: 19 fps
Medium: 30 fps
Low: 45 fps

This right away gives a very conveniant option for those with weaker machines. However, Low quality should only be used as a last result. Medium quality still looks reasonably good, and it may be worth setting this as the default, since many users will not know about quality settings, and the ones that do will easily be able to change it to whatever they wish.

Complexity

Flash's authoring environment has an option for optimizing vector graphics, which reduces the amount of curves and shapes in an image. The amount of reduction can be adjusted, but it is usually possible to reduce the complexity of an image by 20-40% with very little visible decrease in quality. In my experiment, I reduced the complexity by 50%, and here are the results:

High: 21 fps
Medium: 35 fps
Low: 55 fps

A noticable improvement, although not a massive one. However, the quality of the graphics did suffer, so it's best to use this option in moderation. I use it mainly for large boss monsters and short-lived special effects.

Size

I increased the size of all the graphics by 50%, and the results are:

High: 17 fps
Medium: 26 fps
Low: 44 fps

Simply scaling a graphic can have a large impact on performance. So it's best not to make any graphics bigger than they need to be. This should also be taken into consideration when choosing the overall dimensions of your Flash project. A bigger stage size means the exact same project will run much slower.

Transparency

I set all the graphics to be 50% transparent, which showed a significant drop in performance. However, the amount of overlap between graphics did not seem to make a difference. Here are the results:

High: 12 fps
Medium: 20 fps
Low: 33 fps

Transparency should be avoided if possible, or only used for brief moments. Toggling a graphic's

transparency between 100% and 0% on every frame will achieve a similar effect, but without the slowdown.

Gradients

I colored the graphics using gradients instead of solid colors. The results are:

High: 10 fps
Medium: 15 fps
Low: 28 fps

Gradients should be avoided, and there is no reason to use them besides artistic style.

Filters

Flash's filter options force vectors to be cached as bitmaps, and this is often undesirable in animations for the reasons mentioned earlier.

Visibility

Finally, I tested to see when graphics are actually processed, and when they are not. Hiding graphics behind other graphics made no difference to performance, so Flash is clearly still processing them when they cannot be seen. This is worth keeping in mind when a large graphic is covering others, and such graphics should have their "visible" variable set to false, so they will not be processed at all.

Optimization Conclusion

These test results should only be used as rough guidelines, as the exact performance will depend on many other factors. However, in general one should try to avoid using transparency, gradients and filters, keep graphics small, and optimize them by around 20 to 30%.

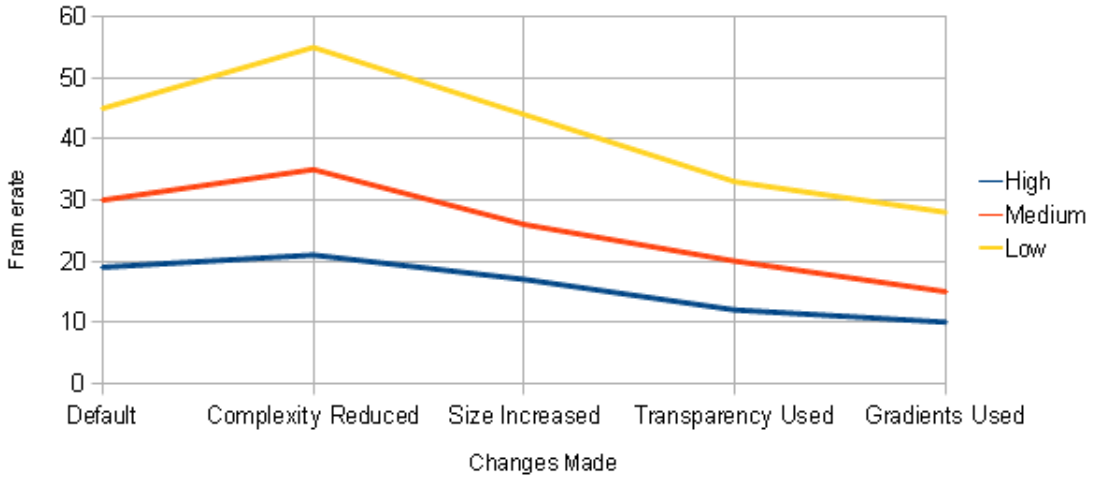


Figure 5.25: Graph showing how the framerate was effected by the above factors.

Chapter 6

Evaluation and Future Work

6.1 Introduction

This chapter explains how Adventure Story has been evaluated, and how this evaluation has led to design changes and ideas for future work.

- **Evaluation Stages and Design Changes:** Explanation of the different ways in which feedback was collected.
- **Distribution and Reception:** Comparison of the product's success to its initial targets.
- **Generated Traffic and Revenue:** Evaluation of monetization strategies.
- **Evaluation of Non-Functional Requirements:** Discussion of whether each non-functional requirement was met.
- **Common Criticisms:** Discussion of the most common criticisms reported by users after the product had been published.
- **Future Work:** Explanation of features to be added in future game projects.
- **Conclusion:** Closing summary of the project's achievements.

6.2 Evaluation Stages and Design Changes

This section explains the methods which were used for collecting feedback, and how the design has been revised as a result of each.

6.2.1 Prototyping

As explained in Chapter 5, 4 demos were shown to the public before all of the game features were fully functional. These prototypes were intended to determine what features were missing that users may like, and what existing features needed to be revised drastically.

Feedback from prototypes was all in the form of comments, and shaped the final design of the game mechanics.

The first demo identified which features were worth developing, and which did not really fit in with the others, or would have been difficult to implement:

- Users showed interest in the spell system, noting that this was quite an original feature to have in a platformer, even though it was poorly designed and unbalanced at the time.
- Similarly, users also showed interest in the leveling up system.
- Users found it difficult to play using 4 different action buttons (light attack, heavy attack and magic, defend, and jump), so this was later simplified to only jump and attack, with the occasional use of magic. The original set up would have worked on a gamepad, but not so much on a keyboard.
- Movement physics in general were very slippery and unintuitive. This raised my awareness of the importance of getting these right, and I began looking more closely at how the physics worked in other games.
- The player could originally block attacks with a ”defend” button, and could also use this to roll left or right. This feature was dropped mainly due to the point made above, and also because it would change the style of battles to that of a fighting game, rather than a platformer. Fighting games rely more on blocking or rolling to dodge attacks, while platformers rely more on jumping or moving away.
- The camera's movement was originally more complicated. The camera would move ahead in the direction the player was facing or moving, rather than be constantly centered on the player. This was mainly changed to make implementation easier, as it was already proving to be difficult to make the camera movement ”feel right”.

The second demo showed several severe issues that were quickly resolved:

- The player's falling speed was much too fast to allow the user much control over their fall, and this was drastically slowed down. The new speed was noted to feel quite unrealistic, but worked well.
- The collision detection on platforms was much too strict; initially the player would fall off a platform if his center was beyond the edge, and users were not comfortable with this. This was extended to an extra 12 pixels beyond the center, roughly to the position of the player's feet. Indeed many games are very forgiving with this, and allow the player to stand on a platform if only a tiny fraction of the player appears to be withing it's edges.

- Key input was revised to be more forgiving. Initially, if the user pressed jump or attack while an action animation was already being played, the input would be ignored. This was changed so that if the key is held to the end of that animation, it is registered, and this makes the timing of key presses much easier for the user.
- Users complained that the spell casting system slowed down the pace of the game, and was unintuitive to use. To remedy this, the option to set spells to hotkeys and bypass the spell menu altogether was added. In addition, the Temper and Seiken spells were added, as they do not require aiming, and effectively did not slow the game down at all.
- A few users mentioned that there was no way of performing aerial combo attacks, and this was something I had not actually considered previously. The ability to bounce up when using aerial attacks was then added, and later proved to be an important mechanic.

The third demo verified that the above changes were effective, and only brought up minor new issues:

- The player originally did not recoil when damaged, and many users felt that this made enemies fairly harmless as they did not make the platforming aspects of the game any more challenging. So recoil was added.
- The player could be attacked by enemies while interacting with treasure chests or portals, which left him defenseless and felt unfair. Enemies are now paused while interaction is occurring.

Up to this point there had been no evaluation done of the menus screens, only the game mechanics. The remaining demos did not lead to any significant design changes, only minor interface tweaks and bug fixes. These changes are covered in the next section.

6.2.2 Pluralistic Walkthroughs

Pluralistic walkthroughs are ideal for evaluating the usability of any software, as the intended users can identify issues that the developers had not noticed themselves. They also allow the developer to verify that the users respond to different elements of a game in the intended way.

I carried out walkthroughs with 10 different users in the Level 4 lab, all computing science students. Each user was asked to play from the start of the game, and was given no instructions from the developer, as these were all included in the game tutorial. User's were allowed to play as long as they wished, but were encouraged to play through at least the first 2 levels (around 10 minutes). The average session lasted around 30 minutes.

The user's gaming experience varied from those who never played video games, to those who only played certain genres of games, to those that played many games of all genres. This difference in experience clearly showed as different players expected different things from the game.

The particular areas of interest which I observed during the walkthroughs were:

- How quickly and easily the user found their way through the menus, particularly the level select screen.
- How the user reacted to the tutorial level; whether they took their time to learn everything, or just rushed through.
- Whether the user could learn all of the game controls within the first 2 levels, with or without paying attention to the tutorial.
- Whether the user was interpreting feedback correctly; damage numbers, knockback, notifications, power ups, etc.
- Whether the user had any problems with visibility during gameplay; whether interactive objects were clearly distinguishable from the background etc.
- Whether the user understood the goals and scope of the game.
- Whether the user could learn and react to enemy attack patterns, particularly boss enemies.

The following is a list of specific usability issues that were identified, and how they were remedied:

- Almost half of the users did not read much of the tutorial, and in a few cases this led to them being unable to learn some of the controls, in particular casting spells and dropping through platforms. Users that did read the signs had no trouble learning the controls. The tutorial text was later replaced by images, and the tutorial signs were repositioned to be more obvious.
- A few users did not know what to do when they found a new item of equipment.
- Most users forgot to spend their money or check their new equipment in between levels. The level select screen now includes a small pop up notification that alerts a player when they have surplus coins to spend, or have found a new equip.
- Many users tried to navigate the menus using the mouse at first, rather than the keyboard, which indicated what they were adjusted to doing. Both control schemes are now supported, but at first it was keyboard only.
- Users often got lost, and weren't sure whether they were going the right way or not, especially in optional dead-end areas. In extreme cases they were making blind jumps off the edges of the level, hoping there would be something they could not see. The level designs were adjusted slightly to include more arrows and signs, which would send the user in the correct direction.
- On the level select screen, a few users tried to start the game by double clicking the level icons instead of clicking the start button. This feature was then added.
- On Normal and Easy difficulty settings, the user could defeat boss enemies by hacking away at them without dodging any attacks. Bosses now have temporary invincibility after being hit a few times, as is the convention in many similar games for the same reason.

Pluralistic walkthroughs continued in a sense even after the game had been published. Many YouTube users record ”Let’s Play” videos, where they record themselves playing a game, and often give an audio commentary alongside the gameplay. The main advantage with Let’s Plays over more controlled walkthroughs is that the user has no time restraints and can play for many hours, often until they finish the whole game. Some of these videos are listed in the Appendix.

6.2.3 Mochi Analytics API

Mochi’s Analytics API is used to collect quantitative data on all of the game’s players, regardless of their location, or where the game is hosted. This data is extremely useful as it can examine the attributes and behavior of the users, and can be used to evaluate the success of vague non-functional requirements such as game difficulty and usability. The data however cannot be used to investigate any new issues that have not been prepared for.

Mochi’s API automatically collects the following data:

- The length of the user’s session. (consecutive time the user has had the game running)
- The user’s current location. (country)
- The domain name of the website hosting the game.

In addition, custom events to be recorded can be specified by the developer. The events that I felt were important to record are:

- What difficulty setting a user plays on. (Easy/Medium/Hard/Epic)
- What quality setting a user uses. (High/Medium/Low)
- The average framerate users achieve.
- How many levels the user completes in a single session.
- How many levels the user completes in total. (1 to 22)
- How many attempts the user makes at each level.
- The time a user actually spent playing within each level.

Mochi’s website conveniently displays all of this data in graph form for the developer, and offers some flexibility in how it is sorted, allowing patterns to be identified. For example, average session time can be shown per country. Unfortunately, Mochi has decided to reorganize their database while I was writing this, so not all of these features were available to me.

There are a few factors that may interfere with the results:

- Ad blockers are browser add-ons that can be used to disable advertisements and other scripts on websites and even within Flash games, and these may affect this API as well.
- When the game is downloaded and played offline, Flashplayer automatically disables all links and outgoing connections for security reasons, unless the user changes some settings.
- The API doesn't distinguish between individual users, only individual sessions, so the same user can be counted twice upon returning later.

The above problems would effect all stats equally, so although absolute values may appear lower than they should be, the stats relative to each other should still be accurate.

6.2.4 User Comments

Most websites that host Flash games allow users to leave brief comments about them. Unfortunately, the vast majority of comments do not offer any helpful feedback, but there are some users who do take some time to offer useful criticism or praise. This qualitative feedback is very useful in identifying which particular game features users liked and which features they did not like, and for collecting ideas for future work. Additionally, this is also a good way of identifying any bugs that may have not been found earlier.

Comments are however not good at representing a game's overall reception. Only a small fraction of users will actually leave a comment, and comments are often motivated by a single issue, rather than covering every aspect of a game like a full review would.

The biggest sources of comments were Kongregate[1], Newgrounds[2] and Armor Games[3], as the game received the most attention on these websites.

6.2.5 User Ratings

User ratings represent reception quite accurately, as they allow users to convey their overall level of satisfaction with a game. Most websites allow users to rate games on a scale of 5 or 10, and these ratings often go towards deciding how much exposure a website will give that game. For example, high rated games will often be listed on a "top rated" list or featured on the front page longer than usual. Ratings however do not say anything about how well specific aspects of a game have performed.

Factors that support high ratings are those which were discussed in Chapter 3; accessibility, robustness, novelty, and so on.

[1] `http://www.kongregate.com/games/kupo707/adventure-story/comments`
[2] `http://www.newgrounds.com/portal/reviews/586861`
[3] `http://armorgames.com/comments/show/12763/game`

6.3 Distribution and Reception

Chapter 3 defined some project goals in terms of distribution and ratings, and this section looks at whether these have been achieved. Adventure Story was published on the 27th of December, 2011, and the data in this chapter is for the first 2 months after publishing unless otherwise mentioned.

6.3.1 User Ratings

The ratings achieved are as follows:

- 1,006,057 views on kongregate.com, and a rating of 4.22/5, or 84%.
- 356,076 views on newgrounds.com, with a rating of 4.43/5, or 89%.
- 809,142 views on armorgames.com, with a rating of 8.5/10, or 85%.

The ratings on Newgrounds and Armor Games satisfy the expectations defined earlier, however the rating on Kongregate is a bit lower than expected, and this is most likely due to the following. When the game was first published on Kongregate, the rating was around 4.3, but this fell once the game received ”badges”. Badges are Kongregate’s form of achievements, and they are tied in to other bonuses around the site. Badges basically encourage users to play the game longer than they might want to, which may lead to frustration if the game is too difficult, and I feel this is what has happened.

6.3.2 Distribution

Overall, Adventure Story has been played 6,208,903 times as of the 20th of March. It has been played most frequently on the above sites, and some Chinese online game sites. Unfortunately, Chinese traffic is not very valuable, especially since many Chinese websites disable in-game links. Figure 6.1 shows in which countries the game was popular. The original goal was 3 million sessions in 2 months, and this has been quickly reached, but perhaps the unusally high number of Chinese sessions should be subtracted from this number.

6.4 Generated Traffic and Revenue

As explained in Chapter 3, Flash games generate revenue mainly through advertising a sponsor’s website, and through 3rd party advertisements. This section looks at the data from these areas.

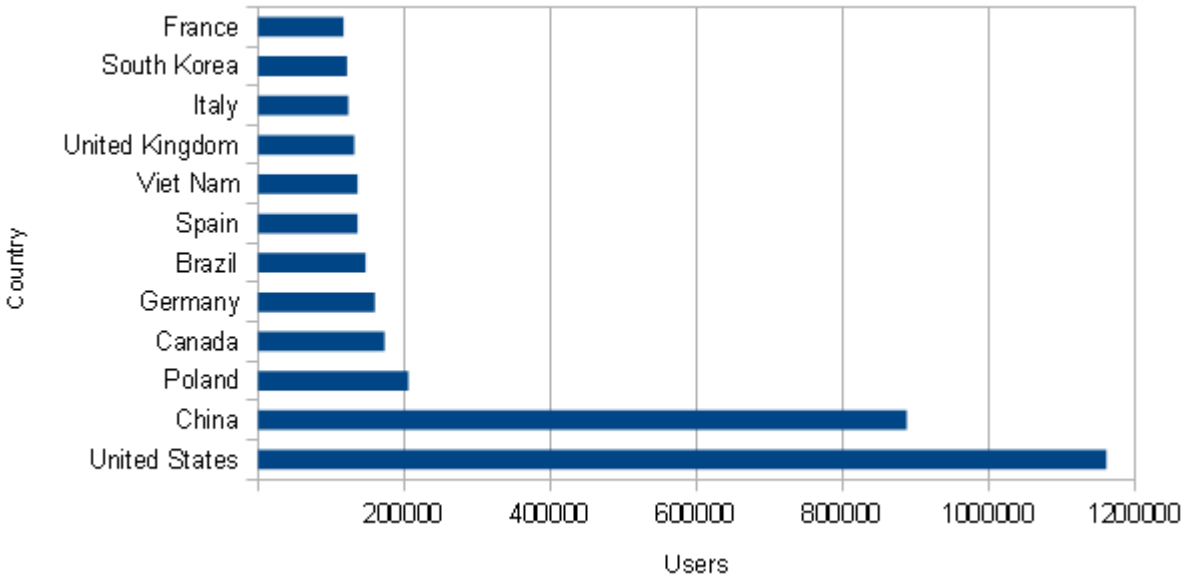


Figure 6.1: The top 12 countries where the game has been most popular.

6.4.1 Sponsored Links

Figure 6.2 shows the traffic data collected for the first 2 months after publishing. ”Entrances” refers to how many users accessed Kongregate.com through links in Adventure Story. In total there were 493,000 entrances. Of these, 350,000 were visits to me previous games hosted on Kongregate, while 143,000 were visits to the site’s front page and ”adventure games” section. The data also shows the bounce rate for entrances; this is the number of users who quickly leave the site instead of visiting more pages. The bounce rate for visits to each of my games is under 2%, while the bounce rate for the other pages is around 55%.

The sponsorship deal me and Kongregate agreed on was [redacted]
Therefor this project’s revenue from the sponsorship is around [redacted]

For comparison, my previous project, Bullet Heaven, only attracted around 200,000 visitors in its first 2 months. Extrapolating from this, I would have expected between 300,000 and 400,000 referrals for this larger project, so these results greatly exceed expectations.

The conclusion that can be drawn from this data is that having a backward catalog of games is incredibly effective at attracting visitors to a site. This is much more effective than advertising the front page of the site, unrelated games, or other sections. The difference in bounce rate shows that users who visit the site for specific games are very likely to stay, as they know what they came for, and they know that the developer’s other projects will be on the same site.

Figure 6.3 shows how the number of referrals quickly dropped over time. At first the number is very high as the game is quickly distributed to new sites, but once this exposure is saturated, the numbers begin to drop. Most of the spikes in the graph are weekends, where users have more free time. This graph highlights the short lifetime of Flash games.

Entrances	Bounces	Bounce Rate
493,159	**84,251**	**17.08%**
% of Site Total: 0.53%	% of Site Total: 0.58%	Site Avg: 15.40%

	Page (None)	Entrances ↓	Bounces	Bounce Rate
1.	/games/kupo707/epic-battle-fantasy/?haref=adventurestory&src=spon-rpg-adventur...	**193,688**	3,284	1.70%
2.	/?haref=adventurestory&src=spon-rpg-adventurestory-site-x	**124,242**	67,976	54.71%
3.	/games/kupo707/epic-battle-fantasy-2/?haref=adventurestory&src=spon-rpg-advent...	**54,539**	655	1.20%
4.	/games/kupo707/epic-battle-fantasy-3/?haref=adventurestory&src=spon-rpg-advent...	**42,587**	333	0.78%
5.	/games/kupo707/bullet-heaven/?haref=adventurestory&src=spon-rpg-adventurestory...	**28,436**	248	0.87%
6.	/games/kupo707/epic-battle-fantasy/?haref=adventurestory	**17,824**	298	1.67%
7.	/?haref=adventurestory	**10,610**	5,926	55.85%
8.	/action-games/?haref=adventurestory&src=spon-rpg-adventurestory-site-x	**8,035**	4,478	55.73%
9.	/games/kupo707/epic-battle-fantasy-2/?haref=adventurestory	**5,185**	58	1.12%
10.	/games/kupo707/epic-battle-fantasy-3/?haref=adventurestory	**4,064**	24	0.59%

Figure 6.2: Printout of referral data collected by Kongregate, showing which pages users visit.



Figure 6.3: A graph showing the number of referrals over time. This data is also from Kongregate.

6.4.2 3rd Party Advertisements

In terms of 3rd party advertisements displayed within the game itself, the stats are as follows; the ads were viewed 5,208,903 times, they were clicked 156,121 times, and in total [redacted]

[redacted]

The conclusion drawn from this is that perhaps 3rd party advertisements are not worth using anymore; the eCPM is very low compared to other ads. The space used by these ads may be put to better use for more sponsored links. Additionally, many websites will not host games with 3rd party ads anymore, so these may slow down a game's distribution to some degree.

6.5 Non-Functional Requirements

This section looks at each of the original non-functional requirements and attempts to assess to what degree they were met.

6.5.1 Addictiveness

This section looks at how effective the game was at retaining users over time.

Session Length

The average game session length is 26 minutes and 38 seconds. The average user played 3.14 levels, with each level session lasting 4 minutes and 28 seconds on average.

Unfortunately, the only data I can compare this to is the data from Bullet Heaven. Bullet Heaven had an average session length of 21 minutes, so there is a clear improvement.

It would be interesting to see how this data compares to other games, if more developers decide to publish such information. But overall, 26 minutes sounds like a respectable length of user time for a game that's designed to be played in short sessions.

Game Progress

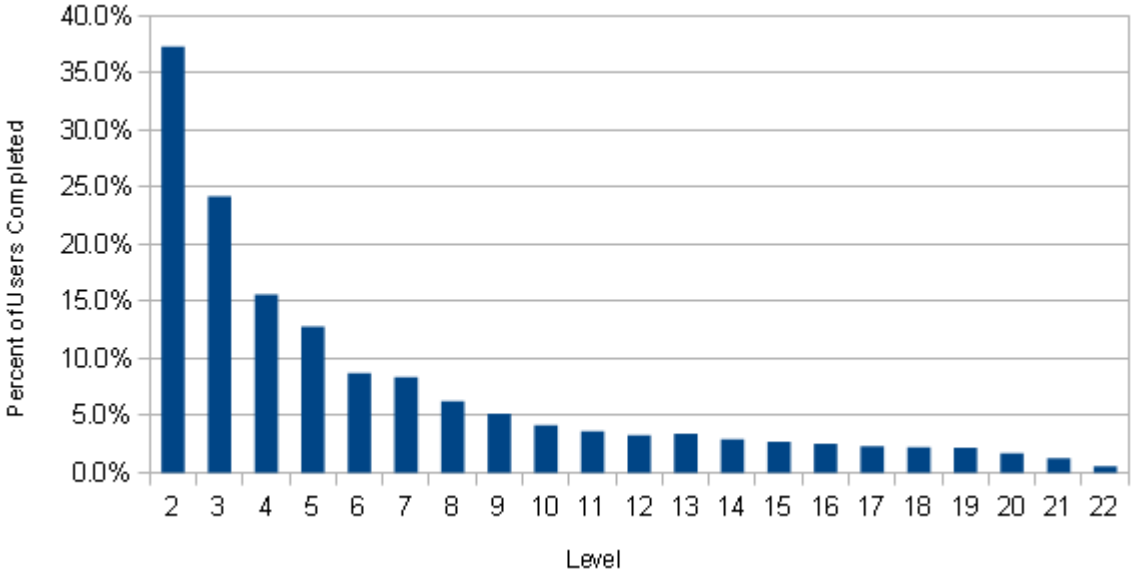


Figure 6.4: Percentage of users who completed each level, assuming they all completed level 1.

Figure 6.4 shows what percentage of users who completed level 1 went on to complete each level after that. Users who did not finish level 1 are assumed to not be interested in the game at all, so these are not discussed in this section.

The graph shows that only 12.8% of users completed 5 levels, 4.2% completed 10, 2.7% completed 15, and 1.7% completed all 20. Levels 21 and 22 don't matter here as they are very challenging bonuses which were not designed to be played by most users.

I do not have similar data from other games, but data on how many users get different badges on Kongregate is publicly available. Easy badges on Kongregate are roughly equivalent of beating the first 2 or 3 levels, while hard badges are roughly equivalent to beating the game. Badge data shows that between 4 and 8% of users who get an easy badge on a game will go on to get a hard badge. This means that the values I have recorded are fairly consistent among different Flash games.

I cannot conclude anything new from this data, but it emphasizes the design points made earlier; the introduction and first few levels of a game are the most important parts to polish, as only a tiny fraction of users will actually reach the end of the game.

6.5.2 Difficulty

As explained earlier, a game's difficulty level must increase gradually to provide reasonable challenges.

Level Success Rate

If the game difficulty were balanced, and each level was gradually harder than the previous, I would expect to see a smooth increase in the number of attempts it takes to complete each level. Even though each level is harder than the previous, the users' skills should be improving at a similar rate, so this increase should not be too large. The exception to this rule are the boss levels, which are short but designed to be more difficult.

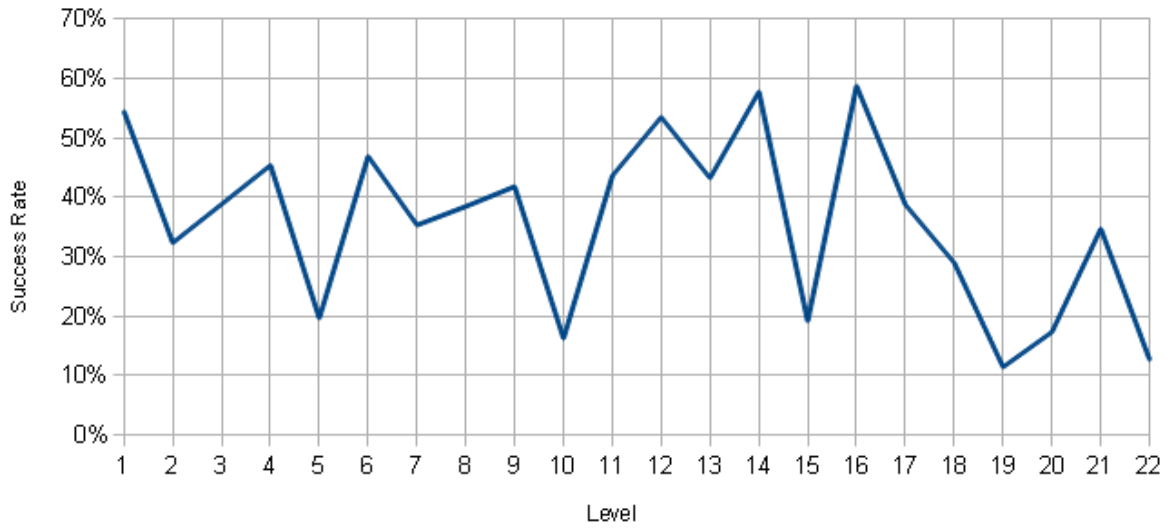


Figure 6.5: User success rate in each level.

Figure 6.5 shows a graph of the user success rate in each level. For example, a success rate of 50% means it took on average 2 attempts to a complete the level. The first thing to note is that users who quit the game are counted as having failed a level. The first few levels are when most users decide to

quit, so the graph shows them to be more difficult than they actually are. Perhaps in future this data should be collected more carefully to avoid this issue.

The graph shows difficulty spikes at level 5, 10, 15 and 20, where the success rate is around 20%. These are the boss levels, so these spikes are intended, and they are more or less as large as expected, based on the earlier play testing. The spike in level 22 is also intended, as it is a final bonus level for experienced players, and was designed to be challenging.

The spikes which were not intended are at level 17, and even more so at level 18. These levels have a success rate of 30% and 12%, and are clearly disproportionally difficulty compared to other non-boss levels. Had I created this graph earlier on in development, or shortly after publishing, I would have quickly edited these levels to reduce the difficulty.

Levels 12, 14 and 16 appear to be easier than intended, but this is not as troublesome as making levels too difficult. Users will not be frustrated by a few easy levels.

Overall, except for levels 18 and 19, the difficulty of each level is more or less as intended relative to other levels. The average level takes slightly more than 2 attempts to complete, which sounds reasonable to me, considering that the users have the option to change the difficulty setting at any time.

Difficulty Setting

The difficulty setting which users play on will be a good indicator of how difficult users found the game overall. Unfortunately, the majority of users will quit the game before they have a chance to change this setting, so it is expected that most users will be playing on the default ”normal” difficulty setting. The comparison then lies in the ratio of users playing on the ”easy” setting, compared to the users playing on the ”hard” and ”epic” settings.

The graph in figure 6.6 shows which settings users preferred. 9.1% of the users played on easy, 4.7% on hard or epic, and the rest on normal. This suggests that, overall, the game may have been too difficult.

6.5.3 Length

The original goal was to have 2 to 3 hours of gameplay. During testing, it took me personally almost 2 hours to finish every aspect of the game, which means that unexperienced players would take much longer, so it is almost certain that this goal has been achieved.

Videos of the entire game that users have recorded show that it takes most players around 3 hours to finish the game. Some of these videos are included in the Appendix.

I cannot retrieve more quantitative data as Mochi’s API only records how long a single session lasts, and cannot track the total play time between sessions. I could have recorded play time within the game, and submitted this data once the game was finished, however, many other uncertainties arise in

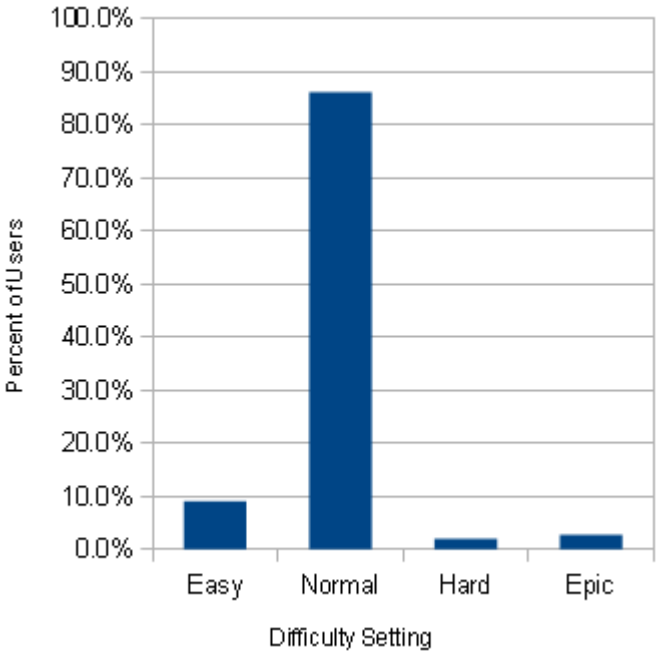


Figure 6.6: The difficulty settings that users used.

this case. If players left the game open while not playing (which is common in browser games), an unreasonably high time could be recorded. Therefore I believe that using gameplay videos is still the most reliable way of determining game length, even if the sample size is small, as I can follow what the player is doing in that time.

6.5.4 Novelty and Originality

The spell and customization system was unique, and many users mentioned that the game offered more play styles than most platformers do, especially free ones.

The characters and their animations were praised for being cute and colorful, and the amount detail which was put into the level scenery was good by Flash game standards.

Besides the above two points, there was not much else that set this project apart from similar games. Although a more important goal was to keep the style and mechanics of the game familiar to fans of the genre, perhaps I concentrated too much on that, and did not develop enough unique ideas. Despite this, there is no indication that the lack of originality was particularly harmful to the success of the project, though in the long term, it may have made it more forgettable.

A lot of users mentioned that the finished game strongly reminded them of the online game Maple Story[4]. Indeed, I did draw a lot of inspiration from Maple Story, perhaps a bit too much.

[4] http://en.wikipedia.org/wiki/Maple_story

Overall, the project did not turn out to be particularly original in style or gameplay, but there were at least a few elements that were new or interesting.

6.5.5 Visuals and Audio

As mentioned above, the visuals were perhaps the most memorable element of the game, and very few users criticized them in any way. The users who did criticize them were mostly upset by the cartoonish style, which was not to their taste, rather than the implementation.

In terms of providing clear feedback to the user, videos and testing has shown that there were a few occasions where this was not done so well. Some users took a moment to distinguish what was part of the background and what wasn't. For example, some tried to stand on trees. Some users did not realize that flashing enemies were temporarily immune to damage, despite this being a common method of indicating this. Most users quickly overcame these difficulties, though it may have been worth making some of this information more explicit.

The users generally enjoyed the music, though many mentioned that it did loop too often. As Figure 6.9 shows, only 16% of users turned the music off, so it appears to have been quite appealing. The sound effects were said to be varied and appropriate for their context. They were also particularly useful at bringing the user's attention to events which were happening off-screen, which is something I did not consider early on in the design.

The Appendix includes some examples of user comments about the visuals and music.

6.5.6 Robustness

Most of the issues in this category arise from general compatibility problems within Flash, and are found in many Flash games. In comparison to these, the bugs in my code were found to be negligible, in terms of the number of complaints received.

Compatibility

The main issue that presents itself here is that of "sticky keys". Some browser security settings, especially in Internet Explorer, cause key input to not register correctly. This is most common when a key is pressed for an extended period of time and then released, and the release is not registered, causing the game to behave as if the key were still pressed. Although this is a serious annoyance, it does not make the game unplayable. Many players have reported this problem, but to the best of my knowledge there is no solution besides requesting that players change their browser settings. It is likely that most players would not pay attention to such advice, or would not be willing to change their settings. All keyboard based Flash games suffer from this problem, and I accepted this when I started the project. This[5] forum thread discusses the problem.

[5] http://www.kongregate.com/forums/7-technical-support/topics/67250-fix-for-keys-sticking-in-games-in-internet-explorer

Another issue is that some Anti-virus and performance tools delete Flash's shared object files, leading some users to accuse the game of deleting their saved data. Besides education, a possible solution is using saved data APIs that some websites offer. These services will store a user's data on a server, and can be retrieved when the user is logged into the hosting website. An additional advantage here is that the saved data can be accessed on different machines by the same user. The disadvantage is that the user will need to make an account if he does not have one, but this could also be seen as another incentive for visiting a sponsored website.

Bugs

Overall, there have been very few bug reports since the game has been published, and most of these were minor and have been fixed within the first 3 days of publishing. The few bugs that have not been fixed are either harmless quirks, or very rarely occur towards the end of the game, which were earlier decided to be acceptable conditions. Any bugs that do break the game, will at worst force the user to restart a level, costing them no more than 10 minutes of time.

6.5.7 Accessibility

Play testing has shown that users can learn the controls quickly, even without reading any text. However, I did not have the chance to observe any users who do not speak English playing the game, so I may not be aware of all the problems they could encounter. Statistically, if language is not an issue, I would expect users in different countries to be playing the game for similar lengths of time.

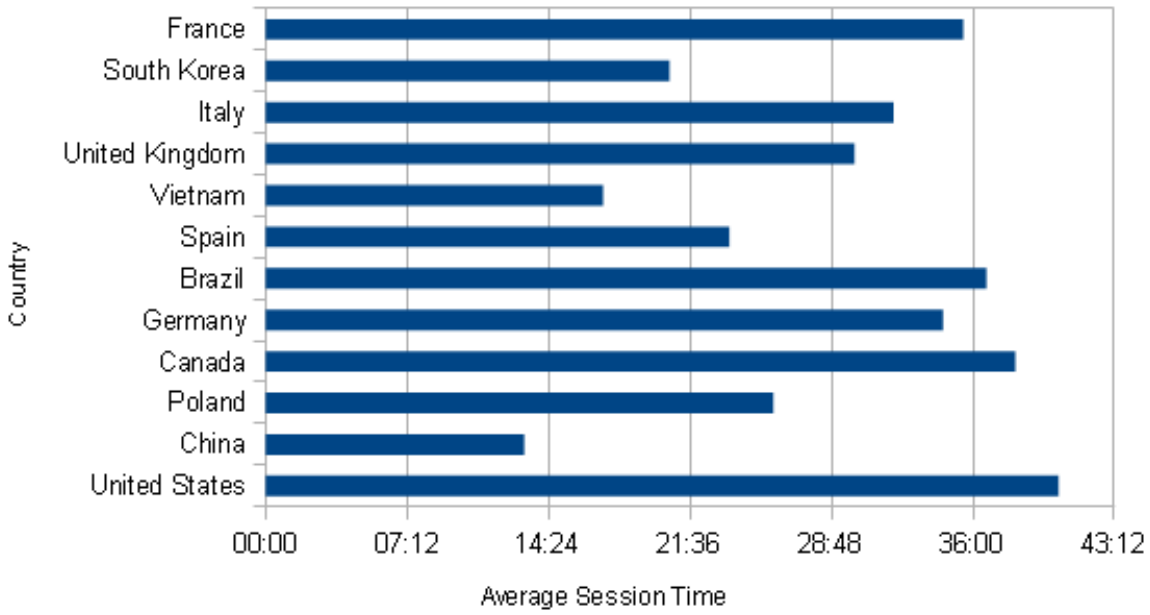


Figure 6.7: Average session time in the 12 countries with the most sessions.

Figure 6.7 shows the average session length for the 12 countries in which the game was most popular.

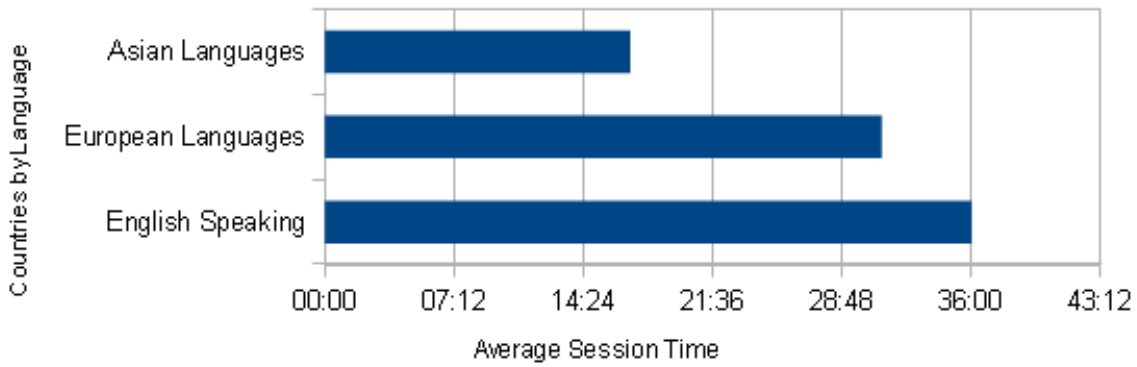


Figure 6.8: Average session time, by language group.

Figure 6.8 shows the same data, with the countries split into 3 groups; English speaking, European language speaking, and Asian language speaking.

As expected, English speakers on average play the game for the longest amount of time, which is 36 minutes. European language speakers are not far behind, at 31 minutes. This may be due to the fact that English is still quite popular in these countries, or that the alphabet is at least similar. Asian countries are much further behind, with an average session length of 17 minutes. I do not know why this may be.

I am not going to discuss cultural differences, such as the possibility that developed countries like the United States have more free time to play games, because I do not have enough knowledge about such factors. However, it should be kept in mind that language alone may not be the only significant factor.

Overall, I believe that the data I have collected is inconclusive at determining usability, and foreign users would actually have to be observed or interviewed to collect more accurate data. The data does however show, that despite any difficulties, and for whatever reasons, the game is still quite popular in many non-English speaking countries.

Another accessibility issue that arose was what the default control scheme should be. Many users had different opinions on this, and examples are given in the Appendix. However, I feel that the controls worked well for the vast majority of users, especially those using game pads, and I am not convinced that I should change the controls in any future work.

6.5.8 Performance

The average framerate recorded during gameplay was 26fps, while 30fps is the intended rate. I suspect that the majority of users are running the game without any slow down, with a few very slow machines dragging the average down considerably.

Figure 6.9 shows, as expected, that most users did not change their quality setting from the default ”normal” setting. Either the game ran smoothly for most users, or they did not realize that it was not running smoothly, or they simply quit the game very early on. Users who did adjust the settings

however, most often selected the lowest settings, suggesting that particularly poor performance must be needed to prompt this.

Even though the data shows that a notable number of users were not playing the game at full speed, few users complained about performance or changed their settings. I assume this is because most games run slowly for these users, and they realize the problem is in their machines. Additionally, a reasonable amount of slow down isn't necessarily a problem, as it makes the game easier.

I do not feel that the project reached the performance goals I intended, but it came close enough to avoid upsetting many users.

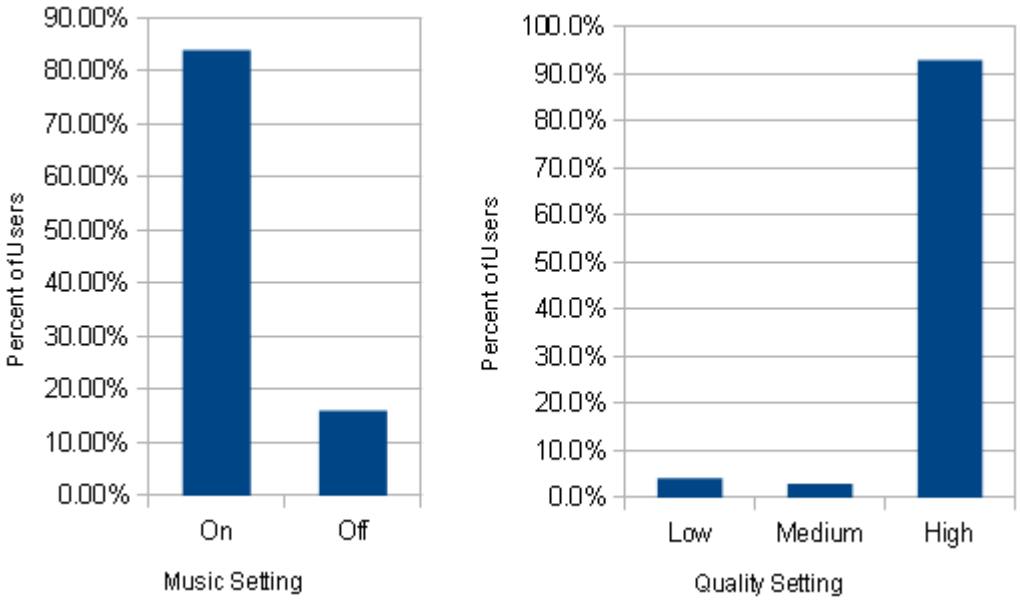


Figure 6.9: The music and quality settings that users used.

6.6 Specific Criticisms

The following is a list of the most common criticisms the game has received from users after being published, and suggestions on how they may be remedied in future work. These are not game breaking, in the sense that they do not make the game unplayable, but may have served to lower the game's overall rating.

Lack of Variety

Levels turned out to be much longer than planned. The goal was to have levels between 5 and 10 minutes in length, but pluralistic walkthroughs have shown that users can take upwards of 15 or 20

minutes on a single level if they are having difficulty. This is mainly due to developer bias, and not taking into account much failure when estimating level length.

Long levels would not necessarily be a problem, except that it causes gameplay to become somewhat repetitive. It would be best just to make levels shorter in future, or add some sort of checkpoint system, which allows users to continue from half way through a level.

The background music loops turned out to be too short. I felt that around two minutes per track would be reasonable, but this should have been closer to three or four minutes. The game's file size ended up much lower than my goal of 8 to 10 MB, so there was space leftover for longer tracks.

Too Much Platforming

Many players found the platforming a bit frustrating, especially towards the end of the game. Either it became too hard or too repetetive. This does not necessarily mean there is anything wrong with the design, but just that there are many players who are not a fan of this type of gameplay. In future, it may be worth making the platforming aspects of the game more forgiving and less central to the level design, and including more challenges in the forms of battles or puzzles instead, as the other elements of the game seemed to be more popular.

Game Camera

As explained earlier, the game camera was simplified greatly since the original prototype. Several users complained that you could not see far enough ahead or below the player, which was especially frustrating when jumping to far away platforms. Although there was never actually any need to see further than currently possible, I can understand why users may have felt this way, especially when trying to find shortcuts back down from high places. The original camera design would have worked better if implemented properly, and I hope to do this in future, but I will have to study closely how other games have achieved this.

Full Screen Support

A feature that I originally wanted to implement was allowing users to stretch the game window. Many Flash games allow this, but it is rarely useful for two reasons. Flash games are almost always played within a browser, where the user does not have this option; at best a user can zoom in through the browser, but this is still inconvenient. The second reason is that Flash is very inefficient, and stretching a game to full screen usually multiplies the resources involved. So a game won't run very well even if the feature is supported. In my particular case, increasing the window size caused backgrounds to exceed Flash's maximum bitmap size, and I felt it was not worthwhile to resolve this issue.

Boss Difficulty

If the game difficulty spiked at any point, it was definitely during the boss battles. The bosses being very challenging was probably the most common complaint. The pluralistic evaluation confirmed that users had trouble here, though it also showed that this varied greatly depending on the user's previous experience. Users who had played many similar games may have found the bosses too easy, while inexperienced users had difficulty learning the bosses attack patterns. A notable example was the first boss's ground-pound attack. Experienced players know to not touch the ground when an enemy causes it to shake, as this is a common attack in many platformers, however, inexperienced players took some time to pick up on this concept.

One possible approach to solve this issue is to perhaps give the user a hint every time he is defeated, in particular he should be reminded that it is possible to change the difficulty level. The Future Work section describes some new features which may help balance difficulty.

Pointless Collectables

Several users preferred not to use very much magic, favoring melee attacks instead. This is an acceptable playing style, but the problem I did not foresee was that this makes collecting coins rather pointless, and a large portion of the levels is designed around this. Treasure chests were also somewhat limited in the different types of content they could have. The Future Work section suggests some new uses for coins and chests.

6.7 Future Work

The first part of this section explains plans for a future sequel project, which will most likely be developed between one and two years from now. The second part considers other gaming platforms which would be suitable for this type of game.

6.7.1 Direct Sequel

As explained earlier, this project has been successful both financially and in terms of user reception. The obvious next course of action would be to develop a direct sequel, building on the same engine and game mechanics.

There are several advantages of developing game sequels. The main one being that the developer already has a fully functional and debugged engine to use, which would greatly reduce the development time. As this project has been specifically designed to support extensions, very little modifications to the engine should be needed.

Another advantage is that there now exists a large user base familiar with the game, who would quickly be attracted to a sequel, simply because they recognize it. Flash games have a very short life time. As

the earlier sections have shown, the majority of game plays will happen within the first three months of the game's release. Sequels help extend this life time as users who play the newer games will often want to play the older ones too. This is another reason why sponsors are particularly interested in sequels; if they host all of the games in a series, users are more likely to visit their site to play the rest of the games if they are presented in a convenient manner.

Many popular Flash games have released sequels, and in many cases these are almost identical to the originals in terms of features, but have different levels or playable characters. The Crush the Castle[6] series is a good example of recycling a Flash game engine. Many high budget commercial games games such as Call of Duty[7] and Pokemon[8] also release many similar sequels, sometimes as often as every year, so this is a common industry practice.

Despite the convenience of releasing sequels, it is unlikely that they will be as successful as the original game if they do not have a significant number of changes or improvements. Of course, the criticisms in the previous section must first be addressed in my case, but I have many other features planned. The rest of this section explains new features that I plan to implement in the sequel. Many of these features were considered early on in the development of this project, but were deemed unfeasible in the given time.

Puzzles

Many platformers involve simple platform-based puzzles. For example, they may require that the player destroys certain platforms, while not destroying others, to create a path through a level[9]. Or they may require the player to find switches in a certain order to activate moving platforms, or open closed areas. These types of puzzles would add more variety and some thinking to the game, and could serve as a relaxing break between battles. Many of these puzzles would not even be difficult to implement; they would just require some special platforms, and a few extra parameters to define their behavior, like those that the chests and signs require.

Multiple Playable Characters

A feature that many users were asking about from early on in development was having multiple playable characters. Once one player is implemented, adding new players isn't very difficult implementation wise, but it raises a lot of difficult design decisions. How would the players differ in abilities, yet still be balanced, so that none is better than the rest? Should the players all share the same power ups and equipment or collect them all individually? Should the user be forced to use the same player for the whole game, or should they be able to change at any time? How would players be unlocked? How would the story stay consistent if the user can control different characters? And so on. The advantages of having multiple playable characters are much like the advantages of the other

[6] http://www.kongregate.com/search?q=crush+the+castle
[7] http://en.wikipedia.org/wiki/Call_of_duty
[8] http://en.wikipedia.org/wiki/Pok%C3%A9mon_%28video_game_series%29
[9] http://www.youtube.com/watch?feature=player_detailpage&v=4Hojw6l_DN4#t=204s

forms of customization; more variety, more replay value, and more accessible to the different tastes of users.

User Level Editor

A level editor within the game itself that users could use would be ideal for extending the game's lifetime. Kongregate and Newgrounds, and several other websites, already offer APIs for sharing user made content. The way these are usually implemented is that they allows users to browse content by others, and to rate it, so that high quality content gets more exposure. In effect, users would have an unlimited amount of levels to play, as long as there remained enough interest in the game.

Unfortunately, as mentioned briefly in Chapter 5, a level editor would be difficult to implement effectively for a game without a rigid tile-based level system. It would firstly be difficult to program, as one would have to recreate many manipulation tools such as skewing, stretching, layering and so on, which I have made much use of. Secondly, it would be quite difficult for users to learn how to use it effectively; the more options the user has, the more work has to go into actually learning how to and then creating a level. Once again there is a trade off between flexibility and ease of implementation. One possible solution would be to perhaps make a flexible editor for the developer, and a simplified version for the users. Either way, this is not a feature I want to give up on, but others take priority.

Gap Monsters[10] and Time Fcuk[11] are good examples of what can be done with shared user generated content in Flash. Happy Wheels[12] has quite an in-depth and flexible level editor, but in my opinion it is quite difficult to use, and allows users to make some very messy and unusable levels.

Mounts

Mountable creatures that the player can ride, or "mounts", have been a popular addition in many games, with the most notable example being Yoshi from the Super Mario series. They can be used both to add variety to the gameplay, and as rewards for visiting hidden areas. Mounts generally give the player increased power and mobility and some new abilities, but are usually limited-use in order to be seen as a bonus. Many enemies and bosses from the first game could return as mounts with their original abilities, to reduce development cost. This would be a much more original move than reusing them exactly as they were.

The main difficulty in implementing mounts is the additional testing required, as each will behave slightly different from an unmounted player. A mount's unique ability, for example the ability to fly, could easily break the game in many ways; either by making levels too easy, allowing access to areas that should inaccessible, or by introducing new quirks and bugs.

[10] `http://www.newgrounds.com/portal/view/575508`
[11] `http://www.newgrounds.com/portal/view/511754`
[12] `http://www.totaljerkface.com/happy_wheels.php`

Single-Use Items

Building on the RPG element, the player could have an inventory of single-use items that could be used at any time during gameplay. These items could be bought in a shop like magic, or found within levels. Once again this would add more variety to the game, but it would also remedy the problem mentioned earlier, that of coins being useless to non magic users. It would also allow more interesting items to be placed in treasure chests, and in general motivate exploration. Another advantage would be that the user could save up items for a difficult part of the game, such as a boss, thus making balancing the difficulty level easier.

The items themselves would be quite similar in functionality to what is already available in the game. They could either be used to heal of buff the player like spells do, or be used as throwable projectile weapons, much like the bombs of some current boss enemies.

6.7.2 Developing for Other Platforms

Xbox Live Arcade

In future, once Flash has become outdated, or I have ran out of ideas for Flash games, it could be worthwhile porting some of my existing games to Xbox Live Arcade (XBLA)[13], or a similar platform. XBLA is a service that allows users to buy and download games online, without the need for physical copies, making it ideal for low budget games. On this platform there would be less limitations in terms of input (more buttons could be used on a gamepad) and computational resources (levels could be larger and more detailed, with massive amounts of enemies on screen). However, the game quality would have to be much higher, and would be subject to regulation by Microsoft. The graphics would need to be redone and improved, the length of the game would have to be around 10 hours or more to be a worthwhile purchase, and a more developed story with many animated cutscenes would be required. Of course, I would also have to become familiar with a new programming language (C#), and the XNA[14] framework, and buy a license in order to develop for the platform. My usual testing strategy would not be applicable here either, as I would be unable to release public demos easily, if at all.

Overall, a lot of new skills would be required, and most likely some extra team members too. The stakes would also be much higher; a successful game would be much more profitable than the average Flash game, while an unsuccessful game would most likely mean over a year of wasted development costs. For these reasons I am not keen to develop for XBLA or other console markets in the near future, but the opportunity remains.

[13] `http://en.wikipedia.org/wiki/XBLA`
[14] `http://en.wikipedia.org/wiki/Microsoft_XNA`

Mobile Devices

Mobile phone or tablet platforms, although a rapidly growing market, would be unsuitable for this type of game, as most current devices do not have keyboards or many buttons. Mobile gaming consoles like the Nintendo 3DS may however be suitable, as they are much more specialized for games. The 3DS now includes a downloadable games market, the eShop[15], which functions much like XBLA. As it is still quite early in the 3DS's lifetime, it is difficult to say how popular eShop games will become.

Vvvvvv[16] is one example of a platformer Flash game which has already been ported to this platform, and there may be more to come.

6.8 Conclusion

Overall, this project has been a success and has met most of its goals.

- The "must have" functional requirements were all met, and the majority of "should have" requirements. The notable exception being the story and cut scenes not being very developed.
- The business goals of this project were met and exceeded. Adventure Story met it's distribution and traffic goals quickly and proved to be a worthwhile investment on its own, but also opened opportunities for future work.
- Users enjoyed the game, giving it mostly high ratings and good reviews.
- I have personally gained experience with the ActionScript 3 language, and in developing action and platformer games. I have also collected and analyzed a lot of data, and drawn some conclusions which should be useful in designing future games.

Not all of the non-functional requirements were met. The game's technical performance was slightly poorer than expected, the game could have been more original, the difficulty was slightly unbalanced, and the gameplay and level design could have been more varied. However, I now have a completed and thoroughly tested game engine, and more time can be devoted to solving these issues when working on a sequel.

[15] http://en.wikipedia.org/wiki/Nintendo_eShop
[16] http://en.wikipedia.org/wiki/VVVVVV

Bibliography

[1] Flash as3 speed tests: Rendering and update models. http://www.8bitrocket.com/2007/12/23/Flash-AS3-Speed-Tests-Rendering-and-Update-Models/ [Last accessed: 13/3/12], 2007.

[2] The basics of tile sheet animation (or blitting). http://www.8bitrocket.com/2008/7/2/Tutorial-AS3-The-basics-of-tile-sheet-animation-or-blitting/ [Last accessed: 13/3/12], 2008.

[3] Farmville community surpasses 80 million players. http://www.joystiq.com/2010/02/20/farmville-community-surpasses-80-million-players/ [Last accessed: 13/3/12], 2010.

[4] Html5 versus flash: Animation benchmarking. www.apple.com/hotnews/thoughts-on-flash/ [Last accessed: 13/3/12], 2010.

[5] Flash games market survey. http://wiki.mochimedia.com/w/page/49099894/2011%20Flash%20Games%20Market%20Survey%20Results [Last accessed: 13/3/12], 2011.

[6] Chris Georgenes and Justin Putney. *Animation with Scripting for Adobe Flash Professional CS5*. Adobe, 2010.

[7] Steve Jobs. Thoughts on flash. www.apple.com/hotnews/thoughts-on-flash/ [Last accessed: 13/3/12], 2010.

[8] Colin Moock. *Essential ActionScript 3.0*. O'Reilly Media Adobe Dev Library, 2007.

[9] Carlos Nazareno. Html5 vs. flash. what you havent heard. http://thenextweb.com/apple/2010/05/14/html5-vs-flash-what-you-havent-heard/ [Last accessed: 13/3/12], 2010.

[10] Samuel Asher Rivello. How to make money with online games. http://www.adobe.com/newsletters/inspire/march2011/articles/article1/ [Last accessed: 13/3/12], 2011.

[11] Andrew Rollings and Dave Morris. *Game Architecture and Design*. New Riders, 2003.

[12] Richard Rouse. *Game Design - Theory and Practice*. Wordware Publishing Inc., 2004.

[13] Ian Sommerville. *Software Engineering: International Version*. Pearson; 9 edition, 2010.

Appendix A

Full Class Diagrams

The following diagrams extend the class diagram shown in chapter 4, including all of the methods and subclasses which were previously not shown.

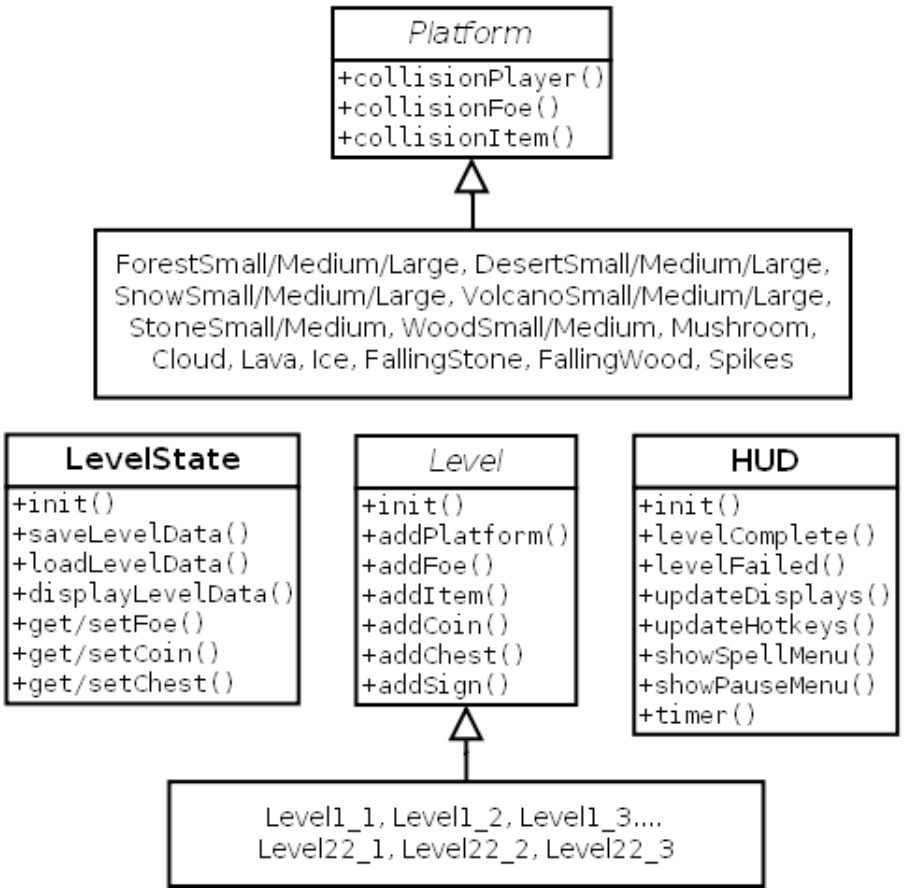


Figure A.1: Full listing of the Platform, Level, LevelState and HUD classes, with lists of all the classes extending Platform and Level.

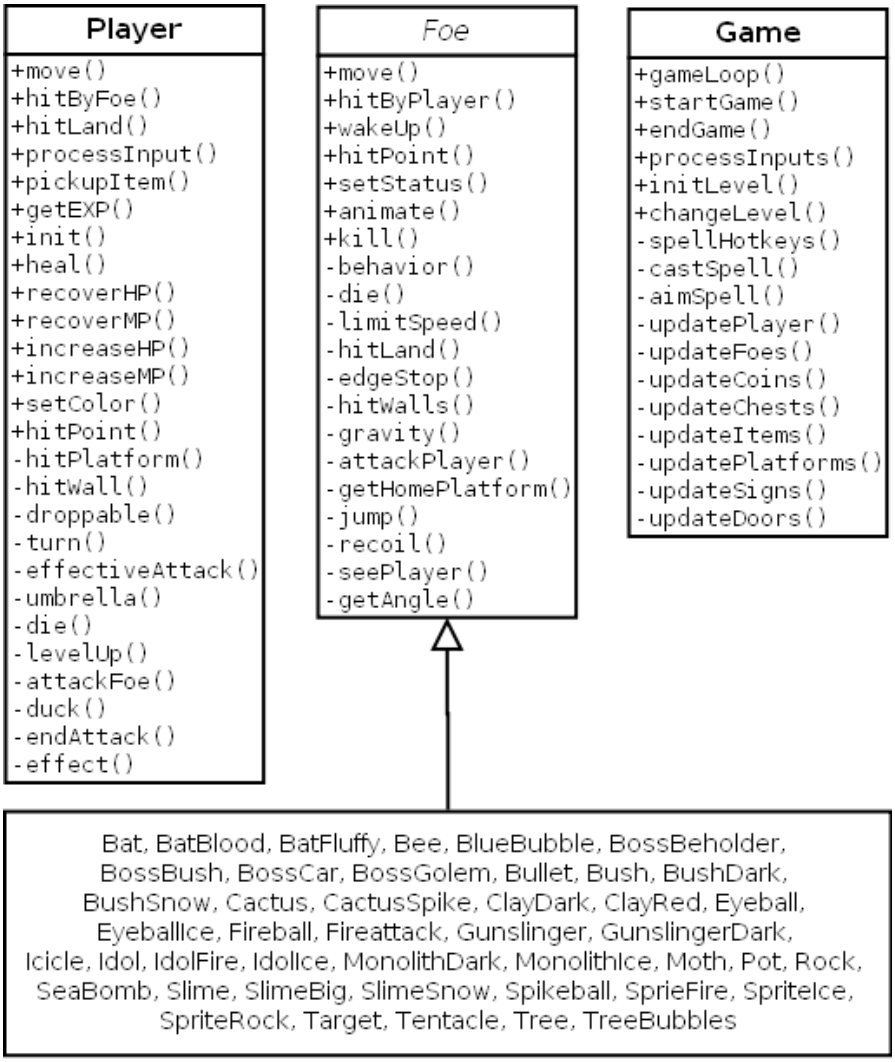


Figure A.2: Full listing of the Player, Foe and Game classes, and a list of all the classes which extend Foe.

Appendix B

Qualitative Feedback

B.1 User Comments

The following is a list of user comments on the effectiveness of the non-functional requirements defined in Chapter 3. These comments are used to support the arguments in Chapter 6, and are merely a small selection of many similar comments. These particular comments have been sampled from Adventure Story's comments section on Kongregate[1], but most websites which host the game have similar comments.

Rewards

The feedback suggests that the reward scheme for exploration and progress was very effective. Many, if not most users (based on pluralistic walkthroughs, videos and comments) were compelled to complete optional areas in the game, and felt rewarded for doing so:

- "I love the game very much it has a great learning curve, encourages exploring to get all chests and coins, and the bosses are just difficult enough to provide challenge. The magic balances out too as the freeze is the most useful early on but other magic is better later on (seiken and temper)."
- "A game that actually makes you want to get all the achievements and doesn't make you stab yourself with how hidden coins/treasures are! Well done, hope for sequels? :3"

Visuals and Audio

Many users mentioned that they particularly liked the visuals and music, with very few, if any, showing a strong dislike to them. In particular, users enjoyed the cute style and references to other games:

[1] `http://www.kongregate.com/games/kupo707/adventure-story/comments`

- ”WOW. There is amazing detail in this game, im sure matt put his heart into crafting each piece of every level. Im sure no one noticed the statue of meowth among the cats, or the kirby snowman that blends in with the background during the heat of battle. I applaud this game, bravo sir, it is fantastic.”
- ”This game is adorable, fun, unique, addicting, and it has great music as well. I’ll definitely be playing more games by the author in the future.”
- ”I really enjoy your extraordinary cute style of graphics and the way you visualize things. You’re a great example to all who wish to do Animation (This is coming from an Animation student). ”
- ”Is there any way you could put up a link to download the music? It’s super pretty :)”

Accessibility and Controls

Despite the option to customize controls, there were still many complaints about the controls being unintuitive. Users had varying opinions about what the default control scheme should be:

- ”Am I the only one who just can’t deal with these controls? I feel like the go in door+use portal+aim attacks up+ jump should all be on the same key, and I can’t find any combo that feels right. I keep doing stupid unintentional things and it really detracts from something that I really do believe I would enjoy.”
- ”I wish the controls were a bit easier. I’d suggest switching a and up (have up make you jump, make space or something continue levels and open chests).”
- ”Hmm. I don’t like the controls, even though I can reset them. I feel as though Jump should be the same as Up, and a big trouble I had was confusing the two, leading to many a death. Another annoying thing is the planks that look solid but aren’t. Otherwise, great game!”

Users who used a gamepad instead of a keyboard had a much easier time adjusting to the controls:

- ”I expected nothing less going into this, it was a great run and I had a lot of fun with the bouncing slashes. Skilled bouncing on most enemies is a surefire way to avoid their attacks, and with my trusty XBOX 360 controller and Joy to Key (which I already use for flash games) I can confidently say I murdered this game.”

Difficulty

The game difficulty spiked at level 17 onwards, and this was reflected in many comments:

- ”Bouncing off of enemies/targets is extremely unforgiving. As a result level 17 is maddeningly frustrating compared to the rest of the game.”

- ”After fighting with 17-A to get to the end I thought I was going to get the ultimate weapon or something. Stupid coins...”
- ”Well, everything is perfect so far, except for level 17-a. which makes me wanna hit someone hard with something heavy. I’m so bad at this jumping part...”
- ”Level 17-A is Crazy!!!!”

Many users felt that the platforming aspects were too difficult compared to the rest of the game:

- ”I don’t like it that this game involve so much jumping, and many bottomless pit. other than that it is great. especially the bosses.”
- ”this game needs more solid ground to stand on for fighting. it feels like constantly falling into holes kind of gets in the way of an otherwise awesome RPG.”

Balance

Although the game was generally well balanced, many felt that the Mana Staff weapon was a bit overpowered:

- ”The mana staff is easily the most overpowered weapon in the game. For the last boss all I did was sit there and whack it with the mana staff and healing and using magic attacks. You regen mana faster (and can thus heal faster) then the boss can damage you.”

A few users mentioned that they would like a melee attack that swings downwards:

- ”Would really like a way to swing my sword down, so when you’re on a platform above an enemy you can hit him. Down+attack does this ducking poke attack that I never use.”

Many users would have liked some magic to help out with the platforming:

- ””Teleport” would have been a really nice spell addition. I can’t tell you how many times I’ve been blown off a cliff and pressed the spell key in a last ditch attempt to save myself, in vain.”

Length

The game was never criticized for being too short, instead it was often mentioned that it became a bit too long and repetitive over time. Though most of these complaints were from those who were having difficulty in general:

- ”This game doesn’t deserve the score it has. Although it’s very well made there really isn’t much to it, and after a while you end up doing the same things over and over again. The platforming especially becomes quite frustrating if you die or fall and have to start over. Where’s the puzzles or story that an RPG should have?”
- ”it’s incredibly annoying to spend 20 minutes or more looking for the last few treasures in a level just to die and have to backtrack to where you were. i think there should be checkpoints or the levels should just be smaller.”
- ”as much as I like the music, it becomes quite grinding after listening to it loop several times in one level, and then again in the next.”

B.2 Videos

The following is a list of user videos which were used in the evaluation. These are much like pluralistic walkthroughs. I do not personally know any of the video owners, but these videos are all publicly available.

In the cases where a video is part of a long series of videos, only the first is listed.

Video commentaries on gameplay, each by a different user. These are a good example of how users learn to play the game, and how they react to different game aspects:
http://www.youtube.com/watch?v=-8HEbF4wk0g
http://www.youtube.com/watch?v=O5fbwPMzy2s
http://www.youtube.com/watch?v=0u_Gz7B6MnY
http://www.youtube.com/watch?v=b5NfCj4IFwg
http://www.youtube.com/watch?v=mvxYBFUyBS4
http://www.youtube.com/watch?v=Jgg-F6Gfck8&feature=related

Videos showing different strategies used during boss fights, with details in the video descriptions. These emphasize the amount of strategy that Adventure Story offers in its fighting mechanics:
http://www.youtube.com/watch?v=xHe3XnV1fVM&feature=related
http://www.youtube.com/watch?v=QErSSUcVk9Y&feature=related
http://www.youtube.com/watch?v=bEt1YD7T_gs&feature=related
http://www.youtube.com/watch?v=JczfAimzQmc&feature=related
http://www.youtube.com/watch?v=WbO4VL57O9Y

A user evaluating the second released demo:
http://www.youtube.com/watch?v=vmlCzK3P6ck