**Match Quest**

**Technical Design Document**

Version 1.0.0

Riley Bloomfield

250670211

CS4483 – Game Design

Winter 2016

**Version History**

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Edited By** | **Date** |
| **1.0.0** | Riley Bloomfield | 2016-03-13 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

Glossary 4

Game Overview 5

Game Summary 5

Platform 5

Development Overview 6

Development Team 6

Development Environment 6

Development Hardware 6

Development Software 7

External Code 7

Game Mechanics 9

Main Technical Requirements 9

Architecture 12

Game Flow 13

Graphics 16

Audio 16

Artificial Intelligence 16

Networking (If Applicable) 16

Game Objects and Logic 16

Data Management and Flow 16

User Interface 16

Game Shell 17

Play Screen 17

Technical Risk 17

# Glossary

|  |  |
| --- | --- |
| Term | Description |
| Game Stage | One specific stage a user completes. This is signified by one puzzle completed of regular style, combat style or boss style. |
| Regular Stage | A standard stage type consisting of a resource-collecting puzzle. |
| Combat Stage | A puzzle in which users match battle moves in order to defeat enemies. |
| Boss Stage | A stage faced once per stage set, where the player must use previous resources collected to accomplish a goal that will propel them to the next set of stages. |
| Stage Score | Calculated by multiplying the number of remaining moves in a stage x total number of resources collected x number of special multiplier items collected |
| Stage Daylight | A measure of remaining moves displayed to the user by the transformation of day to night on a stage background. Stages begin at morning and progress to darkness. A failure is signalled by complete darkness (night). |
| Rule of 9’s | Common server reliability metric, used to indicate the percentage of uptime annually. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Game Overview

## Game Summary

Match Quest is an un-timed puzzle/adventure game appealing to casual gamers looking for more than the average “solve this puzzle to continue” flavour.

* Matching rows or columns of three or more elements on the playing grid will result in the removal of the items and an addition to the player’s resources.
* Once sufficient resources are gathered, the player is provided with a refreshing new type of puzzle involving the completion of a path around obstacles that can be completed using the same mechanics as all other puzzles.
* Combat system appeals to a large variety of players. Puzzle-solvers will complete the task just as they would with any other puzzle and warriors will take advantage of combat strategies by choosing their moves through the puzzle.
* Efficiently solve the puzzles to obtain higher scores to challenge friends, earn higher ratings and obtain combat items allowing the player to take on more impressive enemies for even higher scores.

For a fun twist on the classic match-three puzzle genre, check out Match Quest and begin matching, exploring, battling and earning scores to advance the story and compete against your friends!

## Platform

To provide the most coverage for the target market, a mobile platform will be the primary target. A web app will be the first target as it can be played on any device with a browser, including almost all mobile devices. Secondly, a native iOS or Android app would be created to improve and refine the experience. Web platforms require minimum hardware investment, which is a huge benefit targeting casual gamers. Apple’s Game Centre is a useful development tool for sharing scores with friends, but scores can also be shared through social network integration.

Limited 3D rendering during cut scenes and creature battles will require minimally powerful CPUs and graphics processors, however, the game should run on medium settings in an html5 enables browser without trouble. The majority of the playing area will be simple 2D graphics and will require less powerful hardware.

# Development Overview

## Development Team

|  |  |  |
| --- | --- | --- |
| Name | Role | Description |
| Riley Bloomfield | Developer, Designer |  |

## Development Environment

This describes the environment used to develop the game. In theory, someone could set up an environment like what is described here and everything needed to work on the game.

### Development Hardware

Web Server:

In order to retrieve game materials and assets, an offsite web server will be required to provide access to the game. Server development will take place on the remote server through SSH access. An appropriate server has been selected to serve the purposes of this game with a limited user base and includes:

* 1 processing core
* 512MB RAM
* 20GB SSD
* ITB file transfers/month
* Cost of $5/month CAD + 13% HST
* Ubuntu 14.04
* Reliability of 99.99% annually (4-rule of 9’s)

Main Development Hardware:

Main development requires a personal computing device capable of editing text files, and using the Git versioning system, which requires terminal access. In addition to file system access, desktop browser access is mandatory. The browser must have a console to debug program errors and flow of control. Either Google Chrome or Apple Safari is recommended as desktop browsers. Minimum computing requirements to edit text files or launching desktop browsers are available in almost all modern computing devices. In order to render the HTML5 canvas, a graphics card supporting shader processing is mandatory, which should be all cards greater than DX9 standard.

### Development Software

Integrated Development Environments:

The entirety of code developed can be developed using a text editor.

Compilers:

An environment will not be needed to compile any developed programs as a web browser accomplishes this task upon instantiating the program.

Debugging:

All debugging can be accomplished through a browser console available in most modern browsers. Both Google Chrome and Apple Safari contain consoles that can be opened to view errors logged in the application.

Version Control:

Git will be used as version control. The repository will be held remotely on Github at the address: <https://github.com/rileyBloomfield/matchQuest> . All commits pushed to the remote must be functional without errors.

Backups:

A mandatory offsite backup of the entire working directory will be completed after every major functional upgrade, at the discretion of the developer. In the even of a loss of data on both the remote repository and the local repository, the baseline will be reverted back to the last offsite backup.

### External Code

EaselJS:

A JavaScript library that makes working with the HTML5 canvas element easy. It is useful for creating games, generative art, and other highly graphical experiences.

This will be used to interact with the html5 canvas by abstracting low-level graphical commands with high-level method calls. An additional reason for using this library is to facilitate graphical object tracking. Objects rendered to the canvas will be identified as objects that can be moved, deleted or edited individually instead of interacting with the canvas as an entire bitmap image.

The EaselJS library and documentation can be found at: http://www.createjs.com/easeljs

TweenJS:

A simple but powerful JavaScript library for animating HTML5 and JavaScript properties. This works well as a stand-alone library or integrated with EaselJS.

This will be used to animate the movements of graphical objects on the abstracted HTML5 canvas. Display objects created with EaselJS can have their properties animated individually, such as x and y position and rotation. This will provide a simple and independent way to animate objects without manually changing the position of objects in a developer-defined loop.

The TweenJs library and documentation can be found at: http://www.createjs.com/tweenjs

PreloadJS:

A JavaScript library that lets you manage and co-ordinate the loading of assets and data. With this library, assets such as JavaScript files, xml documents, audio files, images and style sheets can be loaded on command. An additional benefit includes the ability to load all resources before allowing the user to proceed and improving their experience by ensuring all assets are available before continuing. PreloadJS triggers events upon beginning and ending the loading process, allowing the developer to be aware when resources are available.

The PreloadJS library and documentation can be found at: http://www.createjs.com/preloadjs

SoundJS:

A JavaScript library that provides a simple API, and powerful features to make working with audio a breeze. This has been developed with integrating audio file loading to PreloadJS. Audio files operations will be abstracted to provide the developer with high level functions to load, play and stop audio files in a well timed environment.

The SoundJS library and documentation can be found at: http://www.createjs.com/soundjs

Content Pipeline:

With the additional support provided by PreloadJS, a publicly accessible web server will serve all material assets. Assets will be loaded as required on a per level basis. Because of this, server contact must be made at a minimum of at least once every stage, to load the required assets.

# Game Mechanics

## Main Technical Requirements

Functional Requirements:

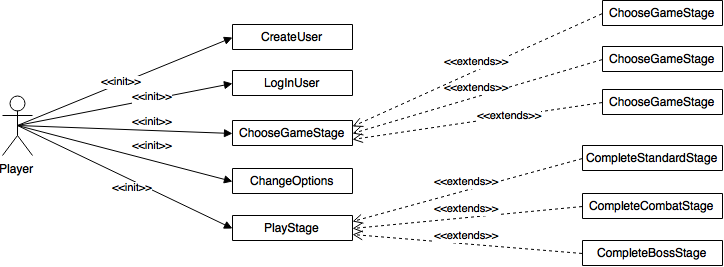
1. **Create User:** Users shall be able to create a user profile in order to restore progress when a game is ended. Selecting a log in item from the main menu and providing user details to verify the user will accomplish this.
2. **Log In User:** Users will be able to restore a previous play state by providing user profile details. Log in will be selected as an option from the initial start menu, which provokes the user to input their details. Upon submission, state will be resumed and the stage selection screen will be shown.
3. **Change Options:** Users will be able to set options for their playing session from the initial start screen. Game volume will be an option available in this menu.
4. **Choose Game Stage:** Once authenticated, users will be able to select a desired play stage from a stage selection screen. Only stages that have been completed previously will be available to the user in addition to the current next stage of play. The score of each stage will be displayed if previously played. Only one set of stages will be visible to the user at a time but users will be allowed to move through unlocked sets of stages once they are available. Stages displayed will be of three types: standard stage, combat stage or boss stage.
5. **Display Standard Stage:** If a standard stage is selected, the standard stage template will be shown onscreen in place of the stage selection map. Initially, the player is shown the resource requirement goal of the stage. The standard stage consists of an 8 x 8 tile grid full of icons and a resource collection panel. The user has the ability to swap two adjacent icons per move. If no match occurs, the move is reversed. Each type of icon will correspond to a resource. The number of resource matches occurred will be displayed onscreen, indicating the player’s progress in collecting the required number of resources. Every move made by a player on a standard stage reduces the daylight by a fraction.
6. **Fail Standard Stage:** If daylight reaches zero (night time) the stage is failed and the player is returned to the stage selection map screen, allowing them to select the same stage and have another attempt. This limits number of moves.
7. **Complete Standard Stage:** If a player has matched the required number of all resources in a standard stage, they are shown a success message and the score for that stage is calculated. The player returns to the stage selection map screen with the current stage shown as completed, score displayed, and the next available stage unlocked.
8. **Display Combat Stage:** If a combat stage is selected from the stage selection map, the player is shown an 8 x 8 grid of icons along with a player and enemy health bar. The opposing creature is displayed onscreen to indicate the combatant. Each type of tile in the grid corresponds to a type of battle move. If a set of three or more of that tile type is matched, the battle move is executed.
9. **Complete Combat Stage:** The battle continues until either the player or the enemy is defeated. If the player is defeated, they return to the stage selection map screen to retry. If they succeed, score is calculated and the stage selection map is updated with the completed stage (score displayed and marked completed).
10. **Display Boss Stage:** If a boss stage is selected, an 8 x 8 grid is displayed as well as a resource count of resources collected in the previous standard stage. One resource of a type will be used on every match of that type of tile on the grid. Elements will be contained in the grid that must fall to the bottom of the grid, accomplished by matching elements below them. If all special items are collected before resources are used up, stage is complete.
11. **Complete Boss Stage:** If a boss stage is completed successfully, a new set of stages is available for the player to view on the stage selection map. This is seen as a progression through the adventure, and more story plot will be revealed on boss stage completion. If resources are depleted before all items are retrieved, the stage is failed and the player returns to the stage selection map.

Puzzle Completion Functional Requirements:

\*Each stage contains an 8x8 match-three puzzle grid with common functional requirements and will be encapsulated in the **PlayStage** case.

1. **Make Move:** Players have the ability to select two tiles sequentially. If the tiles are adjacent, vertically or horizontally, a swap will be performed between the two tiles.
2. **Revert Move:** If a swap of tiles is made and the move does not result in a match, the swap is reverted and the move cost is reimbursed.
3. **Fill Tiles:** If a move results in a match, the matching icons are removed from the grid, and the tiles above gravitate to fill the empty spaces. Random tiles are added to the top of the grid so that the 8 x 8 grid will always be full of tiles. This process should be animated and visible to the player.

Functional Use Case Diagram:

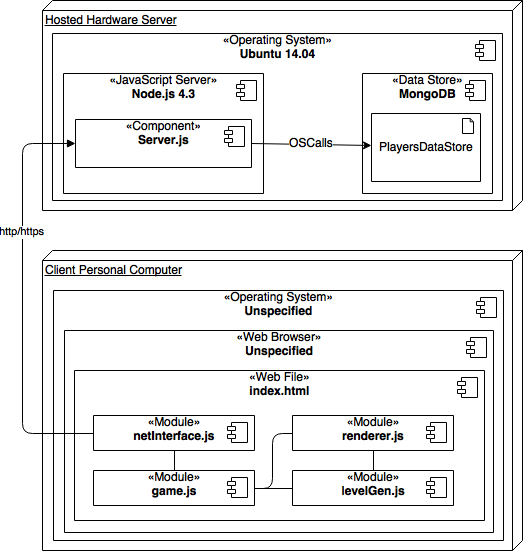


Non-Functional Requirements and Constraints:

1. If a match occurs on the tile grid, matched tiles must disappear and cascading tiles must be filled within 3 seconds of the last match recognition.
2. When a stage is selected, asset loading should not take more than ten seconds, at least 80% of the times stages are selected.
3. Server reliability should approximate 99.99% annually uptime to serve assets.
4. Server contact and asset loading errors should be shown to the player via dismissible alerts, which persist at least 5 seconds.

## Architecture

System Diagram:



System Architecture:

The system required to run Match Quest consists of a web server and a browser containing the client game, fetched from the web server via http/https protocols.

Server Architecture:

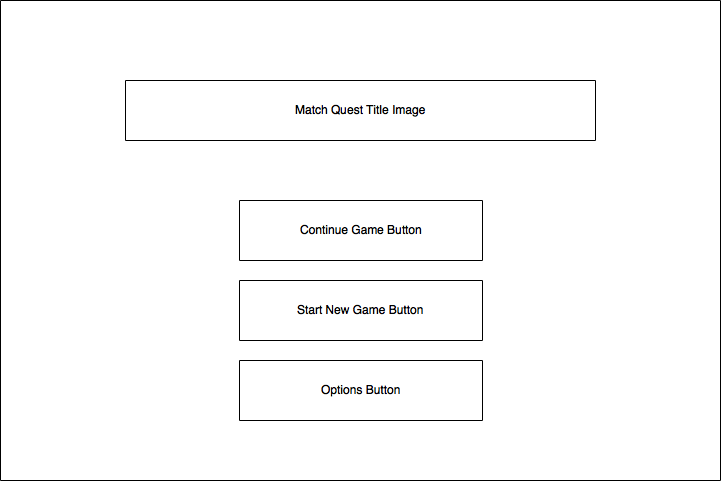
1. **Hardware Server:** Details as specified in development server hardware.
2. **Operating System:** Ubuntu 14.04 without GUI elements, Linux distribution running on the server.
3. **JavaScript Server:** Node.js version 4.3 will be used with Express.js middleware. This will contain the JavaScript server running environment responsible for listening to network ports and responding to requests as defined in the server.js file.
4. **Server.js:** This file contains the routes and port specifications for the Node environment. The public folder will be severed from the empty route while users will be queried and verified via GET and POST methods to the ‘/players’ route. Querying the data store will fulfill player queries. Assets will be served from the public folder statically.
5. **Data Store:** An instance of MongoDB will be running with a player data store table. This will hold maps of usernames to current stage in order to maintain player progress and stage scores.

Client Architecture:

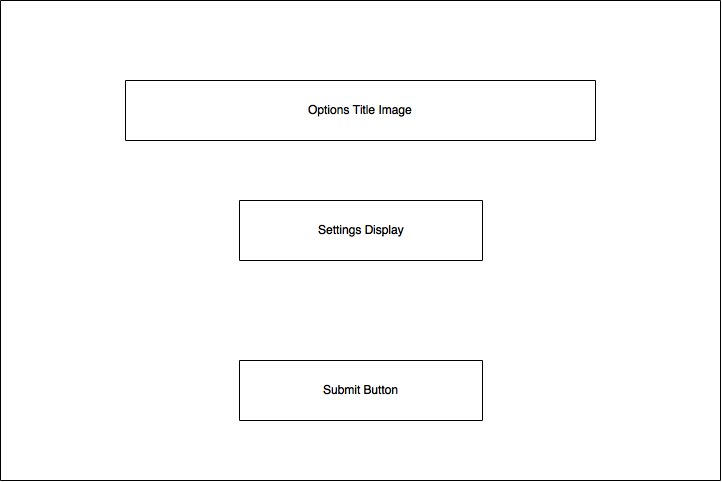
1. **Operating System:** Client operating system will remain unspecified as any operating system supplying a web browser capable of webGL and HTML5 rendering can be used.
2. **Web Browser:** Any browser with JavaScript enabled capable of rendering HTML5 and fetching assets from the web server over http/https.
3. **Index:** The HTML file loaded into the browser, used for displaying the HTML5 canvas and other DOM elements needed by the JavaScript files.
4. **Net Interface:** This file will contain an interface abstracting all network communications to the server. It will contain AJAX calls to the server required to fetch resources that can be executed as method calls from other files.
5. **Game:** A singleton object file responsible for maintaining all possible states of the game. Interacts with the network interface module and holds the current state of the game for the renderer and level generator.
6. **Level Generator:** Used to generate the 8x8 match-three puzzle required for all stages.
7. **Renderer:** A module used to render the stage created by the level generator to the canvas and handle user interaction such as clicks and button presses.

## Game Flow

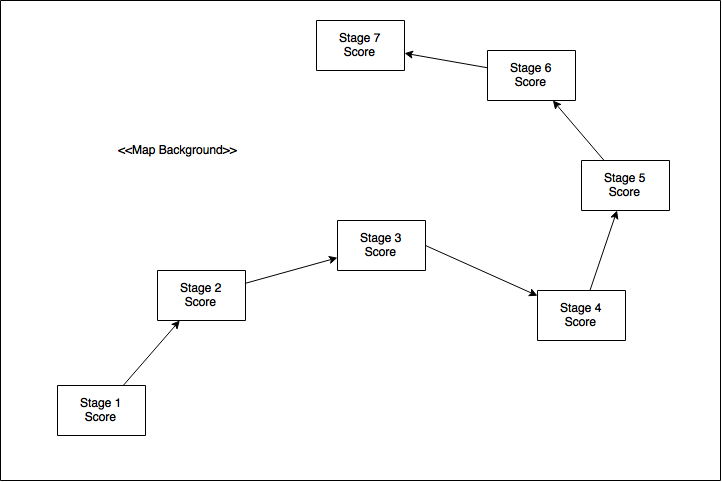
1. Title Screen: Initial starting screen allowing the player to branch to the options menu, start a new game as a new user or continue their previous game by providing user details.



1. Options Screen: After selecting the options setting from the initial start menu screen, the player will see an options screen allowing them to set various options settings. The player will return to the main menu with settings changed on submission.

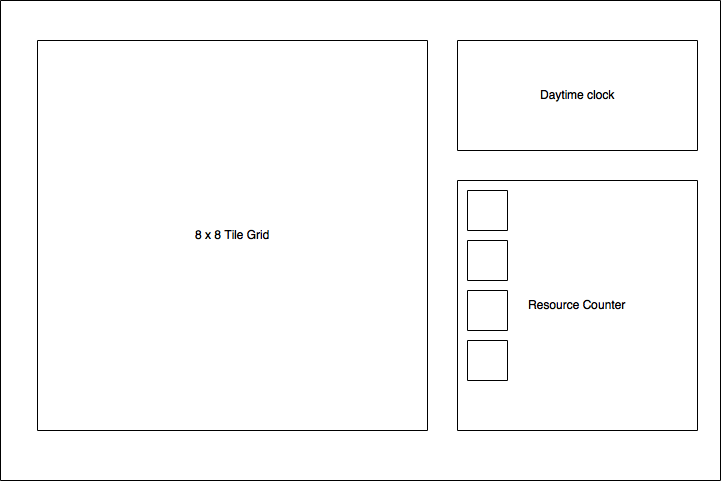


1. Stage Selection Screen: Upon successful creation or retrieval of user from the server, players will enter the stage selection map screen. This will provide clickable buttons for each stage completed or unlocked along with their score.

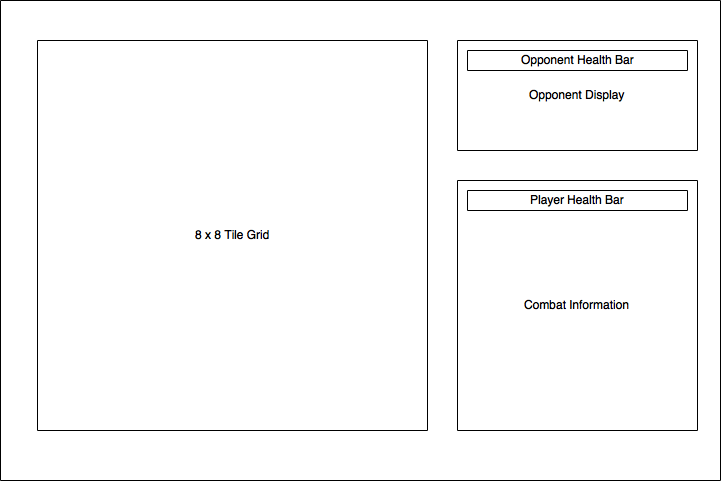


1. Upon selecting a stage, the user will transfer to one of three states, depending if the stage selected was a standard, combat or boss stage. The main puzzle loop will execute on any of the indicated stages while the match-three puzzle is completed. Elements on the stage display will be discussed in the play screen section of this report.

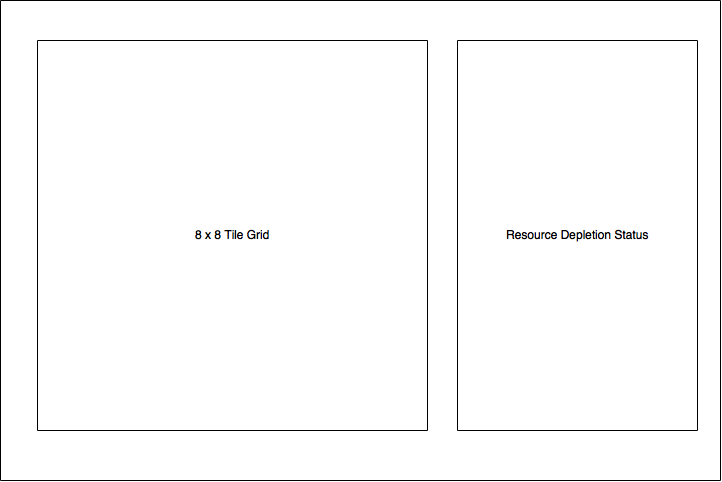
Standard Stage:



Combat Stage:



Boss Stage:



1. Once the puzzle is completed or failed, and the stage is completed, the player returns to the stage selection state in order to select the next stage.

## Graphics

This section discusses the 3D/2D graphics elements of the game. Focus on the core aspects as opposed to interface elements, as these will be discussed later. (This would include rendering, sprite management, and so on.) If there are any constraints on the type of content that can be used here, it should be discussed.

## Audio

This section discusses the sound elements of the game, including music playback, sound effect playback, and so on. If there are any constraints on the type of content that can be used here, it should be discussed.

## Artificial Intelligence

This section discusses the control of all non-player active entities in the game. Identify how this control is done (scripting, state machines, rules, and so on) and provide details for each entity.

## Networking (If Applicable)

If there are online features to the game, they should be discussed here. This includes a description of protocols used, and other technical details.

## Game Objects and Logic

This is a discussion of all gameplay objects and logic used to support the game, as opposed to the various core subsystems discussed above. (One way of thinking about it is that these are all the game specific elements to your game … the other aspects described in the subsections above could theoretically be provided by an engine, and might be, depending on your game.)

This should include subsections for each main game object or set of objects.

## Data Management and Flow

Game data will need to be loaded and saved at various points in time. This section should describe the procedures used to do loading, saving, and any processing of this data (such as compression/decompression, encryption/decryption, parsing, and so on), as well as the various different types of files used to contain this information.

# User Interface

This section describes how information is presented to the player, and how input is received in return. This includes display layout, feedback, controls, and so on.

## Game Shell

This discusses all screens presented to the player outside of game play. If there will be sound, music, or tactile feedback in any of these screens, it should be discussed. Controls should also be discussed to describe how screens are changed or how interface elements (buttons, etc.) are manipulated by the player.

The game will likely have menus given to the player to control the flow of out-of-game activities. This includes setting game options, controlling game flow (starting, stopping, pausing, and so on), and a variety of other tasks. The various menus must be identified, as well as the flow between them, and circumstances under which they appear and disappear.

Additionally, other screens may be presented to the player. This includes splash screens with logos and/or legal notices, credit screens, and so on. These should also be discussed.

Diagrams should be provided both as concept art and to illustrate screen flow.

## Play Screen

This is a discussion of what is presented to the player during game-play. This includes the depiction of all game elements, as well as other information presented in the game’s heads up display (HUD). Any audio and tactile feedback provided should also be discussed. Diagrams should be provided as concept art to illustrate how the game screen appears during play.

In addition, a discussion of all controls should be provided. Not only should this describe what each control is and how it impacts gameplay, but this should also include how control events are processed and disseminated to the various subsystems and game objects as described above.

# Technical Risk

In this section, you discuss the various elements of technical risk associated with the development of your game. This can include things such as unfamiliarity with the platform or development tools being used, complex aspects of the game, other constraints on your schedules or resources, and so on. Each risk should be accompanied by a strategy to minimize, reduce, or mitigate the risk, to ensure that you will be able to handle the challenge appropriately.

Each risk (or set or related risks) can be discussed in their own subsection.