# Match Quest

# Technical Design Document

Version 1.1.0

Riley Bloomfield

250670211

CS4483 – Game Design

Winter 2016

## Version History

| Version Number | Edited By | Date |
|---|---|---|
| **1.0.0** | Riley Bloomfield | 2016-03-13 |
| **1.1.0** | Riley Bloomfield | 2016-03-16 |

## Table of Contents

## 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. |
| Tile | An icon present on the puzzle grid. Can be a variety of types (appearances) |

# 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 | Sole team member responsible for entire project. |

## Development Environment

Web development can be completed on almost any system with a full browser, console and text editor.

### 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:

#### *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.

#### *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.

#### *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.

#### *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.

#### *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.

#### *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.

### *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.

### *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.

### *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).

### *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.

### *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.

### *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.

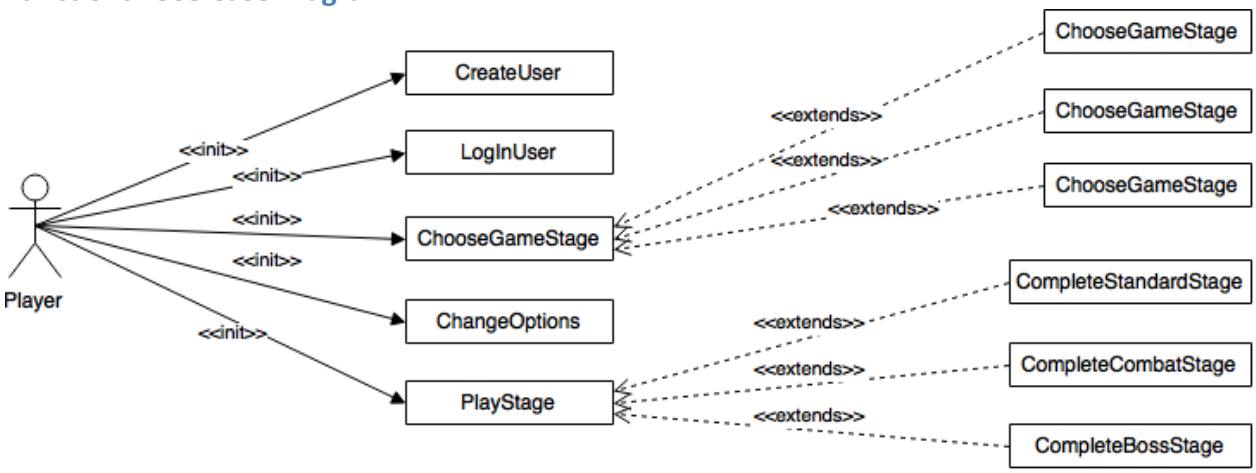
### *Revert Move:*

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.

### *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.

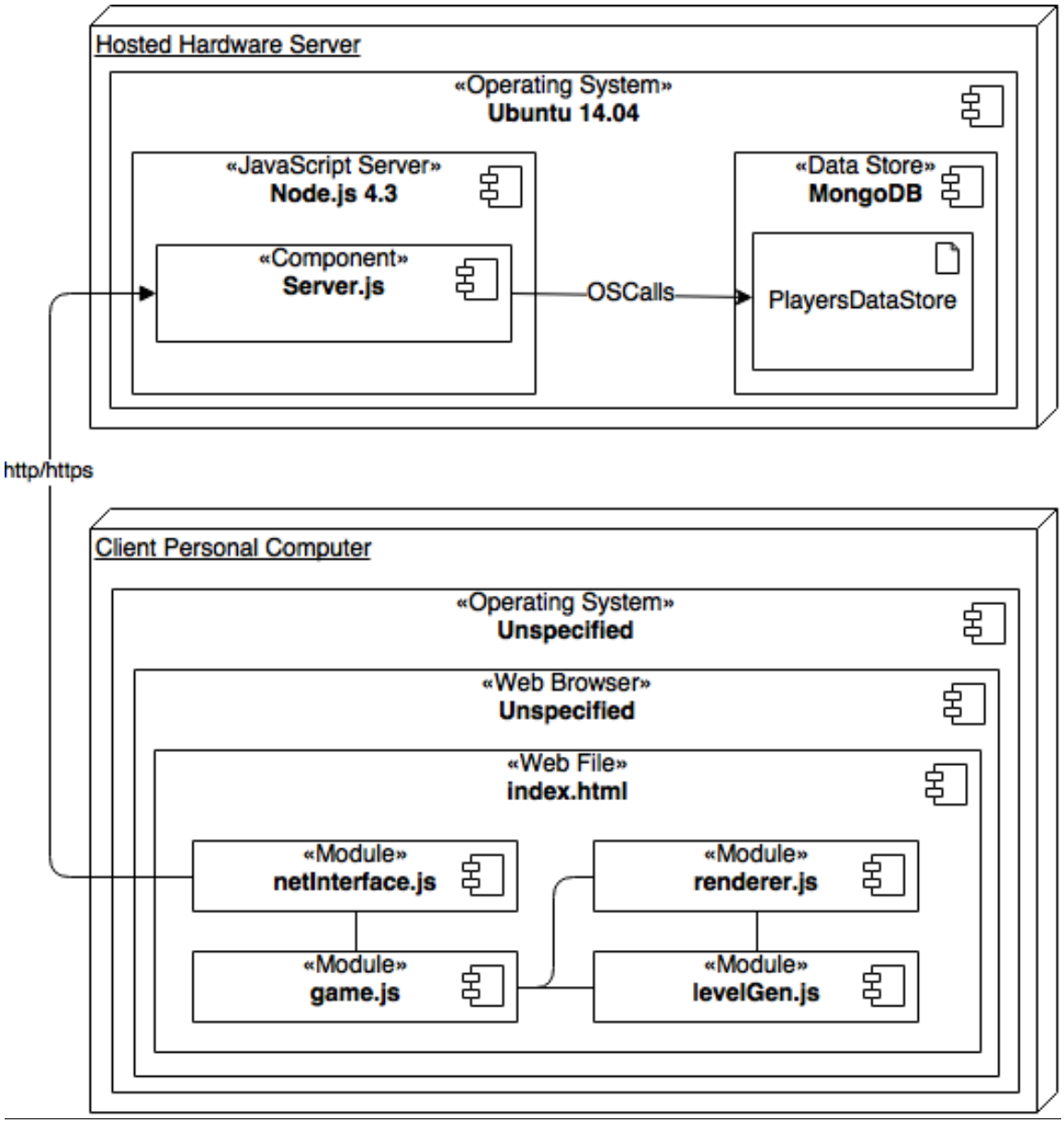
2. When a stage is selected, asset loading should not take more than ten seconds, at least 80% of the times stages are selected.

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:*

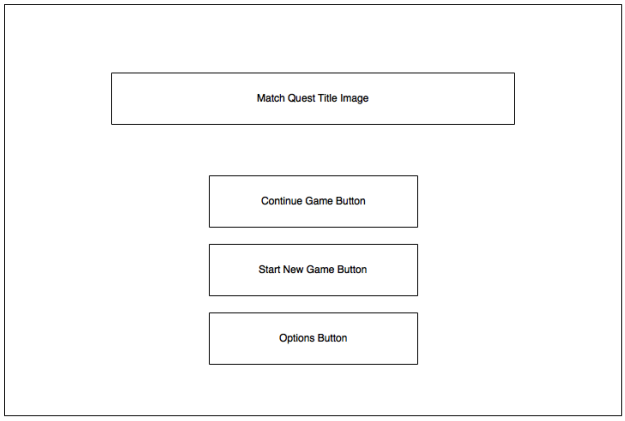
1. **Hardware Server:** Details as specified in development server hardware.
2. **Operating System:** Ubuntu 14.04 without GUI elements, Linux distribution running on the server.
3. **JavaScript Server:** Node.js version 4.3 will be used with Express.js middleware. This will contain the JavaScript server 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:*

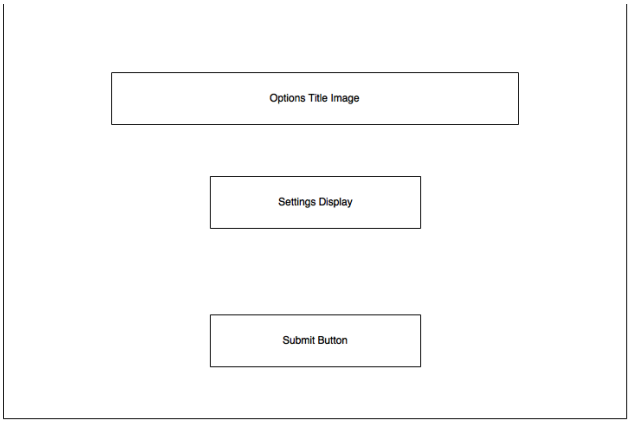
1. **Operating System:** Client operating system will remain unspecified as any operating system supplying a web browser capable of webGL and HTML5 rendering can be used.
2. **Web Browser:** Any browser with JavaScript enabled capable of rendering HTML5 and fetching assets from the web server over http/https.
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

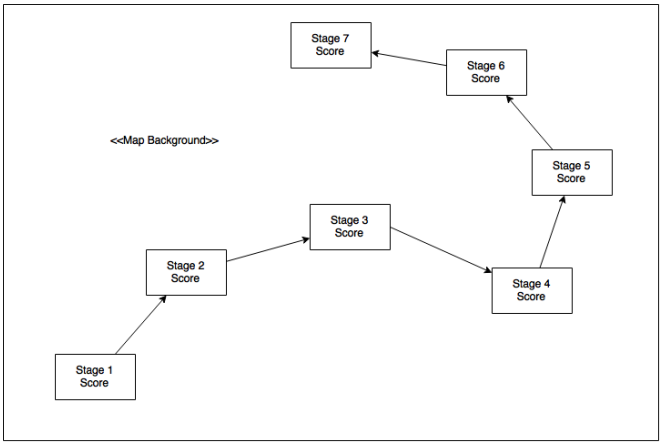
1. Title Screen: Initial starting screen allowing the player to branch to the options menu, start a new game as a new user or continue their previous game by providing user details.




2. 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.

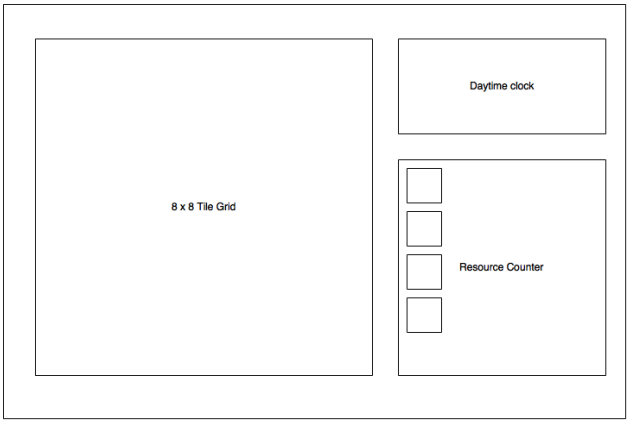



3. Stage Selection Screen: Upon successful creation or retrieval of user from the server, players will enter the stage selection map screen. This will provide clickable buttons for each stage completed or unlocked along with their score.

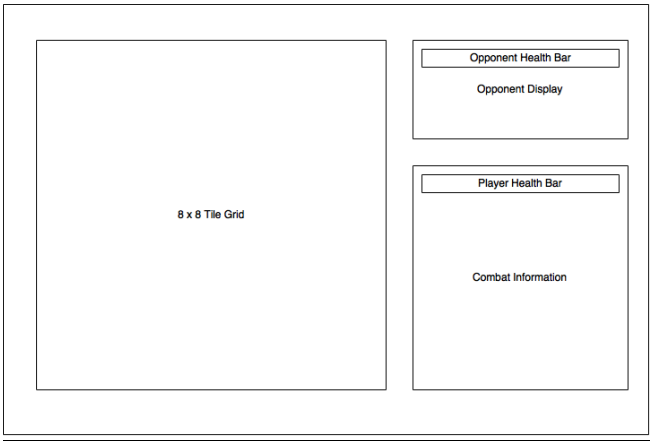



4. Upon selecting a stage, the user will transfer to one of three states, depending if the stage selected was a standard, combat or boss stage. The main puzzle loop will execute on any of the indicated stages while the match-three puzzle is completed. Elements on the stage display will be discussed in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

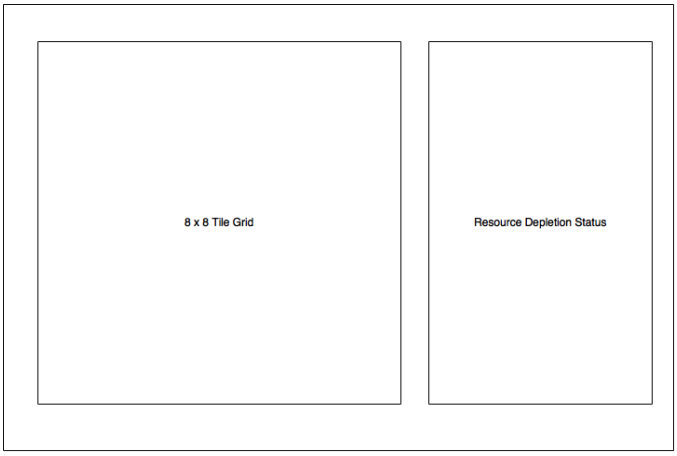
Standard Stage:




Combat Stage:




Boss Stage:




5. 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

Graphics will be rendered entirely through the HTML5 canvas. Tutorials and cut scenes will be entirely 2D. All assets will be created for Match Quest, with no protected materials.

### Tiles:

Tile assets for puzzle grid elements will be loaded as individual sprite sheets 300 px in width and 100 px in height. Each tile sprite sheet will contain three state representations of a tile: hovered, selected, and clicked. Tile sprite sheets will be saved in jpg compression format.

### Background Assets:

Background assets will be saved in jpg-compressed format in a resolution of 720px in width and 480px in height. They will be found in the public folder under res/backgrounds.

### Menu Assets:

Menu buttons will be loaded as sprite sheets containing three stages of buttons in the order: hover, selected, and clicked. Button sprite sheets will by 300px in width and 45px in height. Animated Text will be rendered as jpg images or gif formats of various sizes. The menu assets will be found at public/res/menus

## Audio

Audio files will be contained as assets in MP3 compressed format. These assets will be loaded with preloadJS on start of stage and played on command with the SoundJS library methods. Audio assets will be broken into two formats: actions and ambient loops. Sound assets should be found in the public hosted folder under res/actions or res/ambientLoops. All assets will be created for Match Quest, with no protected materials.

### Actions:

- 5-second MP3 assets used on event triggers such as button clicks, error messages, and warnings.

MenuSelect.mp3 – Triggered on button click. Record sharpening rod against steel knife.

NoMatch.mp3 – Heard when move does not result in match. No. 8 Jaw harp std. note.

LimitedDaylight.mp3 – Signaled when std. stage daylight reaches 10% of minimum.

Alert.mp3 – Triggered on player notification. Metal struck against thin glass.

Chime.mp3 – Successful match

Success.mp3 – Successfully completed a stage

Failure.mp3 – Failed a stage

### Ambient Loops:

- Various length MP3 assets to be looped throughout gameplay.

MenuLoop.mp3 – Unspecified

StandardStage.mp3 – Unspecified

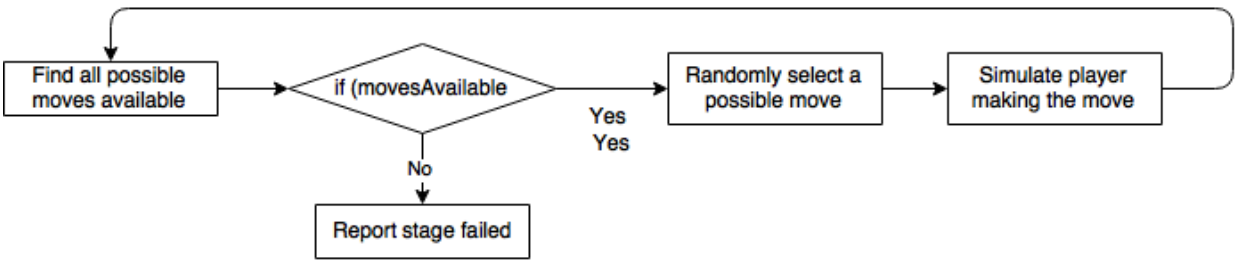
CombatStage.mp3 – Unspecified

BossStage.mp3 – Unspecified

## Artificial Intelligence

Although only the player is the only active character through Match Quest's progression, a very simple artificial intelligence will be implemented to sold the match-three puzzle. This puzzle solving feature will be used for testing stages to ensure they are possible to complete as well as possibly offering the player a suggested move feature.

The artificial intelligence design to solve puzzles similarly to a player is outlined below:



## Networking

The game will be accessed through a web server, to load the game page and all game assets. All pages and assets will be transferred through http protocols. Resource caching, persistent connections, and guaranteed transmission are encompassed in the http protocol and will be required by the game. The public, statically hosted folder will be accessed at:

http://<<serverAddress>>/

Player queries will be executed on the /players route to the same web server. Players can be created through a POST as specified below.

Stage scores can be updated as players repeat a stage by a PUT to /players containing the payload indicated below.

Players can be verified by sending a GET request to players. All stage scores will be returned in an array of stage objects containing stage number and score received.

| GET /players?player=<<username>> | |
|---|---|
| Input | Output |
| Search query in URI | {“user_name”:<<username>>, “stages” :<br>[{<br>“stage”: << stage No.>>,<br>“score”: << stage score>><br>}]} |

| POST /players | |
|---|---|
| Input | Output |
| {“user_name”: <<username>>} | {“user_name”: <<username>>} |

| PUT /players | |
|---|---|
| Input | Output |
| {“user_name”:<<username>>,<br>“stages”: [{<br>“stage”: << stage No.>>,<br>“score”: << stage score>><br>}]} | {“user_name”:<<username>>, “stages”:<br>[{<br>“stage”: << stage No.>>,<br>“score”: << stage score>><br>}]} |

## Game Objects and Logic

### Stage Puzzle Logic:

Matching three or more items in a row or column will remove that set of item from the puzzle and contribute to the goal. New, random items will be added to the top of the playing grid

and will cascade down to fill in the removed items. A single item can be moved at a time by selecting the item, and then selecting an adjacent location to move it to. If no match occurs with this move, the move is reversed. Items can be moved vertically or horizontally one location, but not diagonally. Many matches must be completed to solve a puzzle.

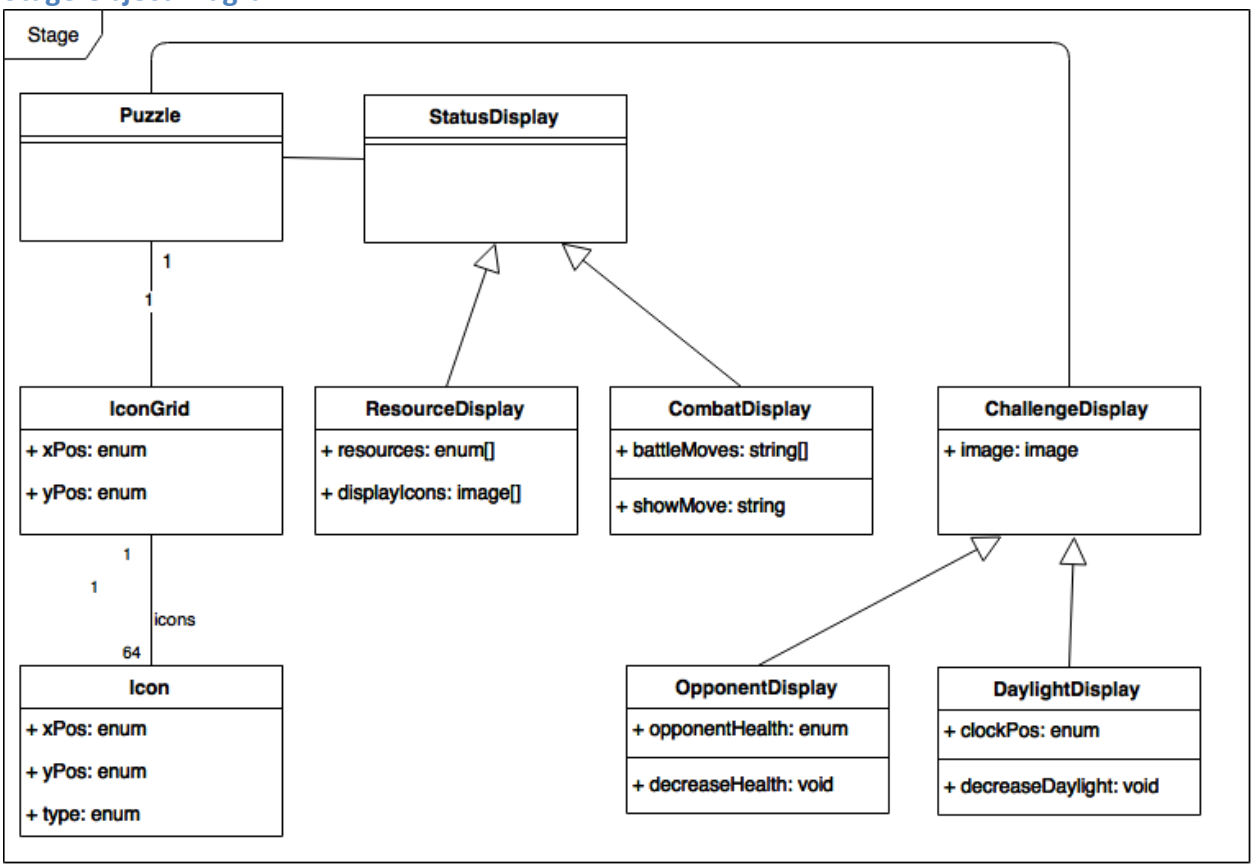
Standard stages will have the goal of collecting an indicated number of certain resources. Matching sets of a resource item on the puzzle grid collects the resource. Matching a larger number of that item will increase the score received for that match. Puzzles will be completed when the indicated number of resources has been obtained.

On combat stages, various creatures will challenge the player. The creatures will be defeated through the same style of matching puzzle, but instead of collecting resources, players will match items related to the desired battle move. Items will include health, block, attack, run and no nothing. Creature battles will be more difficult than regular resource collecting puzzles because the creatures will attack the player with an element of randomness. Creatures must be defeated in order to progress. The player will have limited health and must defeat the creature before losing all of their health.

In a boss stage, the player will be presented with a puzzle in which they must match items to overcome obstacles and complete a path from the bottom of the playing grid to the top. Resources will not be collected in this round, but will be depleted to limit the number of available moves. Once a match is made in a goal row or column, the grid background will fill in, indicating completion of that section of path. Other matches can be made, but will not contribute to the goal of the round (completing the path) and will still deplete resources. Later variations of this puzzle will involve matching only indicated items overlaying the desired path, increasing difficulty.

Each variety of stage will contain a common game grid container, containing the match-three puzzle. An additional display panel will be located on screen to show stage status, and an additional section to display the challenge. An object diagram is provided below to describe the structure of a stage.

### Stage Object Diagram




Each stage object will contain a puzzle container featuring a grid of icons that can be manipulated. Each icon must handle its click. In addition, a stage will feature a status container that will either display resources (for standard and boss stages) or combat statistics (for combat stages). A final container will be present to display an update of progress through the stage; either an opponent status display or daylight regulator (clock).

## Data Management and Flow

Game data will be transmitted through the indicated networking routes. All loads will be done through a GET route, saving will be accomplished by updating via the PUT route and player creation will be completed through the POST route.

# User Interface

## Game Shell

### Pause/Play Functionality:

Because Match Quest is an un-timed puzzle game, there is no need for a pause menu during main game play. Play can be stopped at any time during a stage without consequence.

### Audio Feedback:

Audio feedback will be provided as described in the audio section of this report. The format of audio files is also specified in the same section.

### Menu Buttons:

Buttons should change according to both mouse hover and clicks. Play sessions will be loaded, created and updated completely through web interaction, which is driven by mouse events.

### Stage Buttons:

While playing a stage, all icons available in the puzzle should be reactive to clicking and hovering. Control of the puzzles will be handled entirely through mouse events.

### Asset Loading Screens:

All assets required for a stage must be loaded in advance to entering a stage. This will be communicated to the user through a simple loading screen featuring a loading bar to inform the player of the loading progress. Once assets are loaded, the player will proceed to the stage to resume play.

### Notifications and Alerts:

Any network alerts or achievements will be communicated to the player through pop up alerts. Animation and formatting of the alerts will be facilitated through the included notification library indicated in the support files section of this report.

For screen flow diagrams and visual element positions, see section of this report on game flow.

## Play Screen

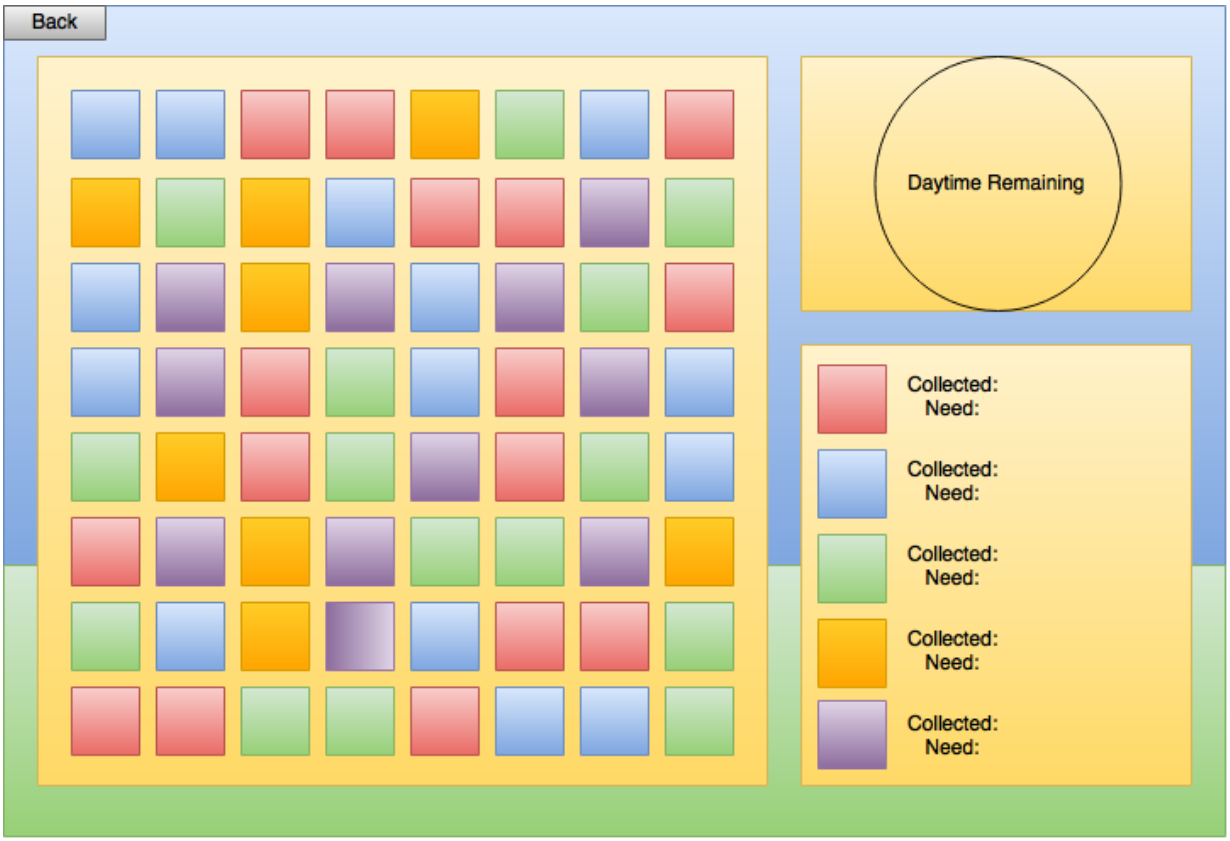
### Gameplay Interaction:

All game control will be allotted to the icon grid in order to complete the match-three puzzle. In addition there will be a return button at the top of the screen to return to the stage selection map screen.

### Icons:

All icons must have a hover event attached to them. The EaselJS library noted in the supporting libraries section allows display objects to have clickable and hover events attached to them. Each display object has collision detection to detect a click. Each icon will have a click handler function assigned to it that lives in game.js to provide game logic. Pairs of icons must be checked for adjacency and match status. Two sequential clicks will be checked to see if a match occurs; icon click handlers will trigger this check.

### Stage Mockup:




Daytime Remaining Indicator: This will provide a clocklike display to the user, informing them of how many moves they have left by means of daylight. As daylight decreases, the background image will turn to night. Each match decreases the daylight level.

## Technical Risk

### *Team Dependency:*

Since the project team for Match Quest is so small, dependency and technical adequacy of the team becomes a high priority risk. Not finishing the game is a risk when the team size is this small. Mitigation of this risk has been implemented by already starting to develop and create the game before technical document approval. Fast tracking is a viable and practical solution to this risk.

### *Library Dependency:*

The development libraries specified could lose support through the development process. This is a low priority risk as the development time of this project is short in comparison to the development of the libraries. Storing a local copy of the libraries to use, pending licensing requirement changes has mostly mitigated this risk.

### *Maximum Asset Loading:*

It may be found through development that the asset-loading library selected is inadequate for the number of resources that must be loaded at the start of each stage. This is a low priority risk, as the total memory size of these assets is not expected to approach more than a few MBs.

### *Development Hardware Failure:*

In the event of current development hardware failure, additional hardware must be available to continue development. This is a high priority risk as, although it is not likely, would strongly impact the success of the project. Mitigation strategies have been provided by adequate budget enhancements as well as the potential use of exterior resources.

### *Complexity of Development:*

Technologies needed to develop this project seem to be within the skills of the development team, so a low risk is assigned to the project being over complex.