**Software Requirements Specification**

SLYYDE

Software Requirements Specification

1.0

2/19/2019

<Sundar Sampath, Andrew Dodel, Andrew Jia, Brenden Campbell, Yuxin Zhang>

Software Engineer

Prepared for

SLYYDE Group

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 2/19/2019 | Version 1 | Everyone | First Revision |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# **Document Approval**

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
| *Andrew Dodel* | Andrew Dodel | Software Eng. | 2/19/2019 |
| *Brendan C.* | Brendan Campbell | Software Eng. | 2/19/2019 |
| Sundar Sampath | Sundar Sampath | Software Eng. | 2/19/2019 |
| *Yuxin Zhang* | Yuxin Zhang | Software Eng. | 2/19/2019 |
| *Andrew Jia* | Andrew Jia | Software Eng. | 2/19/2019 |

**Table of Contents**

**REVISION HISTORY II**

**DOCUMENT APPROVAL II**

**1. INTRODUCTION 1**

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

**2. GENERAL DESCRIPTION 1**

2.1 Product Perspective 1

2.2 Product Functions 1

2.3 Constraints, Assumptions and Dependencies 1

**3. SPECIFIC REQUIREMENTS 1**

3.1 External Interface Requirements 2

*3.1.1 Software Interfaces 2*

3.2 Functional Requirements 2

3.3 Use Cases 2

3.4 Classes / Objects 2

3.5 Non-Functional Requirements 2

*3.5.1 Performance 3*

*3.5.2 Reliability 3*

*3.5.3 Availability 3*

*3.5.4 Security 3*

*3.5.5 Maintainability 3*

*3.5.6 Portability 3*

3.7 Logical Database Requirements 3

3.8 Other Requirements 3

**4. ANALYSIS MODELS 3**

4.1 Use Case Diagram 3

4.2 Class Diagram 3

4.3 Sequence Diagram 3

4.4 Activity Diagram (DFD) 3

4.5 State-Transition Diagram (STD) 3

**5. CHANGE MANAGEMENT PROCESS 4**

**A. APPENDICES 4**

# **1. Introduction**

*This first clause, the introduction, seeks to establish, in some detail, the purpose of this SRS, the scope of the project, and to clearly define any obscure or technical terminology that is used throughout the document.*

## **1.1 Purpose**

*SRS Purpose and Audience*

This SRS is intended to provide a thorough, semi-technical description of the web application Slydde, a slide puzzle browser game. Thus, it shall establish expected behaviors, outline use cases, define functional and non-functional requirements, identify dependencies, and detail best practices for enhancement and modification of this document. This SRS is intended for potential users of Slyyde and those who want to contribute to the project.

## **1.2 Scope**

*This subsection should:h h*

*(1) Software product’s name*

*(2) What the software product will do and won’t do (just high level, no details)*

*(3) Describe the application of the software being specified. Describe all relevant benefits, objectives, and goals as precisely as possible.*

1. The software product’s name is *Slyyde*.
2. Slyyde will be a web application that allows users to play a sliding puzzle game on various different boards. The user will also have the option to upload their own image, which then becomes a playable board. Users will be able to create an account, allowing them to save their high scores. Leaderboards will be maintained for every board and stored in a database. There will be multiple difficulty levels, providing challenge and entertainment for all skill levels.
3. Slyyde is intended to provide entertainment and friendly competition. Slyyde aims to provide a clean and well-designed environment for users to complete slide puzzles. By supporting image uploading by users, the intention is for Slyyde to provide a vast, diverse set of playable boards, that facilitate diverse and exciting games. As a web application with leaderboard support, a prime objective of Slyyde is to foster competition and a sense of progression.

## **1.3 Definitions, Acronyms, and Abbreviations**

*This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.*

***User:*** A *user* in this context is anyone who visits and interacts with the web application.

***Sliding Puzzle:*** A sliding puzzle is a sequential move puzzle, meaning that it consists of a set of pieces that can be moved around to form various different combinations. In this context, a sliding puzzle is *solved* when its pieces are arranged in one specific combination, which generally forms a picture.

***Game Board:*** A game board refers to the N x N grid comprised of the various pieces in the puzzle and a single empty piece. The user directly interacts with this grid, either by clicking one of the pieces or through the use of the directional arrow keys.

***Board Tiles:*** A board tile refers to an individual piece on the game board. A board tile may only move to an empty tile and only if it that empty tile is directly adjacent to it.

***Game:*** A game, in this context, begins after a user has selected a board (based on pictures) and a difficulty setting, and has selected the “Start” button, after which the board is displayed (with its pieces randomly shuffled) and a timer begins. During a game, a user moves around the pieces on a game board until the appropriate combination is reached, after which the timer will stop and the game will end.

***Game Score:*** The game score refers to the metric used to assess performance. The current metric is duration, where lower game times correspond to a higher ranking/performance.

# **2. General Description**

## **2.1 Product Perspective**

The sliding puzzle is a fun, simple, and challenging game that anyone can readily enjoy. Slyyde is a web application that allows users to play browser based sliding puzzle games, for free. Slyyde’s core functionality is very similar to sites such as ruwix.com, which also supports user supplied photos as backdrops for boards. Slyyde looks to differentiate itself from such similar products by providing extensive leaderboard support, supporting multiple difficulty levels, boasting a more inviting and sophisticated UI, and providing different game modes with slight variations in rules.

## **2.2 Product Functions**

The Slyyde web application will support the following functions:

1. Allow users to upload images.
2. Allow users to view leaderboards for every board and difficulty.
3. Allow users to view all available boards.
4. Allow users to play a game with a selected board.
5. Allow users to register for an account.
6. Allow users to log in using their credentials.
7. Allow users to save high scores to database.

## **2.3 Constraints, Assumptions and Dependencies**

**2.3.1 Constraints**

The performance of Slyyde may be constrained by the particular browser that is being run on. Additionally, performance may be constrained by high levels of CPU utilization.

**2.3.2 Assumptions**

The operation of Slyyde assumes that the user is using a relatively modern machine, running a supported and relatively up-to-date operating system , has a reliable and secure internet connection, and hasworking ability in using a browser and the internet.

**2.3.3 Dependencies** Slyyde is not operating system dependent (given that the operating system is relatively up to date) but is browser dependent, with official support only guaranteed for Chrome and Firefox.

# **3. Specific Requirements**

# 3**.1 External Interface Requirements**

### **3.1.1 Software Interfaces**

## **3.2 Functional Requirements**

### **3.2.1** Difficulty Selection

The player can select, from the GUI, one of four different difficulties (easy - 5x5, medium - 7x7, hard - 10x10, or crazy - 15x15). Upon difficulty selection, a game board of the appropriate dimension will be generated.

### **3.2.2** Game Creation

Upon difficulty selection, a game board of the appropriate dimension shall be generated.

### 3.2.3 Game Score

The player's current score, time and turn, shall be displayed and continuously updated ten times every second.

### 3.2.4 Game Board

The game board shall consist of tiles, the number of which comes from the game difficulty. Tile placement shall always start randomized.

### 3.2.5 Game Moves

One tile will be black, and this is the tile that can be moved around and swap places with adjacent tiles . The player shall be able to move this tile by clicking adjacent tiles or using the “wsad” or arrow keys.

### 3.2.6 Valid Moves

Tiles that are not adjacent to the black tile cannot be moved to.

### 3.2.7 Board Creation

Game boards can be generated from either stock photos inherent with the game, or from user-uploaded pictures.

### 3.2.8 Board Creation

Pictures, as well as user data, shall be stored on a server that communicates with the web app.

### 3.2.9 Game Completion

Upon successful completion of a game, the application shall display a congratulatory message to the user, displaying the score for that game.

### 3.2.10 High Score Accounting

Game score will be sent to the server to update the running list of highscores. Every times high scores are checked, the webapp must interface with the server to get this “official” list rather than using what is local.

## **3.3 Use Cases**

**Use Case 1:** Difficulty Selection

**Created By:** Andrew Jia **Last Updated By:** Andrew Jia

**Date Created:** 2/16/19 **Last Revision Date:** 2/16/19

**Actors:** The primary actor is the player.

**Description:** To account for the different skill levels of players, the game will allow users to choose a board size, which will vary the difficulty of the puzzle. The game will

then generate a board with the dimensions chosen.

**Trigger:** 1. The user chooses a provided picture to play the game.

2. The user uploads their own picture to play the game.

**Preconditions:** 1. The user has chosen to play a game.

2. The user has chosen a picture to play.

**Postconditions:** 1. Board is split into appropriate dimensions.

**Normal Flow:** 1. User selects a difficulty represented by board dimensions.

2. Board divides picture (selected or uploaded) into appropriate dimensions.

**Alternative Flows:** None. A new board is created every time a new game is initiated.

**Exceptions:** The board runs into an error regarding picture uploaded and cannot create board

of requested dimensions. The system will display an error message and prompt

the user to exit or upload a different picture.

**Includes:** Use Case 3

**Frequency of Use:** Every time a new game is initiated.

**Special Requirements:** Creating a new board for every new game does require excessive

resources.

**Assumptions:** 1. User understands English.

**Notes and Issues:** If creating a new board every time is creating too much overhead, saving

and randomizing common boards should be considered as an alternate flow.

**Use Case 2:** Game Initiation

**Created By:** Andrew Jia **Last Updated By:** Andrew Jia

**Date Created:** 2/16/19 **Last Revision Date:** 2/16/19

**Actors:** The user will be the primary actor.

**Description:** To be able to play the game and interact with the program, a game will have to be created when the user chooses to play. The outcome will be the creation of a board that is ready to be played, and score counters that are ready to start as soon as the player makes their first move.

**Trigger:** The user chooses to play a game from main menu.

**Preconditions:** User selects to play a game from main menu.

**Postconditions:** 1. Board is generated.

2. Timer and move counter are ready to start.

3. Game can be started.

**Normal Flow:** 1. Player chooses to play a game.

2. Player chooses to upload their own picture or select one from the set given.

3. Player chooses a difficulty represented by different puzzle dimensions.

4. Board is generated and split into specified dimensions.

5. Board is randomized and displayed to user.

6. Game resets score counters and waits for player’s first move.

**Alternative Flows:** After a game is played, user chooses to replay the game. If this is the case,

the board randomizes, the game resets score counters and waits for the player’s first move.

**Exceptions:** None.

**Includes:** Use Cases (By ID): 1, 3

**Frequency of Use:** Every time a new game is played.

**Special Requirements:** None.

**Assumptions:** User understands english.

**Notes and Issues:** None.

**Use Case 3:** Board Generation

**Created By:** Andrew Jia **Last Updated By:** Andrew Jia

**Date Created:** 2/16/19 **Last Revision Date:** 2/16/19

**Actors:** The primary actor of this use case is the user.

**Description:** To create a new game, a new board must be generated randomly so that playing

the game feels entertaining and different every time a new game is initiated.

**Trigger:** The user selects a difficulty represented by the dimensions of the puzzle they wish to

play.

**Preconditions:** 1. The user has selected to play a game from the main menu.

2. The user has chosen or uploaded a picture for the puzzle.

3. The user has chosen a difficulty.

**Postconditions:** 1. A board is generated with appropriate dimensions.

2. Board is randomized.

**Normal Flow:** 1. Board splits picture into specified dimensions.

2. Board randomizes.

**Alternate Flow:** None.

**Exceptions:** None.

**Includes:** Use case 4.

**Frequency of Use:** Every time a new game is initiated.

**Special Requirements:** Board randomization does not often allow users significantly easier

puzzles.

**Assumptions:** Board does not randomly generate super easy puzzles.

**Notes and Issues:** Should easy puzzles be generated too often, should consider extra steps to

ensure difficulty of game and integrity of high scores.

**Use Case 4:** Picture Selection

**Created By:** Andrew Jia **Last Updated By:** Andrew Jia

**Date Created:** 2/16/19 **Last Revision Date:** 2/16/19

**Actors:** The primary actor of this use case is the user. Secondary actor is the database.

**Description:** To give users a more varied and enjoyable experience, the system will allow them

to select or provide their own pictures to play the puzzle with. The puzzle will

then be generated with the picture chosen.

**Trigger:** User chooses to play a game from main menu.

**Preconditions:** User has chosen to play a game from the main menu.

**Postconditions:** The selected or provided picture is stored and ready to be used for board

generation.

**Normal Flow:** 1. User is prompted to select or provide a picture.

2. Selected picture is retrieved from database.

3. Picture is passed to Board and ready to be used.

**Alternate Flow:** 2a. User opts to provide a picture.

3. Picture is uploaded to database.

4. Picture is passed to Board and ready to be used.

**Exceptions:** None.

**Includes:** No other use cases.

**Frequency of Use:** Every time a new board is generated.

**Special Requirements:** None.

**Assumptions:** Users upload pictures that have unique enough parts to reasonably be able to

solve the puzzle.

**Notes and Issues:** None.

**Use Case 5:** Board Tile Movement

**Created By:** Andrew Jia **Last Updated By:** Andrew Jia

**Date Created:** 2/16/19 **Last Revision Date:** 2/16/19

**Actors:** The primary actor of this use case is the user.

**Description:** To be able to play the game, users must be able to move tiles to the empty space

on the board. After clicking on a tile adjacent to the empty space, or using one of

of the “wasd” keys, the corresponding tile will move to the empty space and

count as a move.

**Trigger:** User clicks on a valid tile.

**Preconditions:** 1. System is in game playing state.

2. User clicks on a valid tile.

**Postconditions:** Tile is moved to empty space and displayed to user.

**Normal Flow:** 1. Receive information from game about which tile is to be moved.

2. Move tile to empty space.

3. Update display for user.

**Alternate Flow:** 2a. Tile is unable to be moved to empty space.

3. Print error message ( Game class should handle valid moves)

**Exceptions:** None.

**Includes:** No other use cases.

**Frequency of Use:** Many times per game.

**Special Requirements:** None.

**Assumptions:** Game does not pass invalid tiles to be moved.

**Notes and Issues:** This use case is for the board only. Another Use Case that includes this and is focused for the Game class will call this.

**Other Use Cases:**

1. **Solution Check**
2. **Game Tile Movement**
3. **Board Randomization**
4. **Score Incrementation**
5. **High score Documentation**

## **3.4 Classes / Objects**

### **3.4.1 User**

3.4.1.1 Attributes

* string userID
* string password

3.4.1.2 Methods

* identify(): user identification, return false if it fails, return true if userID and password are correct

### **3.4.2** Picture

3.4.2.1 Attributes

* int pictureID: to access to the specific picture from database
* double[][] highscores: 4\*10 high scores for the specific picture

3.4.2.2 Methods

* getID(): get the pictureID
* getHighscore(int difficulty): get the high scores for the specific difficulty

**3.4.3 Board**

3.4.3.1 Attributes

* int difficulty
* double[] highscore
* int[][] grids: the value of each grids are initialized, starting from 1 to its size, from left to right, from up to down, represented the correct location it should be
* picture selectedPicture: picture object selected by user
* int[] blank: represent the location of blank tail(x,y coordinate)

3.4.3.2 Methods

* split(): get the copy of the picture and split the picture into specific dimension
* randomize(): randomize the grids and randomize the empty tile
* movability(): check if it is a valid move: target is adjacent and in the boundary
* moveblank(): move blank tail to target location, update current board
* display(): display current board
* checkIfWin(): check if current board is correct, display winning message, score and top 10 highscores if win

**3.4.4 Game**

3.4.4.1 Attributes

* board: the board object

3.4.4.2 Methods

* getDifficulty(): ask user for difficulty selection
* getID(): get the pictureID for picture user selected from database
* getNewPicture(): if user select to upload picture, generate a new pictureID and initialize the high scores, for user-uploaded picture, return the picture object
* start(): start new game, create one board per game
* move(): monitor next move
* updateToDatabase(): update high score and user data

## **3.5 Non-Functional Requirements**

*Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).*

### **3.5.1 Performance**

Game moves should have a sub 1 second response time.

Puzzles should be created within 5 seconds.

### **3.5.2 Reliability**

The game shall be reliable, having less than 1 crash per 100 games.

### **3.5.3** Usability

75% of users should be able to complete at least one puzzle.

Users should misclick less than 10% of the time when using a mouse.

### **3.5.5 Maintainability**

New game types should be able to be implemented and integrated seamlessly with little overhead within 2 days of development time.

### **3.5.6 Portability**

Game should run on Firefox and Google Chrome, and it will work on 90% of machines.

### 3.5.7 Enjoyment

75% of users would say they enjoyed doing one of the puzzles.

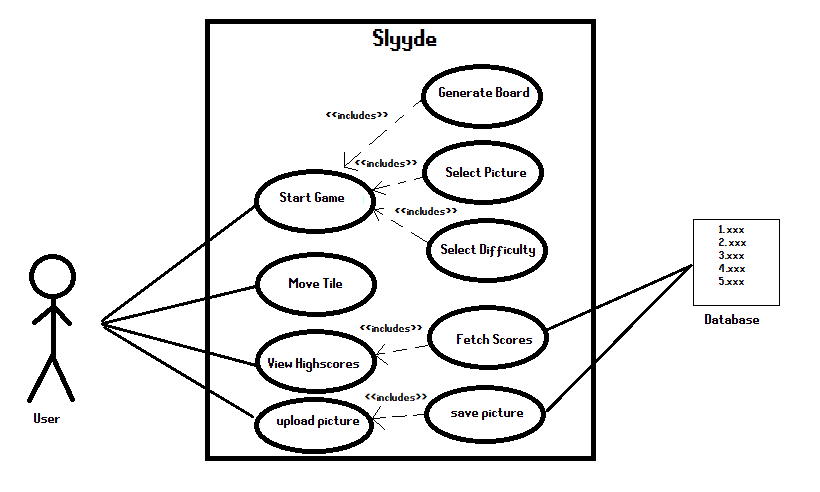
## **3.6 Logical Database Requirements**

A Flask server running SQL lite will be used to store user high scores on the cloud. JPEG will be used to store images uploaded by users to save space. Quality loss should not be an issue, as the image will not be edited. For each picture object, there will be a high score list for each difficulty. Each list will have the top 10 fastest users and their times. There will be ample space to store the data. Each time a user gets a new high score, it will be added to the database. After every game, the high score list for that board is retrieved and displayed.

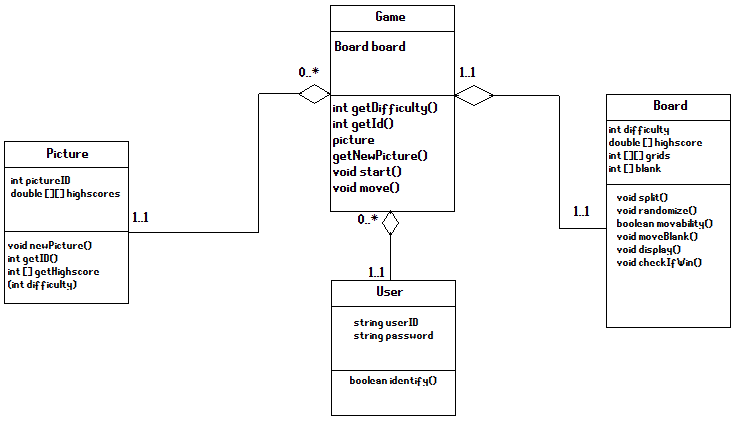
## **3.7 Other Requirements**

# **4. Analysis Models**

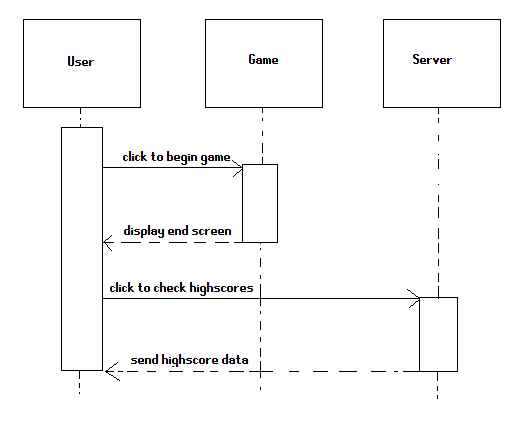
## **4.1 Use Case Diagram**



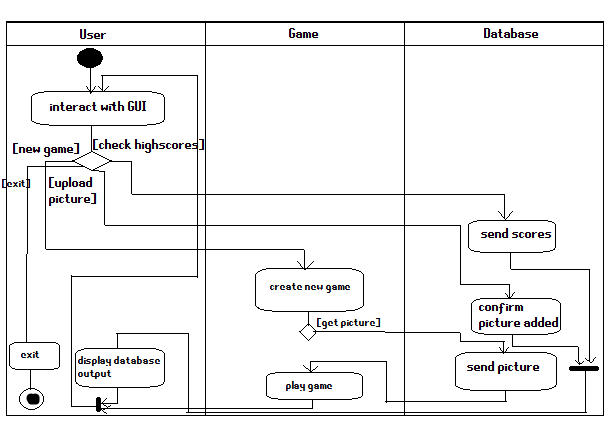
## **4.2 Class Diagram**



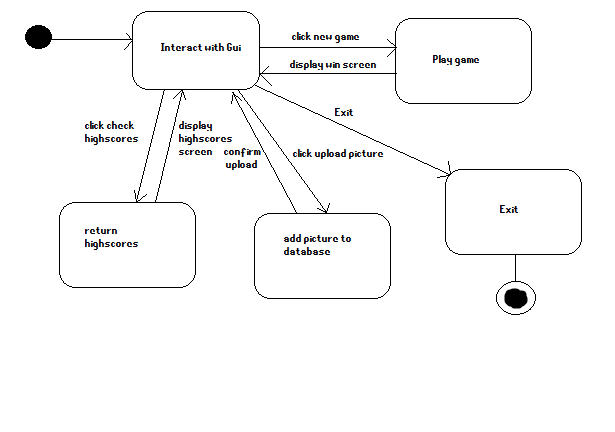
## **4.3 Sequence Diagram**



## **4.4 Activity Diagram**



## **4.5 State-Transition Diagrams (STD)**



# **5. Change Management Process**

The following steps should be followed if one wishes to make a change to this document. Please note that only the authors listed in the first page of this document may propose changes to this document.

*The terminology here is based upon the Git version control system.* *Github and Git will be the version control technologies utilized for the change management process.*

1. Obtain a local copy of this document through your desired version control system (Git) from this projects Github repository. If you have not yet done so, first fork this projects repository at <https://github.com/andrewdodel/CS1530_Project_Repo> , then clone it to your local machine.
2. Ensure your forked repository is up to date. <https://stackoverflow.com/questions/7244321/how-do-i-update-a-github-forked-repository>
3. Create a separate branch, titled descriptively, upon which you will make changes to this document.
4. Modify your local copy of this document with changes you feel are necessary. These changes may reflect/address:
   1. Current state of the project
   2. New client needs
   3. Enhancement decisions
   4. Feature reductions due to time constraints
   5. Ambiguity in the document
5. Commit your changes to the recently created branch.
6. Push your recently created branch to your Github.
7. Open a pull request, and describe the changes you’ve made.
8. Once all other members have reviewed the pull request, a decision will be made regarding the incorporation of the proposed changes.
9. At this point in time, changes may be made at any point in the project’s development life cycle.

# **A. Appendices**

None so far.