DUNGEON CODER

Team 23: Design Document

Ryan Teo Yii Wen	0028791734	Zhuohang Li	0028283889
Ley Yen Choo	0028729283	Devon Lee	0028067918

Index

Purpose

Learning how to code can be a very difficult and boring task for kids and people who have no coding experience. The initial experience of learning the basics of programming can be very challenging and tedious which results in many giving up at the beginning. Our project aims to create a captivating and interesting game that can teach people on the basics of programming while at the same time providing tools to help teachers in the classroom.

Functional requirements

1. Users can join a class and learn about the basics of programming

   As a student,

   a) I would like to register an account and manage my information.

   b) I would like to easily browse and repeat a stage.

   c) I would like to see difficulty, objective and explanation of a stage.

   d) I would like to save my progress manually and automatically.

   e) I would like to see how many attempts I took to complete my tasks.

2. Users can ask for help when they fail to complete a stage.

   As a student,

   a) I would like to get hints on a stage.

   b) I would like to discuss, ask questions and respond to others in a discussion page.

   As a teacher,

   a) I would like to respond to student questions on the discussion page.

   b) I would like to review my students' attempted codes.

3. Users can manage their classroom.

   As teacher,

   a) I would like to keep track of my students' progress.
   b) I would like to edit the point values of different tasks.
   c) I would like to see statistics on the class progress as a whole.
   d) I would like to assign tags to certain students that require more attention.
   e) I would like to create announcements for the class.
   f) I would like to control the access of stages of the game.
   g) I would like to set a deadline for the students to complete the stage.
   h) I would like to save some or all of my student's attempted codes.
   i) I would like to assign grades and leave comments on the gradebook.
   j) I would like to show a good example of codes after the deadline is over.

4. Users can play other modes besides the instructional mode.

   As a student,

   a) I would like to easily browse and repeat a stage.
   b) I would like to play as a normal player after I finish working on assignments.
   c) I would like to design my own movements, skills and equipments.

   As a player,

   a) I would like to have an achievements system in the game.
   b) I would like to make and upload my own stages and designs.
   c) I would like to download and play challenging stages made by other people after I beat the game.
   d) I would like to record my battle and share it to others.
   e) I would like to see others' designs sorted by upload time, popularity, and difficulty.

Non-functional Requirements

1. Client requirements
   a) The application will run on Windows Operating System.
   b) The interface needs to be simple and user friendly to prevent confusion for the users.

2. Design requirements
   a) The application must be able to integrate with a backend server to save students' and teachers' data and transfer vital information for the game.
   b) The application must provide the freedom for users to manipulate movements, skills and equipments to a certain degree to prevent crashes and imbalance.
   c) The stages made by students must be validated to ensure that stages are beatable to prevent infinite stages and unexpected game crashes.

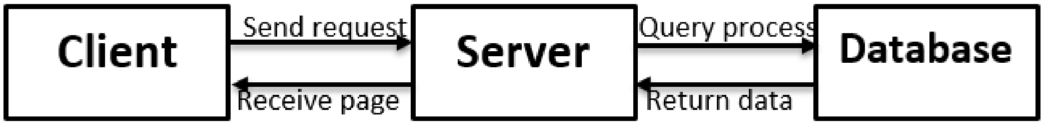
3. Server requirements
   a) The application must be able to be scaled to support multiple programming languages.
   b) If time allows, integration with Purdue accounts for easy authentication.

4. Security requirements
   a) The source code should be encrypted to prevent users or players from accessing and altering.
   b) As user data is sensitive, accounts must be secure so that tampering is very difficult.

Design Outline

High-Level Overview

Our project is a fun and interesting game that allows users to learn the basics of programming. Our game will also include tools to help teachers in the classroom. Our implementation will use a Client-Server model. We will be implementing a Java backend server to handle requests from the Client. Also, we will be implementing a MySQL as the framework for our database.



1. Windows Client

a) The Windows client will be the interface to our system.

b) The Windows client will be using the libGDX framework for the implementation of the game.

c) The Windows client will retrieve and send data such as the user information, teacher information, game data and etc. to our server via HTTP requests.

2. Server

a) The server will receive and handle all requests sent by the clients.

b) The server will validate requests before processing.

c) The server will query the database and generates appropriate response to send to client.

d) The server will transfer user information and game data from the client to the database to be stored.

e) The server will be hosted on Amazon Web Services.

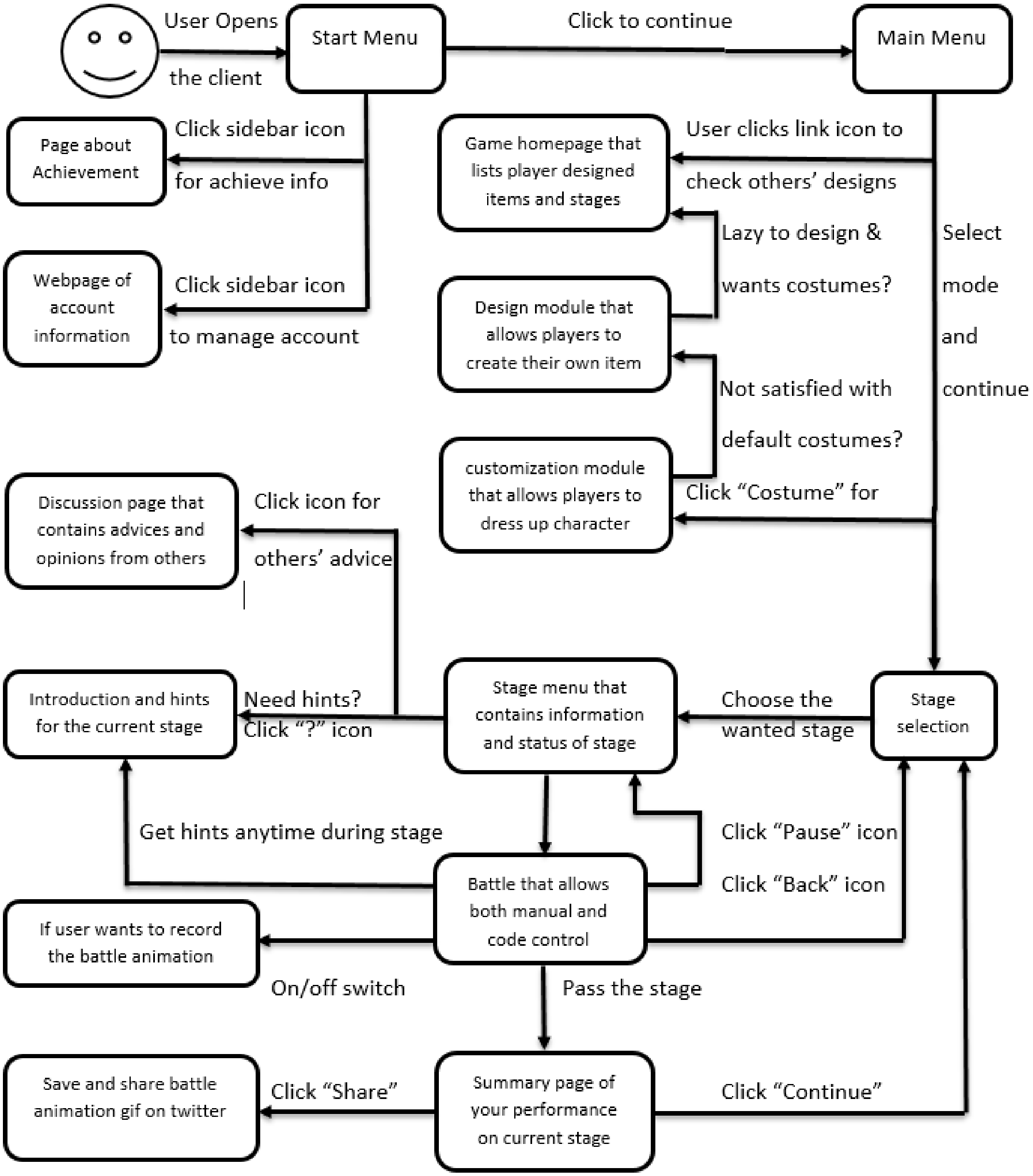
3. Database

a) The database will store user information, teacher information, game data etc. used in Dungeon Coder.

b) The database will be hosted on Amazon Web Services.

Activity/ State Diagram

The diagram below shows the activity diagram which shows overall sequence of events that can happen when a user interacts with the UI of Dungeon Coder.


User Opens
the client
Start Menu
Click to continue
Main Menu
Page about Achievement
Click sidebar icon
for achieve info
Webpage of account information
Click sidebar icon
to manage account
Game homepage that lists player designed items and stages
User clicks link icon to
check others' designs
Lazy to design &
wants costumes?
Design module that allows players to create their own item
Not satisfied with
default costumes?
customization module that allows players to dress up character
Click "Costume" for
Select
mode
and
continue
Discussion page that contains advices and opinions from others
Click icon for
others' advice
Introduction and hints for the current stage
Need hints?
Click "?" icon
Stage menu that contains information and status of stage
Choose the
wanted stage
Stage selection
Get hints anytime during stage
Click "Pause" icon
Click "Back" icon
Battle that allows both manual and code control
If user wants to record the battle animation
On/off switch
Pass the stage
Save and share battle animation gif on twitter
Click "Share"
Summary page of your performance on current stage
Click "Continue"

Design Issues

Function Issues:

Issue: Do users need to create an account and login to use our service?

- Option 1: Allow the creation of login ID using Facebook, Google, etc.
- **Option 2: Create a username and password that is unique to our service.**
- Option 3: No login ID creation.

We chose to require user to create a username and password that is unique to our service. This is because we would need to have accounts for student-teacher interactions. They would need to be a student to be able to join a class made by the teacher. Each student will also be able to track their progress and performance through their accounts. On the other hand, teachers would also be able to monitor each student's performance. We would also need an account system for the designer page as published under the designers account.

Issue: How should users receive notifications about announcements?

- Option 1: Email notifications.
- Option 2: Client notifications.
- **Option 3: User chooses one or more of the above.**

We've decided to give the user the choice on how they want to receive notifications. This gives the students more flexibility when it comes to important information. Everybody will always have their preferred notification method and limiting the options available could cause issues for some students. When creating their accounts, students will have the option to opt into any of these notification methods. The client notifications would simply be pushed straight to the client with no extra information needed. If students would like to opt into the other notification methods they would be asked to provide the corresponding information and it would be stored with their account.

Issue: How should teachers be able to securely access their class's data?

- Option 1: Create a teacher only client.
- Option 2: Create a teacher only login option on the existing client.
- **Option 3: Create a teacher login option on the website/discussion page.**

We decided that it would be best if the teacher was able to access their class's data via the website. The teacher would be able to log in using their teacher id and password and be taken to a page where they can see statistics, send messages, assign points, etc. This method means the teacher doesn't have to download a whole client just to see and control their class. The teacher can easily get all of the information they need anywhere, and on any machine, they want with only an internet connection.

Issue: Do users need an internet connection to play our game?

- Option 1: Users must be connected to the internet at all times in order to play
- Option 2: Users must be connected to the internet when uploading/downloading maps and class data
- **Option 3: Users must be connected to the internet in specific game modes**

We decided to require an internet connection when the user is using specific modes, specifically when playing as a student or when creating/downloading user made levels. These modes require an internet connection so that you can upload and download maps as well as information about your progress as a student at any time. However, we don't want to limit the game by always requiring an internet connection. Therefore, there we plan to make a separate story mode that can be played without an internet connection.

Non-Functional Issues:

Issue: What type of architecture should we use?

- **Option 1: Client-Server Architecture**
- Option 2: Unified Architecture

We decided to go with the Client-Server architecture. Having a Client-Server architecture would allow us to divide the work easily within our team. We would be able to make changes to both frontend and backend without worrying about compatibility issues. Going for unified architecture would limit our efficiency as the frontend could only start work when the backend works.

Issue: What type of game should we make?

- **Option 1: 2-dimensional game**
- Option 2: 3-dimensional game

We decided to go for a 2-dimensional game. As our team is new to game development, we would like the development and implementation of the game to be as simple as possible. Going for a 3-dimensional game would be a time-consuming process as we would need to invest time into learning complex 3-dimensional game development techniques. Also, we also considered our demographic and realized that going for a 2-dimensional game would be more beginner friendly.

Issue: What front end framework should we use?

- Option 1: Unity Engine
- **Option 2: libGDX**
- Option 3: LÖVE
- Option 4: Unreal Engine

We considered multiple game engines before deciding on one. The primary concern was the whether we wanted to do a 2-dimensional or 3-dimensional game. We considered Unity Engine and Unreal Engine for a 3-dimensional game but we felt that it may be time consuming. Comparing libGDX and LÖVE, libGDX uses Java which is familiar for the entire team while LÖVE uses Lua which is new for us. We decided to choose libGDX for several reasons. First of all, libGDX is widely used as a 2D game engine. Also, libGDX will work on various platforms such as Windows, OSX, Linux, iOS, Android and etc. This would allow us to port to another platform in the future. We considered LÖVE at first because it is amazing for pixel graphics which is the intended game aesthetic but we had trouble finding appropriate APIs for transferring data back and forth the client and server. Hence, we opted for libGDX.

Issue: What backend language should we use?

- Option 1: Python
- Option 2: Ruby
- **Option 3: Java**

Looking at the popular backend languages, Python and Ruby are commonly recommended. However, our team does not have much experience with Python and Ruby. Therefore, we decided to go for Java as the entire team is very familiar with Java. This would also be easier for us to implement as the frontend framework is also in Java.

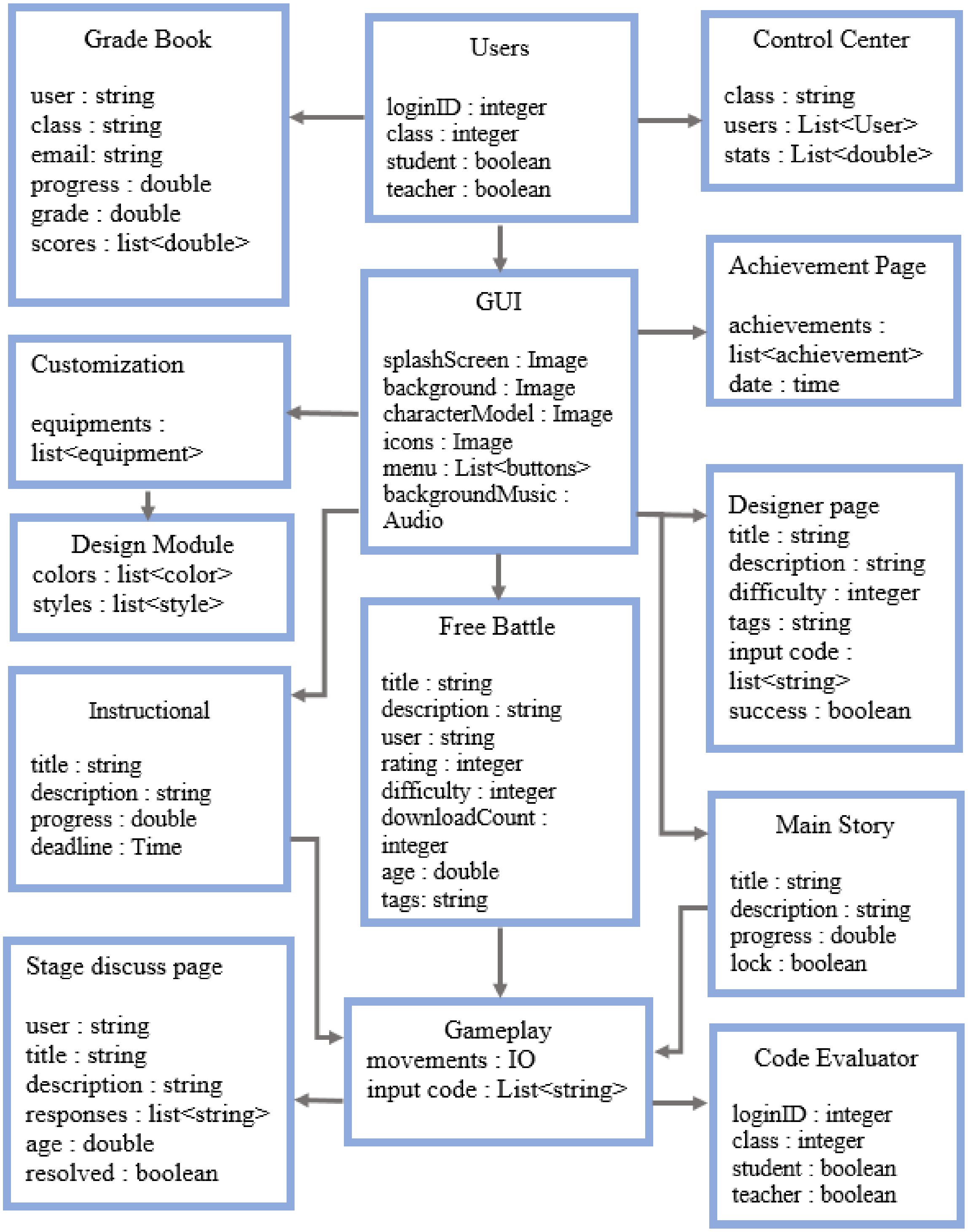
Issue: What database should we use?

- Option 1: **MySQL**
- Option 2: NoSQL

Database language is a tricky consideration as none of us in the team have any experience with database languages. After much research, we considered PHP, MySQL, and NoSQL. We chose to go for MySQL as some of the team members will be learning about SQL and Relational Database Management Systems (RDBMS) in their other classes. Also, MySQL has been a free open-source database management system that has been around for many years. Hence, it is very stable and there is a big community to help maintain the system. This would allow us to easily access documentation on the language as well as ensures future reliability.

Design Details

Data Class Level-Design



Data Class Description

Our data class design is based on the understanding of the types of objects in our domain. Therefore, they represent the "Model" classes that our Java client will use.

Graphical User Interface (GUI)

The GUI consists of several components:

- **SplashScreen** - an animation where the logo will be shown for a few seconds before entering the main menu.
- **Background** - images for the various backgrounds in game.
- **Model -** representation of all of the game assets such as characters, enemies and environmental objects.
- **Icons -** images for various icons and buttons for accessible user interaction.
- **Menu** - a list of buttons that will be present in different pages in the game.
- **BackgroundMusic -** various audio files to accompany the graphics and gameplay of the game.

Users

The user class will cater for various types of users.

- **LoginID** - a unique integer value for each users.
- **Class** - a unique integer value for each class that user is enrolled in.
- **Student** - a boolean value to indicate student permissions.
- **Teacher** - a boolean value to indicate teacher permissions.

Instructional

The instructional class consists of the attributes that should be available for users when they access the instructional mode of Dungeon Coder.

- **Title** - given name for the stage.
- **Description** - contains a short summary of the stage.
- **Progress** - a bar to indicate the progress of the user on the stages.
- **Deadline** - a date to indicate the time for students to complete the stage; if students fail to complete the stage before the deadline, the main story will be locked until the stage is completed.

Main Story

The main story class consists of the attributes that should be available for users when they access the main story mode of Dungeon Coder.

- **Title** - given name for the stage
- **Description** - contains a short summary of the stage.

- **Progress** - a bar to indicate the progress of the user on the stages.
- **Lock** - a boolean value to indicate whether the main story is locked or unlocked.

Free Battle

The free battle class consists of the attributes that should be available for users when they access the free battle mode of Dungeon Coder.

- **Title** - given name for the stage
- **Description** - contains a short summary of the stage.
- **User** - the author of the map
- **Rating** - average rating of the map given by other users.
- **Difficulty** - an integer value to indicate the difficulty of the map
- **DownloadCount** - an integer value to indicate the number of times the map has been downloaded by other users.
- **Age** - a double value to indicate the age of the map since it was first published.
- **Tags** - a string value to indicate tags for the map.

Code Evaluator

The code evaluator class will be used for evaluating if the input code from users are compilable.

- **Success** - a boolean value to indicate the whether the input code is compilable.
- **Input** - the input code that is submitted by the user.

Gradebook

The gradebook class consists of the attributes that should be available for users when they access the gradebook in their account.

- **User** - the name of the user.
- **Class** - the class that the user is enrolled in.
- **Email** - the email address of the user.
- **Progress** - a bar to indicate the progress of the user on all modes.
- **Grade** - a double value to indicate the grade of the user.
- **Scores** - a list of all the scores obtained from completing stages in the instructional mode.

Control center

The control center class consists of the attributes that should be available for teachers when they access the control center.

- **Class** - the name of the class that the teacher is managing.
- **Users** - the list of all the students that are in the class.
- **Stats** - the list of all the statistics for the students.

Gameplay
The gameplay class consists of the available interactions that can be made between the user and the game.

- **Movements** - it will be controlled through the IO device.
- **InputCode** - the entire code that is submitted by users.

Customization module
The customization module is the area where users can customize their characters with equipments.

- **Equipments** - a list of equipment that is worn by the user. User will have a option to change their equipment.

Design module
The design module is the area where users can customize their equipments or even create new ones through the inbuilt editor.

- **Colors** - a list of all the colors that can be applied when creating an equipment.
- **Styles** - a list of all the styles that can be applied when creating an equipment.

Equipment
This is the list of the type of equipments that can be worn by the user.

- Hat
- Weapon

Achievement
The achievement class consists of the attributes that should be available for users when they access the achievements page.

- **Achievements** - list of all the achievements that can be achieved by the user.
- **Date** - representing the date when a certain achievement is completed.

Designer page
The designer page is where users can design maps that can be published for other users to try. The designer class consists of the attributes that should be available for users when they access the designer page.

- **Title** - the given name of the map
- **Description** - contains a short summary of the map.
- **Difficulty** - an integer value indicating the difficulty of the map.
- **Tags** - a string value to indicate tags for the map.
- **InputCode** - the entire code that is submitted by users.

- **Success** - a boolean value to indicate that the map has been successfully beaten by the designer. This is crucial before allowing designers to publish their map to prevent unbeaten maps.

Stage discuss page

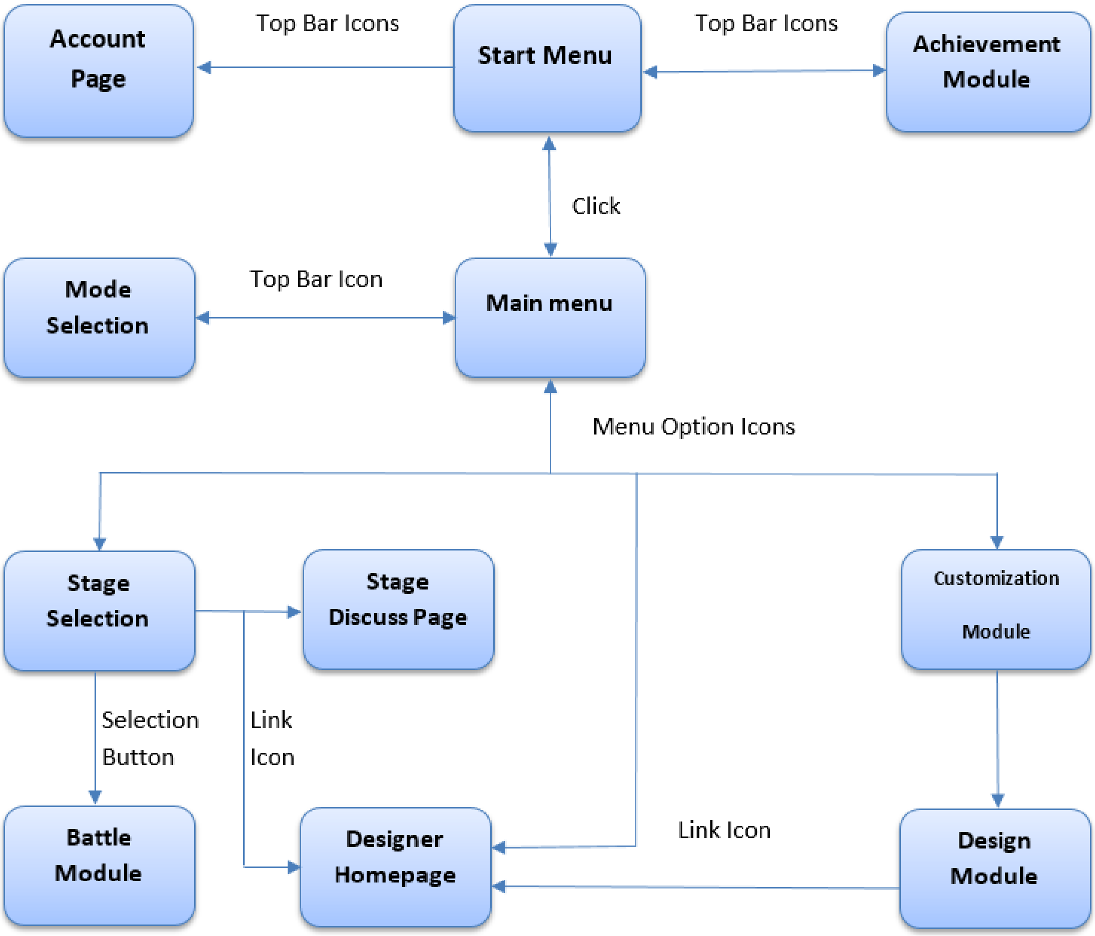
The stage discuss page is the place where users can ask for help from other users. The stage discuss page class consists of the attributes that should be available for users when they access the stage discuss page.

- **User** - the author of the post.
- **Title** - the title of the post.
- **Description** - the contents of the post written by the user.
- **Responses** - the list of all the responses from other users.
- **Age** - an integer value indicating the age of the post.
- **Resolved** - a boolean value indicating whether the post has been resolved.

Navigation Flow Map

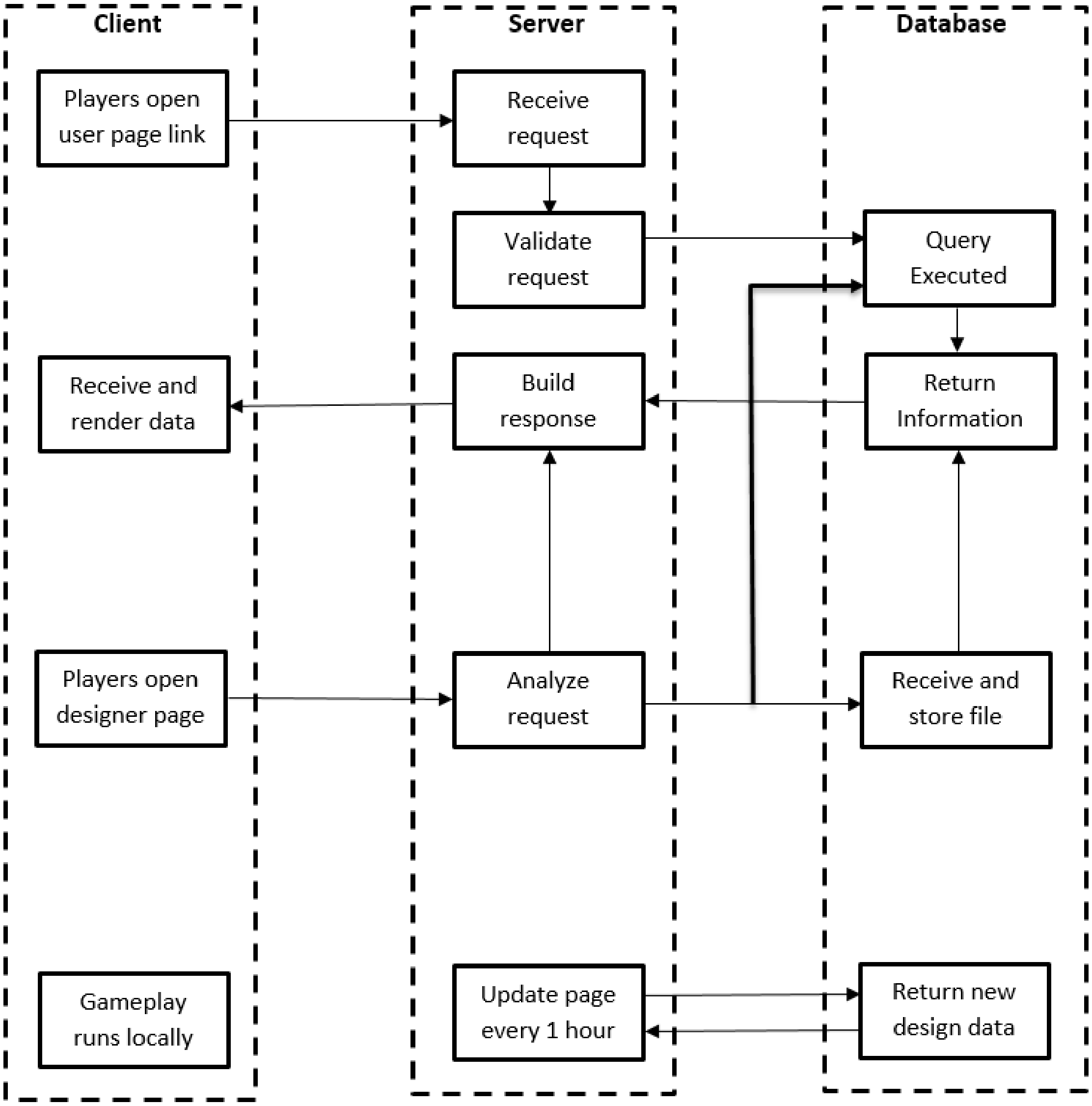
Our project is a client-server based programming game aiming at beginner's coders, we want to make our navigation simple, clear and easy to understand and access. The client starts with the start menu. At the start menu, player can navigate to the main menu by clicking, navigate to the Account page and manage his account though the link icon, or navigate to the achievement module to check their achievement status and get rewards. The main menu contains a selection bar that allow players to choose the game mode. The game will have 3 modes which leads to four main modules of the game.

The Stage Selection menu is where players can select the stage play. Different types of stages (instruction, main story and free battle) are displayed. The Customization Module allows players to customize their character, and they can go to the Design Module to create their own helmets, weapons, or stages. And finally, the Designer Homepage can be accessed through an option brings users to a website. Players can upload their own designs, or explore and download other designer's works at the Designer Homepage.



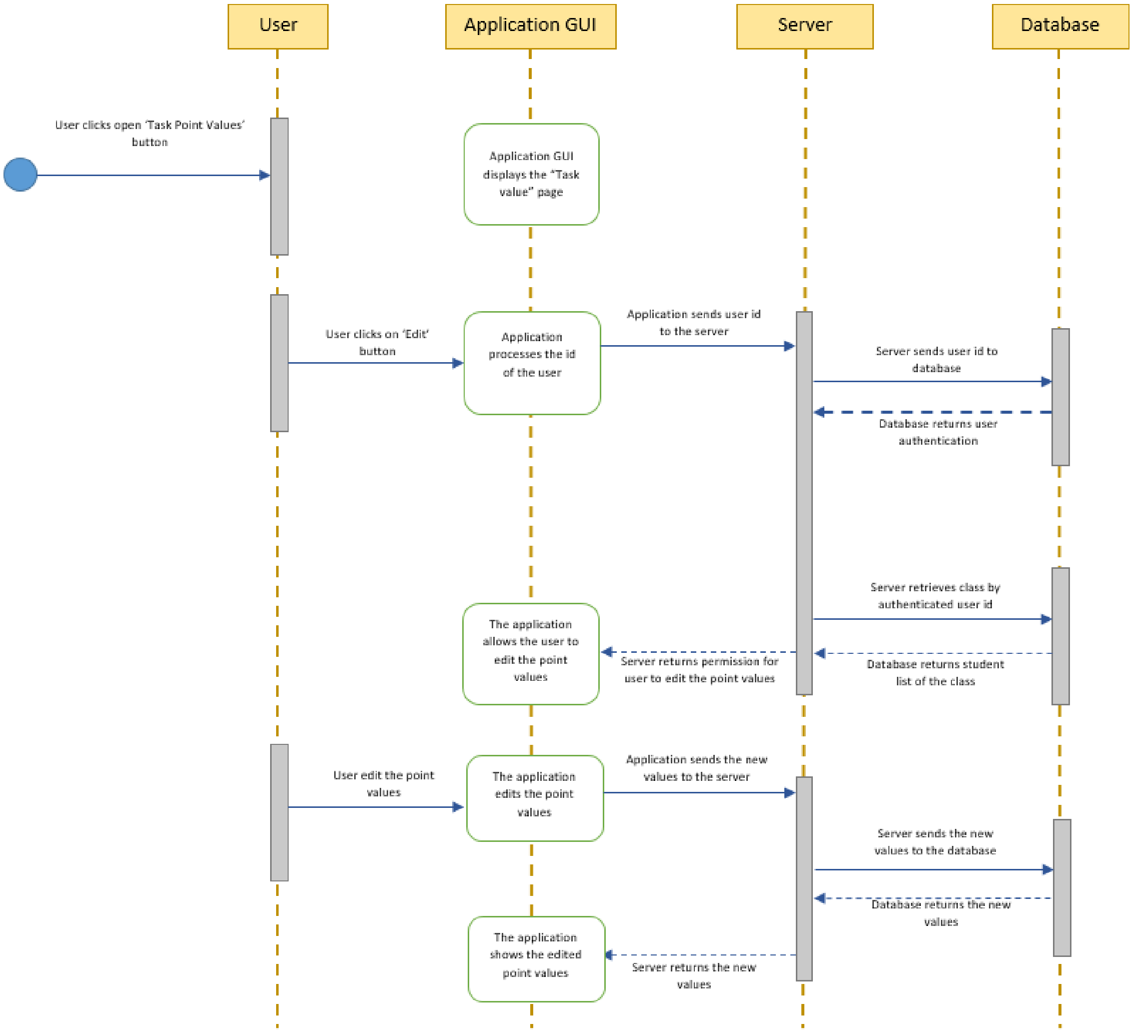
Flow of Events

The diagram below shows the flow of events of our project. In general, gameplay runs in local environment, and the flow starts when the user wants to the manage their account or check the designer homepage. They connect to the server and receive response data due to their request. For better efficiency, the server should hold a file that represents the current designer page. It will exchange data with the database and build a new page every 1 hour.

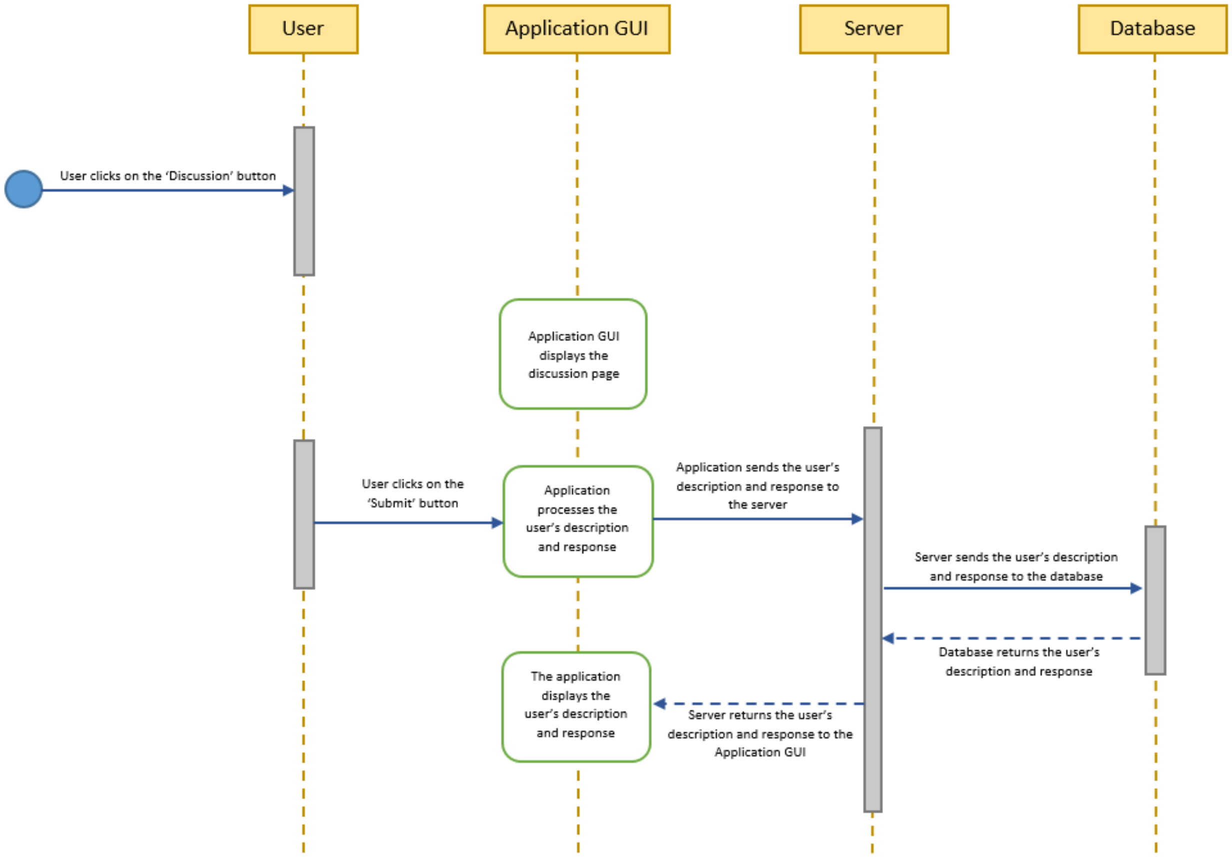


Sequence Diagrams

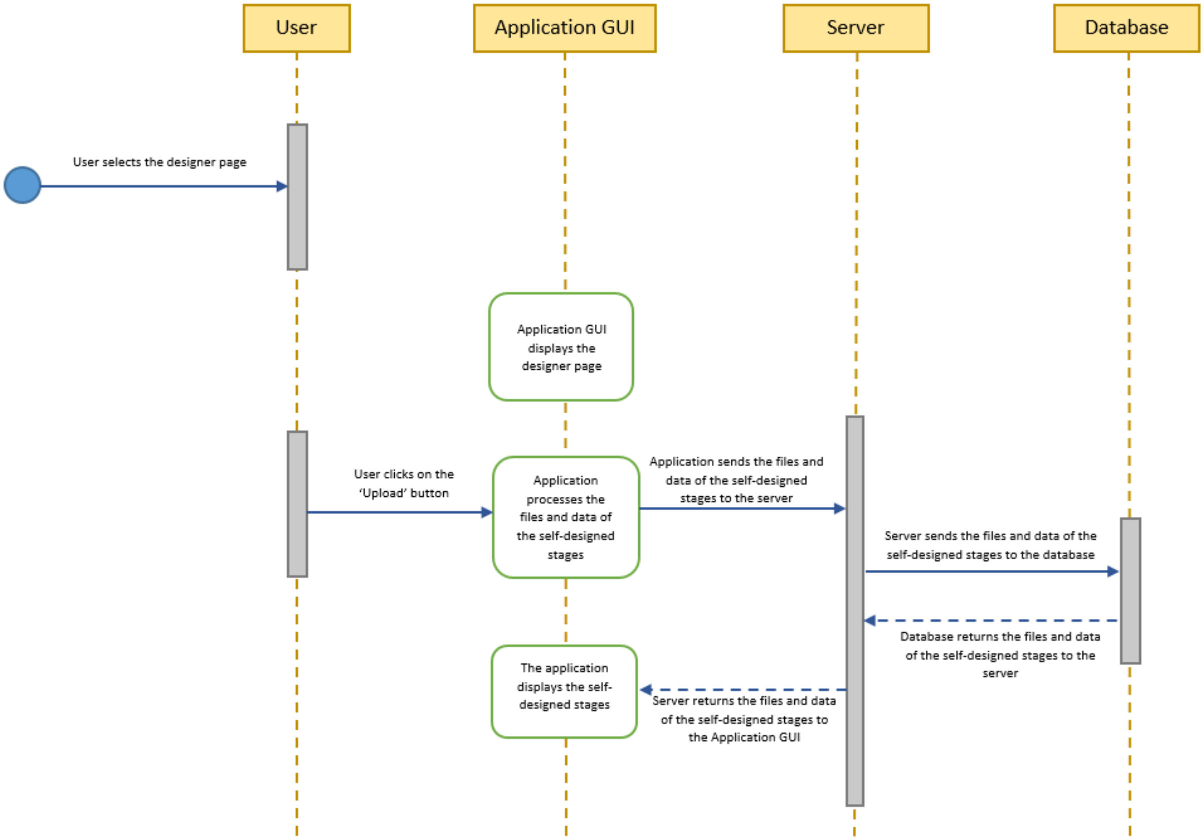
Sequence of events when teacher edits the point values of different tasks.


User
Application GUI
Server
Database
User clicks open 'Task Point Values' button
Application GUI displays the "Task value" page
User clicks on 'Edit' button
Application processes the id of the user
Application sends user id to the server
Server sends user id to database
Database returns user authentication
Server retrieves class by authenticated user id
Database returns student list of the class
Server returns permission for user to edit the point values
The application allows the user to edit the point values
User edit the point values
The application edits the point values
Application sends the new values to the server
Server sends the new values to the database
Database returns the new values
Server returns the new values
The application shows the edited point values

Sequence of events when user discuss in the discussion page.

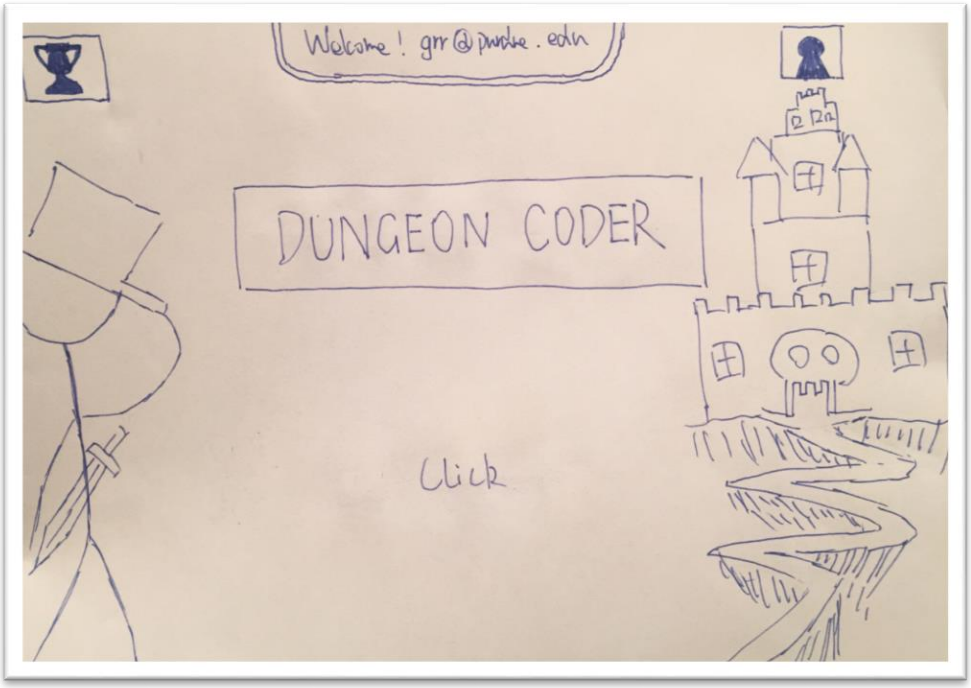

User
Application GUI
Server
Database
User clicks on the 'Discussion' button
Application GUI displays the discussion page
User clicks on the 'Submit' button
Application processes the user's description and response
Application sends the user's description and response to the server
Server sends the user's description and response to the database
Database returns the user's description and response
Server returns the user's description and response to the Application GUI
The application displays the user's description and response

Sequence of events when user upload his or her self-designed stages.

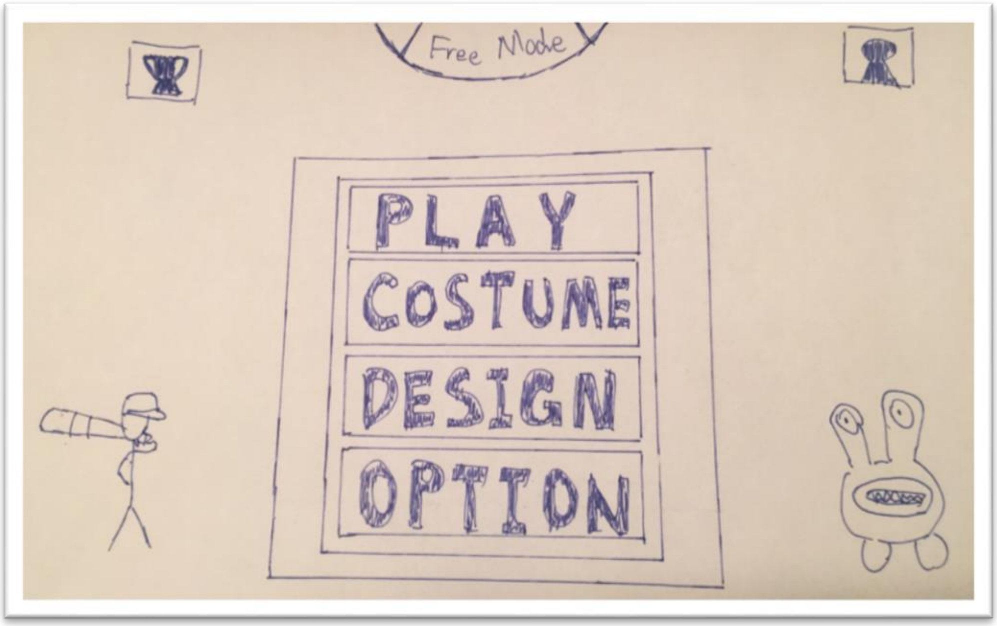

User
Application GUI
Server
Database
User selects the designer page
Application GUI displays the designer page
User clicks on the 'Upload' button
Application processes the files and data of the self-designed stages
Application sends the files and data of the self-designed stages to the server
Server sends the files and data of the self-designed stages to the database
Database returns the files and data of the self-designed stages to the server
The application displays the self-designed stages
Server returns the files and data of the self-designed stages to the Application GUI

UI Mockups

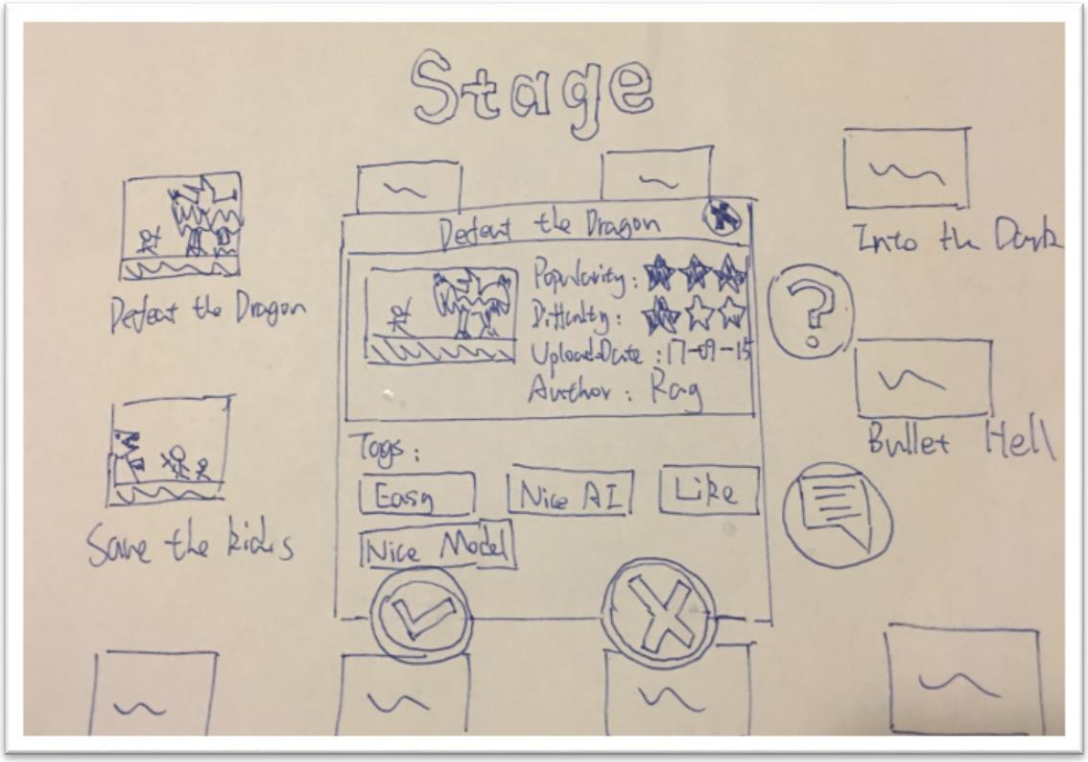
Splash screen



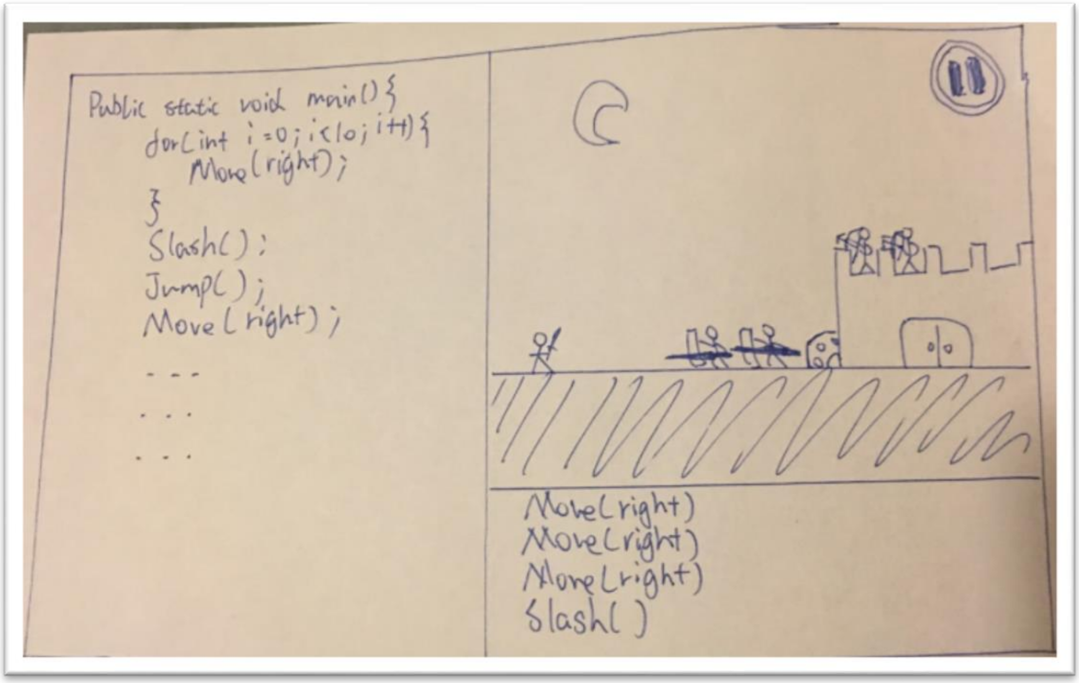
Main menu



Stage menu



Gameplay



```
Public static void main(){
    for(int i=0;i<10;i++){
        Move(right);
    }
    Slash();
    Jump();
    Move(right);

    - - -
    . . .
    . . .
```

```
Move(right)
Move(right)
Move(right)
Slash()
```