STORYCREATOR

Complete Documentation

Team 16- T.I.P.

Lingxiao Gao | Jie He |
Zijing Liu | Zheman Shi |
Thanakorn Suppakarnpanich | Yubo Wang

T.I.P TeamInPower

Team Member:

Zijing Liu zijingli@usc.edu

Lingxiao Gao lingxiag@usc.edu

Zheman Shi zhemansh@usc.edu

Thanakorn Suppakarnpanich suppakar@usc.edu

Jie He hejie@usc.edu

Yubo Wang yubowang@usc.edu

High-Level Requirements

A text game engine (software/android app) that helps users to easily create their own text games.

User does not need any coding experience to use our text game engine. We provide graphical UI, which should be user-friendly and easy-use, to help user fully utilize provided functionality.

User could package up his own text game after he made it. He could choose share the game with his friend, with the public, or just upload to the remote server. If he shares his game with his friends, his friends could download his game from the server.

Users could backend download games. They could do some other stuff when they download games. (multi-threads; there will be more multi-threads when optimizing codes).

Technical Specifications

Client

Game GUI

The game engine will prove a default game GUI for user to visualize their text game. However, users could customize their game GUI by manually set up some GUI variables, which includes background image, font, textures. A user could also indicate for each scene how he or she wants to display all texts and game contents at a given scene state.

Login Window GUI

After user opens our software, the Login Window GUI will display. The Login Window GUI has dimension 960 * 640. There are three sections in the Login Window GUI – the title, the logo animation and the buttons group. The size of these sections should be proportional to the provided screenshot below. There is status label at the left bottom corner to indicate the connection status with the server.

All the sections will be loaded as soon as the Login Window GUI shows up, but the username and password fields will be blank until the user enter their username and password.

Section functionality

When every section is sated up, the user is able to login with correct username and password combination.

a). Title and logo functionality

It is a JLabel that contains an image that includes both the title and our logo.

b). User and password functionality

These two are made with JTextField and user must enter correct username and password to login. If user enter wrong combination, a massage will show up.

c). Login button functionality

This one is made with JButton. After users enter username and password, they can press this button. If combination is true, they will go to next window. If not, a window will pop up that shows some massages.

d). Sign up button functionality

This one is made with JButton. After users press this button, a window will pop up to let users create username and password. If username is already in database, it will ask again until users enter a unique username.

e). Login as Guest button functionality

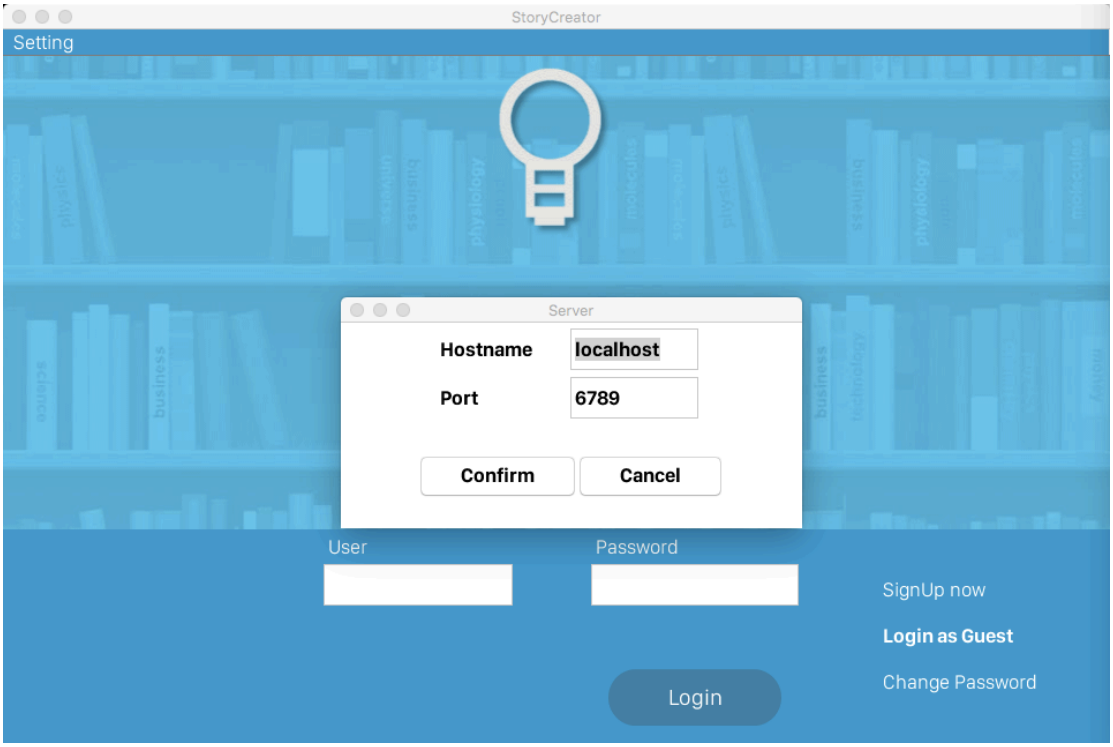
This one is made with JButton. It let user login without entering any information. After press, users will go directly to the next window.

f). Forgot username button functionality

This one is made with JButton. It will ask users to enter the email address that they used for registering. Then they will receive emails that contains their username.

g). Forget password button functionality

This one is made with JButton. It will ask users to enter the email address that they used for registering. Then they will receive emails that contains their password.

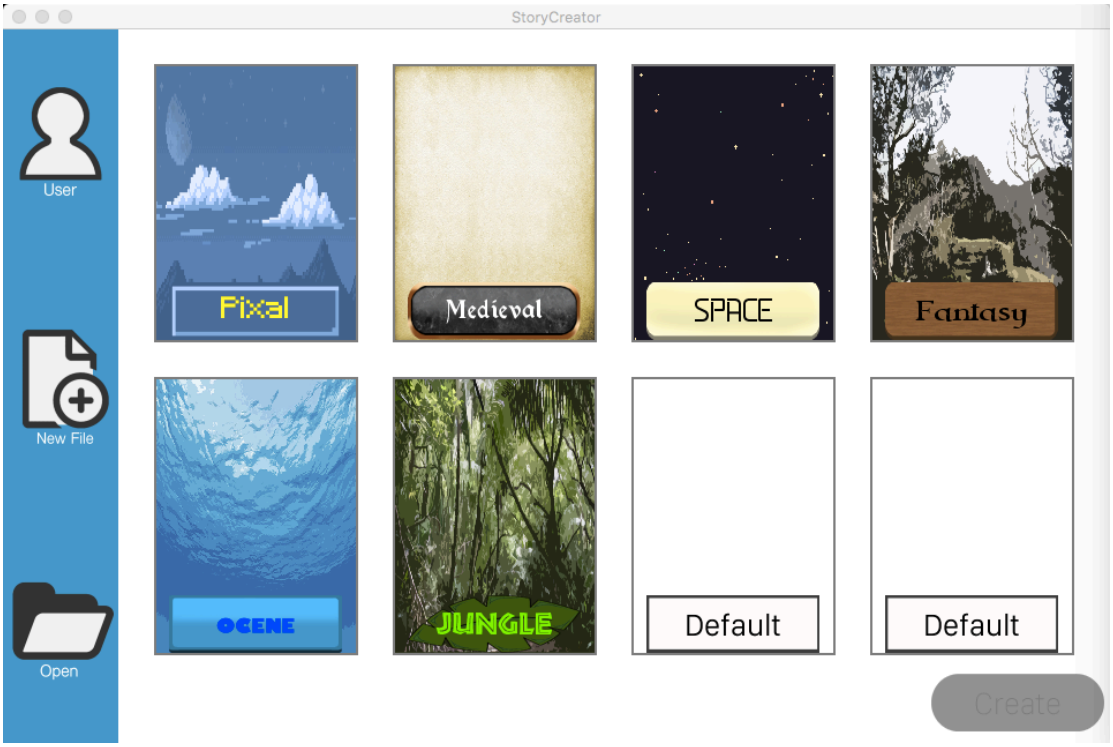


h). JMenubar functionality

There is a menu at the top left corner which has a menu item “connect”. When pressed, it will pop a window in which users can set hostname and password. Here requires an admin password in order to make valid change of hostname and password. (This part has been removed because we have put our server and database to the Amazon Server and everything is set up automatically.)

Create File Window GUI

After the user logs in, the GUI with options to create a file will display. There are three sections in this GUI – the three buttons on the left, the main section on the middle of the screen that by default displays the sample templates provided, and the create button in the bottom right corner.



Section functionality

a). User Button functionality

The first button on the left, “User”, is a display of the user’s avatar, which allows the user to click on it and choose to log out and return to the previous Login Window GUI, and check the number of existing game projects.

b). New Button functionality

The “New” button is selected by default, and shows all the templates the user can choose from to start creating his or her own game.

c). Open Button functionality

The “Open” button allows the user to open a local file and load a game he or she was previously working on.

d). Template Panel functionality

The template section is made from JPanel. Inside the JPanel, all predefined template icons are made from JLabel (may be changed in the future). The predefined templates will be shown with a simple description of a design style. The user can also design original templates that will be shown as well. The predefined design styles we provide at present are pixel, sci-fi, medieval and fairytale.

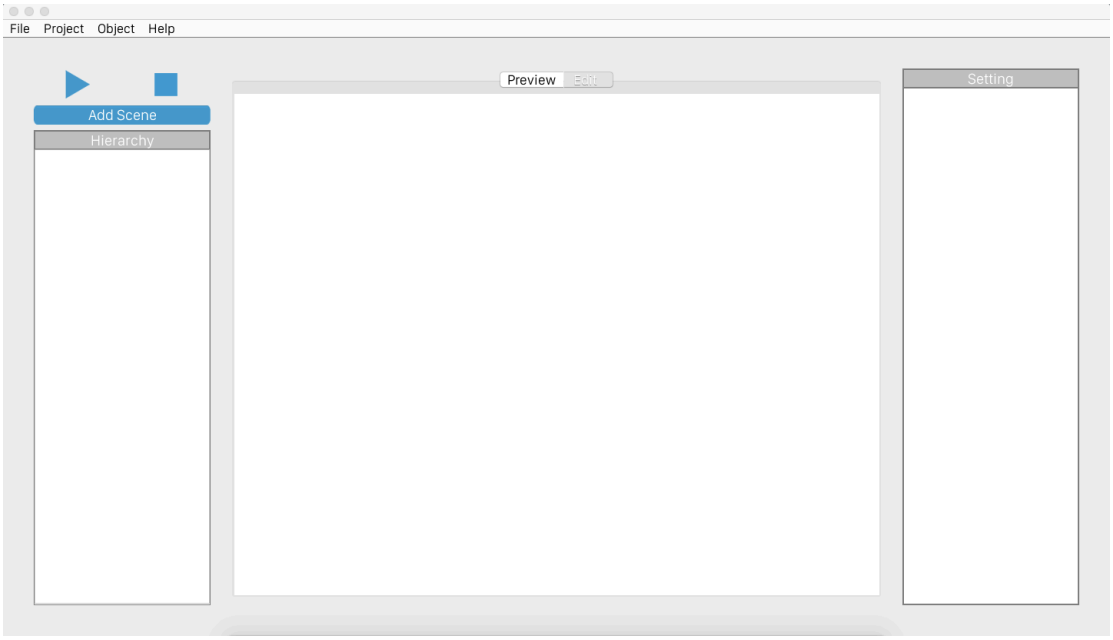
e). Create Button functionality

The “Create” button lets the user open his or her file, and the software will automatically jump to the Main Window GUI.

Main Window GUI

After user created file or opened the existed file, the Main Window GUI should be displayed. The Main Window GUI should open to the full size of the screen. There are main five sections in the Main Window GUI – the menu, the run/stop button, the hierarchy Tabbed Pane, the WorkArea Tabbed Pane and Outline panel. The size of these sections should be proportional to the provided screenshot below.

All the sections will be loaded as soon as the Main Window GUI shows up, but the WorkArea will be blank until the file is received from Create File Window.



Section functionality

When every section is sated up, user is able to create or edit his or her own Story by using the functionality our software provided.

a). Menu functionality

For the menu section, it is made from JMenu, which contain five JMenuBar in it – File, Asset, Object, Window, Help. For the "File" MenuItem, it contains options like create new, open, close, and save. GUI setting. The "Project" MenuItem contains game setting. The "Object" MenuItem provides user with all kinds of elements he or she needs for creating the story. The "Help" MenuItem should contain a tutorial of the software. The tutorial should be a documentation contain text and images.

b). Run/Stop Button functionality

For the run/stop button, it is designed to let user test or run his or her story during or finishing manufacturing.

c). Hierarchy Tabbed Pane functionality

For the hierarchy tabbed pane, it is made from JTabbedPane which contains Hierarchy. The Hierarchy will list all the scenes user made and all the components in the scene. User could add a new scene by clicking “Add Scene” button. When the user right clicks an element in the hierarchy, he or she could choose to delete the scene or add new state.

d). Work Area Tabbed Pane functionality

The work area is the main section for user operation. It contains two parts – Preview part and Edit part. For the Edit part, it is the area for user to do any modifications on the scenes such as changing/adding/deleting objects and properties within the current selected scene. When user finishing editing, he or she can view the result of his or her modification in the preview section. (Note: the preview section will automatically update the operations user does in the Edit section.) For the preview part, user could see the preview of the state and change the locations of game contents by dragging the elements.

e) Setting Panel functionality

The setting panel is made to change the game contents locations and sizes in the state preview area.

GUI Save and Exit

After creating the story, user could save their story by clicking the “File” → “Save”. User could choose to save to the local folder, the server. After saving the story, user can choose either close the software or continue working.

Game logics

The text game made through the game engine is based on a series of scenes and game contents, and eventually displayed by game GUI generated by the game engine with customized user settings. All game

data is stored in a game project objet and the entire game play process is controlled by a game play manager.

a) Scene

This is the main component of a text game. It represents what will be shown on the screen at one time. Scenes contain description texts, game contents, and next-step-choices. Description texts are what users want their players to see at the current scene time. Game contents are some function derived components in the game (will discussed later). Next-step-choices are actions that player could choose between one of them when they exit the current scene. A scene also has its own state property, which is used to determine how it will be shown on screen at one time. Specifically, a scene can show one part of its description texts at one state, and show another part of description text at another state. So do the game contents and next-step-choices.

b) Game Content

This is an information derived game component. It could be customized by user. It has a name, a description (this could be image as well). When a user customizes a game content, he or she should provide a name and descriptions. There are also some pre-define game content that user could use.

1. *Player Game Content*: This represents the player object in the game. It has a name, an inventory to contains items.
2. *Item:* Every customized game content could be an item.

c) Game Project Object

This stores all the data of a game, which includes all scenes, game contents, GUI settings.

Server

Server has its own database to store all user information. It should be able to connect with multiply clients at the same time. It should be working fine without any human involvement.

All server GUI is removed for the latest version because we put our server and database to Amazon Server and ever thing is set up automatically.

a) Port GUI

When run the server client, port window will be the first GUI. It requires a port number to listen. If an invalid port number is entered, the text area will be cleaned and port number will be required again.

b) Server GUI

After a port is successfully set, there will be a server GUI. It is just a window with a logo and a terminate button, which used to terminate the server service. All communications between server and clients are automatically running without the need of human involvement.

Server Database:

Server has a main database containing users and their data. We do this because we want the users to only be able to load their data when they are connected to the server and data will be stored more securely. For server database, there are two main data object: MySQLDriver object and User object. All data will be stored in MySQL database.

a) User Object

User object will contain all user data such as username, password (encrypted), and Arraylist of current user's game projects.

**Game Project Object*

This is the object that stores all game data of a game project. It contains game project name, game project GUI settings, scenes, game centents.

b) *MySQLDriver*

MySQLDriver is a class for managing database. It connects the Java program to MySQL database. There are two most important functions in this class. The function saveToDatabase() takes in the User Object and parse all data of user into table in MySQL. Secondly, LogIn() reconstructs a user object by retrieving data from MySQL tables and returns to the logged in user. This class supports get and add data function to the table.

How Server database works with Client

Upon launching the software, you can either log in, sign up, or change password.

a) Option 1: Log in

When the user types in username and password, MySQLDriver would look into user table in database and checks if the user exists. If it does not exist, then it will return null. If the username and password exists, it will reconstruct a user object that integrating all user data from the table into a user object, which is returned to the current user.

b) Option 2: Sign up

This will create a new user object. It needs to check if the username already exist in database. If not, create a new user object with password, store it in database, and return the newly created user object. So if return is null, we know that username and password already exist so we can't create new user with that username.

c) Option 3: Change password

If user wants to change the password, he or she could type in the username and old password. Then he or she could type in the new one to change the password. If input is correct, it will change the password and store new password in database.

After a user enters the software, he or she has the option to create new project or load up existing project.

a) Option 1: Creating new project

This will create a new game project object. If the user chooses to save it in the remote server, this game project will be added to the ArrayList<GameProject> of the user and will be stored in the server database in MySQL tables.

b) Option 2: Open existing project

The user object that is returned when LogIn() is called already reconstructs a user object that has all the existing game project. Therefore, when the user opens an existing project, he can just find from the returned user object.

When user finishes creating their game, they could choose store locally or remotely. If they choose save remotely, the current game project object will be sent to the server. The server will add that game project object to the user object and update the MySQL database.

Server & Client Communication Protocol

There should be server communication protocols for server and clients to follow in order to make efficient and safe communication. It should specify how server and client should start their communication, send their request, and receives data.

Detailed Design

Hardware Requirements

Windows:

Windows 8 (Desktop)

Windows 7

Windows Vista SP2

RAM: Recommended 512MB

Disk Space: Recommended 1GB

Processor: Minimum Pentium 2 266MHz processor

Mac OS X:

Intel-based Mac running Mac OS X 10.8.3+, 10.9+

Linux:

Oracle Linux 5.5+

Oracle Linux 6.x (32-bit), 6.x (64-bit)

Oracle Linux 7.x (64-bit)

Red Hat Enterprise Linux 5.5+ (32-bit), 6.x (64-bit)

Ubuntu Linux 12.04 LTS, 13.x

Software Requirements

Java 8

Eclipse IDE for Java EE Developers (Luna/Mars)

Server

Constants
+int: lowPort +int: highPort +int: defaultPort +String: defaultHostname +String: portDescriptionString +String: portLabelString +String: submitPortString +String: portGUITitleString +int: portGUIwidth +int: portGUIheight +String: portErrorString +String: portAlreadyInUseString +String: initialtoryStoryCreatorTextAreaString +String: StoryCreatorGUITitleString +int: StoryCreatorGUIwidth +int: StoryCreatorGUIheight +String: startClientConnectedString +String: endClientConnectedString +String: clientDisconnected +String: selectStoryCreatorButtonString +String: defaultResourcesDirectory +String: unrecognizedLine +String: StoryCreatorFileDelimeter +String: StoryCreatorLoadedMessage

InfoServerGUI: JFrame
+long: serialVersionUID -JTextArea: textArea

-JScrollPane: textAreaScrollPane -ServerListener:
-InitializeVariable(): void -createGUI(): void -addActionAdapter(): void +setServerListener(ServerListener): void +CloseOperation(): void +addMessage(String): void

Server
- UserDatabase mUserDatabase
+ Server(int) + main(String [] args)

MySQLDriver
-Connection: con; -String: selectUserName -String: addUserName +String: updateUser -String: updatePlayer -String: selectPlayer -String: selectPlayerFromGPID -String: addPlayer -String: updateGameContent -String: selectGameContent -String: selectGameContentFromSceneBelongTo -String: selectGameContentFromPlayerBelongTo -String: addGameContent -String: updateGameProject -String selectGameProject

-String selectGameProjectFromName -String addGameProject -String updateScene -String selectScene -String selectSceneFromGPID -String addScene -String updateSceneStatePair -String selectSceneStatePair -String selectSceneStatePairFromSceneBelongTo -String addSceneStatePair -String updateSceneState -String selectSceneState -String selectSceneStateFromSceneBelongTo -String addSceneState -String updateGameSetting -String selectGameSetting -String addGameSetting -String userpasswordtemp
+setLogInString(String): void +MySQLDriver() +connect(): void +stop(): void +logIn(String, String): User +createNewUser(String, String): User + doesUserExist(String): boolean + doesUserPasswordExist(String, String): boolean + doesPlayerExist(int): boolean + doesGameContentExist(int): boolean + doesGameProjectExist(int): boolean + doesSceneExist(int): boolean + doesSceneStatePairExist(int): boolean

+ doesSceneStateExist(int): boolean
+ doesGameSettingExist(int): boolean
+ addUserName(String, String): void
+ changePassword(String, String, String): void
+ updateUser(String, String): void
+ saveToDatabase(User): void
+ getUserObject(String, String): User
- getPlayer(GameProject): Player
- getSceneStateAndAddToScene(Scene): void
- getSceneStatePairAndAddToScene(Scene): void
- getGameContentAndAddToScene(Scene): void
- getGameContentAndAddToPlayer(Player): void
- getSceneAndAddToGP(GameProject): void
- getGameSetting(int): void
- getGameProject(String): void
+addPlayer(Player, int): void
+ updatePlayer(Player, int): void
+ addGameContent(GameContent, Integer, Integer): void
+ updateGameContent(GameContent, Integer, Integer): void
+ addGameProject(GameProject, String): void
+ updateGameProject(GameProject, String): void
+ addScene(Scene, int): void
+ updateScene(Scene, int): void
+ addSceneStatePair(SceneStatePair, int): void
+ updateSceneStatePair(SceneStatePair, int): void
+ addSceneState(SceneState, int): void
+ updateSceneState(SceneState, int): void
+ addGameSetting(GameSetting, String, GameProject): void
+ updateGameSetting(GameSetting, String, GameProject): void

ServerClientCommunicator: Thread

-Socket: socket -ObjectOutputStream: oos -ObjectInputStream: ois -ServerListener: mServerListener -ServerManager: mServerManager
+ServerClientCommunicater(Socket, ServerListener, ServerManager) +ServerClientCommunicater(Socket, ServerWithoutGUI, ServerManager) +SendObject(Object): void +run(): void +close(): void

ServerListener: Thread
-ServerSocket: mServerSocket -Vector<ServerClientCommunicator>: sccVector -ServerManager: mServerManager
+ServerListener(ServerSocket) +run(): void +removeServerClientCommunicator(ServerClientCommunicator): void +close(): void

ServerManager
HashMap<ServerClientCommunicator, User>: ClientUserMap ServerListener: mServerListener MySQLDriver: mysql String: s
+ServerManager(ServerListener) +ServerManager(ServerWithoutGUI)

+addUser(ServerClientCommunicator, String, String): void +UserLogin(String, String, ServerClientCommunicator): void +dropUser(ServerClientCommunicator): void +saveGameProject(ServerClientCommunicator, GameProject): void

ServerWithoutGUI: Thread
-ServerSocket: mServerSocket -Vector<ServerClientCommunicator>: sccVector -ServerManager: mServerManager
+ServerWithoutGUI() +run(): void +main(String[]): void

StoryCreatorServer
-ServerSocket: ss -ServerListener: ServerListener
+StoryCreatorServer() -listenForConnections(): void +sendUserArchiveFile(): void

Server
- UserDatabase mUserDatabase
+ Server(int) + main(String [] args)

Server – It listens to a port and start a new ServerThread when there is a new connection. This will create server GUI as well.

ServerThread: Thread
- Socket s - BufferedReader br - PrintWriter fw - Server mServer
+ ServerThread(Socket, Server) + run(): void

ServerThread – It is responsible of communicating with one client: sending and receiving information

ServerGUI: JFrame
+long: serialVersionUID -JTextField: portTextField -JLabel: descriptionLabel -JLabel: portLabel -JLabel: portErrorLabel -JButton: submitPortButton -Lock: portLock -Condition: portCondition -ServerSocket: ss -Image: buttonimage -Image: buttonpressed
+ServerGUI() -initializeVariables():void -createGUI():void -addActionAdapters():void +getServerSocket():ServerSocket

ServerGUI-is able to connect with multiple clients and deal with multiple requests at the same time. When ran the portPanel will show, displaying the portLabel, a textfield, and a "Start Listening" button. After a valid port number is submitted, the window will show our logo

with a "Terminate" button that stops all the communications when clicked.

User[Serializable]
-long: serialVersionUID String: username String: encryptedPassword ArrayList<GameProject>: mGamgeProject
+User(String, String) +setGameProject(ArrayList<GameProject>): void +getName(): String +getPassword(): String +setPassword(String): void +getGameProject(String): GameProject +addGameProject(GameProject): void +removeGameProject(GameProject): void +createNewProject(String): void +getGameProjects(): ArrayList<GameProject>

User Class – responsible for storing information about one user. Password stored must be encrypted password which is done by UserDataBase class. No plain text password is stored. The user is able to create new game project or open existing project. SetPassword method is used when a user changes his password. Int type differentiates between a guest (1) and a regular user (0).

UserDataBase[Serializable]
-long: serialVersionUID -ArrayList<User>: udb -MySQLDriver: mysql

+LogIn(String, String): User +encrypt(String): String +connect(): void +createNewUser(String, String): void +changePassword(String, String, String, String): void

UserDataBase—the main database in the server. Has private member which are arraylist of registered users and a static pre-instantiated user which is guest. Encrypt() utilizes hash function to take in plain text password and encrypts it. retrieveForgettenUserName() is called when a user chooses the option Forgot username and input his email. It will return the username matched with the input email. retrieveTemporaryPassword() is called when a user chooses Forgot Password and input his username and email. It will return the temporary newly set password generated by the server for the user. Then the user can change to new password by calling the changePassword method. changePassword method will take in username and currentPassword twice for confirmation and the newPassword. Then after verifying input information, it will call setPassword() on the corresponding user, thus changing to new password.

Note: All data is stored in mySQL database. Simple User data like username, password, email, and type is stored in a user database table. But for the arrayList<GameObject>, there will be a separate table to store each game object. There will be a column telling which gameObject belongs to which users.

Game

Game logics:

Global
- Random rGenerator
+ getRandom(): int

Global – It contains a getRandom() method which will generate a random int number based on current time and each of them is guaranteed to be unique.

GameObject: [Serializable]
- long serialVersionUID - String: name - int: objectID
+ GameObject(String) + getName(): String + setName(String): void + getID(): int + equalID(int): boolean

GameObject – The very basic object for other game objects. It is guaranteed that each GameObject class will automatically generate a unique ID number by calling Global class getRandom().

SceneStatePair: [Serializable]
- long serialVersionUID + int sceneID + int stateID - int ID - String description - int x - int y - int w - int h + int usability
+ SceneStatePair(Scene, int) + setID(int): void

+ getID(): int + getX(): int + setX(int): void + getY(): int + setY(int): void + getW(): int + setW(int): void + getH(): int + setH(int): void + getDescription(): String + setDescription(String): void + toString(): String

SceneStatePair – This is a pair of scene and state. This is used for easily store a Scene with a specific state number.

SceneState: [Serializable]
- long serialVersionUID - ArrayList<Integer> sceneChoices - ArrayList<Integer> gameContentChoices - String description - int x - int y - int w - int h - String imagePath
+ SceneState (String) + getX(): int + setX(int): void + getY(): int + setY(int): void + getW(): int

+ setW(int): void + getH(): int + setH(int): void + addSceneID(Integer): void + removelinkedScene(integer): void + getSceneChoices(): ArrayList<Integer> + setDescription(String): void + getImageChoice(): String + getImageChoice(): String + setImagePath(String): void + getDescriptionID(): String + setImagePath(String): void + getDescriptionID(): String + addGameContentChoice(Integer): void + removeGameContentChoice(Integer): void + getGameContentChoice(): ArrayList<Integer> + toString(): String

SceneState – It is used to store which description texts, linked SceneStatePairs and GameContenets are chosen for this scene state.

GameContent: GameObject
-int: x -int: y -int: w -int: h +int usability + long serialVersionUID - String imagePath - String description
+ GameContent (String, String) + getX(): int

+ setX(int): void + getY(): int + setY(int): void + getW(): int + setW(int): void + getH(): void + setH(int): void + setDescription(String): void + lookup(): String + setImage(String): void + getImagePath(): String + toString(): String

GameContent – It contain an image and description texts that would be shown in a scene when this GameContent is chosen.

Player: GameObject
- long serialVersionUID ArrayList<GameContent> itemList
+ Player (String) + pickup(GameContent): void + drop(int): void +getItems(): ArrayList<GameContent>

Player- It represents the player in the game, which has a bag that could store all items he meet in game.

Scene: GameObject
- long serialVersionUID = HashMap<Integer, SceneState> ownSceneStates = HashMap<Integer, SceneStatePair> linkedScenes

= HashMap<Integer, GameContent> mGameContents - int currentSceneState
+ Scene (String) + addSceneState(SceneState): void + removeSceneState(int): Boolean + getSceneState(int): SceneState + getAllSceneStates(): HashMap<Integer, SceneState> + getAllSceneStatePairs(): HashMap<Integer, SceneStatePair> + getAllGameContents(): HashMap<Integer, GameContent> + setCurrentSceneState(int): void + addlinkedScenes(SceneStatePair): void + removelinkedScenes(int): void + getSceneStatePair(int): SceneStatePair + addGameContent(GameContent): void + removeGameContent(int): vid + getGameContent(int): GameContent + getCurrentSceneStateNum(): int + getCurrentSceneState(): SceneState + getCurrentSceneState(): SceneState + toString(): String

Scene – It stores all description texts, linked SceneStatePairs, GameContent, and SceneStates. It is responsible of changing SceneState and providing corresponding description texts, GameContents, and SceneStatePairs.

GameProject: GameObject
- long serialVersionUID - HashMap<Integer, Scene> SceneList - Player mPlayer - GameSetting mGameSetting

+ GameProject(String) + addScene(Scene): void + removeScene(int): void + getScene(int): Scene + getSceneMap(): HashMap<Integer, Scene> + setPlayer(Player): void + getPlayer(): Player + setGameSetting(GameSetting): void + getGameSetting(): GameSetting + setId(int): void

GameProject – It stores all the data of a game including Scenes, GameContents, Player, and GameSetting.

GameGUI: JFrame
- long serialVersionUID - GameProject mGameProject - JPanel overalPanel - JPanel sceneDescription - JPanel sceneContents - JPanel sceneButton - JLbael DescriptionText - Scene currentScene
+ GameGuiClass(GameProject) + refresh(): void + class buttonActionListener: [ActionListener] + class contentActionListener: [ActionListener]

GameGui - The GameGui is responsible for showing every Scenes in a GameProject. It will display one SceneState of one Scene at one time and provides buttons for players to interact (Change State/Scenes).

GameSetting[Serializable]
- long serialVersionUID - Image bgimageString - Image releaseButtonimageString - Image pressedButtonimageString - String fontString - int textFontSize - String textFontColor - int buttonFontSize - String buttonFontColor - int initialScene - int initialState
+ getInitialScene(): int + setInitialScene(int): void + getInitialStante(): int + setInitialState(int): void + GameSetting() + GameSetting(String,String, String, String, int, String, int, String, int, int) + setBackgroundImage(String): void + getBackgroundImage(): String + setReleasedButtonImage(String): void + getReleasedButtonString(): String + setPressedButtonString(String): void + getPressedButtonString(): String + setFont(String):void + getFont(): String + setTextFontSize(int): void + getTextFontSize(): int + setTextFontColor(String): void + getTextFontColor(): String

+ setButtonFontSize(int): void + getButtonFontSize(): int + setButtonFontColor(String): void + getButtonFontColor(): String

GameSetting – The class that stores the user's GUI settings for game, like how the buttons and text would look like and the background.

Client

StoryCreatorClientWindow: JFrame
- long: serialVersionUID - Dimension minSize - LoginWindow mLoginWindow - CreateFileWindow mCreateFileWindow - MainWindow mMainWindow - LoadingWindow mLoadingWindow - JPanel OverallPanel - ClientListener cls
+ StoryCreatorClientWindow() + getLoginSuccessSignal(LoginSuccessSignal): void + getSignUpSignal(SignUpSuccessSignal): void + getUserSignal(UserSignal): void + BackToLoginWindow(): void + getChangepasswordsccSignal(ChangePasswordSuccessSignal): void + SaveFileSuccess(): void class: WindowServerActionListener class: LoginLoadinglistener class: CreateFileConfirm class: CreateFileLogOut class: LoginAsGuestListener

StoryCreatorClientWindow – it is the main JFrame of the software. It will store two panels: LoginWindow panel and CreateFileWindow. The main method will generate a new SotryCreatorClientWindow.

StoryCreator
StoryCreator() + main(String[]): void

ClientListener: Thread
- Socket mSocket - ObjectInputStream ois - ObjectOutputStream oos - StoryCreatorClientWindow mStoryCreatorClientWindow
+ setStoryCreatorClientWindow(StoryCreatorClientWindow): void + ClientListener(int, String) + Login(String, String): void + signUp(String, String): void + changePassword(String, String, String_: void - initializeVariables(): Boolean + run(): void + SendObjectToServer(Object): void

ServerWindow: JFrame
- long: serialVersionUID Font font
+ ServerWindow(ActionListener)

LoadingWindow: JPanel
- long: serialVersionUID

LoginWindow: JPanel
- long serialVersionUID JLabel: title JLabel: userLabel JLabel: passwordLabel JTextField: userField JPasswordField: passwordField JButton: loginButton Icon: gifIcon JLabel: signUp JLabel: guest JLabel: changePassword BufferedImage: image JLabel: logoLabel JPanel: buttonPabel JMenuBar: menuBar JMenuL menu JMenuItem: menuItem MouseListener: loginButtonActionListener MouseListener: loginAsGuestListener WindowServerActionListener: windowServerActionListener Font: font Font: font1 SignUpWindow: sw Color: color +boolean: isPressed ClientListener: mClientListener
+class BackgroundMenuBar:

+long: serialVersionUID =paintComponent(Graphics): void +class BackgroundMenu: +long: serialVersionUID +BackgroundMenu(String) =paintComponent(Graphics): void + LoginWindow(LoginLoadingListener, LoginAsGuestListener, WindowServerActionListener) +ChangeToNextLogo(): void +setClientListener(ClientListener): void + setup() : void +addUser(String, String): void +closeSignUpWindow()L void +isConnected(): Boolean +getLoginName(): String +getPassword(): String

LoginWindow – The LoginWindow is responsible of connecting with server and complete a series of user options. It will pass a User class to the next level GUI class.

CreateFileWindow: JPanel
+ long: serialVersionUID - User: mUser -JButton: userButton -JButton: createNewButton -JButton: openFileButton - JButton[]: templateButtons - JScrollPane: templateSP -GameTemplate: mGameTemplates -GameProject: mProject -ActionListener: confirmAction

-JButton: createButton -Color: default_color -Font: font
+getGameProject(): GameProject + CreateFileWindow(ActionListener, ActionListener) -initializeVariables(): void -createGUI(): void -addActionAdapters(ActionListener, ActionListener): void -showRemoteFiles(): void +setUser(User): void +getUser(): User

CreateFileWindow – Window opened after the user logs in. It allows the user to either select a template to create a new game, or open either a local or remote file to work on a previous game. The user can also choose to logout. Each of the templateButtons is a simple preview of the templates background and font, and the user must choose a default template or customize one in order to proceed to the main window. No matter what the user chooses to do, a GameProject must be passed on to the next window.

MainArea:

CreateGameContentDialog
-GameContent: mGameContent +Enum: FieldTitle -Color : color -Font : font -Insets: WEST_INSETS -Insets: EAST_INSETS -Image: notifyImg -Map<FieldTitle, JTextField>: fieldMap -JTextField: nameField

-JTextField: dspField -JTextField: imgField -JPanel: mainPanel -JPanel: optionPanel -JPanel: previewPanel -String: imgPath
+CreateGameContentDialog(Component) -createGbc(int, int): GridBagConstraints + getFieldText(FieldTitle): String + getContent(): GameContent + main(String[]): void

GameSettingDialog: JDialog
-Long: serialVersionUID -Insets: WEST_INSETS -Insets: EAST_INSETS -JPanel: mainPanel -JTextField: bgImgField -JTextField: pButtonField -JTextField: rButtonField -JComboBox<Scene>: sceneBox -JComboBox<SceneState>: stateBox -JTextField: fontField -JPanel: optionPanel -GameSetting: mGameSetting -Integer[]: fontSizeList -Color[]: colorList -String[]: fontColorList -JComboBox<Integer>: ButtonfontSizeBox -ButtonfontColorBox : JComboBox<String> -TextfontSizeBox : JComboBox<Integer> -TextfontColorBox : JComboBox<String>

int: sceneID int: stateID Color: color Font: font
+initVariable(): void +GameSettingDialog(GameProject) -createGbc(int, int): GridBagConstraints +main(String[]): void -createFileChooser(JTextField): void

InputHelper
-UpdateTool: mUpdateTool +String: NAME +String: DESCRIPTION +String: IMAGE
+InputHelper(UpdateTool) +setSceneInputListener(JTextField, Scene): void +setSceneStateInputListener(JTextField, SceneState, String): void

MainWindow: JFrame
- long: serialVersionUID -JPanel: mainPanel -JPanel: firstPanel - JPanel: upPanel -JPanel: bottomPanel -JPanel: secondPanel -JPanel: thirdPanel -String: savedFilePath -Image: notifyImg -Image: playImg

-Image: stopImg -Image: treeIcon1 -Image: treeIcon2 -GameProject: mGameProject -User: mUser -JScrollPane: hierarchyContentPanel -JTree: hierarchyTree -DefaultTreeModel: defaultTreeModel -CustomTreeCellRenderer: ctcr -JTabbedPane: workPanel -JPanel: previewPanel -JPanel: editPanel -UpdateTool: mUpdateTool -InputHelper: mInputHelper -JPanel: settingContainer -GameSetting: mGameSetting -Dimension: dSize -JFrame: thisWindow -Scene: ownScene -ClientListener: mClientListener -GameFrame: myFrame
+MainWindow(GameProject, User, BackTpCreateFile) -setUIFont(FontUIResource): void -saveToFileMethod(int): void -initializeGUI(): void -createMenu(): void -addNewScene(): void -addNewState(Object): void -addNodeToDefaultTreeModel(DefaultTreeModel, DefaultMutableTreeNode, DefaultMutableTreeNode): void -createHierarchyPanel(): void +setPreviewandSetting(Scene): void

-createWorkPanel(): void -createSettingPanel(): void -refreshFrame(GameProject): void +refreshSetting(ArrayList<ObjectLocationPanel>): void +setClientListener(ClientListener): void +SaveRemote(GameProject): void +SaveSuccess(): void -class: NewFileActionListener -class: SaveActionListener -class: OpenActionListener -class: CloseActionListener -class: HelpMenuActionListener class: tappedPanelListener +class: TutorialWindow class: UpdateTool class: CustomTreeCellRenderer

MainWindow – The main GUI for the StoryCreator, it houses a JMenuBar for basic operation, a left panel to show the Hierarchy Tree, a center panel to hold the preview and edit panel and a right panel to show the outline of present project.

OutlinePanel: JPanel, [Runnable]
+ long: serialVersionUID - JPanel centerPanel
+ OutlinePanel() - createSceneState(): void - refreshComponents(): void - linkSceneState(State, State): void

OutlinePanel – it is a JPanel that implement all the functionality of showing project outline, including create new scene state on the outline

panel, link two scene states on the outline panel. The class should implement runnable so that it should call refreshComponents() method whenever the edit panel is changed.

PlayControl: Thread
-GameProject mGameProject
+ PlayControl(GameProject) + run(): void

PlayControl – it is the main class to run the game user create. The class should extends Thread so that the user can run several game at one time. In the class, it should take in one GameProject component to help create GameGUI. Also the class should override run() method to create the GameGUI.

ObjectLocationPanel: JPanel
-long serialVersionUID -String SCENESTATE -String SCENESTATEPAIR -String GAMECONTENT -JTextField xfield -JTextField yfield -JTextField wfield -JTextField hfield -JSlider jsldHortX -JSlider jsldHortY -JSlider jsldHortW -JSlider jsldHortH -SceneStatePair mSceneStatePair -SceneState mSceneState -GameContent mgContent

-String type -int ix -int iy -int iw -int ih -GamePanel mGamePanel
ObjectLocationLanel(SceneState, GamePanel) ObjectLocationLanel(SceneStatePair, GamePanel) ObjectLocationLanel(GameContent, GamePanel) -instaniation(String): void -class: SliderChangeListener -class: InputChangeListener class: GameContentUpdateValue class: DescriptionUpdateValue class: sspUpdateValue

PlayerDialog: JDialog
-long serialVersionUID -Insets WERT_INSETS -Insets EAST_INSETS -JPanel mainPanel -JTextField nameField -JPanel optionPanel -PlayerDialog thisPD -Color default_color -Font font -Image notifyImg
PlayerDialog() -createGbc(int, int): GridBagConstraints

SaveTypeDialog: JDialog
-long serialVersionUID -JComboBox<String> jcb -String saveLocation +Boolean resume
SaveTypeDialog() +getSaveLocation(): String

SceneEditPanel: GameObject
- long: serialVersionUID - ArrayList<Integer> sceneChoices - ArrayList<Integer> gameContentChoices - String description - Int x - Int y - Int w - Int h - String imagePath
+ void SceneState(String) + Int getX() + void setX(int) + Int getY() + void setY(int) + Int getW() + void setW(int) + Int getH() + void setH(int) + void addSceneID(Integer) + void removelinkedScene(Integer) + ArrayList<Integer> getSceneChoice() + void setDescription(String)

+ String getImageChoice() + void setImageChoice(string) + void getDescriptionID() String + void getGameContent(Integer) + void removeGameContentChoice(Integer) + ArrayList<Integer> getGameContentChoices() + String toString()

SceneStateEditPanel: [Serializable]
- long: serialVersionUID + Scene mScene + SceneState mState - GameProject mGameProject - InputHelper minputHelper - Boolean CtrlPressed - Boolean AltPressed - Image notifyimg + SceneStateEditPanel(Scene, GameProject, InputHelper)
- void createGUI() Class LinkedSceneinfor: - GameProject mGameProject - SceneStatePair mSceneStatePair + LinkedSceneInfor(GameProject, SceneStatePair) + SceneStatePair getContent() + String toString ()

Class GameContentInfor:
- GameContent mGameContent
+ GameContent GameContentInfor()
+ getContent()
+ String toString ()

Class GameContentInfor:
- GameContent mGameContent
+ GameContentInfor(GameContent)
+ GameContent getContent()
+ String toString ()

Class sceneStateCreateDialog:
- Int SceneID
- Int StateID
- String desString
- Font font
- Font font1
- Color color
SceneStateCreateDialog(GameProject, Component)

Class SceneStateUpdateDialog:
+ void CheckAll()

PopupWindow:

ChangePassword: JDialog
- long: serialVersionUID
+ class MouseAdapter: [MouseAdapter] + class WindowAdapter: [windowClosing] + WindowAdapter()

+ Void Close() + Font font + JTextField username + JTextField oldPassword + JTextField newPassword

ChangePassword – it is the popup window for login in. It will let user enter username, old and new password.

OneImageTwoButton: JDialog
- long: serialVersionUID - Int choice - Font font
+ class MouseAdapter: [MouseAdapter] + class WindowAdapter: [windowClosing] + Int getContent()

OneImageTwoButton– it is the popup window for dialog that has image and two buttons.

OneSentenceTwoButton: JDialog
- long: serialVersionUID - Font font - Int choice - Boolean isPressed
+ class MouseAdapter: [MouseAdapter] + class WindowAdapter: [windowClosing] + Int getContent()

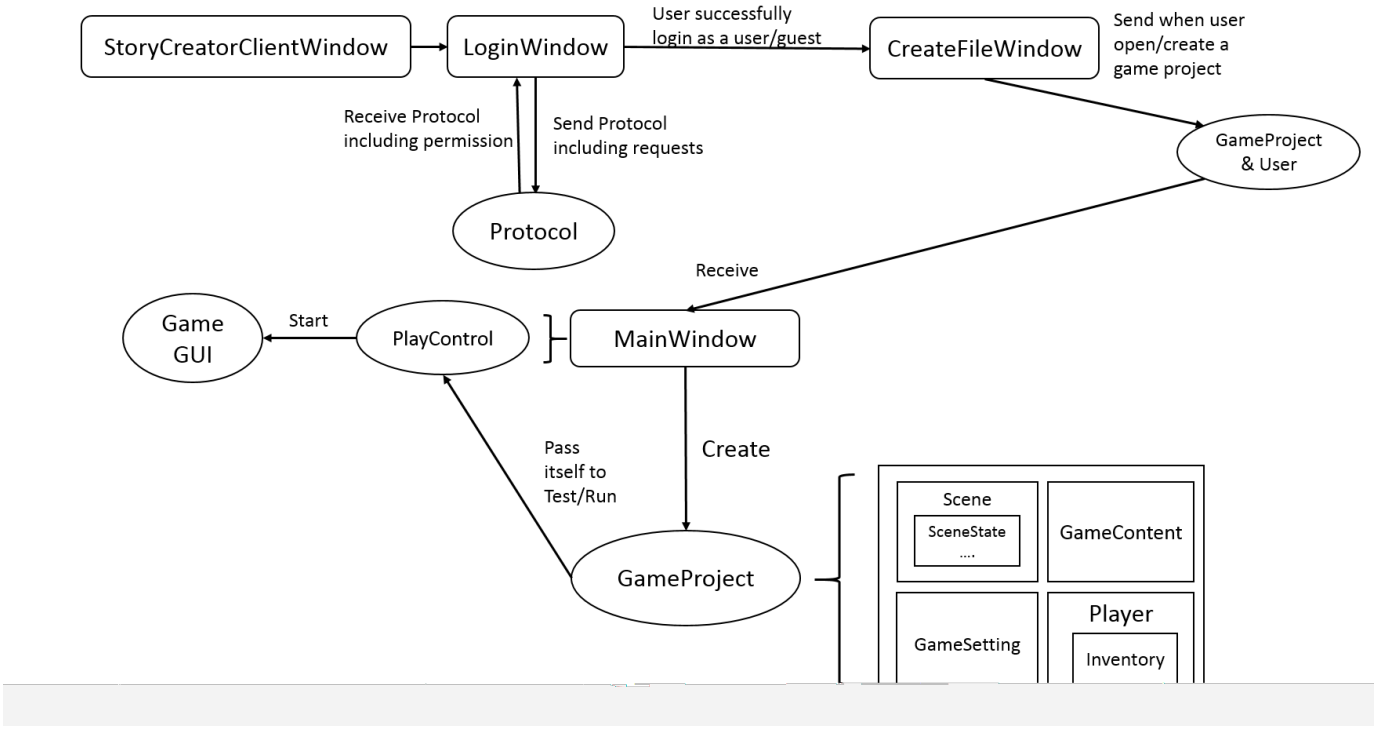
OneSentenceTwoButton– it is the popup window for dialog that has JLabel and two buttons.

OneTextFieldTwoButton: JDialog
- long: serialVersionUID - Font font - String input
+ class MouseAdapter: [MouseAdapter] + class WindowAdapter: [windowClosing] + class OneTextFieldTwoButton(Component, String, String, String, String) + class OneTextFieldTwoButton(Component) + class OneTextFieldTwoButton(Component, String, String) + Int getContent()

OneTextFieldTwoButton – it is the popup window for dialog that has JLabel and two buttons.

SignUpWindow: JDialog
- long: serialVersionUID - Font font - String input - JTextField username - JTextField password
+ class MouseAdapter: [MouseAdapter] + class WindowAdapter: [windowClosing] + class MouseAdapter: [MouseAdapter] + void close()

Use Case Diagram:



Testing

Game

Login:

Test#	1
Test Description	If auto connection is successful, status label will show some message
Steps to run test	1.Change default port or hostname to some valid value 2.Run StoryCreatorClientWindow
Expected Result	The status label has message that indicates connection is successful
Actual Result	The status label has message that indicates connection is successful

Test#	2
Test Description	If auto connection is failed, status label will show some message. User can change the default port and hostname and try again
Steps to run test	1.Change default port or hostname to some invalid value 2.Run StoryCreatorClientWindow 3.Click Setting

	4.Click Connect 5.Enter correct value in the pop up window that contains text fields and one confirm button 6.click “connect” button
Expected Result	Initially, status label should show some message that indicates connection is failed. After user enters correct value and presses connect button, The status label should show some message that indicates connection is successful
Actual Result	Initially, status label should show some message that indicates connection is failed. After user enters correct value and presses connect button, The status label should show some message that indicates connection is successful

Test#	3
Test Description	If username and password are incorrect. A window will pop up.
Steps to run test	1.Run StoryCreatorClientWindow 2.Enter incorrect username and password 3.Click login button
Expected Result	A message window should pop up to notify user that username or password is incorrect.
Actual Result	A message window should pop up to notify user that username or password is incorrect.

Test#	4
Test Description	If username and password are correct. User will proceed to the next window
Steps to run test	1.Run StoryCreatorClientWindow 2.Enter correct username and password 3.Click login button
Expected Result	User should be proceed to the next window
Actual Result	User proceed to the next window

Test#	5
Test Description	SignUp button pop up a window. If username, password or email address is invalid, a window will pop up
Steps to run test	1.Run StoryCreatorClientWindow 2.Click signUp button 3.A window that contains username, password, email address textfield and one confirm button will pop up 4.Enter invalid username, password or email address 5.Click confirm button
Expected Result	A message window should pop up to notify user that username, password and email address is invalid.
Actual Result	A message window should pop up to notify user that username, password and email address is invalid.

Test#	6
Test Description	SignUp button pop up a window. If username, password and email address are all valid, a window will pop up that notifies user signup is successful.
Steps to run test	1.Run StoryCreatorClientWindow 2.Click signUp button 3.A window that contains username, password, email address textfield and one confirm button will pop up 4.Enter invalid username, password and email address 5.Click confirm button
Expected Result	A window should pop up that notifies user signup is successful.
Actual Result	A window should pop up that notifies user signup is successful.

Test#	7
Test Description	Login as Guest pops up a window. User will proceed to the next window
Steps to run test	1.Run StoryCreatorClientWindow 2.Click login as guest
Expected Result	User should be proceed to the next window
Actual Result	User should be proceed to the next window

Create File:

Test #	01
Test Description	A JDialog should pop up when the User button is clicked.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Press the User button
Expected Result	A JDialog showing the username and a Logout button should pop up.
Actual Result	A JDialog showing the username and a Logout button should pop up.

Test #	02
Test Description	The user should be able to logout.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Press the User button 4. Press the Logout button
Expected Result	User logs out and returns to the LoginWindow.
Actual Result	User logs out and returns to the LoginWindow.

Test #	03
Test Description	Default templates should be showing in the main area of the window.

Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow
Expected Result	7 Predesigned templates and 1 default template should show up in the form of the JButtons.
Actual Result	7 Predesigned templates and 1 default template should show up in the form of the JButtons.

Test #	04
Test Description	User should be able to choose a template.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click on a template button
Expected Result	The template should be highlighted.
Actual Result	The template should be highlighted.

Test #	05
Test Description	User should be able to create a new game project with a chosen template.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click on a template button 4. Click on the create button
Expected Result	The client's MainWindow should show up with the chosen template designs displaying and ready to use.

Actual Result	The client's MainWindow should show up with the chosen template designs displaying and ready to use.

Test #	06
Test Description	A JDialog should open when the user clicks the open button and lets the user choose between opening a local file or a remote file.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click on the open button
Expected Result	A JDialog pops up with two buttons: "Local File" and "Remote File"
Actual Result	A JDialog pops up with two buttons: "Local File" and "Remote File"

Test #	07
Test Description	A JFileChooser opens when the user wants to open a local file.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click on the open button 4. Click on the Local File button
Expected Result	A JFileChooser pops up
Actual Result	A JFileChooser pops up

Test #	08
Test Description	The JFileChooser should allow the user to open a .txt file.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click the open button 4. Click the Local File button 5. Try to open a .txt file
Expected Result	The client should proceed to the MainWindow with the chosen .txt file loaded and previously saved work displayed.
Actual Result	The client should proceed to the MainWindow with the chosen .txt file loaded and previously saved work displayed.

Test #	09
Test Description	The JFileChooser should only allow the user to open a .txt file.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click the open button 4. Click the Local File button 5. Try to open a file that is not a .txt file
Expected Result	The client should not proceed to the MainWindow, would give out a warning and let the user choose a file again.
Actual Result	The client should not proceed to the MainWindow, would give out a warning and let the user choose a file again.

Test #	10
Test Description	A new JDialog should pop up if the user chooses to open a remote file saved on the server.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click the open button 4. Clock the Remote File button
Expected Result	A new JDialog pops up showing the user's previous GameProjects saved on the server in the form of a combobox.
Actual Result	A new JDialog pops up showing the user's previous GameProjects saved on the server in the form of a combobox.

Test #	11
Test Description	The user should be able to choose a GameProject saved on the server to work on if he or she has saved GameProjects previously.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Click the open button 4. Click the Remote File button 5. Choose a file and click OK
Expected Result	The user should proceed to the client's MainWindow when chosen a GameProject from a combobox.
Actual Result	The user should proceed to the client's MainWindow when chosen a GameProject from a combobox.

Test #	12
Test Description	The user should be prompt to do something else if he or she doesn't have any files saved on the server.
Steps to run test	1. Run the StoryCreator Client 2. Proceed to the CreateFileWindow 3. Clock the open button 4. Click the Remote File button
Expected Result	The JDialog would prompt the user that there are no saved GameProjects on the server and the user should either open a local file or create a new GameProject from a template.
Actual Result	The JDialog prompts the user that there are no saved GameProjects on the server and the user should either open a local file or create a new GameProject from a template.

MainWindow

Test #	01
Test Description	Every JMenuBar show correct output
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the "File" button on Menu 4. Press the "Asset" button on Menu 5. Press the "Object" button on Menu 6. Press the "Help" Button on Menu

Expected Result	1. Show up “New”, “Open File…”, “Save” , “Upload”, “Close”, and “GUI Setting” when clicking “File” button 2. Show up “Import Image” when clicking “Asset” button 3. Show up “Player” and “Game Content” 4. Show up “Tutorial” when clicking “Help” button
Actual Result	1. Show up “New”, “Open File…”, “Save” , “Upload”, “Close”, and “GUI Setting” when clicking “File” button 2. Show up “Import Image” when clicking “Asset” button 3. Show up “Player” and “Game Content” 4. Show up “Tutorial” when clicking “Help” button

Test #	02
Test Description	User should be able to save the current file and create a new file when clicking “New” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “New” button
Expected Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path)

	2. The software then should return to the CreateFileWindow to let user chooses a template and create a new file
Actual Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The software then should return to the CreateFileWindow to let user chooses a template and create a new file

Test #	03
Test Description	User should be able to save the current file and open a existing file when clicking “Open File…” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “Open File” button
Expected Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The software then should return to the CreateFileWindow to let user chooses an existing project
Actual Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the

	file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The software then should return to the CreateFileWindow to let user chooses an existing project

Test #	04
Test Description	User should be able to save the current file and closes it when clicking “Close” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “Close” button
Expected Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The software then should terminate
Actual Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The software then should terminate

Test #	05
Test Description	User should be able to save the current file and continues his or her work when clicking “Save” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “Save” button
Expected Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The JDialog will be automatically closed and user could continue his or her work
Actual Result	1. The software should pop up a JDialog to allow user save his or her file to a certain path. (If the file has already been saved before, the JDialog will not show up and the file will automatically be saved to the previous path) 2. The JDialog will be automatically closed and user could continue his or her work

Test #	06
Test Description	User should be able to change his game project GUI setting by click the “GUI Setting button”
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow

	3. Press the “File” button on Menu 4. Press the “GUI Setting” button
Expected Result	The software should pop up a JDialog to allow user to make any necessary changes to the current game project GUI Setting. User confirm his changes by click the confirm button.
Actual Result	The software should pop up a JDialog to allow user to make any necessary changes to the current game project GUI Setting. User confirm his changes by click the confirm button.

Test #	07
Test Description	User should be able to upload the current game project to the remote serve
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “Upload” button
Expected Result	The software should pop up a JDialog to indicate whether user successfully upload his game project. It would tell user “Not successful because of connecting error” if disconnect with the server. Otherwise, “Successfully uploading”.
Actual Result	The software should pop up a JDialog to indicate whether user successfully upload his game project. It would tell user “Not successful because of connecting error” if disconnect with the server. Otherwise, “Successfully uploading”.

Test #	08
Test Description	User should be able to add the image to the resource package and panel when clicking “Import Image…” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “Asset” button on Menu 4. Press the “Import Image…” button
Expected Result	1. The software should pop up a FileChooser to allow user select the image he or she wants to import 2. The name of the image will automatically show up in the Resource JPanel, and the image file should be added into certain package
Actual Result	1. The software should pop up a FileChooser to allow user select the image he or she wants to import 2. The name of the image will automatically show up in the Resource JPanel, and the image file should be added into certain package

Test #	09
Test Description	User should be able to look up the tutorial of the software any time he or she wants when clicking “Tutorial” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “Help” button on Menu

	4. Press the “Tutorial” button
Expected Result	1. The software should pop up a JFrame which contain the basic information and instruction about software 2. User is allowed to do any operations while looking up the tutorial
Actual Result	1. The software should pop up a JFrame which contain the basic information and instruction about software 2. User is allowed to do any operations while looking up the tutorial

Test #	10
Test Description	User should be able to run he or her current project when clicking “Play” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “Play” button at the left corner
Expected Result	1. The software should automatically generate a game project which contain all the process user does 2. User is able to test and go through his or her game through the project software generates 3. User is able to run several game at the same time by repeatedly clicking “Play” button
Actual Result	1. The software should automatically generate a game project which contain all the process user does

	2. User is able to test and go through his or her game through the project software generates 3. User is able to run several game at the same time by repeatedly clicking “Play” button

Test #	11
Test Description	User should be able to stop the current running project when clicking “Stop” button
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “Stop” button at the left corner
Expected Result	1. The current running game will be terminated and the window will be closed automatically 2. Other games that user previously ran will not be influenced
Actual Result	1. The current running game will be terminated and the window will be closed automatically 2. Other games that user previously ran will not be influenced

Test #	12
Test Description	User should be able to show or hide the “Scene” catalogues by double clicking the “Hierarchy” root
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow

	3. Double click the “Hierarchy” root at the left corner
Expected Result	1. If the “Scene” catalogues are hidden, the Hierarchy tree will expand 2. If the “Scene” catalogues are shown, the Hierarchy tree will pack up
Actual Result	1. If the “Scene” catalogues are hidden, the Hierarchy tree will expand 2. If the “Scene” catalogues are shown, the Hierarchy tree will pack up

Test #	13
Test Description	User should be able to create a new Scene by right clicking the “Hierarchy” root
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Right click the “Hierarchy” root at the left corner 4. Click “Create New Scene” option in the pop up dialog
Expected Result	1. When right clicking the “Hierarchy” root, a pop up JDialog will show up and contain “Create New Scene” option 2. When user click the “Create New Scene”, a new Scene will be created in the backstage (the Scene will contain a state in default) 3. Also a new Scene node will be created in the Hierarchy tree

	4. The “Preview” and “Edit” panel will automatically show the content in the new Scene
Actual Result	1. When right clicking the “Hierarchy” root, a pop up JDialog will show up and contain “Create New Scene” option 2. When user click the “Create New Scene”, a new Scene will be created in the backstage (the Scene will contain a state in default) 3. Also a new Scene node will be created in the Hierarchy tree 4. The “Preview” and “Edit” panel will automatically show the content in the new Scene

Test #	14
Test Description	User should be able to show or hide the “State” catalogues by double clicking the “Scene” node
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Double click the “Hierarchy” root at the left corner to see the “Scene” node 4. Double click the “Scene” node
Expected Result	1. If the “State” catalogues are hidden, the Scene tree will expand 2. If the “State” catalogues are shown, the Scene tree will pack up
Actual Result	1. If the “State” catalogues are hidden, the Scene tree will expand

	2. If the “State” catalogues are shown, the Scene tree will pack up

Test #	15
Test Description	User should be able to create a new State by right clicking the “Scene” node
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Double click the “Hierarchy” root at the left corner to see the "Scene” node 4. Right click the “Scene” node 5. Click “Create New State” option in the pop up dialog
Expected Result	1. When right clicking one of the “Scene” nodes, a pop up JDialog will show up and contain “Create New State” option 2. When user click the “Create New State”, a new Scene will be created in the backstage 3. Also a new Scene node will be created in the Hierarchy tree 4. The “Preview” and “Edit” panel will automatically show the content in the new Scene
Actual Result	1. When right clicking one of the “Scene” nodes, a pop up JDialog will show up and contain “Create New State” option 2. When user click the “Create New State”, a new Scene will be created in the backstage

	3. Also a new Scene node will be created in the Hierarchy tree 4. The “Preview” and “Edit” panel will automatically show the content in the new Scene

Test #	16
Test Description	User should be able to delete a existing Scene by right clicking the “Scene” node
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Double click the “Hierarchy” root at the left corner to see the " 4. Click “Delete Scene” option in the pop up dialog
Expected Result	1. When right clicking one of the “Scene” nodes, a pop up JDialog will show up and contain “Delete Scene” option 2. When user click the “Delete Scene”, certain Scene will be deleted in the backstage 3. Also certain Scene node will be deleted in the Hierarchy tree
Actual Result	1. When right clicking one of the “Scene” nodes, a pop up JDialog will show up and contain “Delete Scene” option 2. When user click the “Delete Scene”, certain Scene will be deleted in the backstage 3. Also certain Scene node will be deleted in the Hierarchy tree

Test #	17
Test Description	User should be able to delete an existing State by right clicking the “State” node
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Double click the “Hierarchy” root at the left corner to see the "Scene” node 4. Double click the “Scene” node to see the “State” node 5. Right click the “State” node 6. Click “Delete Scene” option in the pop up dialog
Expected Result	1. When right clicking one of the “State” nodes, a pop up JDialog will show up, which contains “Delete State” option 2. When user click the “Delete State” option, certain Sate will be deleted in the backstage 3. Also certain State node will be deleted in the Scene tree
Actual Result	1. When right clicking one of the “State” nodes, a pop up JDialog will show up, which contains “Delete State” option 2. When user click the “Delete State” option, certain Sate will be deleted in the backstage 3. Also certain State node will be deleted in the Scene tree

Test #	18
Test Description	User should be able to see the current outline of his or her work in the Outline panel

Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Check the “Outline” panel at the right side
Expected Result	1. The outline panel should contain all the Scenes, States and connections that user currently creates 2. Outline should be updated whenever the user click the confirm button in the “Edit” panel
Actual Result	1. The outline panel should contain all the Scenes, States and connections that user currently creates 2. Outline should be updated whenever the user click the confirm button in the “Edit” panel

Test #	19
Test Description	User can see the preview of the current scene at the current state
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Select any Scene 4. Setelt any State 5. Click the preview button located in the center of the window
Expected Result	The preview of the current scene at the current state should be displayed accordingly to GameGUI setting
Actual Result	The preview of the current scene at the current state should be displayed accordingly to GameGUI setting

Test #	20
Test Description	Preview should reflect to any GUI Setting changes immediately
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Press the “File” button on Menu 4. Press the “GUI Setting” button 5. Make changes of all properties 6. Click confirm button 7. Click preview
Expected Result	All changes should be reflected in preview accordingly to our changes
Actual Result	All changes should be reflected in preview accordingly to our changes

Test #	20
Test Description	When click edit button, user can see a editing area
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Select any Scene 4. Setelt any State 5. Click edit button
Expected Result	There should be an editing area which has several input text fields and combo box for users to do any necessary modifications.
Actual Result	There should be an editing area which has several input text fields and combo box for users to do any necessary modifications.

Test #	21
Test Description	User can make any change of all properties of the current scene at the current state
Step to run test	1. Run the StoryCreator Client 2. Proceed to MainWindow 3. Select any scene 4. Select any state 5. Make arbitrary changes in the editing area 6. Click preview
Expected Result	All changes should be reflected in the preview area
Actual Result	All changes should be reflected in the preview area

Server

Test#	01
Test Description:	The server should start listening for connection after valid port number is entered
Steps to run test:	1.) Open up the port GUI 2.) Enter valid port number 3.)Click the button “start listening”
Expected result:	If the port number is valid, the Server GUI will pop up and can now allow connection with clients. Otherwise, there will be a message saying the port number is invalid.

Test#	02
Test Description:	Server will cut all connections with clients when it is terminated
Steps to run test:	1.) Start the port GUI 2.) Enter valid port number 3.) Start listening 4.) Log in on the main window GUI successfully 2.) Click the button “terminate” on the server GUI
Expected result:	The connections between server and client are cut. The client cannot do any activities that require server connection such as log in from another main window, sign up, or save their projects.

Test#	03
Test Description:	Server allows multiple client connections
Steps to run test:	1.) Start the port GUI 2.) Enter valid port number 3.) Start listening 4.) Log in on the main window GUI successfully 5.) Repeat step 4 from another main window GUI with another username
Expected result:	Two Log ins must be successful.

Test#	04

Test Description:	Server does not allow connection with multiple clients who try to log in with same username
Steps to run test:	1.) Start the port GUI 2.) Enter valid port number 3.) Start listening 4.) Log in on the main window GUI successfully 5.) Repeat step 4 from another main window GUI with the same username
Expected result:	The second log in attempt should fail. There will be message noting that this username has already logged in.

Test#	05
Test Description:	Server can service the requests from multiple client connections
Steps to run test:	1.) Start the port GUI 2.) Enter valid port number 3.) Start listening 4.) Log in on the main window GUI successfully 5.) Repeat step 4 from another main window GUI with another username 6.) Perform some actions on the first main window such as creating new project and save it 7.) Repeat step 6 on the second main window 8.) Log out and Log in again from the first main window 9.) Repeat step 8 on the second main window
Expected result:	After re-logging in on both windows, the new game project created on each account before logging out

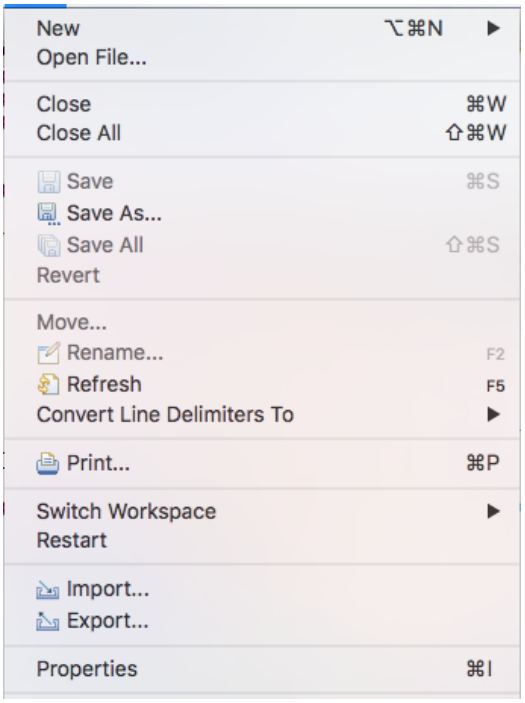
	must be saved and accessible. This would mean that server successfully service requests from multiple clients.

Database

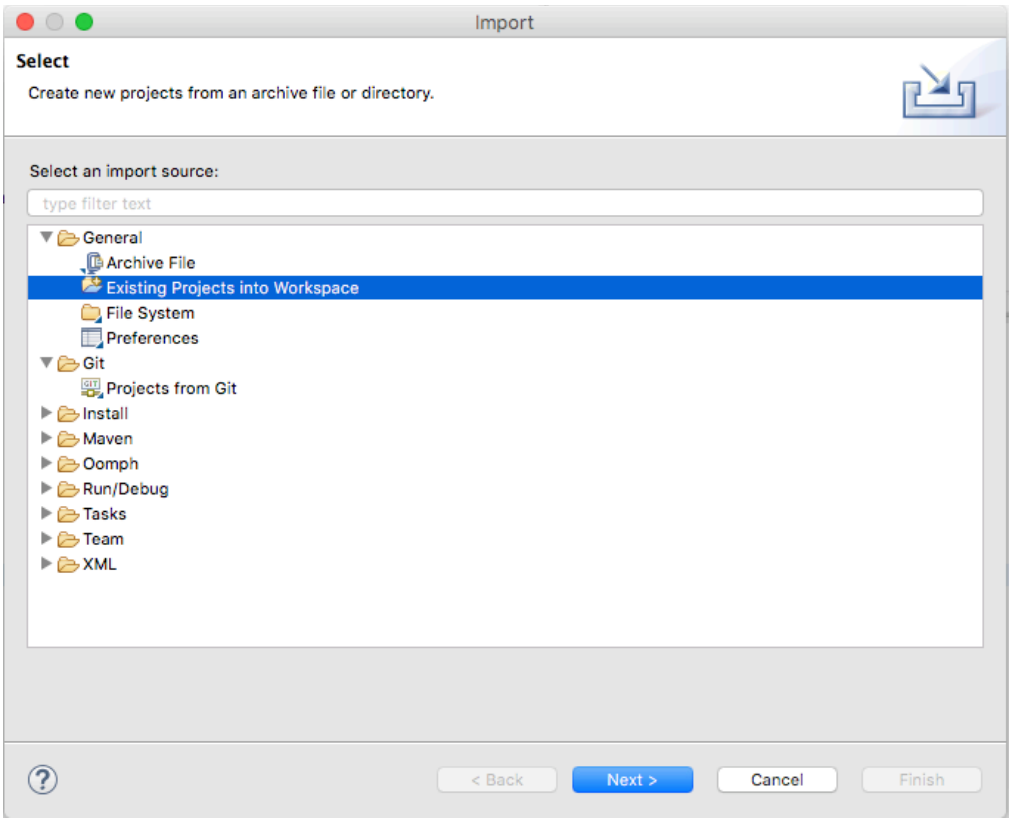
Test#	01
Test Description:	User should be able to save game projects. This also tests for retrieving user object from database.
Steps to run test:	1.) Launch the StoryCreator client 2.) Create a game project and make changes to the project 3.) Save remotely 4.) Close the program 5.) Launch the program again 6.) Open existing project and try to look for the one you just saved
Expected result:	Saving function and user object retrieving function work, users are able to open an existing project, which is the one the user has saved before closing the program. Users are also able to see other existing projects made previously.

Deployment

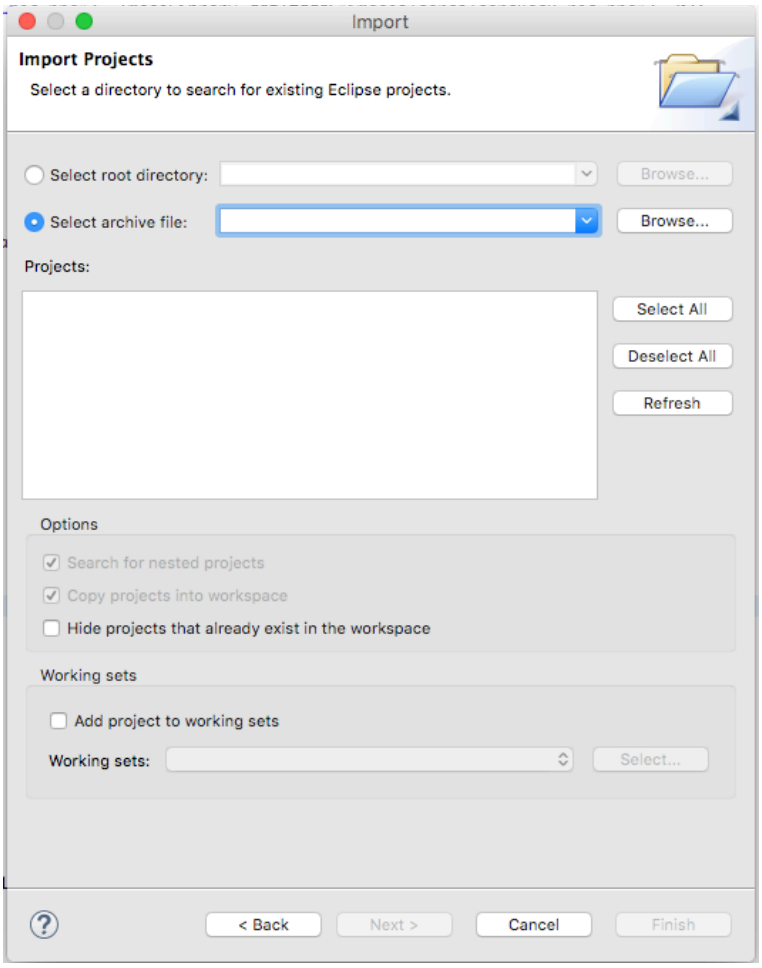
To deploy this application within Eclipse, import the StoryCreator.zip file into Eclipse.
Open Eclipse, click “File” menu bar, click “Import…”



Then click “Existing Projects into Workspace”, click next to continue



After that, click “Browse…” to import project from File Chooser, then click “Finish”.



This should generate a project called StoryCreator with src and resources directories. To execute the StoryCreator project, run StoryCreatorClientWindow.java in the tip.storycreator.client package. No need to run the server in the local computer. The server has already been set up on cloud.

Before user signs up or login, he or she should click the “Setting” → “Confirm” to connect to the port. After this, user should be able to sign up and login.

When users finishing creating their own project, they should be able to run their projects directly through our software by clicking the “Play”

button in the MainWindow. After clicking the “Play” button, the game GUI will show up, and users could play their game in the way they create.

To read from or save to the remote(Server)/local side, use the “Open” and “Save” button created in the MainWindow. After clicking one of the two buttons, a file chooser will pop up which contain the choices of open from/save to the local/remote. Users could choose to save or open their file in different ways base on their choices. Also when users are in the CreateFile Window, they could also choose to open existing file from local/remote side.