**Client Side**

Setting and Story

RealmRPG is a fantasy based RPG set in the caves and dungeons found in the under dark (cave systems below the planets crust). The story is based around a single character who has recently awoken alone in an uninhabited cave in the lower levels of the under dark. He has no memory of how he ended up there but he does remember the reason why, to finish the task of assassinating a certain dark elf.

Details and Design

The game itself will be tile based with tiles size 32:32 pixels and the view port will be 800:600 pixels. The graphics style of the game will be entirely pixel art for the characters, items, tiles, and icons. Backgrounds will consist of pixel art, just at a larger scale than the other objects.

The game engine will consist of all the different systems that will control the entire game, all updated on the same game loop. The main systems in the game are state stack, main menu state, local game state, dialogue state, inventory state, stats state, map system, and the battle state.

**State Stack**

The state stack is the heart of the game, it controls which part of the game the player is interacting with at any given time. The state stack will have 4 default functions that every state must contain in order to be called to the stack.

onEnter: Is ran every time a new state is pushed onto the top of the stack, it sets up the new state by initializing new variables and setting game variables to the correct values.

onExit: Is ran every time the current state is popped off of the stack, local variables are reset to the default values, global variables are changed to their correct values for the previous state on the stack.

Update: updates the current objects and variables present in the current state running on the stack

Draw: Draws the objects of the current state onto the game canvas

Then the state stack contains 4 functions that help maintain the stack throughout the game.

addState: adds a new state into the possible states that the game can be in

changeState: pushes the new game state onto the stack, running the new state while maintaining the old states information

revertState: pops off the current state and returns to the previous state

returnToMenu: returns the player to the main menu, used as a “quit” option when the player is done playing

**Main Menu State**

The first state the player will see when starting the game. There will be two options, “new game” and “load game”. Pressing the new game button will create a new div element that will then utilize ajax to grab a form page that will guide the player through making a new account. Pressing the load game button will also create a new div element that utilizes ajax to grab a form page except this page will guide the player through logging into an existing account.

**Local Game State**

The local game state is the state that he game will spend most of its time in. This state controls the current map, the player, collision detection, the viewport location, and all the game objects that are present on the current map. Every other state is accessed through the local game state through either collisions or hot key presses. There are no special functions that are a part of the local game state only the 4 required functions of all states, onEnter, onExit, update, and draw.

**Dialogue State**

The dialogue state is used to progress the story through the communication between the player and the NPC’s. This state will contain the map information as well as the player location and NPC’s/object locations. It will then “pause” all movement on the map and create a custom dialogue box at the bottom of the screen. The player will then be able to click the dialogue box to move on to the next set of text and or pop the dialogue state off of the stack if they reach the end of the current dialogue.

**Inventory State**

The inventory state will be pushed onto the stack when the “I” key is pressed on the keyboard while in the local game state. This state will keep track of all of the player’s current items, unless they are equipped, and display them to the player. Once the player clicks on one of the displayed items, he will be given the options to either drop the item, thus destroying it, or if able, to equipped the item. The inventory state will also display the cost of each item in gold. The inventory state will have an inventoryUpdate function that will make sure the current inventory being shown is the most up to date and will dynamically change the locations of the items when needed. When done viewing the inventory the player will then be able to click a return button that will then pop the inventory state off of the stack and return the player to the local game state.

**Stats State**

Similarly to the inventory state, the stats state will be pushed onto the stack when the “k” key is pressed on the keyboard while in the local game state. This state will keep track of the player’s current stats, strength, defense, agility, health, mana, and experience. This state will also show the player’s current equipped weapon and armor and when these items are pressed, the player will be presented with the items attack/defense statistics and the option to unequip said item. The stats state will also double as the “logout” menu where there will be two buttons at the bottom of the page. One button will give the player the ability to save their current progress and the other button will allow the player to logout of the game and return to the main menu state.

**Map System**

The map system will have two options, either read in a single array of integers and convert those integers to their corresponding tile images, or read in multiple arrays of a “layered” map and draw each tile on top of each other starting from the first array and moving to the last array. The layered approach will allow me to separate the game tiles, game objects, and collision items in a more organized manner as I can set them up separately but still draw them as if they were the same array. The map system will have a generateMap function along with a changeMap function.

**Battle State**

The battle state will be pushed onto the stack every time the player encounters an unfriendly NPC during their adventures through the under dark. The battle state will contain its own State Stack that will keep track of the entire battle. The battle stack state will have 3 sub states just like the main state stack has 6. Battle Tick State, Battle Action State, and Loot State.

The battle tick state will be running most of the battle. This state keeps track of whose turn it is to choose an action, the player or the enemy. Every time the player or the enemy chooses an action, a new PChoice or EChoice will be added to an array that will then tell the battle state whose turn it is. The battle tick will pop the top choice option from the array and this will either prompt the player with 2 action buttons, “spells” or “attack” or prompt the enemy to run through its binary tree until it reaches a leaf. Once an action is chosen the PChoice/EChoice object will be pushed onto the Battle Action State and that state will be pushed onto the stack. The action will then be ran, either attacking the opposition or casting a spell. Once the action is completed the Battle Action State will be pushed off the stack and the battle tick state will continue popping off choices from the choice array until one of the enemy’s health drops to 0. Once the battle is won, if the player is the victor the Loot State will be pushed onto the stack which will then add the experience and any other loot (weapons, armor, items) to the player object. The Battle State will then be pushed off of the main State Stack and the Local Game State will continue on where the player left off.

**Server Side**

The main functionality on the server side will be saving and loading games depending on a user log in system. I will be implementing these ideas through the use of JSON, sessions, ajax, and a database.

**Loading Game**

The player will have the option to load a game on the main menu state when they first boot up the game. When the button is clicked an ajax function will be called and a new div will be created that will then have a forum added to its children through ajax. The player will then be able to enter their username and password into this forum. The forum will then be posted to a controller which will then use a model specifically modified for the games database. This model will check to see if the username and password combinations are present in the database and if so, will start a session for the player and pass in all the information stored in the database in the form of JSON. The game will then change the JSON back into readable variables for javascript, set the corresponding variables to the newly loaded database variables, and then start the game where the player left off.

**Saving Game**

The player will have the option to save the game when they are in the stats state. When the save button is pressed, the character will be transferred to JSON, along with the map information. This information will then be sent to a controller using ajax and will then replace the values in the database using the model. The game will then continue from the stats state, giving the player the option to quit the game to the main menu or to continue their adventure.