RPGBattle

Jonathan Cote V00962634

JonCote95@gmail.com

Project/Problem Overview:

Development proposal for a text-based RPG battle game. This project's goal is to develop a turn-based RPG battler game using the ncurses text-based user interface library, boost JSON library, and C++. The proposed game's core gameplay loop will consist of 1vs1 battles verse AI opponents where the player and AI will select one of four combat actions each turn. The combat actions available to the player are attack, block, dodge, and a player assignable skill. Combat will follow a stat based rock paper scissor match defined in more detail within the combat section below. When the player successfully wins a fight, they will be awarded experience points. Once a player has acquired enough experience points, they will gain a level-up awarding them with slight increase of base stats and player assignable attribute points that can be assigned to Strength, Defense, Dexterity, and Intelligence. Strength improves the players physical attack making impowering the attack and physical attack power scaling skills damage output. Defense improves the players chance to successfully block incoming damage. Dexterity increases the players accuracy rate allowing for improved odds of landing damage vs the block/dodge combat actions. Additionally, dexterity slightly increases the players physical attack power. Intelligence increases the players magic attack improving the damage of the attack spell and magic attack-based skills, allowing for more skill casts during combat. Additionally, intelligence increases the players mana pool allowing them to cast more skills over a combat encounter. The game will provide a save/load feature that enables players to continue character progression throughout play times.

Classes/objects/methods:

The character class is used for tracking all player character progression information, attribute assignment, statistic calculations, and other player character specific functionality. Interaction between the game and the players character will all go through the character class. The enemy class provides enemy creation, attributes, stats, and information components.

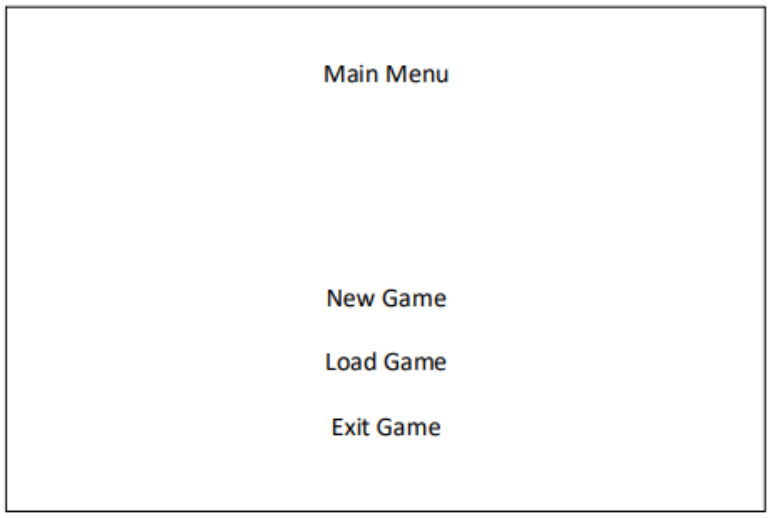
Enemy creation will be based off 3 architypes the warrior, rogue, and mage. The warrior is a strength plus defense-based attribute build focused on dealing consistent damage while having a high health point pool and block rate. The rogue is a swift dexterity-based class focused on swift accurate attacks and

high dodge rates. The mage focuses on high damage skills with a limited sustainable damage output and defense.

The core gameplay and UI is controlled by several methods accesses dependent on current game state. Each of the core UI game scenes Main Menu, Lobby, New Game, Load Game, and Combat will be called by the programs main function. Within the core gameplay scenes additional functionality/rendering will be called based on user inputs.

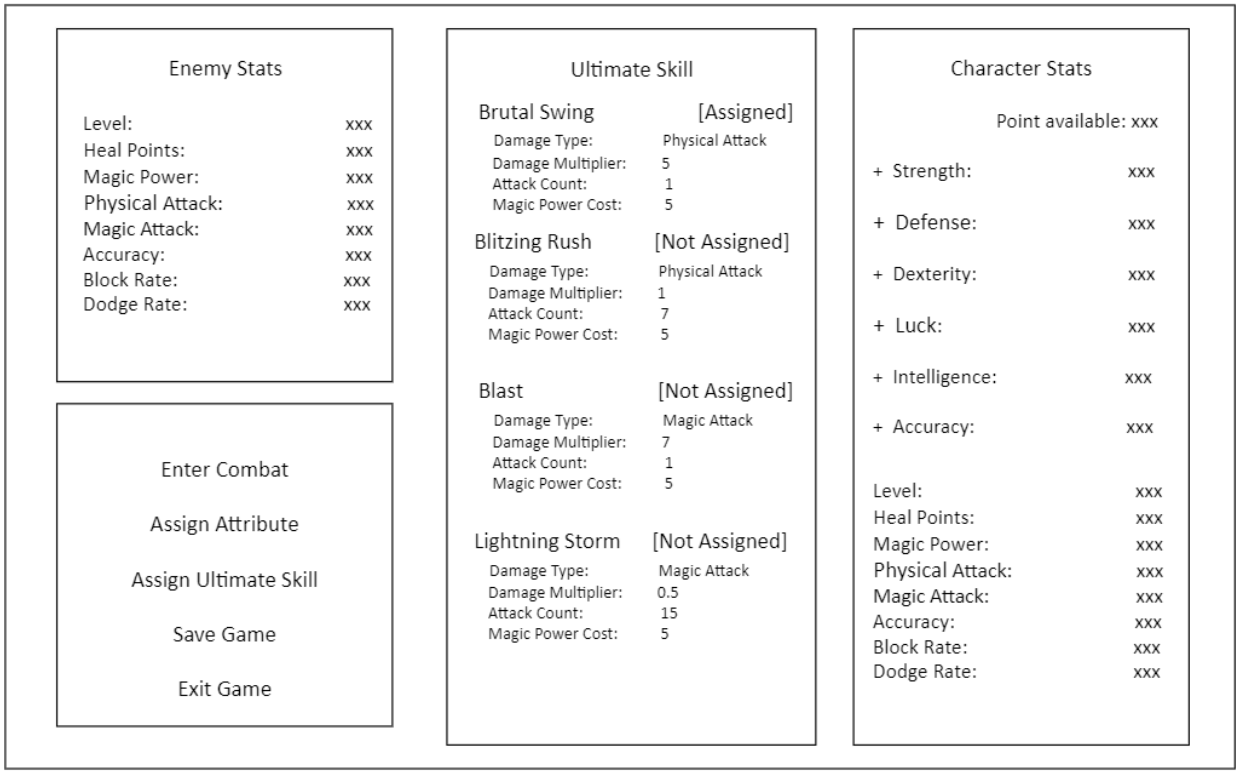
UI:

Main Menu:



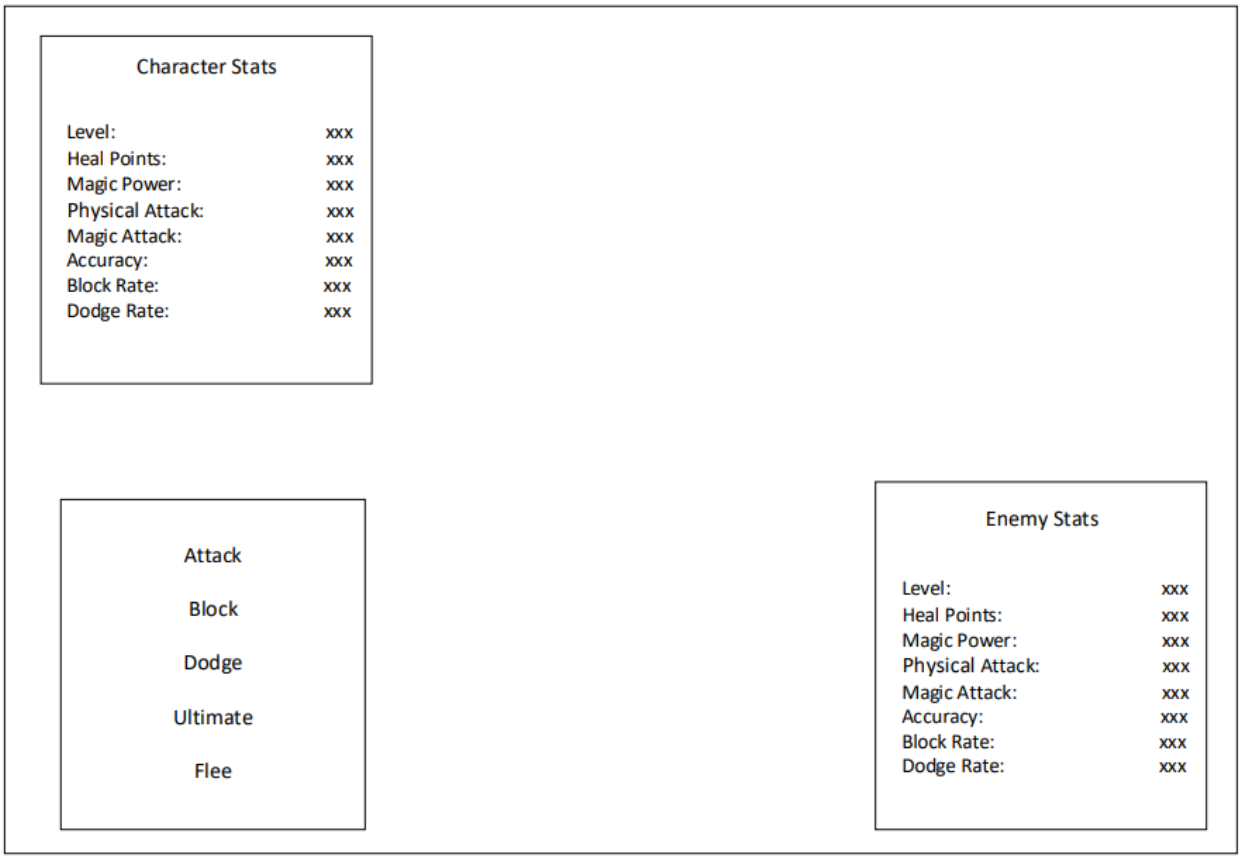
The Main Menu is the entry point for the game allowing players to start a new game or load existing saved games. The player will navigate the UI using the arrow keys to highlight desired action, when highlighted the player can hit the enter key to make their selection. If a player selects the New Game menu option, they will be prompted to select the desired save slot for their new game. After selected the desired save slot the player will be prompted to provide a character name that is less than 20 characters long, consists of only letter symbols, and is not empty. Upon inputting a valid character name, the player will be redirected to the game Lobby with a base level 1 character created. When a player selects Load Game, they will be directed to the Load Saved game scene where the player can select the desired saved game, they would like to continue gameplay with. During the navigations expressed above the player can hit the backspace key to return to the Main Menu. However, if at the character name input scene, the player will have to use the ESC key to cancel character creation and be returned to the Main Menu.

Lobby:



The Lobby scene is the primary scene for the game that gives the user the ability to assign character attribute points, ultimate skill, enter combat, and save current game. Navigation of the Lobby scene is done using the arrow keys to highlight the desired action they would like to perform then hit enter to input the selected action. If a user selects Assign Attribute the UI will begin interacting with the Character Stats UI where the player can navigate and assign available attribute points. The player can additionally assign an ultimate skill by selecting the Assign Ultimate Skill menu option that will send them to the Ultimate Skill menu where they can select the desired ultimate skill. Player input of the backspace key allows for navigate back to the primary menu from the Ultimate Skill and Character Stats menus. If the player selects the Save Game menu option, their character attributes and progression information will be saved to the save slot selected during character creation. If a player selects the Enter Combat option, they will be directed to the Combat scene to conduct battle against the next enemy.

Combat:



Combat is conducted by the player selecting one of the possible combat action Attack, Block, Dodge, Ultimate, and Flee. Each of the combat actions available scale in effectiveness based on the players character stats, opponents combat action, and the enemies' stats. The basic combat logic will follow a similar design to older Pokémon games, with the following action hierarchy,

Attack vs Attack: Both enemy and player take full heal point damage from the attacks

Attack vs Block:

If Block roll successful: No damage is done

Else: Attack damage is done

Attack vs Dodge:

If Dodge roll successful: No damage is done

Else: Attack damage is done

Attack vs Ultimate: Both enemy and player take damage

Ultimate vs Block:

For each instance of damage from the skill:

If Block roll successful: No damage is done

Else: damage is done

Ultimate vs Dodge:

For each instance of damage from the skill:

If Dodge roll successful: No damage is done

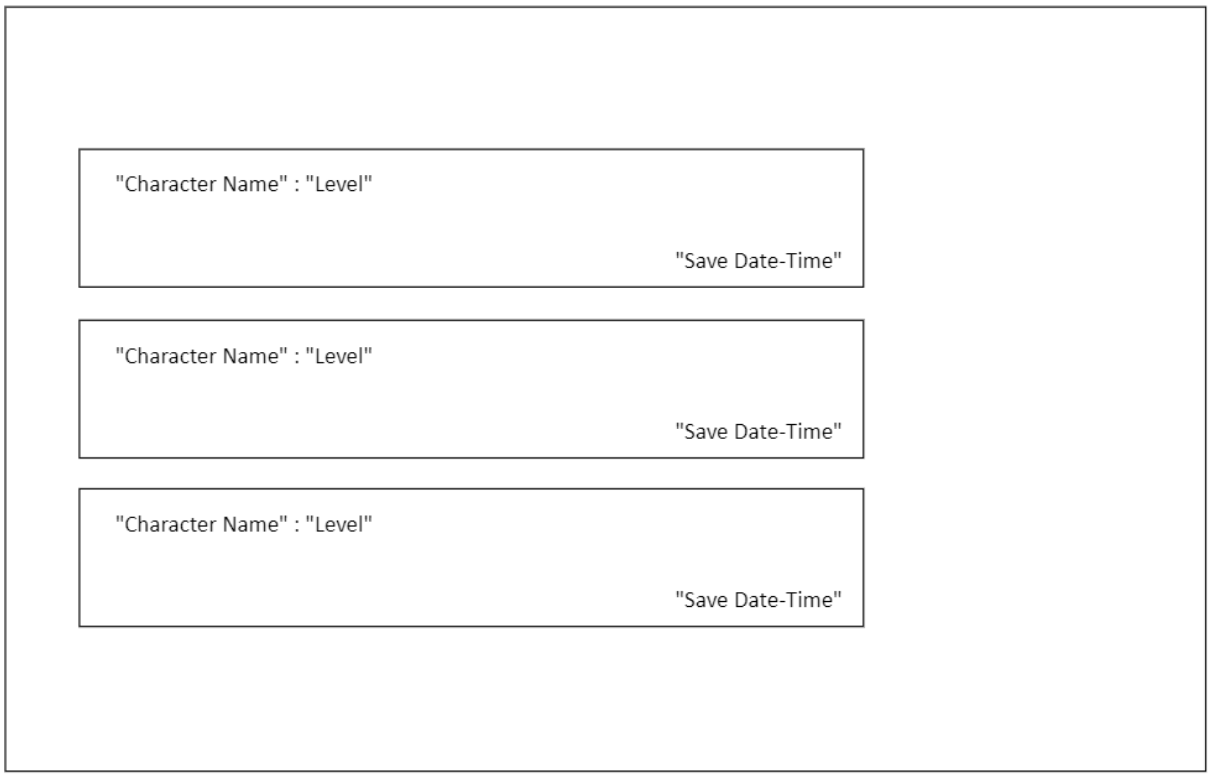
Else: damage is done

Ultimate vs Ultimate: Both enemy and player take damage

Block/Dodge vs Block/Dodge: Nothing happens

Additionally, the success rate of a Block/Dodge is based on the Accuracy and Block rate/Dodge rate stats of the player/enemy. Higher the Accuracy the lower the rate of success for Block/Dodge combat actions. Combat is completed when the user selects the Flee action or one of the combatants Heal Points hits 0.

Load Game:



In the Main Menu for the game the user can select the Load Game menu option to be directed to the Load Game scene (seen in image above). The user will see character name, level, and save date-time for up to 3 saved games. Once the user selects the desired save, character data from the saved game JSON file will be loaded in before redirecting the user to the game lobby. During execution of the game file user can pass in a desired well-formatted JSON file to be used for character saving and loading. If no file is passed in a default save/load JSON file will be used.

Launching/Navigation:

The game will be launched within the user's terminal (testing done using Linux Bash terminal) following execution of the cmake build file. During execution of the built game file named RPGBattle the user can pass in a well-formed user-defined save file as an argument to advert save/load data from being directed to/from the default save file. When RPGBattle is launched, the game will take over the current terminal window to display the text-based game UI. During gameplay the user can input ctrl+c to violently close the game or select the Exit Game menu option in the Main Menu or Lobby scenes. When game is successfully launched the user will interact with the application using the arrow, backspace, enter, and

ESC keys. The arrow keys are used by the user to navigate menu selection options, while the enter key is used to select the highlighted menu option. Dependent on scene/menu user is currently in the backspace/ESC key is used to return to previous game scenes. Generally, backspace will be used as the escape input for users, in cases where ESC is used instead of backspace it will be specified in the scenes UI.

File Format:

Saved game data will be stored and collected from JSON files with the following structure:


```
{
    "Saves": [
        {
            "Name": "Magical",
            "Level": "10",
            "Exp": "28",
            "Exp_needed": "121",
            "Skill_id": "3",
            "Attribute Points": "0",
            "Strength": "4",
            "Defense": "4",
            "Dexterity": "1",
            "Intelligence": "30",
            "Save_DateTime": "Sun Aug  6 18:38:19 2023"
        },
        {
            "Name": "TheGoat",
            "Level": "1",
            "Exp": "0",
            "Exp_needed": "10",
            "Skill_id": "0",
            "Attribute Points": "3",
            "Strength": "2",
            "Defense": "2",
            "Dexterity": "1",
            "Intelligence": "1",
            "Save_DateTime": "Sun Aug  6 19:26:02 2023"
        },
        {
            "Name": "",
            "Level": "0",
            "Exp": "0",
            "Exp_needed": "0",
            "Skill_id": "0",
            "Attribute Points": "0",
            "Strength": "0",
            "Defense": "0",
            "Dexterity": "0",
            "Intelligence": "0",
            "Save_DateTime": "0"
        }
    ]
}
```

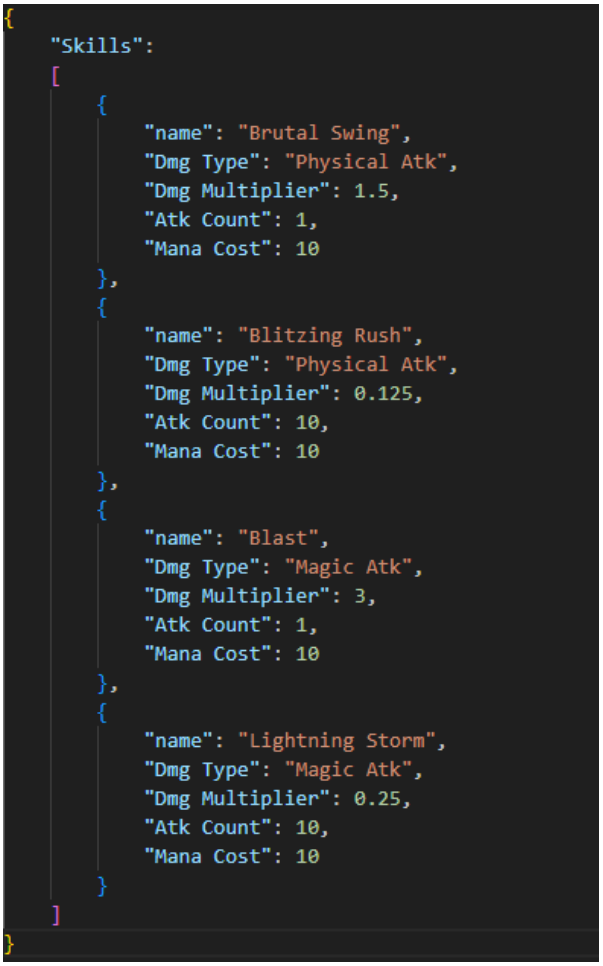
The JSON save file consists of a list with the required key “Saves” and three save objects that relate to an associated game save slots. When a save object is not associated with a player character, the key-value pairs should be initialized to the following values: Name to the empty string, and rest of the keys to “0”. The key-value pairs have the following value specifications:

- Name: Empty string if no save data, otherwise string of letters up-to 20 characters long
- Level: “0” if no save data, otherwise string of integer value between 1 and 30
- Exp: “0” if no save data, otherwise string of integer value
- Exp_needed: “0” if no save data, otherwise string of integer value
- Skill_id: “0” if no save data, otherwise string of integer value between 0 and 3
- Attribute Points: “0” if no save data, otherwise string of integer value
- Strength: “0” if no save data, otherwise string of integer value between 1 and 30
- Defense: “0” if no save data, otherwise string of integer value between 1 and 30
- Dexterity: “0” if no save data, otherwise string of integer value between 1 and 30
- Intelligence: “0” if no save data, otherwise string of integer value between 1 and 30
- Save_DateTime: “0” if no save data, otherwise string of date-time in format “[three letter day of week] [three letter month] [integer of day of month] [24hr hour]:[minutes]:[seconds] [year]”

When a new character is saved the data in the object associated with the selected save slot will be overwritten. Alternative save file that follow the above layout can be passed into the program by inputting the desired files path during command line launch as follows, “/RPGbattle [file path]”. If no save data file is passed the default file will be used. For user defined save file creation, user should use the following image’s structure and key-value pairs to avoid undefined behaviours.

```
{
    "Saves": [
        {
            "Name": "",
            "Level": "0",
            "Exp": "0",
            "Exp_needed": "0",
            "Skill_id": "0",
            "Attribute Points": "0",
            "Strength": "0",
            "Defense": "0",
            "Dexterity": "0",
            "Intelligence": "0",
            "Save_DateTime": "0"
        },
        {
            "Name": "",
            "Level": "0",
            "Exp": "0",
            "Exp_needed": "0",
            "Skill_id": "0",
            "Attribute Points": "0",
            "Strength": "0",
            "Defense": "0",
            "Dexterity": "0",
            "Intelligence": "0",
            "Save_DateTime": "0"
        },
        {
            "Name": "",
            "Level": "0",
            "Exp": "0",
            "Exp_needed": "0",
            "Skill_id": "0",
            "Attribute Points": "0",
            "Strength": "0",
            "Defense": "0",
            "Dexterity": "0",
            "Intelligence": "0",
            "Save_DateTime": "0"
        }
    ]
}
```

Skill data JSON is used to enable easy updating of skill attributes, the structure of this file is shown and expressed below.



```
{
    "Skills":
    [
        {
            "name": "Brutal Swing",
            "Dmg Type": "Physical Atk",
            "Dmg Multiplier": 1.5,
            "Atk Count": 1,
            "Mana Cost": 10
        },
        {
            "name": "Blitzing Rush",
            "Dmg Type": "Physical Atk",
            "Dmg Multiplier": 0.125,
            "Atk Count": 10,
            "Mana Cost": 10
        },
        {
            "name": "Blast",
            "Dmg Type": "Magic Atk",
            "Dmg Multiplier": 3,
            "Atk Count": 1,
            "Mana Cost": 10
        },
        {
            "name": "Lightning Storm",
            "Dmg Type": "Magic Atk",
            "Dmg Multiplier": 0.25,
            "Atk Count": 10,
            "Mana Cost": 10
        }
    ]
}
```

The skill JSON file is constructed as a list objects, where each object is associated with a different skill. Each skill object must contain these key-value pairs:

- Name: string of the skill name
- Dmg Type: only string “Physical Atk” or “Magic Atk” based on desired attack type scaling
- Dmg Multiplier: float value
- Atk Count: integer value
- Mana Cost: integer value

Restrictions/Limitations:

Due to operation inside the user's terminal, game UI formatting/layout may be affected based on terminal size during game launch and the users desired terminal. Character name can not be the empty string since checks for if slot is empty or associated with a character is determined by looking for the empty string for the name field. Additionally, due to the free access to save data a user can manipulate the save data causing game to potentially enter an undefined state. To limit potential undefined behaviour user created save files should follow specified empty save file structure seen above in File Format section. Furthermore, alterations to the skills JSON file may cause undefined behaviour.

Libraries:

Ncurses library is used for rendering game UI, the Boost property tree JSON libraries are used for reading, parsing, and writing save files. All other functionality is made using C++ standard libraries or personally made functions.

Schedule:

July 31st: Complete a basic implementation of the UI and game screen navigation.

Aug 4th: Complete an initial implementation of the Character, Enemy, and combat control methods/classes. Get combat working to a basic level.

Aug 7th: Implement save/load functionality, improve game UI, complete combat controls, and enemy AI logic.

Aug 11th: Make project video, cleanup project/prepare project for submission, submit.