SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Documentation of the code

Cohort 4 Group 1

Jaron Ho (1005011)
Ann Mary Alen (1005382)
Cheh Kang Ming (1005174)
Zhang Tianqin (1004878)

Libraries required: **random**, **time**, **copy**

Title(info). This function takes in a parameter **info**, and displays the title screen. **info** is a tuple of preset information that will be utilised throughout the game.

- Depending on the player's input, the function displays the story or starts the game.
    - If the player enters anything other than the displayed commands, the function will call itself again to give the player another prompt. This applies to all functions that ask for player input.

Start(info). This function takes in a parameter **info**, and is used to start the game.

- The function creates the player character **(player)** and the map for stage 1 (**stage**).
- It also asks the player to choose which map style they want to use.
- Then, it packs **player**, **stage** and **info** into a list (**data**) and passes it to **Menu()**.
    - **data** contains all of the necessary variables and information to run the game, and will be passed around by nearly all other functions in this program. Thus, by packing all of this information into a list, the information can be easily accessed and updated, while making the code tidier.

MapChoice(). This function takes in no parameters. It asks the player for their choice of map style and returns it.

CharacterCreation(). This function takes in no parameters and returns a list of variables that make up the player character (**player**).

- It asks the player for a name for the player character and stores it as a string (**name**).
- Then, it randomly generates the base stats and the growth rates for each stat
    - Growth rates represent the chance for a stat to increase by 1 upon a level up.
    - **boon** and **bane** refers to the stats which the player character is especially strong and weak in respectively. Thus, the **boon** stat will have a higher than normal growth rate and should grow steadily with each level up, whereas the **bane** stat will have an abnormally low growth rate and should lag behind other stats.
- The function then sets the number of the current stage and a dictionary of stat changes.
- Next, the function prints the information that the player should know.
- Finally, the function packs the above variables into the **player** list and returns it.

GenerateStat(lower, upper). This function takes in two numbers, which represent the lower and upper bounds for the **random.randint()** function, calls the **random.randint()** function twice, and returns the sum of their results.

- This is used to generate the player character's base stats on a bell curve, so that the player is more likely to get stats in the midrange and less likely to get stats in either extreme.

GenerateMap(branchLevel, chance). This function takes in two numbers and returns a list containing the map for the next stage (**stageMap**) and the player's starting coordinates (**x** and **y**).

- *Setting up the grid*
    - The function first creates a 19x19 zero matrix using nested lists. This represents a grid where each 0 is an empty space where a room can be created.
    - The room the player starts in is set in the middle of the grid.
- *Branching algorithm*
    - A nested **for** loop is used to generate connecting rooms which branch out from the starting room.

    - A check is done on the number of rooms generated to ensure the map will not be too simple and linear. Another check is done to make sure there is at least one room at the maximum branch level, as the boss room will be situated there. If the map fails either check, a new map is generated.
- *Finishing touches*
    - The function then selects a boss room out of all the rooms at the maximum branch level.
    - It also prints the stage title and story.
    - The function returns a list (**stage**) which contains the map and the player's starting coordinates.

GenerateRoom(info, matrix, counter, lstX, lstY, direction, change). This function takes in 7 parameters and determines if a room should be created in a particular space. It updates and returns **matrix**, **counter**, **lstX** and **lstY**.

- First, **info** has to be unpacked to retrieve the necessary information about the stage and the space.
- The function randomly decides if a room should be made in the space, so that further calculations can be avoided if a room will not be created.
- Depending on **direction**, the function checks if the space in question is within the boundaries of the map and that it does not have a room in it yet.
- Then, it does a similar check to the three other adjacent spaces (i.e. excluding the room it is branching out from). This is to prevent different paths from getting interconnected.
- Finally, if all of the requirements are satisfied, a room is created in the space and **matrix** is updated accordingly.
    - Using the starting room (coordinates: 9, 9) as an example, **GenerateRoom()** is called on the space above it (9, 8). After the random check, the function checks if (9, 8) is within the map and there are no rooms in (9, 8) (i.e. **matrix[8][9] = 0**). Then, it checks the other three adjacent spaces – (8, 8), (10, 8) and (9, 7) – for existing rooms in a similar fashion. Only if (9, 8), (8, 8), (10, 8) and (9, 7) are empty spaces will a room be created in (9, 8).
- If a room is created is at the maximum branch level, its coordinates are added to **lstX** and **lstY**.
- The updated **matrix**, **counter**, **lstX** and **lstY** are then returned.
    - In the case where one of these checks fail, the function returns the original **matrix**, **counter**, **lstX** and **lstY**.

CheckSpace(matrix, x, y, index, gridSize, isTriplet). This function accepts 6 parameters and checks if a space is empty. The corresponding boolean value is returned.

- As the branching algorithm only changes one coordinate at a time, **index** represents the changed value. So, the function first checks if **index** is within the bounds of the map.
    - **isTriplet** refers to the checking of the three adjacent spaces. For this check, if the room to be created is on the edge of the map, one of its adjacent spaces will be out of bounds, but no rooms will be in that space. So, this function should return **True** in this scenario.
- Then, it checks if the space at the coordinates (**x**, **y**) is empty (i.e. the value at those coordinates is 0).

DisplayMap(stageMap, x, y). This function takes in 3 parameters and returns a string that displays the map. This corresponds to the Exploration Mode Map.

- The function iterates through each space on the map, and depending on which flag the space has, different strings are added to the main string.

    - The main string, which displays the map, shows the player's position and the rooms the player has visited. The map also shows all available adjacent rooms for each visited room, even if the player did not visit them yet, to make navigation easier.
    - If a boss room is discovered, it is given a special icon on the map.

colouredmapoutput(ls1, vo, vi). This function takes in 3 parameters and directly displays the map. This corresponds to the Creative Mode Map.

- The function iterates through two global lists cs1 and ls1 and prints out the output string on screen one at a time.
    - If ls1 has a value one and value stored in cs1 at those indexes in the list corresponds to the maximum value, a red X is printed on the output screen using the colorama methods FORE and STYLE.
    - If ls1 has a value one and value stored in cs1 at those indexes in the list is lesser than the maximum value, a white X is printed on the output screen.

Menu(data). This function receives one parameter (**data**). It is used to give the player a list of possible actions they can take, then call a function depending on their input.

- The function first extracts the map and player's coordinates from **data**.
- As this function is only called after any event in a room has been triggered, it will set a flag to indicate the room is cleared and no events should be triggered if the player returns to this room later on.
    - For the boss room, a special flag is used.
- Then, the function prompts the player to select one of several actions, and call a function accordingly.
    - For the boss room, the player is given an extra option to go to the next stage.

Move(data, stageMap, x, y, room). This function takes in 5 parameters and moves the player in the direction indicated.

- Checks are done on each of the four adjacent spaces to determine if there is an available room to move to. The input message is configured accordingly.
- Depending on the player's input, the function updates the player's location.
- The map is printed to reflect the player's new location.
- Further, depending on the direction the player decides to move, five global variables, count,vo,vi,to,ti are updated which then update the values of lists cs1 and ls1 which control map b output.

PlayerStats(data, inBattle, enemies). This function takes in 3 parameters and prints the player's stats, after factoring in existing stat changes.

Quit(data, inBattle, enemies). This function takes in 3 parameters and, after a confirmation from the player, terminates the game.

EventCheck(data, room). This function takes in 2 parameters. Depending on the flag set in **room**, it decides whether to trigger an event.

- For a room that has already been visited, no events should be triggered.
- For the boss room, the boss battle is initiated.
- For all other unvisited rooms, a random event will occur.

TriggerEvent(data). This function takes in **data** and triggers a random event.

- The chance for each event is unpacked from **data** and depending a number is randomly selected. By comparing the random number to each chance value, a specific event is triggered.
- Events include: battles, healing or damaging the player, increasing or decreasing the player's stats, or showing the location of the boss room. If none of these events are triggered, nothing happens.
    - For healing, damaging, buffs and debuffs, the values are determined randomly. The range of possible values depends on the stage. A check is done afterwards to ensure the player's HP or stat changes do not go beyond the limit e.g. player should not over-heal to above the maximum value.
    - For the boss room location, the flag for seeing the boss room is set, even if the player did not find the boss room yet. This is only triggered if the player chooses to use the Exploration Mode Map.

EndStage(data). This function takes in **data** and ends the current stage.
- If the player is not on stage 5 yet, the map of the next stage is generated and **currentStage** is increased. The player's HP is fully restored and all stat changes are removed.
- If the player is on stage 5, the ending is displayed and the game ends.

Battle(data, stageData). This function takes in 2 parameters and spawns 1 to 3 enemies.
- Enemies are randomly selected from the pool of enemies specific to the stage.
- Each duplicate enemy is given a unique name.
- The enemy's details are fetched from the main enemy dictionary using a deepcopy, then added to another dictionary (**enemies**).
- The list of enemies spawned are printed for the player's information.

GiveName(enemy, enemyName, enemies, counter). This function takes in 4 parameters and returns a string with a unique enemy name.
- The function checks if **enemyName** is already in **enemies**. If so, the counter increases and is appended to **enemyName**. The function then calls itself recursively until **enemyName** is unique.

Boss(data). This function takes in **data** and spawns a boss enemy.
- Similar to **Battle()**, a boss is randomly selected from the pool of bosses specific to the stage, then the details of the boss is retrieved from the main boss dictionary, and added to a dictionary.
- Additional messages are printed to signal the player is entering a boss battle.

BattleManager(data, enemies). This function takes in 2 parameters and controls the entire battle.
- First, a dictionary (**entities**) is created to store the SPD value of the player and each enemy, as well as their action gauge (initialised as 0).
- Then, as long as there are enemies alive and the player has not died or quitted the game yet, the action gauge for each entity is increased by their SPD value.
    - Once an entity's action gauge is full, it is added to another dictionary (**actors**). These are the entities that can perform an action. Their action gauges are also reset, retaining only the excess gauge.
    - Then, as long as there are actors, the function checks which actors have the greatest amount of excess gauge (i.e. the fastest entity). This actor will then take their turn and be removed from the list of actors.

    - If the actor is the player, the player's stat changes gradually return to 0 at the end of their turn. The player's SPD is then updated to reflect any changes to the SPD stat.
    - If an actor is an enemy and has been killed, they are removed from the list of actors.
- If the player is dead, the battle and the game ends.
- If the player quits the game, they return to the title screen.
- Otherwise if there are no enemies remaining, the player wins and returns to navigating the stage.

PlayerPhase(data, enemies). This function takes in 2 parameters. Based on the input of the player, a function is called to carry out the respective player action.

- The Special option is only available if the player's Special gauge is full.

Attack(data, enemies). This function takes in 2 parameters and lets the player attack their chosen enemy.

- *Selecting an enemy*
    - The function prompts the player to select an enemy if there are multiple of them.
    - The input message is configured by assigning each enemy to a number.
    - If there is only one enemy, it is automatically selected.
- *Attacking the enemy*
    - The relevant stats are retrieved. The SPD stat of the player and the enemy are compared to see if the player is fast enough to perform a double attack.
    - For each attack, an accuracy check is performed to determine if the attack lands.
        - If the attack hits, the enemy sustains damage.
        - If the enemy's HP reaches 0, the player gains the EXP value of the enemy.
        - The player's Special gauge increases with each attack.
- The function then returns the updated **data**, **enemies** and **False** for **hasQuit**.

Special(data, enemies). This function receives 2 parameters and lets the player use a Special attack to inflict massive damage to all enemies.

- This works similarly to **Attack()**, but with increased **playerATK** and **playerHit**, and the entire process of attacking is repeated for each enemy.
- If an enemy is killed, it is temporarily added to a list, and only deleted from **enemies** after the **for** loop is completed.
- The player's Special gauge is then depleted fully.

IncreaseSpecial(value). This function accepts a number representing the player's current Special gauge, increase the Special gauge if it is not full yet, and return the updated Special gauge.

- It also prints a message when the Special gauge is fully charged.

TakeDamage(attackerATK, defenderDEF). This function accepts two numbers, and returns a number representing the calculated damage.

- The damage is set to 0 if **attackerATK** is smaller than **defenderDEF**.
- Otherwise, the damage will be the difference between **attackerATK** and **defenderDEF**.

AccuracyCheck(attackerHit, defenderAvo). This function accepts two numbers, and returns a boolean representing whether the attack lands.

- **hitRate** is calculated from the difference between **attackerHit** and **defenderAvo**.

- A random number is chosen and compared to **hitRate** to determine if the attack hits.

SpeedCheck(attackerSPD, defenderSPD). This function accepts two numbers, and returns either 1 or 2.

- The function checks whether if **attackerSPD** is 5 points or more than **defenderSPD**. If so, the attacker can attack twice in one turn.

GainEXP(data, exp, lv). This function takes in 3 parameters and returns the updated **data** after the player has gained **exp**.

- **scale** is used to adjust the EXP gained depending on the difference between the player and the enemy's levels.
- The scaled EXP value is added to the player's existing EXP.
    - If EXP exceeds 100, the player's EXP is reset to the excess EXP and they gain a level.
    - For each stat, the function randomly determines if the stat should increase by 1 point. The higher the growth rate for that particular stat, the more likely the stat will receive the +1 increase.
    - The player is then informed that they have leveled up. The stats gained and the new stats are also displayed.

CheckEnemy(data, enemies). This function takes in 2 parameters and prints the relevant information of all enemies for the player to see.

EnemyPhase(data, enemies, enemy). This function takes in 3 parameters and determines what an enemy does on its turn.

- First, the function checks what special behaviour the enemy has.
    - If the special behaviour is to wait, the function randomly decides if the enemy should wait and waste its turn.
    - If the special behaviour is to charge up an attack, the enemy will alternate between attacking and passing its turn to charge up.
    - If the special behaviour is to flee, the function checks if the enemy has low enough HP, then randomly decide if it runs away. The enemy is then deleted from **enemies**.
    - If the special behaviour is to debuff, the function randomly decides if the enemy will do so, then the penalty for each stat is randomly decided.
- If the enemy has none of the above behaviours, it will attack the player.
    - If the enemy's special behaviour is to explode, the function checks if the enemy is the only one left. If so, the enemy will inflict a single, very powerful attack. A flag is also set to indicate it has exploded.
    - Otherwise, the enemy will attack the player normally, in a similar fashion to the player's regular attacks against the enemy.
        - If the player's Special gauge is full and the incoming attack would be lethal, the player will block the attack automatically at the cost of half of the Special gauge.
    - After attacking, enemies with the charge behaviour will have their value reset so that they will charge up on their next turn instead of attacking again. An enemy which has exploded is deleted from **enemies**.
    - If the player's HP reaches 0, the game ends.

GameOver(info). This function takes in **info**. Depending on the player's input, the function either restarts the game or terminates the program.