Project report: Python DIE

Student name: Xie Qiyuan
Student ID : 2024533048

1. Provide a brief summary of the project's objectives, main methods, and final outcomes.

The project is named Spaceman Blast and is a turn-based combat game. It's totally made my Kim Xie (or so-called AbandonXIE) himself.

This project is totally made by using Python, with a special code base named Pygame. Even if the base does not include complicated calculations, I managed to create the simple game.

The project was accomplished after several version updates, and finally reached the version 0.4.0. In fact, the game was playable since the version 0.3.0 after the Battle System was totally constructed.

2. Introduce the structure of your project files and their purposes.

The project includes a main (and the only) python project file named "main.py", a folder named "assets" which contains all the images and audios used in the game (together with some developer's tools), and a "README.md" file which tells players how to play the game.

There are 4 main systems in the main project file which are the Basic Setting System, the Moving System, the Chatting System and the Battle system. The systems are separated in the main project file:

· The Basic Setting System - Managing the image display and scene rendering. Also managing the basic rules of the game, such as the numerical system, the key detection and the covering. The main project would crash without it. (Separated in lines 1~161, 245~285, 301~307, 598, 739, 804, 834~859, 969~1102, 1190~1210).

· The Moving System - Managing the movement of the main character when not in a battle, and detection of interactive and non-interactive objects. (Separated in lines 861~967, 1171~1185)

· The Chatting System - Managing the chatting dialog and its presentation, as well as the procedure of conversations of the main character when not in a battle. (Separated in lines 163~243, 308~377, 599~685, 740~790, 805~812, 1104~1130).

· The Battle System - Managing everything that may happen in a battle. (Separated in lines 287~299, 378~596, 687~737, 792~802, 814~832, 1132~1172)

(The designer once tried to add a Saving System to save progress when quitting, but he canceled it at last because the gameplay can be short so it's useless. Another reason to cancel it is that it can be really troublesome.)

In the "assets" folder, there are 46 images, one font file and 8 music files, which are called in the main project. Besides, there are also a list of numbers in the game for players to get to know the game better, and an instructions of cheat codes which can be used in the game.

The "README.md" file, as is known to everyone, includes basic details and instructions to play the game.

3. Project Workflow

The game's four main systems are constructed one by one. The following system would not be started to construct before the current is fixed with no errors. The following workflows are listed by time sequence of constructing.

However, as the systems are sharing the same instructions (for example, both the Battle System and the Chatting System uses Z key to proceed the sentences), the functions of the same key are put together. This leads to the separation of codes in the same system.

· Lines 1~3: Importing vital bases for the game.
· Lines 4~17: Importing LLM system.
· Lines 19~31: Classifying the type of players and the wall.
· Lines 33~35: Displaying the window.
· Lines 37~117: Loading images which may be used in the game.
· Lines 119~161: Initializing game numbers.
· Lines 251~254: Playing the music of covering.

· Line 256: Keeping the project running.
· Lines 257~285: Refreshing data according to the player's current level.
· Lines 301~304: Detecting quit conditions.
· Lines 969~1102: Displaying the worlds, the player, and other objects.
· Lines 1190~1205: Displaying level and coins so that they're visible.
· Lines 1207~1210: Refreshing the screen.

· Line 846: Detecting keys get pressed.
· Lines 848~859: When in cover, press Z to begin.

· Lines 861~883: Use WASD or arrow keys to move.
· Lines 885~967: Checking whether interactive. If so, checking which object is interacting.
· Lines 1163~1171: If interactive, displaying instructions.

· Lines 163~243: Listing dialog sentences.
· Lines 308~377, 599~685, 740~790, 805~812: Proceeding conversations.
· Lines 1104~1130: Displaying dialog sentences.
· Lines 1183~1188: Displaying concerned images when in a dialog.

· Lines 287~299: Displaying the movements of the characters when in a battle.
· Lines 378~596, 687~737, 792~802, 814~832: Operations during the battle.
· Lines 1132~1171: Displaying battle instructions.

The core of the game is its turn-based combat system. In order to make it easier to realize, I gave up the original design of one-to-many battle system and complicated setting of the action values. Instead, I designed a one-to-one battle system only with some basic settings, such at the HP, TP, attack and defence of both sides. Meanwhile, one can only make one movement per turn. In order to make the game easier, the player always acts first.

In order to make designing easier, I separated each turn into 9 parts, and marked them into a variable named “row”. Every time the player press Z to continue, or press X, C or V to use techniques, “row” changes, so will the instructions in the dialog bar.

When the battle begins, “row”will be -1 and the battle instruction will be “The enemy approached you!” Then after pressing Z “row”will be 1 and you must choose a technique. If your TP is not enough and you chose the end technique, “row”will be 0 and the instructor will mention that there’s not enough TP. If you choose attack, “row”will be 2 and you will move to the enemy. As soon as you touched the enemy, “row”will be 3 and you’ll go backwards, and when you reach your original place “row”will be 4 and you’ll no longer move. If you changed form, “row”will be 4 directly and you’ll not move to the enemy. After pressing Z, “row”will be 5 and the program will check whether the enemy was knocked down. If so, “row”will be 10 and the battle ends. If not, it’ll be the enemy’s turn. If the enemy attacks, “row”will be 6 and then it moves to you. As soon as it touches you “row”will be 7 and it goes back to it’s original place, and then “row”will be 8. If it recovers HP, “row”will be 8 directly and the enemy won’t move. After you press Z again, “row”will be 9 and the system will check whether you were knocked down. If so, “row”will be -2 and you will lose the battle. If not, “row”will be 1 and the next turn will begin. The battle wouldn’t end and “row”will be a number between 1 and 9 until either side was knocked down.

In the 0.3.2 playable version I fixed some numbers so that the LV6 (player’s maximum level) player can only defeat the enemy of LV8 (common enemy’s maximum level) if they know how to make the biggest attack. At that time the enemy has only two techniques:

the common attack and the end technique. However, I was commanded to use LLM in the battle, so I added a technique to the enemy which can recover its HP. Then I applied LLM into the enemy. At first, it would mostly recover HP and seldom attacks, even after it becomes weaker and gets more hurt than recover in a turn. However, after some training, it attacks more even though it's still prospective. At first, when the LV6 player defeat the enemy of LV8, the player would retain more than 200 HP (LV6 player has a maximum of 400 HP), but after training, the AI model became stronger. Once the player only retained 6 HP when he won.

Another important thing is how to build a large map by fixing the camera to the main character. The project has three maps. I applied it to the grass world by calculating the relative positions and relative distances to display every item in the theme (including the background, the wall and the enemy) at the right place. For the other two maps, I used traditional cameras which are immovable.

4. Learning Reflections

This project is the first pygame I finished independently. In fact, I learned some basic pygame skills last summer and made a game step by step following the tutorial. If that's the introduction for me to the python programming, then this project can be a step forward. From this project, I got to know more about computer programming, and got more interested in the whole subject. From now on I will proceed my progress and keep my interest. Maybe I'll try something more challenging, such as other forms of pygames or other programming languages. But most importantly, I know what I learned is far more beyond programming things like a successful commercial game. So I'll keep learning, just as Steve Jobs said, "Stay hungry, stay foolish."

5. Results and Analysis

In my first week of project, I was confused because I couldn't design a function to detect collisions between the player and the wall. That was because my wall was fixed and was a whole object, so players would only move in its hollowed-out space. As a result, the collision would always happen. So I used a "hard method" which mechanically detect collisions to each piece of wall one by one. Even though the method was stupid, it worked.

In my original settings, I designed sound effects when the player attacks the enemy or when the enemy attacks the player, but I finally rejected it. For one thing, the sound effect might conflict with the BGM; For another, I really don't know which sound effect fits the most.

I've told that I trained the AI model to battle, but it wasn't as ideal as I thought. Maybe it's because I trained it less. I just applied the model a week before I finished the project But I fount it difficult to make it learn from failure and do the most powerful choice: I

thought it would never recover HP after it was weakened and the HP it recovered was lower than my attack. But it may think that it must recover HP after being attacked. Only when I stopped attack and switch form in a certain turn would it give a common attack in the same turn. Maybe after times of training, it will be smarter and maybe it will be bound to win when it reaches LV8. (Actually I hadn't calculated the best technique yet!)