Report of Final Project of POA

Jiaxuan Liu and Jiangnan Huang

May 9, 2020

Instructor: Patrick Amar

Abstract

The goal of this project is to learn to write a program from informal specifications by making a *Kill'em ALL* type game. Considering that the programs will be reused by others later, we should code them clearly by commenting them judiciously, we should also train to encapsulate the data and methods to obtain robust programs.

1 Introduction

This is a simple *Kill'em ALL* type game, with a 3D display simplified supported by OpenGL. In this game, there are two kinds of characters: Hunter and Guardian. Players play the role of hunters to gain treasure by walking through the labyrinth. Players can control the hunter's movement through the keyboard, click the mouse to shoot. Hunters can get blood packs around the box, which increases the hp value. Guardians are computer players. The guardian has four different states to better protect the treasure. The guardian can properly avoid the hunter's attack, and can find the direction of the fireball that passed by him but not hit him. Both the Hunter and the Guardian have an initial hit point (HP), which is reduced every time they are attacked by their opponents. Their shooting accuracy will decrease to varying degrees as the hp value decreases. And their attack power will increase as the distance between them become shorter.

In this report, we will introduce how these functions are implemented in three parts: labyrinth, guardian and hunter.

We used the original version given by the teacher.

2 Labyrinth

The Labyrinth class is responsible for building a labyrinth based on the input *.txt* file. The input file is read line by line, the data is processed and stored in a two-dimensional array. There are 4 two-dimensional arrays in the Labyrinth class:

- **char ** _data** : It indicates if the grid is free or occupied.

  Initialize its value to 1 (occupied). When processing input, when the input is a space, it indicates that the grid is free, so set to 0, other conditions remain unchanged.

- **int ** _distance_to_treasure:** It indicates the distance of grid to treasure.

  The value of this array is initialized to INFINI, the grid where the treasure is located is initialized to 0, and the grid in eight directions around it is 1. After the input stream is processed, the distance to treasure begin calculating. Traverse the entire array, if the grid is free, and its distance has been changed(<INFINI), then the surrounding grid, which is not assigned, is assigned the correct distance. Repeat the operation until all grids have been processed. (Breadth First Algorithm)

- **int ** _around_boxes:** It indicates if the hunter is around the boxes (where the medical kits exist).

  The grid of eight directions around the box are 1 and the others are 0

- **int ** _over_teleportation:** It indicates the number of teleportation. When the character read is a number, it means there is a portal. (See details later).

Wall construction:

When the string '+' is read, it means that this is the start or end position of the wall. Because the string is read from left to right, from top to bottom according to the file. So, if we judge that the right side of '+' is '-' or '+', it means that the current point is the starting point, and we look to the right for the end of the wall. If not, the current point is the end of the wall (Similarly, downward), if we meet an 'a' or 'b' when reading the wall, paste the corresponding texture here along the direction of the wall.

3 Guardian

This section mainly introduces the implemented functions of guardian.

3.1 State

In the Guardian Class, the ***num_of_mode*** variable controls the state of the guard.

- State 0: Patrol. The default state of the guardian when the game starts, the guardians walk randomly around the Labyrinth.

- State 1: Defense. When the threat level is greater than 6, the guardian begins to approach the treasure (According to the array *int ** _distance_to_treasure*)

- State 2: Emergency defense. When the threat level is greater than 8, the guardian approaches the treasure at a faster rate

- State 3: Attack. When the guardian sees the hunter, he will attack the hunter. The guardian heads towards the hunter and fires at intervals. When the hunter is not visible any more, the guard will no longer be in state 3, but will return to the previous state.

The state of guardian decided by the $defense$ value which is calculated based on the guardian's own distance to treasure $dist_g$, the distance from hunter to treasure $dist_h$ and the number of dead guardians $N_{g_{dead}}$.

$$defense = \frac{dist_{max} - dist_g}{dist_{max}} * 3 + \frac{dist_{max} - dist_h}{dist_{max}} * 8 + N_{g_{dead}} * 0.5$$

where $dist_{max}$ is the max value existed in the matrix **int ** _distance_to_treasure**.

3.2 Other implemented functions

- Life condition:

  Guardians' life have been changed to Hit Point (HP) value instead of one life. This is more in line with the actual situation

- Accuracy:

  The accuracy of the guard's shooting is related to its physical health. The worse the health, the worse the accuracy. Since hunters are controlled by humans, shooting is far less perfect than computer software. So in the file *Guardian.cc* we increased the shaking angle to reduce its accuracy.

- Attack power

  The guard's attack power increases as the distance between the two decreases.

- Dodge fireball:

  When the fireball approaches the guardian, he will randomly dodge in a certain direction in a small step.

- Notice an attack:

  If a fireball passed by the guardian, he can notice it. Then he will patrol in the direction of the fireball even he can't see the hunter.

4 Hunter

This section mainly introduces the implemented functions of hunter.

4.1 Teleportation:

In the .txt file for building the map, the same pair of numbers is used to represent a group of portals. When the hunter walks over the grid where the portal is located, there will be a text prompt in the upper left corner. The hunter needs to keep moving on the grid in place for a period of time, and then he can instantly move to the other portal. The following details the implementation method:

1. In the labyrinth.h file, we constructed a structure: **Teleportation**, which has five members: *Id*, *pos_x1*, *pos_y1*, *pos_x2*, *pos_y2.*

- *ID*: is the number read from the *.txt* file. Here is to distinguish between different pairs of portals, because the portals exist in pairs, otherwise you will not know where the destination is.

- *pos_x1, pos_y1*: position coordinates of a portal.

- *pos_x2, pos_y2*: position coordinates of another portal.

2. When reading a number, check whether the portal of the id already exists, if it does not exist, then "new" a Teleportation, and assign values to *pos_x1, pos_y1.* If it does not exist, find the teleportation with the same id and assign values to *pos_x2, pos_y2.*

3. When the hunter walks over the grid of the portal, we use *int _wait_to_transmission* to calculate the waiting transmission time. The purpose of setting this time is to prevent the hunter from being teleported as soon as he walks on the grid, which will be very confusing. When *_wait_t_transmision* equals 0, directly change the position of the hunter to complete the teleportation.

4.2 Other completed functions:

- Life condition:

  Hunter's life have been changed to Hit Point value instead of one life. This is more in line with the actual situation.

- Accuracy:

  The accuracy of the hunter's shooting also decreases as the health level decreases.

- Attack power:

  The guardian's attack power on the hunter increases as the distance between them decreases. But the two have different degrees of decrease. Hunter's attack power is stronger.

- Resist attacks:

  When the hunter is attacked by the guardian, the hunter can hide behind the box. In this way the guard can no longer see the hunter and will stop attacking.

- Recover HP:

  There exist 4 medical kit around each box, when the hunter walks around the box (according to int ** _around_boxes), he will eat the medical kit and recover 20HP (The HP value can not exceed 100). The medical kit are disposable, and the guardian's HP value cannot be increased.

5 Conclusion

During this project, we've met many problems and difficulties. For example how to construct the labyrinth by reading 'map.txt' files, how to encapsulate the functions and data, how to make our program more legible, etc. Finally, we solved these problems with our knowledge and efforts, we implemented successfully the main functions of the game and some more extensions. By implementing this game, we've improved our programming skills with C++ and our ability to collaborate. Also, we've got a better understanding of Object-Oriented Programming.