El Chido Proyecto

a.k.a. "The Cool Project"

Team Information

Team Destructors Group Members: Ivan(Captain), Travis, Sean, Zach, Michael, Andy

---

Table of Contents

Introduction

For this final project, our task was to develop a small yet interesting turn-based game between our group of six people. The game takes place inside a building represented by a 9 x 9 grid. The player takes the role of a spy infiltrating the building in order to retrieve a briefcase containing top-secret documents, located in one of the nine rooms distributed throughout the building. This building is also pitch black and has six ninja-assassins equipped with ultra-laser katanas patrolling the area.

Upon starting the game, the player is given three lives and enters the building through the lower left end of the grid. The player is equipped with a gun containing only one round of ammunition , and night vision goggles that can see two spaces ahead of the player. The player's turn is divided into two parts: look and move / shoot. During the first part of their turn, the player(user) is prompted to "look" in any direction. Following that, the player is prompted to choose between shooting or moving.

Following the player's complete turn, enemies go through a two step process. First, the enemies check all adjacent squares to check if the player is nearby. If the player is in an adjacent square, the ninja-assassin attacks with their ultra-laser katana, taking a life away from the spy and resetting the player into their start-game position on the grid. If the player is not in an adjacent square, the enemies randomly move one square in any valid direction.

The game ends once the player enters the correct room with the briefcase inside of it. Rooms can only be entered through a door on the north-side of the room.

There are also multiple power-ups located throughout the building, which modify the attributes of the player. These power ups include: additional bullet, invincibility, and radar.

Additional bullet equips the player with another bullet. If the player already has a bullet, then the power up does nothing. Invincibility gives the player super-strength for five turns, making stabs from a ninja-assassin's ultra laser katana ineffective. Lastly, radar displays the location of the briefcase.

Approach

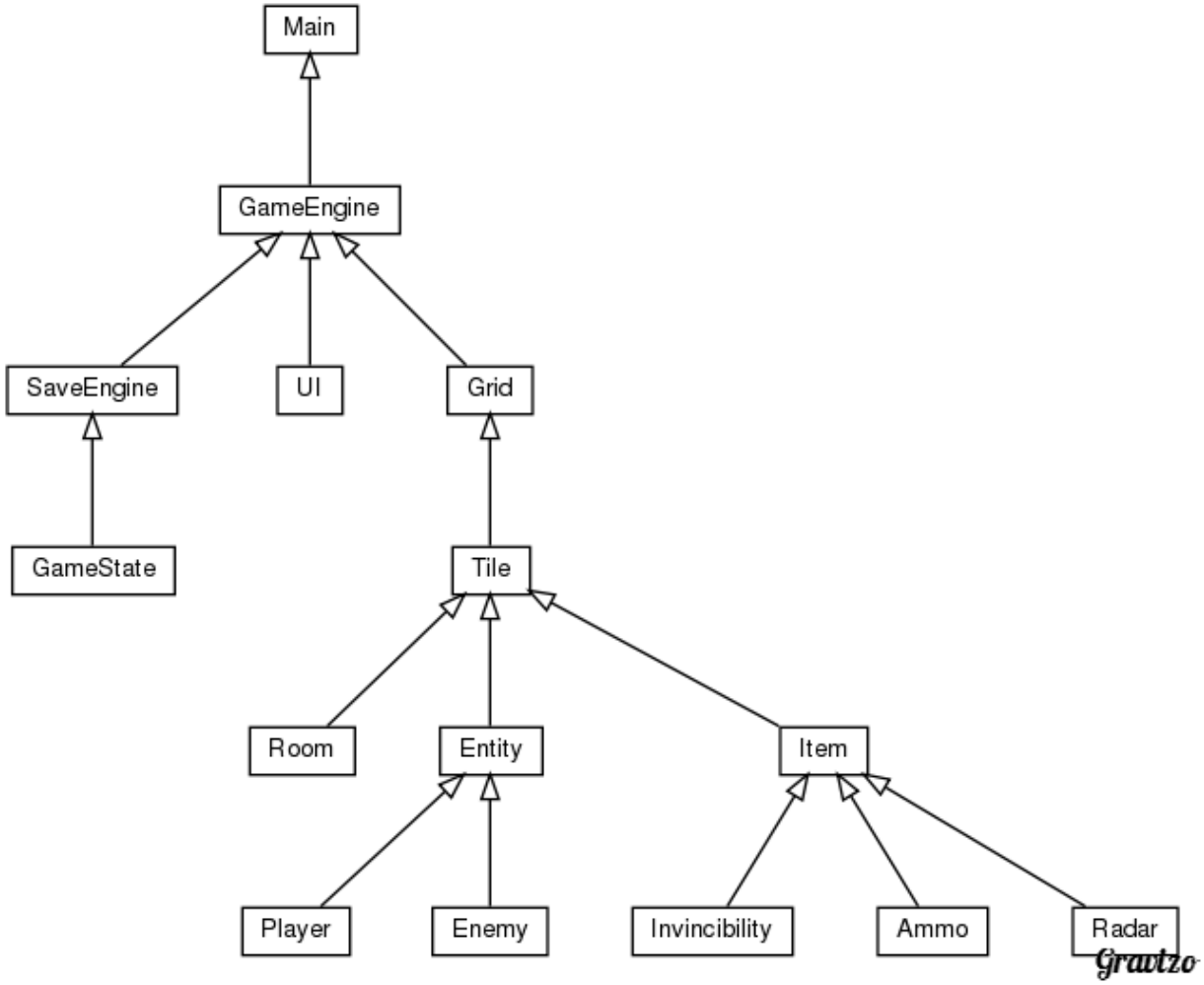
Our approach for this project was to first develop a general idea of about what classes and fields so that each individual can work independently on separate parts of the project. We took this approach since we figured that since we all have varying schedules, we should maximize the time we have to work on the project, both as a group and independently. This approach worked up until milestone 1, but not for milestone 2.

Upon inception of our group, we decided to use Slack to communicate effectively via a text channel that can be created with the app. We also chose to use Github as our version control management tool. Along with Github's issue feature, we also used a browser-based app called Waffle.io to list and assign tasks to different members of the team. Although in theory, communication seems very simple and straightforward; we discovered it requires consistent effort to communicate and understand our varying approaches on the project, which was vital to success. We later also used Discord to voice chat while collaborating on the project. For most of us, we decided to use Intellij IDEA as our IDE since it included a user-friendly, graphical git merging tool that was utilized innumerable times.

For our project, we initially wanted a game board (Grid) that consisted of a 2D array of Objects but we decided that design wouldn't allow items and entities to be contained within the same tile of the board without nesting one within the other. To make the design more intuitive, we our board was made into a 2D array of Tiles in which entities and items can be placed on/in.

Design

UML Diagram of Classes



Discussion of Implementation

The board is an array of tiles that can contain one of three possible items. The tiles may contain a player object, an enemy object, or an item object. A tile may contain both an enemy and an item, but a tile cannot contain any other combination of objects. We created an enum called direction which helps the programmer refer to each of the four directions for looking or moving. We also implemented a class we called Math which does the conversion from cartesian coordinates (X & Y coordinates) to the positions in the array (the board). This class helped make the positions more intuitive for the programmer during coding.

First, when the game is started, rooms are first to be set in their stipulated positions in the tile array. Next, the player is set at the origin of the map, the bottom left corner. The remaining objects (enemies and items) are all randomly positioned. This is done by generating two random numbers between 0-9. One randomly generated number will correspond to the row of the room, while the other randomly generated number will correspond to the column of the room. Before either an enemy or an item is placed, a check is ran to ensure that the tile they are being placed on does not contain any objects, namely, a room or another enemy.

After all rooms, enemies, and items are placed, the game loop begins. The loop continues to run the following functions: playerturn, checkbriefcase and enemyturn. The playerturn consists of a look action and a choice to either move or attack. After the playerturn, there is a boolean check called checkbriefcase that sees if the player has retrieved the briefcase from the room. If they have, a win prompt is displayed and the program is terminated. If no briefcase has been retrieved, the game proceeds with the enemyturn. The enemyturn is called for all of the remaining enemies.

Each enemyturn call has two functions, first the enemy attacks if the player is positioned at an adjacent tile to the enemy's current position. The player can only take one hit before dying and respawning back at the bottom left corner. After the enemy attack, that respective enemy will move to a random adjacent tile. (Except in the case that random direction is a wall, then nothing happens.)

The game loop will continue until either the player runs out of their 3 initial lives or wins the game by retrieving the briefcase. The user may save a game during any look turn phase and load a game when the game first begins.

Conclusion/Suggestions

With a total of 208 commits our team was able to successfully develop a small, text-based game. Our game begins with a prompt asking the user whether they want to start

a new game or load a saved game. If new game is chosen the player starts in the corner of a 9x9 grid of squares. If load game is chosen, then the game will load the game with the game state that was saved. In order to win, the player needs to reach one of nine random rooms that contains a briefcase. The nine rooms are spread evenly along the grid. The grid also contains six ninja assassin enemies as well as three different power-ups (ammo, invincibility, and radar).

The gameplay is in turns, with the player moving first, and then the enemies. The player's turn has two parts. On the first part, they are able to look in any direction and see two spaces ahead (the grid is pitch black so that's all they can see). They also have the option to save their game when choosing their direction. On the second part, they are able to choose move or shoot. If they choose move, they will be given the option to move to an adjacent space inside the grid. If they choose shoot, they will be able to then choose a direction to fire a bullet. Any enemy in the direction of the fired bullet will die. The player has only one bullet.

On the enemies turn, all remaining enemies start their turn by checking if the player is in an adjacent square to them. If they are they will mortally stab the player, killing them (the player starts with three lives and is sent to the player spawn after each death). If the player is not next to an enemy, then the enemies will then move in a random direction. It is then the player's turn again.

The three items are also randomly generated on empty spaces in the grid (though enemies are able to walk over/cover them as they move around). The player uses them as soon as they walk over them. The ammo is an additional bullet that sets the player's ammo to 1. It has no effect if the player has not already used their bullet. The invincibility power-up makes the player invulnerable to being killed for five turns. The radar power-up shows the location of the room with the briefcase.

The player wins the game by successfully finding the room with the briefcase by entering it from the north side. The eight empty rooms are not enter-able by any entity in the game. The player loses if they lose all three of their lives.

There were several things that we could have done differently to make the project go more smoothly. The main issue we had was not being organized enough. In addition to a Github repository, we set up a project on waffle.io so that we could have an organized way to assign tasks to our team members. At first we used the waffle.io, but most of our team didn't know how to use it that well and didn't think it was very helpful. So we stopped using it and decided to just divide up what we would each do on our own. The program isn't that big, however, so it was difficult to divide up tasks between all of our members without being very specific and planning ahead exactly what would be in each

class. We didn't plan well enough in the beginning, and this led our team members to writing code for methods and fields that had already been made by other team members. In addition, our team members each took different approach to developing the game. This left our classes with much code that was either a duplicate of another function or would not be fully used in the final product. It wasn't until the second team meeting we had that we were able to all decide on the approach we would take. After that we became better at communicating exactly who would do what once we were all taking the same approach. Although, at that point our code was fairly disorganized and caused the debugging process to be challenging.

Another thing we could have done to make the project easier would be to have written the Javadoc as we wrote the program. The code was partially documented throughout the time we worked on it, but not fully. The Javadoc was one of the last things we worked on and it would have been helpful to be able to have that documentation as we worked on writing the program. It could have potentially prevented much of the duplicate functions from being written as well as making the debugging process easier.

Everyone on this team put their hardest work and much of their time into this project. Even though we had some organizational trouble in the beginning, that was mostly due to our whole team's willingness to contribute as much as they could to the project. Most of the work we got done was during team meetings to which everyone showed up and stayed for as long as they could. We finished the program with 208 total commits which shows how dedicated we were to working as a team and all doing our fair share of the work.