Final Project Report

Team “ГРЕЧКА”

Victor Ruiz, Ian Garrett, Khanh Nguyen, Alexandra Klimenko

Introduction

In this project we designed a game using Object-Oriented techniques in Java, and implemented the Graphical User Interface using Java’s Swing library. The purpose of the game is to find the briefcase hidden in one of the rooms in a pitch-black building. The player needs to make sure to not get stabbed by ninjas, which are randomly located on the map. The visibility of the player is only 2 squares horizontally and 2 squares vertically, however for our version we added 1 diagonal square because sometimes if the ninja and the player moved towards each other, the player would get stabbed without seeing the ninja.

The background story of our game is an imaginary trip of Professor Rodriguez and his wife going to a Victoria's Secret store. As it gets boring, professor decides to find “Grechka” [briefcase] which is hidden in one of the rooms that are protected by Victoria Secret Angels [ninjas]. Grechka is the russian name for “buckwheat” , and we chose it because it was our team name.

Development Approach

Efficient methods for communication was a key element in getting this project started. We utilized Google Docs so we could begin sharing ideas and create an outline for our project. We used GitHub to share code, which made it much easier to collaborate.

We created 14 classes, and decided that all the classes besides GUI, UI, Main, and Engine will extend a Square class. This made it easier to populate the map, which is a two-dimensional array of Square objects. The Invincibility, Radar, and Bullet classes extend the PowerUp class, which extends the Square class. The Engine class performs all game functions such as moving the spy or ninjas. The UI or GUI contain the logic of the game and contain all methods that print to the screen and take data from the user. The Main class only calls the methods to get the UI or GUI started. All the classes that have elements of the game stored were implemented as Serializable, and generate a serial ID, such as: serialVersionUID = -9053250392397413615L; in order to identify if this is the corresponding class to a serialized object.

As we expected many bugs with this new project, we made an effort to keep track of them as soon as they appeared. We kept a separate Google Doc where we would describe the bug in as much detail as possible.

Implementation Challenges

The more significant challenges appeared after we implemented the movement of the players and ninjas.

Some examples of our most difficult bugs and how we fixed them:

1) One of the biggest issues we ran into at first was the ninjas moving over the powerups and picking them up in the process. The way we fixed this was by recreating the power-ups in their previous locations after the ninja had left.
2) The visibility from the spy's *previous* locations would remain visible. We stored all the previous locations of visible squares in an array and made sure we hid them before next move.
3) We displayed many error messages depending if the player didn't follow instructions correctly or if there is a wall or the room if the player is not on the entrance side. Establishing how we would prompt the user and the options we would give them was determined through many iterations.
4) Though it wasn't exactly a "bug" in the code, we constantly ran into issues when using git through the command line. All of the problems we faced can be pointed to the simple fact that most of us had never used github before. Getting used to the website interface and the many different commands took some time, but we began to get the hang of it.

Testing The Code

One of the really important parts in developing our project was testing the game for errors. The fact that we had all of the game data stored on GitHub, as previously stated, made it very easy for all of us to individually maintain copies of the game that we could test and alter without changing the main copy of the game's code. This made it very easy for us to test the game's code for bugs whenever we had a few spare minutes.

Many of the errors that we would have encountered in early testing we were actually able to plan ahead for and prevent in advance. Despite our original planning, we did

everything we could to break the game throughout our testing. This included things such as firing a bullet into the wall, moving the spy into the wall, moving the spy into one of the rooms from the wrong direction, shooting at the power-ups, and running into the ninjas while invincible.

Once we found an error, we would attempt to recreate this error multiple times, which would help us to identify exactly what was causing the errors. This habit of constantly testing the code after each feature was added helped to limit the amount of issues we faced when we did our testing.

Error Log:

- Had problems printing the rooms in the multi-dimensional array. (Fixed by assigning rooms to certain locations when the map is created)
- Ninja's were not being placed randomly. (Fixed with a random int generator and an isOccupied method.)
- Visibility of the spy would not update as the spy moved. (Fixed by recalling spy visibility after each turn.)
- Ninjas destroy power-ups when they move over them. (Fixed by remembering the locations and recreating them after the spy has left.)
- Player moving outside of the array create ArrayOutOfBounds Exceptions. (Fixed by including a switch case statement around the player and ninja movement that catches if they move outside of the array)
- Once the game is started, there is no option to quit or save the game. (Fixed by adding a second options menu that is accessible from any point during gameplay.)
- Once debug mode is enabled, it cannot be disabled without restarting the game. (Fixed by adding a toggle to the debug class)
- In the GUI, visibility is created from where the player just moved from. (Fixed by assigning visibility before the map is printed.)

Conclusion & Future Improvements

In conclusion, this project has helped us to learn how to effectively write an object oriented program on a larger scale than would have been possible on our own in the time frame we had. This project helped us to better understand inheritance, polymorphism, overloading, and object oriented programming. More specifically, during our time working on this project, we learned how to write code to fulfill different needs that we had. Here are a few of the things we learned how to do: creating a multi-dimensional array of objects that we could create and call after the array was created, developing an AI that would move the enemies in a way that made the game challenging, and creating a GUI that would display our game in a visually friendly way. It was a challenging project, however as all the team members worked together and communicated effectively, it was much easier than it normally would have been.

Even though we made an amazing project, there is always room for improvement. The biggest improvement we would make if we had more time to write this program is to simplify our code. Many of our methods, especially our movement methods that contain the logic of the game, are pretty long and should be simplified where possible. One of the things we could have improved aesthetically would have been to implement sound to the game, for example when the player moves, gets stabbed, wins the game, or even background music. Also, we could allow the player to have even more bullets and give the possibility to win the game if all the ninjas are killed.