Members: Rachel Chiang (Captain), Kartik Soni, Dominique Acuna, Seongmin Kim, Danyel Gil

Final Report

Introduction to programming and problem solving, a computer science class offered at Cal Poly Pomona, introduces students to many different concepts used in computer programming. Specifically, the class explains object oriented programming. Object oriented programming is defined as a programming methodology that is based on the use of objects as the unit of design and supports the following three paradigms: inheritance, encapsulation, and polymorphism. Taking into account what was learned from previous classes and the use of these three main concepts, the ideas of overloading, overriding, initializing, information hiding, exception handling, and recursion were used to program an assigned group project. To assess if the group understands object-oriented programming, they were to make a game involving a nine-by-nine grid with nine equally distributed rooms, six enemies (ninjas), and three items. The objective of the game is to control a spy who is equipped with a gun with only one bullet and maneuver around the field to find the room which contains the briefcase.

The group first planned out a rough outline of classes and their fields and methods, similar to how the previous assignment’s planning for the gun was. Among Seongmin, Dominique, and Rachel, the overlapping classes drawn out were the ActiveAgent, Spy (the player), Ninja (the enemies), UserInterface, and GameEngine classes, with some disparity over the rooms and every class’s respective attributes and behaviors. Kartik and Danyel then assembled the skeleton code. The ActiveAgent, Spy, and Ninja classes were somewhat straightforward, considering the previous assignment similarly entailed the ability to interact with one another by killing each other. Their movements, their positions, and the map ended up being completely different from the original skeleton code. In class after getting the first milestone checked, Rachel put together some simple code: a class for the map; temporary displays (since these would be moved to the right classes later); and the positions (for the rooms, randomly placed items, semi-randomly placed enemies, and the player). There were some errors, particularly in enemies and items overlapping one another, since it was early in development, but it was a start. Following this, the rest of the code fell together relatively easily.

The first meeting was held on the same Wednesday as the first milestone check, with Kartik, Rachel, and Dominique, since the other two were busy. During this time, Rachel fixed the grid class, moving the player’s spawn to the Player class, connecting the Map class to the Player class via the GameEngine, and creating the appropriate position fields and methods. Kartik created methods which received the user’s input as the direction and adjusting the positions accordingly. The three fixed the grid some more, moving the Enemy to its appropriate class and made a new boolean array, which held true or false depending on whether there is an entity (the player, an enemy, an item, or a room) in the position, which would be used to eliminate the issue in which the entities would spawn on top of each other and provided something like “collision” that would prevent active agents from standing on top of each other, items, or rooms. The other array held integer values, which identified whether a space was empty or contained one of the various objects and affected the display in the UserInterface. Everyone except Dominique met on Friday of the same week, during which the captain assigned roles for everyone, giving everyone five days to work on it.

However, none of the group members sent their progress, so Rachel took over some of the roles. Rachel put in methods which would randomly move the enemies, which was Kartik’s part, but he did actually do it, but he forgot to send it to everyone. It turned out alright though, since the two had similar approaches for it. Rachel improved the general movements of the active agents, specifically, by adding the ability to reset positions in the map class to “empty” and implementing the function of the collision boolean array “isTaken” in the Map class, and added a toggle for debug mode. Simply to experiment with the enum, Rachel tried to make the items and rooms both enum types but ended up removing them either because her lack of experience with and knowledge of it made it seem overcomplicated or because it seemed unnecessary, which was the case for the rooms. Rachel put in the ability for enemies to stab and for the player to die and created items, though only the invincibility worked since the briefcase and player’s attack were not added in yet.

At the second milestone check, Professor Rodríguez informed the group that the rooms were in the wrong spaces and the items needed to be identifiable. Moving the rooms to the right positions was an easy fix, but the items were tedious; perhaps there is a simpler way to do it. Although it was past the captain’s personal five-day assignment interval, Seongmin sent in the code for his role: the briefcase class, room interaction, and the ability to win the game. Rachel added the ability for the player to attack. Thus, the items were now all functional. The captain requested for volunteers to make the Look action of the player and for the try/catch. Kartik decided to take up the Look action after giving 24 hours for another person to pick up the opportunity to contribute. The directions on the website were somewhat vague, so he simply made the Player “look” in the row and column, which would return a message depending on whether the player’s row or column was clear or not. Since no one else volunteered to put in the try/catch, Rachel did it; she adjusted the UserInterface by adding while-loops and using the try/catch blocks. However, she ended up getting an infinite loop with the try/catch blocks, but this was fixed by resetting the scanner’s buffer.

Simultaneously, Rachel began working on the pursuit mode for enemies. This first version simply relocated the enemy to the player’s previous location, which was problematic for several reasons: it failed to implement the line-of-sight mechanic, the player could never escape the pursuit mode, and only one enemy could follow. Kartik suggested that the pursuit mode mechanic should work similarly to the enemy’s stab, in which the adjacent squares are checked and a boolean is switched to true if the player is nearby. For the second version of the pursuit mode, Rachel changed it as such, but made it so that the enemy saves the direction it had just moved in and checks that single direction, but the line of sight was too easy to break this way, so she changed it so that instead the enemy will check all four directions, two spaces ahead, for either the player’s current coordinate or the player’s previous coordinate, similar to a breadcrumb trail. As a result, the enemy had a more challenging following mechanic, and the line of sight could only be broken when too many enemies pursued at the same time or when the player cut the LoS using a room.

At the third milestone check, Professor Rodríguez found issues in the vision of the player and in the look action. The group met the same day (on Wednesday); the attendants were Danyel, Dominique, Kartik, and Rachel. We fixed the two issues the professor had with our game, which required simple adjustments, and the change for the Look action actually made its code much shorter. The game was just about finished; the group only needed to implement saving and loading and make the GUI, so we discussed the GUI and the plan for Friday. Rachel drafted a role sheet for testing and polishing, documentation, and the presentation, and Dominique started working on the PowerPoint. On Friday, Dominique, Kartik, and Rachel met and played with saving and discussed the GUI. The saving and loading was fairly straightforward as well. Dominique, Kartik, and Rachel set it up and simply saved the objects in the GameEngine in a specific order into a file and then loaded the objects back into the game in the same order.

The group encountered a handful of bugs during testing; all were fixed. Danyel found an issue with the former shoot mechanic: sometimes, the bullet would not hit any enemies in the inputted direction. As it turns out, the logic itself was flawed, so the group rewrote that piece completely, and now the bullet will hit the closest enemy to the player and will not penetrate. Kartik also found a bug with the rooms—specifically, the room with the briefcase. Originally, Seongmin did not take into consideration what would occur when the player attempted to enter the room from the southern, eastern, and western sides, but the fix for this was short and simple. The last issue was the ability for the player to be killed multiple times in one turn, which generally did not happen unless the user reached eighty or so turns, since the chances of the enemy to randomly find itself in the player’s original position at the same time that the player dies is rather slim. The fix required a boolean, ensuring that the enemies who have not yet checked for the player in its vicinity would not be stab. Also, to prevent the enemy who was currently on the player’s original position at the time of death from disappearing when the player respawned, the enemy had to be moved to another square.

To conclude, object oriented programming was learned this spring quarter of 2015 at Cal Poly Pomona and used to program this programming project in CS 141. Major concepts that are included in Object oriented programming are inheritance, encapsulation, and polymorphism. Through the implementation of ideas under these concepts such as overloading, overriding, initializing, information hiding, exception handling, and recursion and after overcoming many different obstacles, our group was able to successfully accomplish the creation of this game.

The game could probably be improved, though it seems fine as it is. The group considered making it three-dimensional or making it into first-person. However, in terms of realistic improvements for likely early programmers, perhaps instead more items, an inventory, and their appropriate interactions could be added, such as traps or explosives, or maybe even give those items to enemies to make more interesting and diverse gameplay. For non-game-related improvements, perhaps reducing the size of the groups to a maximum of three would even be better. The Pink Panthers contained five people, which was an issue brought up by the professor since the beginning. It turned out that a workforce of three people was even sufficient. Furthermore, communication was difficult to establish because emailing and messaging systems turned out to be unreliable. In the first week, one person’s email was nonfunctional, and in the last two weeks, one member’s email automatically sent the messages to spam and his messaging software would not update. Additionally, meetings were difficult to coordinate because individual schedules were frequently conflicting. From experiences with group projects in other classes, instructors assisted in facilitating communication by opening up group functions on Blackboard. Otherwise, perhaps pushing the groups from the start to use code-sharing software, such as GitHub, may be helpful, since although the captain brought it up that it would be useful to use, all the members were reluctant to use it.