Phase 2 Report

Dennis Zhang, Jovan Bapla

Josh Amato, Kris Zhang

Implementation

Our overall approach to developing the game was that each person is to work on their own tasks, and then come together when each person is done with their tasks to put the game together. Because of conflicting schedules, our group couldn't meet as much in person to pair program, but we did our best to work through online group calls. We initially had planned to use the UML diagrams from phase 1 to dictate our classes, but as we found out during development, some of the classes were either redundant or replaced by new classes. Therefore we used the UML as more of a reference than an actual guideline. We also didn't use any external libraries but instead used the Swing and AWT libraries to make the game's GUI. We decided on these mostly because they were already in the Java built-in libraries and were fully functional for what we wanted to create and there would be less errors and conflicts than if we were using an external library.

We decided to have our game revolve around a 2D array made from our Map class and this allowed things to become more structured, because everything that the user sees has a set place in the grid system. This also allowed us to easily manipulate and move items more easily and as we see fit. The way we decided to implement graphics was using Java's built in Graphics2D class which allowed us to make and set the position of our items to match the layout of our grid. Our grid was 34x60 and we made our window 1200x625 and this translated to have everything be moved by 20 pixels and set in place by being divisible by 20 pixels. We thought separating the graphics from the grid in this way would be best to help separate our components and reduce coupling. Everytime the user used the keyboard to move, the hero would move 20 pixels on the game screen and that would result in the user moving one grid cell in the 2D array used in the map class.

For our graphics, we got them all into a folder, called them from within the program, imported animations from gifs and changed them depending on what key

was pressed. The player would get damaged if they are standing on the same grid cell as the enemy or a negative item (trap).

Code Enhancement and Modifications

As mentioned earlier, we added some new classes and replaced some of the classes laid out in the UML diagram. During development, we encountered some problems and created solutions by creating some of the classes. We also placed regular comments and JavaDoc comments where applicable. At the end we also made sure to go through the files and take out any unused code to ensure that we had clean and readable code.

We initially had our main component be in a class called GameLogic but later decided it would be best to implement it in our Graphic class because that is where all the moving parts were. We also renamed a few classes because we thought the new names were better suited for the classes. There were new functions added to the Map and Graphic classes because we underestimated how vital and how much the whole game needed these two classes.

We also resorted to taking away certain actions like speed boost and movement blocks due to time constraints and instead made items only affect the users overall score. We also spent a long time working on the enemy hero detection and had to account for walls so that code had to have multiple iterations. Having to work separately and then put everything together required us to either rename or re structure our already existing code. We also decided to modify our timers after we discovered a more efficient and simpler way of doing so by using the swing timer instead of the util timer. Near the end of our project we decided to look through our functions and see if we could simplify or remove any redundant code. This helped shorten and simplify our Map class greatly and led to an easier implementation to our keyPressed function in our Graphic class. The end screen menus were later moved to be brought up when conditions met to our keyPressed function as well in the Graphic class.

Management

We split up the tasks based on what we had on the UML diagram. Dennis was tasked with working on the Character class and by extension the Hero and Enemy classes. Jovan was tasked with working on the EndMenu and StartMenu as well as the actual in game graphics screen classes. Kris was tasked with working on the Item class and by extension the Reward and Bonus classes. and Josh was tasked with working on the Map class. We decided to leave out GameLogic until we had all done our respective tasks as it depends on everything else which we remade into the Graphic class. We wanted to meet up weekly but then we later realized that we would need more time to finish this project so we started meeting up two or three times a week. This was done remotely because of the current conditions in the world.

Since things were done separately and we had to get together to put them together it required us to be very communicative and make sure everyone understands their task and the code that is being implemented in the entire game. When we were not meeting in person we were always communicating through Whatsapp or our Discord group. Asking questions about the game or asking for help for certain challenges we faced implementing our code as well as to check up and to see where people are at currently in their designated tasks.

When it got to the task of combining our code we almost always made sure that the people combining the code were together and guiding each other in how the code worked and how to implement it so that it works with the whole project. Tasks like these are when we decided to use pair programming to make sure things could always be handled if complications do arise.

Challenges

Aside from having to learn some of the concepts on the fly, one of the biggest challenges was finding time to work together as a group. We would've liked to meet more and work on the project in person, but our schedules never quite lined up. At most, we would have one hour to meet and discuss things and then the rest would be done in a group call. It was midterm season for all of us so finding even an hour to meet was a miracle. We couldn't pair-program like we planned but we kind of got around that by sharing screens on discord.

One of the biggest problems we had was with collision detection. We had trouble getting it to work with the player, the enemy, and some of the walls. On one of the early iterations of the game, we had planned to put the player back to its initial position when hit by an enemy. On the initial pass through, everything worked as planned but when the player got placed back at the beginning, it didn't have the same properties as it had in the beginning. The player was passing through walls and not being detected by enemies.

It was definitely a challenge to work on a project that relies on other members' code. Progress could be made by everyone and each person might have a working component of the game, but you never really find out until you combine your code with another member.