Phase 2 Report

Group 20
Nov 6, 2022

We began implementation by first creating the Entity and Map classes and assigning them assets and methods based on the evolving communication needs they would have with other parts of the program. Namely, the Java Swing elements we would use to visually represent the logic of the game. One of the bigger hurdles of the process was getting familiar with the Swing tools as some of us had limited knowledge of GUI implementation in Java. Once the Entity, Entity subclasses, and Map were outlined, the focus then became easily converting a Map instance into a visual equivalent. We ended up using an equally sized, 2D array of JLabels that would be updated at every game tick to keep the visuals cohesive with the logic. Another consideration was creating the tick and keylistener to update the Map and react to user inputs. We also figured using threads for elements like the timer and shark movements would be suitable, as these operations run independently from each other and need to be ended/terminated at roughly the same instant. Once most of these pieces began to fit together and prove functional, we began attempting to “trim the fat” of our implementation and streamline the code a bit more cohesively without compromising the function of the game.

The main deviations we took from the original design plan were mainly in the hierarchy of the Entity class, and the addition of various Swing elements we were not aware we would need to utilize when first designing the project. The general visual and logical framework laid out in the mockups and use cases were adhered to for the most part, however, the UML Diagram was deviated from in some key areas. Firstly, we did not end up making the Entity hierarchy tree as expansive as we first planned. This is mainly due to some of these classes being redundant/unnecessary and therefore we omitted them in the final implementation. Next, we decided to use the Swing toolset from Java to build the visual elements of the game. This choice resulted in many new, Swing-specific classes being introduced that were not mentioned in the original plan. Mainly the Board class which extends JPanel and incorporates arrays of JLabels to create a cohesive, adaptable method of displaying and updating the game’s map. Additionally, some auxiliary classes were also added such as the Timer and KeyListener in certain areas that were not considered in the original design. Overall we stuck to the original intention and outline of our designs, but made some key adjustments/expansions where they were needed.

Dividing up the responsibility of this phase ended up being mostly dependent on the availability of each group member, as many of us were occupied by other

classwork/obligations at different times, making the delegation of the tasks difficult at times. Our approach was to initially work on each key area that could be separated from the bulk of the design and put the pieces we created back together when they were mostly complete. For instance, one of us laid the foundation of the game logic by creating the Entity and Map classes, along with their associated attributes and subclasses. Another member was assigned to create the key listener for user inputs and Tick class to run methods at given time intervals. The logic of the shark enemies was also broken off and assigned to another member to implement, as we felt this aspect of the game may present more of a challenge to create than others. The visual elements were mostly implemented collaboratively and as needed for testing/experimentation. Their design and layout were tweaked and iterated on throughout the majority of phase 2.

The libraries we decided to use for this project consisted of mainly Java Swing and some key AWT classes as well for handling keyboard events. We decided to use Swing mostly due to it being relatively simple and modular enough to achieve the goals of the project. In hindsight, JavaFX may have been a better alternative as it has more native support for event handling, however using Swing provided the opportunity to create our own solutions for managing the flow of the game. As mentioned, Swing does not have native event handling support so we used AWT libraries for creating event handlers to run the main gameplay loop.

Throughout the implementation process, we used basic and rudimentary testing methods at each step to try to ensure our code was working as intended. This included implementing print statements to check on the status of class methods with basic test cases. Later in development, we also implemented factory methods in order to make the potential of adding different map layouts possible in the future. Creating an iterator was also considered for parsing the board object in order to update the UI, however time constraints did not allow for this during this phase. We also made some changes to the game's aesthetics as the final UI was created. For instance, we added some different background images to the game, but decided a simple one was best as to not confuse the player visually and keep the game elements distinct from the background. We also changed the wall image used, as we felt the new one was more pleasing to the eye with the color palette of the game.

There were many challenges during this phase. The first was getting familiar with the Swing toolkit in order to create and layout the elements in a cohesive, desirable way. I personally have some limited experience with JavaFX and GUI elements but most of our group hadn't used Java in conjunction with GUI's before. The next challenge was coordinating tasks and creating time to work collaboratively on the

project. As we all have other commitments during this semester, scheduling was a big challenge to overcome during this phase. In addition, connecting the components we were each working on was a challenge, as much coordination was needed and some things needed to be modified to work together correctly. This challenge extended into keeping up with the changes we were all making and ensuring we all understood the intention and function of each part we contributed. One of the more recent challenges we faced was implementing the shark pathfinding logic, as the algorithm we used seemed to use a lot of system resources and slow the game down considerably. This is an issue we may need to rectify further during phase 3.