Overall Approach:

Our approach to the game required two internal phases: implementing data structures and algorithms to build logic of the game, and applying graphical components to the logic of the game.

In our first internal phase, since we already figured out the theme through creating the mock ups and the game behaviour by writing use cases, it was our priority to translate the behavior described in the use cases to well-written codes. After transcribing our class diagram to actual classes in our game, it was time to bring the various components together in the Game.java class.

The component that was first on our to do list was modeling and storing a maze where each index would correspond to a tile object. We decided to use a 2D array of Tile objects to achieve this because in our mind, the maze is simply a grid with various indices/coordinates. To differentiate what was located at each coordinate we gave the tile constructor a set of booleans which would identify the tile as a barrier, empty spot, a reward, an enemy, the mouse character ,or an entrance/exit.

The second major component was creating a jFrame for our gamepage, which would listen to keyboard inputs to then move the mouse character accordingly. We used the built-in graphics library called Swing to create a window for our gamepage and had our game class implement the keyListener interface which allowed us to monitor for key presses when on that screen. Once we were able to accurately discern which arrow key was being pressed, we created a move mouse function which would update the coordinates of the mouse tile in the “levelMap” 2D array and print it to stdout. It was within this move mouse function that we then gradually added checks for barriers, rewards and enemies and implemented the logic for what should be done in each of those cases.

The third major component in our logic was moving the Cat in response to the movement of the mouse. Within the move mouse function we added calls to a function called “Chase” which is a member function of the Cat class. This function would look at the location of the mouse and determine which adjacent location is a valid movement that would bring the cat closer to the mouse.

After the three components mentioned above, we added logic for rewards (consisted of regular (Cheese) and the bonus (Organic Cheese)), and the mouse traps before moving on to displaying the maze on the actual jFrame we had created using jLabels.

During our second internal phase our focus was now on getting a graphical representation of the maze and its various components. To do this we broke our tasks down into two parts: static components, and dynamic components.

For the static components (barriers, cheese, mousetraps ,and entrance/exit), we created a function called createGameMap which traversed the 2D array of tiles and drew 100x100 jLabels one the screen which represented the object. For example, if a tile was a barrier, our function would draw a red jLabel at that location. Using a set of if/else conditions we were able to draw all unchanging components of the game on to the jFrame.

Once all static components were drawn, we simply chose our desired starting locations for the cats and the mouse and then drew jlabels representing each of them in their respective functions. Now to get the game to become responsive, we added logic to move the cat and mouse labels 100 pixels in the desired direction in sync with the updates performed to the underlying tile map. Also, when the mouse arrived at a location where a cheese or a mousetrap is located, we would simply make that cheese jLabel disappear and repaint the entire jFrame.

Overall, as can be inferred from reading this report, our approach was an incremental one where focused on a subset of the overall functionality to implement and test before building upon that in the next internal phase.

Adjustments Made:

For the majority of the project, we adhered to the specifications we created in phase 1 with a few exceptions in the context of class structure and functionality.

In terms of the class structure, we initially proposed that we would have a class called Maze which would maintain 2D arrays of Maze Barrier objects. This would have allowed us to maintain multiple “levelMaps” which the game could swap between; however, since we decided to create only one level, due to time constraints, the overhead to set up a constructor with an interface which would allow efficient initialization of a Maze object seemed unattractive. To simplify our design we omitted the Maze class and added a 2D array of “Tiles”(we renamed Maze Barrier to Tile) to the Game class and created a function called “CreateTileMap” to initialize it.

In terms of functionality, we also made some changes to streamline the design which entailed removing some features that seemed semantically unnecessary. One such feature was the “Back to Homepage” button which was to be displayed on the win and loss screens to allow players to reset the game. We proposed that this button would return the player to the home screen of the game where the player would click the “Play” button to play again. This seemed cumbersome because there was nothing else that the player could do on the homepage. This was in part due to the omission of multiple levels. Originally we had thought that returning to the homepage would be more useful if the player wanted to choose a different level to play. Since we decided to create only one level,the “Back to Homepage” buttons were replaced with “Play Again” buttons which would simply reset the game page to a new game. We felt that this change lent itself well to the responsiveness of the game’s interface.

Additionally, we replaced the cheese eaten labels with a simple score label and omitted the “Back” button from the gamepage. The cheese eaten labels were removed because the score of the player was the more important statistic. Furthermore, the back button was omitted for the same reason that the “Back to Homepage” buttons were changed, there wasn’t much need for the user to visit the homepage due to the game using a default level.

Management Process & Division of Roles and Responsibilities:

Planning and Organizing:
Based on the UML diagram created in Phase I, our team first assigned the components of the game and their requirements among each other:

Yogesh was required to implement the mouse class, tile class and the portion of the game class concerned with creation of the gamepage and movement of the mouse. Yogesh set up the data structures to create and store the maze, he also implemented the algorithm for mouse’s movement within Game, which allows the character to update its coordinates by first registering a key pressed, and then distinguishing the elements of the Maze to determine if it can move there, and the behaviour of the game when the Mouse exits the Maze (i.e: Has it collected all of the regular Cheese in order to win the game?)

Scott was responsible for the mouse’s enemies (cats and mousetraps.) His responsibilities included creating the characters’ enemies, cats and Mousetraps. With cats, Scott was required to implement a chase algorithm which allowed them to determine its direction that would get them closer to the player, while making sure they would not affect the maze’s structure, nor the rewards and punishments. Since mouse traps are non-moving enemies, Scott was responsible for setting up ideal locations for them within Game, along with an acceptable penalty value for them. Lastly, he had to make sure that if the character and enemies “collide” then an appropriate behaviour would occur, such as the player losing their score, or losing the game overall due to either having a negative score or colliding with the moving enemies within the game.

John was responsible for the regular (cheese) and bonus (organic cheese) rewards, score update algorithm, and a stopwatch mechanism. Similar to mousetraps, he was required to set up the locations for the rewards locations and values within the maze. However, he also had to implement an algorithm for the organic cheese to randomly appear and disappear after a certain amount of time. In addition, necessary methods were implemented so that when the character collides with the rewards, the score would be updated appropriately. Lastly, to help the users determine the time played, the best approach for John would be to create a stopwatch with a mechanism that would start when the user begins the game, and stops when the game is over.

Tian was responsible for managing and maintaining the quality of the game. Some of his tasks include analyzing created classes’ conditions and behaviours (and revising them if needed,) assisting other members with generating and improving their implementations (i.e:

Essentially creating a ‘skeleton’ for Cats’ chasing mechanism for Scott to work on, or assisting John with improving the Stopwatch mechanism,) and lastly organizing our codes. He was also responsible for implementing additional features of the game including the Win/Lose screens with the appropriate information displayed such as an image, win/lose indicator, total score, the time played, and a button that would generate a new game when clicked on by the user.

Controlling**:**

Each component’s requirements are always observed during the process. Throughout Phase II, we actively update each other on our progress, as well as discussing any challenges that need to be prioritized to ensure our work is within the scope of the project and prevent any unnecessary delays. In addition, any adjustments to the initial plan are discussed between the team before implementing in the actual project.

Review:

Lastly, once the implementations in each component are completed, the program is tested and debugged to ensure all components can function together without errors and bugs caused by one another. Once again, if there are any changes/updates found that need to be adjusted, the process is reverted back to Controlling.

External Libraries Used:

In our project we did not have to rely on external libraries as we were able to use built in libraries such as Java.swing and Java.awt for our graphics.

The decision was made due to two subjective advantages Swing had over the other graphics libraries like JAVAFX, which is an external library. The first advantage was that Swing had a smaller learning curve for us due to the vast assortment of tutorials available for it. For components like creating JFrames , JPanels, and adding keyListeners we followed the tutorials published by the youtube channel called “BroCode” which gave us the relevant information we needed in a clear and concise manner which saved us from having to dig through comprehensive and lengthy documentation. The second advantage was the fact that it is a built in library so we didn’t have to add a dependency, and saved us time we’d have to spend learning how to do that correctly.

Maintaining Quality of Code:

Since Java is an Object Oriented programming language, we paid special attention to the principles of OOP such as inheritance, and encapsulation to make sure we were making design choices which would make our code base semantically clear and maintainable.

With respect to inheritance, we tried to create hierarchies where classes with similar attributes and functionality would share a common superclass containing common components. An example of this would be the Rewards hierarchy where the Cheese and OrganicCheese classes share a superclass called rewards which stores common attributes like location and

value. This is important because if in the future we wish to add a third type of reward with its own specialized attributes, it can simply extend the rewards class to gain those common components.

With respect to encapsulation we made an active effort to bring all the attributes and functionality of an entity into one class. An example of this would be the cat class where we have all the attributes that are relevant to a cat object such as its x and y coordinates in addition to functionality like the “Chase” function which allows the cat to pursue the mouse.

Furthermore, we were careful to follow the basic conventions of programming where we gave each function and variable meaningful names to maintain the clarity of the code. While doing so, we tried to consistently comment on codes to explain their roles and functions. Additionally, we extracted functions to convert large blocks of code into reusable units to make the code easier to comprehend.

Biggest Challenges:

The first challenge that we initially encountered was choosing and learning which GUI tool we should use to develop the application. Instead of using JavaFX like we originally planned on, we went forward with Java Swing due to its simplicity and flexibility in conjunction with the plethora of resources available. We had to spend an equal amount of time learning about Swing as we did employing it in our project.

The second challenge we encountered was algorithmic in nature. When trying to implement the logic for the mouse’s movement, we experienced issues such as invalid movements onto barriers, failed collection of rewards where the score and label are not updated accordingly as the mouse’s coordinates are the same as the reward. We also experienced issues where initially we couldn’t get the cats to move in a sophisticated way, including them going over barriers, or both cats inhabiting the same tile and made it seem as if one of them had disappeared. Implementations of the bonus reward were also a little bit tricky since we kept having errors where the score stopped updating and the JLabel didn't disappear after the mouse had crossed the tiles where the organic cheeses were placed. In general the algorithmic implementation of the above mentioned features was very challenging.

One last challenge that we had was the time zone difference, due to one of our team members currently located in Vietnam which is 14 hours ahead. Although this made the planning process slightly inconvenient, we were able to tackle this challenge through scheduled meetings at times that work for everyone, while also constantly communicating whenever there are any problems through our group chat on discord.