Overall Approach

All our group members were new to what was required of a project of this scale so we started by each choosing a part of the code base to implement outlined in our initial UML diagram. We communicated our work to each other in a group chat to allow everyone to work out consistency issues with each other. Our lack of experience with software development and planning quickly encountered problems with this approach and we worked to resolve the issues as they arose.

We learned of Java libraries that handled functionalities that we designed naive solutions for in the UML diagram such as UI, image display, and screen transitions, and adapted our design plan to incorporate those pre-existing solutions. Later into development we realized we missed a number of specifications outlined in phase 1 and redesigned the code for executing the game logic. Once all the requirements had been implemented and the game worked correctly, we decided to refactor our code base to be simpler and more consistent with the design principles we learned in lecture.

UML diagram Modifications

There were a few minor changes to the overall structure of the UML diagram and many additions as we filled in details we were unaware of when we first designed it.

1. We changed our code to use Java class libraries to handle many functionalities throughout the code, changing the methods and adding inheritance and realization relationships to reflect those changes. JButton, JFrame, JPanel, ActionListener were used for UI functionalities in the Screen superclass, KeyListener, Timer were used in the Level subclass to handle player input and execute game logic at a consistent interval. The BufferedImage class was used to hold the images for entities.
2. The ScoreRecords screen was removed after we decided it was not necessary for the scope of this phase of development.
3. The Sprite class was not implemented due to graphics functionality being handled by the BufferedImage class, and we decided to leave out animation due to time restrictions in the current development phase.
4. The BasicObjective and StationaryEnemy subclasses did not add any meaningful functionality over what was defined in their abstract superclasses Objective and Enemy, so we removed the abstract property from the superclasses and made them object classes from which the BonusObjective and MovingEnemy still inherit.

Use Case Modifications

There were a few use cases that we did not have enough time to implement.

1. <u>Enhanced game loop</u>. We did not have enough time to implement the "bomb" and hybrid scoring system.

2. Viewing Scores. We chose to not implement a "View Scores" option from the main menu. We deemed that this use case was superfluous and would not add towards the quality of the game. Being able to view the final score once a game ended was sufficient.
3. Rebinding the keys. This was another superfluous use case.

Management Process

We managed the project by creating a list of tasks to address. The list included tasks that must be completed and other tasks that would enhance the game but were not necessary. Anyone could work on a task of their choosing and we would communicate what we were working on to avoid conflicts and double check code.

Roles and Responsibilities

Megan - Created the art. Implemented the GUI, including the Screens the user can navigate to, implemented the logic behind switching screens. Reminded the group of project requirements and deadlines. Initiated the work for the project and report. Created the list of tasks. Wrote the Phase 2 Report.

Zoe -Created the SelectLevel screens. Assisted with the art for the enemies and instruction screen. Assisted with adjusting the positions of the buttons. Worked on randomly generating the bonus rewards. Created new UML diagrams.

Musou - Wrote the Level, LevelPlan classes and classes in the "entities" package. Game logic, including movement and collision detection, and painting of elements in the level class. Wrote part of the Phase 2 Report.

Brandon - Overall design and quality control. Worked on game logic and led the code refactoring to be consistent with software design principles. Edited and contributed to the Phase 2 Report.

External Libraries used

javax.swing

This library allowed for us to display images on a ***JPanel***, create ***JButtons*** and display everything in a ***JFrame*** window

javax.imageio / java.io.

java.io.File allows us to represent a file of an image pathname as an object, ***java.io.IOException*** is the exception thrown if the image file is not found. This library allowed us to load images and display them as backgrounds or entities.

java.awt

BufferedImage, allowed us to contain image data so that it could later be displayed on the screen with a paintComponent() method from the JPanel class. ***KeyListener*** allowed for the program to listen for keyboard inputs. The ***CardLayout*** class allowed us to display each of the screens one at a time. The card shown depended on the button the user clicked on, the

ActionListener interface listened for pressing of the button, the ***ActionEvent*** class represented the action of pressing the button.

java.util

ArrayList allowed us to create resizable lists of various Entity types. ***The Date, Timer, TimerTask***, allowed us to keep track of each "tick", the scheduling of player movement and the playtime of the game.

Measures taken to Enhance Code

1. Final Static Constants — Initially we used number literals to represent values such as the window height and width. We enhanced the code by using final static constants that would be used throughout the code. Therefore, if the value of the number was changed, we would only have to change it at one location, rather than multiple. This also made the code easier to read, the meaning of each value could be easily understood.
2. Abstraction —- Features that were common for all subclasses were added into the abstract superclass. There was the Screen abstraction of sub-screens. Initially, a pointer to the Frame object was going to be passed to each of the sub-screens. Passing the reference to the Frame was necessary for the screens to be displayed on the window frame. We reduced the number of parameters passed to the sub-screen objects by passing them to the Screen superclass.
3. Information Hiding — Setting class attributes to private and having them only be accessible through getters and setters. This prevented us from inadvertently giving some classes access to another's attributes.
4. Encapsulation — We placed classes into packages based on their uses. For example, all of the Screen objects were placed in the screens package, and the Entity objects were placed in the entities package. Again, this prevented us from inadvertently giving some classes access to another's attributes.
5. Separation of Concerns — Before the code refactor, despite most of the game logic being performed in the Level class, the responsibilities of the game logic were distributed arbitrarily. Entity objects held an array of positions of existing entities, the array itself was initialized in the LevelPlan class, and game logic took the form of methods of one entity class given the position array of another entity class as an argument. After the refactoring, The Level class was the only place that held entity references arrays and contained all the necessary functions to execute and display the game. It reduced the amount of inter-class dependencies and made the code easier to understand.

Challenges

One challenge was coordinating tasks given that we each had to manage our own commitments, including other classes, work and unexpected injuries. This made it difficult to collaborate with other group members while they were in the process of working on a specific task. Therefore, mistakes made while working on a task would not be caught by other group members until later on. This was challenging because these errors could cascade and lead to larger issues. Mistakes made in the design of the code were especially problematic as they required re-working or sometimes even re-doing the code.

Another challenge was our lack of experience with Java. This holds especially true for the Java libraries: JFrame, JPanel. As we were not experienced with these libraries, a lot of time was spent researching the documentation and finding tutorials. Furthermore, when there was a bug with the GUI, it would be hard to tell if there was a bug with implementation of the libraries or if it was a bug with the logic of the other code.