Group 6: Phase 2 Report

A modular design methodology was used to create the code. The implementation of Zombie Dash was based on smaller classes divided based on the game entities (Player, Enemy, Game Board, etc.) organized in an overarching Main class file once all the individual parts were ideally working as intended to create the whole of the game. The separation of concerns was sufficiently organized within the modular design approach, as separate classes for different entities would be easier to understand with their self-explanatory class names. When everyone had completed their individual parts in their respective branches on GitLab, as per Figure 1, we tried to run the Maven project and fix any errors as needed as a result of combining all of our code together. The process of fixing and re-running would continue after the initial completion of the code, as we highlighted parts to change in order to get a relatively playable version of the game ready to submit. We believed that we optimized the effort to integrate the classes of different entities together relative to the number of modules we had so that the code of our implementation remained as straightforward to understand as possible. However, we may have slightly deviated from our original Agile software development plan as we started coding once we better understood the nature of this project. Some aspects were more difficult to test than others in smaller increments compared to having something more complete to compile. For example, classes whose functions were more dependent on each other—such as the Player class having to check for enemies, rewards, bonuses, etc.—were better run with more of the other corresponding classes completed. However, a prototype for the game's screen and interface with its corresponding sprites on the map was able to be run on its own as a placeholder until the other functions were implemented and combined as well, so the software development process still remained chaotic akin to an Agile cycle.

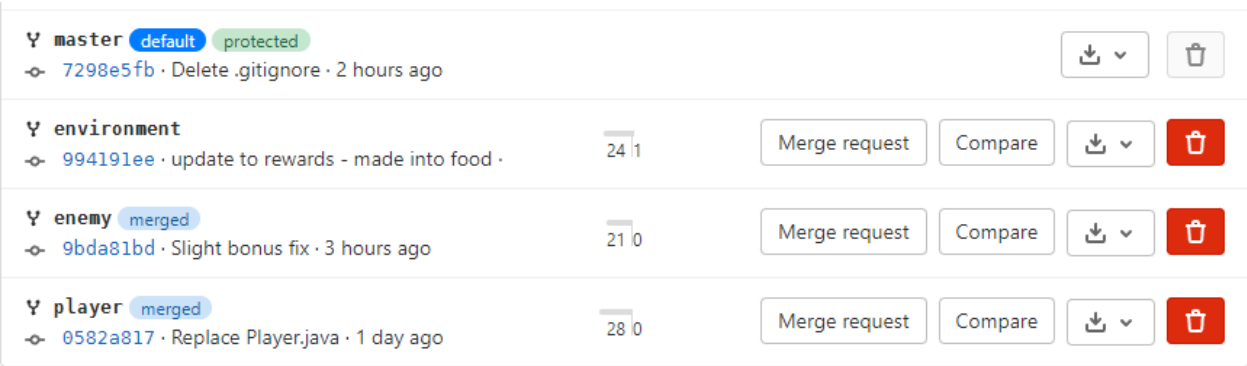

Figure 1. *Separate branches for larger classes.*

There were several adjustments needed to be made after attempting to implement our game. For instance, we underestimated the complexity of setting up the GUI, and had to further split the Game class into Game, GamePlay, and GameScreen classes to represent the underlying implementation of the game's structures, the player and enemy's data, as well as the map and graphics respectively. These changes were justified as they improved the readability and organization of our code. Additionally, some of the attributes of classes we had

thought were needed in our UML diagram planning turned out to be unnecessary in the actual implementation—such as the boolean *turn* and the integer *turnCounter* in the original Game class. We cleared up what the ticks described in the game requirements of Phase 1 were meant to represent after watching the example videos of projects from previous semesters. Since the 2D game wasn't necessarily turned-based, there were slight modifications to some of our variables. Additionally, more functions were added to the Reward, Bonus, and Enemy classes to set and randomize the starting position of these entities—or in the case of the Enemy class, another function was added to find the closest tile out of the up/down/left/right keys to reach the Player as well. These modifications were intended to help streamline and complement the game's other functions. Although our initial thought was to combine punishments along with enemies, we also added a separate Punishment class for them in this phase to remain consistent with the distinction between Reward and Bonus classes in our code.

Our workflow was organized with a list of tasks that could be marked as complete, in-progress, or to-do at a later time. We used the Notion website/app, as in Figure 2, to keep track of our responsibilities and duties as a team, and the division of our roles was based on the main entities of the game as described in Phase 1. Rather than having every group member work on the same classes simultaneously, we thought it would have been more efficient to split up the classes to individuals to work on instead so that merge conflicts were kept minimal. However, for more challenging classes to implement—such as the Player class, as it had to handle many of the keyboard inputs—there was more collaborative work between team members. Additionally, each member may have had a specific implementation to each class in mind, so communication in the team's group chat was essential to keep everyone else notified of potential changes to the original game plan. In addition to updating Notion, the team attended weekly meetings to stay updated with what the team was working on or have completed. Lastly, Discord was used to communicate directly with the team since it gave a platform to exchange tutorials, images, and videos, as well as the option to live stream your screen to each other.

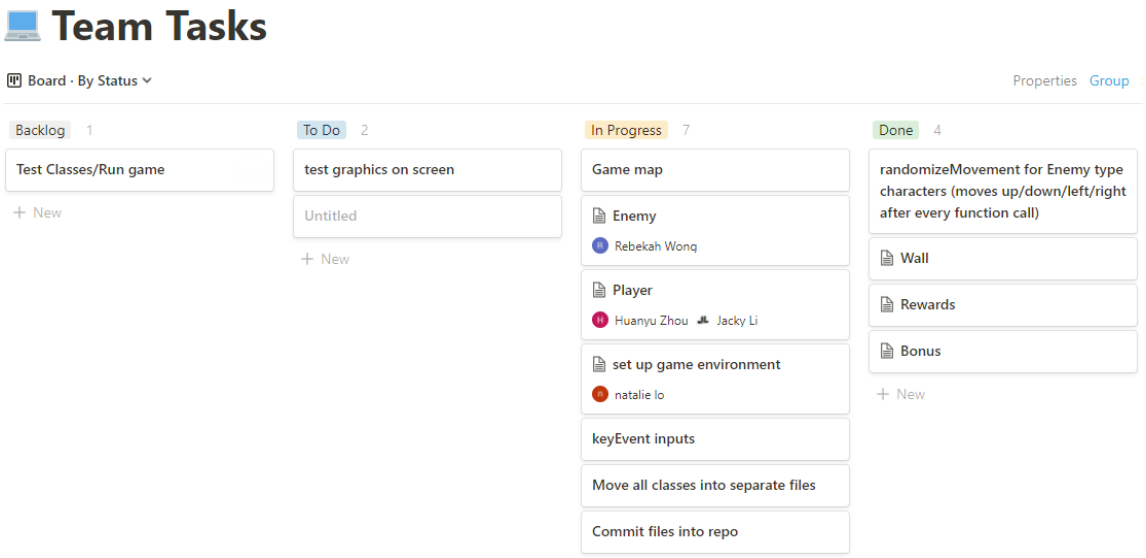

Figure 2. *Using Notion to assign tasks*

We used resources such as the standard Java library for classes like java.util.Random in order to randomly generate numbers to determine where bonuses may spawn throughout the game. Using the standard library was well-justified because as a standard library, it should be widespread enough to contain many helpful classes and functions to use in a variety of projects. Our GUI also stemmed from Java Swing and importing corresponding libraries to get the JFrame working for the user interface. We chose Java Swing based on our own preferences as to what may have been more user-friendly to grasp for beginners to Java such as ourselves, and we decided on libraries based on how well we thought we could make use of them.

To enhance the quality of our code, we tried to comment our functions to varying degrees in order to clarify any aspects of the implementation as needed, especially for parts that may have seemed more confusing to understand at a cursory glance. For instance, the labelling system of our HashMap for the game board may not be entirely clear, and that the numerical keys representing each area type (path, enemy, reward, etc.) had to be further clarified throughout the rest of the classes. Setting a map's location to a single integer would not be sufficient to describe the region if not for a comment explaining that "0" meant "path", or "3" meant reward, for example. Additionally, any repeated fragments of code were placed in functions—such as with the movement functions for up, down, left, and right in the Enemy class—to help with the organization and structure of the code, as well as to improve its readability compared to various code fragments repeated throughout the class. Lines of code were spaced out as needed to also enhance the readability and avoid clutter, which is correlated to enhancing the quality of our code. For consistency, we aimed to use a uniform style and standard coding practices. The team reviewed the code in the branches before performing a merge.

During Phase 2, many of our challenges came from figuring out the more technical components of the project, such as setting up Maven Apache correctly and using Java Swing for the GUI, because of our lack of experience with these applications. Some group members were having trouble running the project and testing the application on Visual Studio Code and spent a lot of time trying to learn how to set up configurations on IntelliJ from the group member who actually knew how to use it. Additionally, all group members were relatively new to Java, so we had to learn some of the proper syntax along the way and find similarities between coding languages that we did know, such as C++. Phase 2 took place in one of the busier times of the semester, which meant that our work may have been more sporadic because of the other responsibilities that each of us had to maintain for other courses as well. In Phase 1, we were able to work on the deliverables simultaneously in a call, but this proved to be more challenging in Phase 2 due to our conflicting schedules and increased workload of to-do items to implement. Nonetheless, we managed to communicate our updates well enough in our group chat when we worked on our sections independently in our own time.