Phase 2 Report

- **describe your overall approach to implementing the game,**
    1. Divide the requirements/expected final product into different parts
        a. UI
        b. Enemies(trap & zombie)
        c. Maze and object control/set up
        d. Player initialization and update
        e. Rewards(regular and special reward)
        f. Other(music, sound effect, sprites,etc)
    2. Split the requirements in to specific tasks using simple factory design pattern
        a. Game panel class for overall display and update
        b. Key handler class to handle keyboard input
        c. Main class to call the game panel
        d. Sound class to load back ground music and sound effect
        e. UI class to improve user interaction
        f. Utility Tool class for shared method between all classes
        g. Collision checker to check object collisons
        h. Object file which contains reward, player life, trap, and exit
        i. Entity file which contains player and zombie
    3. Create sprite sheet for in-game UI(map, player, title screen, zombie, trap,etc)

4. Create a to-do list that contains all the tasks need to be done, using git push, pull and discord communication constantly updating the to-do list

- **state and justify the adjustments and modifications to the initial design of the project (shown in class diagrams and use cases from Phase 1),**
    1. Renamed Board class as game panel and introduced thread to the original board design
    2. Add a new class called Sound to load and prepare background music and sound effect for in game performance
    3. Add a UI class to render images onto the screen and update accordingly.
    4. Rename location class to ObjectSuper Superclass
    5. Break off services in Service class into individual classes
        a. Attack method to player and zombie class
        b. Step trap method to trap class
        c. Move method to player, zombie, and keyhandler class
        d. Update method to game panel, player and zombie class
        e. Calculate point method to UI class
        f. Collect reward method to player class
    6. Add a tile and tile manager class to read input text file and create maze accordingly
    7. Add an asset setter class to initialize player, zombie, rewards, and traps.

8. Put rewards class, and trap class into Object directory, put player class and zombie class into Entity directory, put tile and tile manager into Tile directory
9. Create an additional resource directory to keep all the render resources(pixel art and sound)

- **explain the management process of this phase and the division of roles and responsibilities**
    1. At the beginning of the phase, the entire group met online to discuss possible roles based on the outline of the game that we had determined in Phase 1. This process also included brainstorming how to actually implement the game and finding resources that would help us along the way.
    2. We wrote a to do list to separate each part of the project and each person chooses what they want to do.
    3. Since the project was due around the mid term week, we had to split the time slot and figure out who had more time during a certain time slot.

- **list external libraries you used, for instance for the GUI and briefly justify the reason(s) for choosing the libraries,**
    1. While we did not use external libraries, except for java jdk 15 for the coding portion of the project, when sourcing audio we went to two different

websites; https://freesound.org/ and https://orangefreesounds.com/music/background-music/ to find both the music and the sound effects for the game.

- **describe the measures you took to enhance the quality of your code,**
    1. We went through the code and tried to follow applicable design patterns as discussed in class. We created factory methods for creating entities and worked to make our code extremely modular throughout
    2. Throughout the coding process our focus was on making the code run. Once the game was able to be played, we made sure that every change we made did not regress the game back past the state that it was last in. For example, adding music should not make the game unplayable if we made a mistake. Using the try{} except{} method of checking for exceptions was key to making this happen.
    3. We made the name of the methods, variables and classes easy to understand so that anyone literate in the programming language used should be able to read the code and quickly grasp how it works, which is also easy for our group members to understand and be able to make improvements.

- **and discuss the biggest challenges you faced during this phase**

1. Implementing the algorithm for the zombie to follow the player when they are in a certain range. For example, at first the algorithm was too simple, which made the zombies able to walk diagonally and go through walls.
2. Dealing with collisions between the player and the zombies and traps. We had to constantly change the CollisionChecker class to fix bugs such as the player becoming unable to move after colliding with zombies.
3. When starting the game, since we set up multiple background music and sound effects, they were first playing at the same time due to music frequency and collisions, which was not what we wanted. Thus, we had to fix this problem by changing the sound effects.
4. In the beginning, we did not understand how to use Maven, so we had to watch several tutorials in order to figure it out.