Approach to implementing the game

- Reviewed the basic user interface described from phase 1.
- Created the main game window with a fixed resolution size to develop our frames around it.
- Created the basic frame structure for required frames (Title Screen, Playing Screen, Game over Screen and Game win screen) using imports from AWT and JFrame.
- Added “Game states” so it switches states when certain conditions are met (for frame switching).
- When basic frame structure was completed, moved on to creating the basic game loop (eg. the game thread) to get the app running.
- After the main loop was implemented, we began importing the sprites involved in the game (eg. Player, Enemies, Items, Bonus Items etc).
- Successful sprite imports lead to coding the “Player” character and the keyboard function in order to check the response of the user moving.
- Implemented the mouse inputs in order to switch between frames when conditions are met (eg. if clicked “resume” in the pause menu it will switch to the current “playing” game frame).
- Afterwards, we developed the objects (eg. walls, flooring), then the “enemies” so that they appear in the game frame”.
- Consequently, we coded the conditions required from the project where the enemies can either be stationary or moving towards the “player”. Afterwards, we enabled a condition where no more than one character can be on a tile.
- We enabled the consequence for “player to enemy” collision. The first is hitting the “stationary” enemies which induced a “hurt effect” (renders a sprite image and pauses game thread very shortly). While the second was when moving enemies hit the player, which instantly sends the user to the “Game over” frame.
- After completing the main character requirements, we moved on to writing the score function. The score was developed to help create the “win condition(7 rewards = win)” we planned to make shortly, as well as the “second defeat condition (negative score = game over)”.
- We proceeded to create a function for rewards which when the player touches the sprite it would be removed from the screen and added a point to the score.
- Revitalized stationary enemy condition which now removes a point from the score if a player hits one.
- We added the time function afterwards to showcase overall playtime for one run through.
- Revitalized “Game Over” screen to show the achieved score and time of the individual as well as a humored result for enemies hit.
- When the game was (mostly) developed, we ran the use cases described in phase 1 until all cases succeeded.

Adjustments and Modifications to Initial Design

Our initial design documents focused mostly on objects & characters, and we were missing functionality such as: keeping track of the score of the game, a stopwatch to keep track of time, and drawing objects.

We added utilities which we didn’t think of in our initial design: a class to handle collisions between objects, as well as classes for handling key & mouse input. We also created separate classes for the screens for winning, losing, or pausing the game.

We also added abstraction to our initial design by creating an Objects class and a Characters class

Management process & and the division of roles and responsibilities:

We made a list of all the tasks to be implemented and assigned them. As we encountered new errors or thought of additional tasks to add, we added them to the list. We communicated frequently to confirm if we had completed a task or if it was still in progress.

All External Libraries used for the Game

- AWT
  - The main purpose of AWT was for building the game GUI. This was used to make the main window of the game for the user to interact with. This is responsible for running the game on the screen.
- Swing
  - Swing was used for making the frames and window applications for the game.
  - Also contains Timer, which allowed us to make a stopwatch.
- IO (Input/Output)
  - This was used to read all necessary files in the resource folder such as images, text or sound files. This allowed us to load maps by reading a text file that was accessed using java.IO. It also allowed us to draw using our own custom sprites for tiles, objects and entities. Music and sound effects also used IO to read the sound files in the resource folder and play them as a clip using the sound library.
- ImageIO
  - Drawing and buffering images on the screen was best accomplished using ImageIO. It was what enabled the program to read and write image files from the resource folder and process those images onto the game.

- Sound
    - This allowed us to implement music and sound effects to the game using the music files (.wav) in the resource folder. Ex: music plays when on main menu or when playing the game, sound effects are played when action occurs such as picking up rewards or pausing the game

Measure taken to enhance code quality

We enhanced the quality of our code using OOP principles such as abstraction and inheritance. We also tested each other's code as well as our own. We reviewed our code to try and reduce coupling and increase cohesion. We also added in file comments to functions that require an explanation to understand.

The biggest challenges faced during this phase

The first challenge we faced during this phase was figuring out how to use Git properly and efficiently (as well as troubleshooting error messages), and settling into a workflow. It was troubling to pull and push files from the repository due to the linear workflow of the project, due to all the unknown errors and troubleshooting when trying to perform a task on git.

In terms of code, the biggest challenge we faced initially was actually getting an object to move across the screen. Once this was figured out it was easier to add additional objects. However we faced another challenge with the moving enemies when trying to figure out how to make them move towards the player's [changing] position while also restricting them from going through walls.

The tasks that took the most time to debug were: the buttons on the menu/pause/win/lose screens, fixing collision bugs with the player, and randomizing spawn points for bonus rewards.

Another problem involved the rendering of certain files and folders failing after pulling and pushing to the repositories. Additionally, some files were loading inconsistently or were not being read properly. So, bugs like images loading properly occured or the sound files were not playing when being accessed and loaded.