CMPT276 - PHASE 2 REPORT - GROUP 9

The primary objective of phase 2 was to find a way to implement our game and learn how to connect code in java and a user interface together. As a team, we had no prior experience with game development, so our first step was exploring java libraries and getting more familiar with their respective functions. We did a lot of research and had to watch many tutorials online to be able to understand how the mechanics of a 2D game work. Our initial focus was on implementing basic functionality to render elements on the screen. This involved creating classes to represent entities like the Player, Enemies and Objects within the game worlds. We started with simple methods to display these entities on the screen and we gradually expanded their capabilities.

Additionally, we created classes to manage game objects such as chests, hearts, and enemies, each with their own unique characteristics and behaviors. These classes were essential for populating the game world with interactive elements and adding depth to the gameplay experience. By starting with basic methods and gradually expanding our game, we laid a solid foundation for our Java game development skill. Our approach prioritized simplicity, clarity, and modularity allowing us to build upon our initial implementations iteratively and incrementally refine our game over time.

Task division within the team:

Duc Duy Pham (Paul):

1. Implemented collision checking for different types of objects, including walls, chests, and doors.
2. Fine-tuned collision detection for lava and water tiles, ensuring proper interaction with the player character.
3. Handle special behaviors associated with lava and water, such as damaging the player or affecting movement speed.
4. Implemented visual effects or animations for lava and water interactions, enhancing the game's immersion.
5. Created the pixel art for characters and tiles in the game.

Danial:

1. Implement enemy behavior, including movement patterns, targeting, and attacking.
2. Integrate pathfinding algorithms to enable enemies to navigate the game environment and pursue the player character.

3. Fine-tune enemy AI parameters, such as detection range, aggression level, and fleeing behavior.
4. Implement enemy spawning mechanics, including spawn points and wave-based spawning for increasing difficulty.
5. Created Game Over screen and Game complete screen as well as their respective mechanics

Sandra:

1. Expand the variety of interactive objects in the game, such as chests, keys, switches, and traps.
2. Design and implement visual assets for objects, ensuring they align with the game's aesthetic and theme.
3. Fine-tune object behaviors and interactions to provide meaningful gameplay challenges and rewards.
4. Handle keyboard input detection and processing that allows the game to respond to the player key presses for controlling character movements and other game interactions

Vinh:

1. Develop the game map layout, including tile placement, terrain features, and environmental hazards.
2. Implement the scoring system, tracking player progress, and awarding points for various actions and achievements.
3. Design and implement the user interface elements, including HUD components, menus, and in-game displays.
4. Handle graphics rendering and optimization, ensuring smooth performance and visual quality across different devices.

At first we started by implementing exactly the UML diagram in Phase One but after becoming more familiar with the game development process we found that certain design decisions were impractical and not very efficient when translated into actual code, so we had to do some modifications to that. The different classes we now have in our project are the following:

Character Classes: These classes represent entities within the game world, such as the player character, enemies, and objects. Examples include Player, Enemy, and SuperObject.

Tile Classes: Responsible for defining the properties and behaviors of tiles used to render the game map. Examples include Tile and TileManager.

Checker classes: EventChecker, CollisionChecker and KeyChecker. These classes handle the game functionality that involves interaction of the Character objects, within itself or with other objects. For example, eventChecker handles situations for ending and damages in the game.

UI Classes: Manage the user interface elements, including menus, HUD elements, and game screens. Examples include UI and GamePanel.

Utility Classes: Provide utility functions or methods used across multiple parts of the project. Examples include UtilityTool.

Pathfinding and AI Classes: Handle pathfinding algorithms and AI behaviors for enemies or NPCs. Examples include PathFinder.

On the other hand, the UI did not really change from what we planned in phase 1. We tried making improved graphics with better quality but did not change any core concepts in the game. Also, we found the use cases extremely helpful as they helped us understand the overall logic of the game, as well as the different events that the game must handle. The functional requirements of the system helped us define all the actions that the player could possibly perform and the resulting response of the system to these actions.

The first library we started by using was the **java.awt.image.BufferedImage.** Its first use was in the Tile classes to help effectively manage tile images, ensuring that they are loaded efficiently and rendered accurately on the game screen. Also once the images are loaded we need to scale them to fit the size on the screen. The UtilityTool class contains a method for scaling BufferedImage' objects, ensuring that they match the dimension of the tiles on the game screen. Another library that was very useful throughout the project was **java.awt.Graphics2D.** It was especially useful for rendering the game graphics, such as "GamePanel", "UI", "Object", and "TileManager". It was important for Object class because it was used to draw object sprites onto the game screen, the class has a draw() method that accepts a "Graphics2D" object as a parameter and uses it to draw the object's sprite at its current position on the screen. We also used it in GamePanel class, it was used to render the game world including tiles, objects, enemies, and the player character.Graphic2D is also used to draw various UI components, such as the player's health and score within the game panel. **java.swing.JPanel** library played a crucial role in creating the GUI for our game. It served as the canvas on which we could draw all

the visual elements of the game, such as the prisoner character, the enemies, objects and the game environment itself. This library allowed us to customize and organize the layout of our game screen, including the placement of various components and their interaction with user input. JPanel provided us with methods for rendering graphics using the Graphics object, enabling us to draw images, shapes, and text onto the screen. This allowed us to create visually appealing game elements, such as sprites, backgrounds, and user interface elements like buttons and labels.

To enhance the quality of our code we implemented several measures throughout our development process. First we established coding conventions and standards to ensure consistency and readability across the project. This included naming conventions for variables, classes, and methods, as well as comments and documentation. As a team, we conducted regular code reviews sessions and peer evaluations to identify potential issues or talk about how to fix logical errors and improve the overall code quality. We also used Git to manage our codebase effectively. This allowed us to track changes, collaborate seamlessly and allowed team members to work on different tasks at the same time. Furthermore, we prioritized modularity and code reuse to support maintainability. We tried to simplify complex functionalities into smaller, reusable components and libraries, we reduced code duplication and minimized the risk of introducing errors during development. Regular refactoring sessions were also conducted to improve the structure and design of our code. This involved restructuring code, simplifying algorithms, and eliminating redundant code and to enhance clarity, performance, and maintainability. We also used Maven in our development process which provided a lot of advantages in enhancing the quality of our code and it also facilitated dependency management by allowing us to specify project dependencies in the pom.xml file. This ensured that all team members were using consistent versions of libraries and frameworks.

During the course of our project, we encountered several challenges. Firstly, coordinating meetings that suit everyone's availability was challenging but it helped a lot because as a group we agreed on meeting in person because it is more efficient for us and helped us get our work done faster and to discuss how we will implement all the specific details of the game. Additionally the majority of our team does not have prior experience in Java and game development, so we had to do a lot of coding sessions to practice coding in java and to be more familiar with all the different libraries we agreed on using. Another significant challenge we faced was managing collisions and defining game termination conditions and implementing them. These took a lot of discussions on the logic of the implementation and we had to try a lot of different ways to get it right. Also figuring out how to implement the game

loop like how the player will be interacting with all the different rewards, enemies and damage and how to connect all these events with the score of the player and keep it updated that was a huge challenge for us as well.

In conclusion, Phase 2 of our project served as a significant learning phase for all of us. Starting from scratch, we successfully navigated the complexities of integrating Java code with user interfaces, laying down the essential building blocks of our Dungeon Escape game. Through the use of Java libraries, extensive research, and collaborative effort, we overcame our initial lack of experience with the technology used in the project. We made huge progress by implementing core functionalities and learning the ropes of game mechanics. The division of tasks among team members enabled us to tackle all the challenges and aspects of the project. We followed the scrum SDLC where we treat this phase two as a scrum sprint, and have daily 15-30 minute meetings to sort out the tasks for everyone. Overall, this phase not only improved our technical skills but also strengthened our teamwork, setting a solid foundation for the continued development and refinement of our game. Looking forward to phase 3, we will bulletproof our implementation of the game with thorough testing of every method and objects involved in the design, in order to produce the best final product possible.