Overall Approach:

Based on the UML, we divide into two parts to implement which are the Character and the Map entities which need to be done by the first deadline. Since we are a group of 4 people, we do a pair programming method, dividing into 2 groups, one group implementing all classes in Character entities (except Enemy class) and one group implementing all classes in the Map. For the Enemy class implementation we leave it for the second deadline, because it is implementing a searching algorithm which is quite complicated.

During our first deadline tasks, we are working collaboratively with the partner we are assigned with for pair programming and checking each other's code and giving some feedback. Then meeting as a full team to update what we have done and make sure all the code can work as expected. Implementing a refactoring method to check the overall code connected properly.

For the second deadline, is the stage where we have major changes after doing the refactoring process. We found out that our code doesn't work as expected after combining everything. So, one of our team members takes control of it, since he is quite familiar with the assignment, while others are supporting with the documentation and giving some suggestions on how it is supposed to work and also designing the layout.

Adjustment and modifications:

We changed the game theme from the ocean theme into a hospital theme due to the animation picture for the motion effect not found in the internet (for example, fish picture facing left, right, upward, downward, etc). We also update the UML concurrently reflecting to our current code.

Division Role:

Map: Liam Cummings and Ryan Martin
Character: Jea Oh Lee and Ryan Mitchellin
Refactoring: Jea Oh Lee
JavaDoc and Report: Ryan Mitchellin

External Libraries:

- AffineTransform: Represents 2D affine transformations like translation, rotation, scaling, and shearing, used for transforming shapes and images.
- AudioInputStream: Represents a stream of audio data, used to read audio data from sound files for playback in the game.
- AudioSystem: Provides methods for accessing and manipulating the audio system, used to retrieve the Clip object for playing audio files.
- BufferedImage: Used for working with images in Java, including loading, displaying, and manipulating image data.

- Clip: Manages audio playback, specifically for short audio clips like sound effects in the game.
- Color: Represents a color in the RGB color space, used for setting colors of graphical elements in a GUI.
- Dimension: Represents the size of a rectangular area, commonly used for specifying component sizes or dimensions in graphics programming.
- Font: Represents the graphical font used for rendering text, including its style, size, and family. In the maze game, it is used to set the font for displaying text in the user interface.
- Graphics: An abstract class for drawing graphics, providing methods for drawing lines, shapes, text, and images on components.
- Graphics2D: A subclass of Graphics with more advanced 2D graphics capabilities, offering precise control over rendering, transformations, and compositing.
- JFrame: Provides a window for hosting Swing-based GUI components, allowing the game to be displayed in a graphical window with title bars, borders, and controls.
- JPanel: A Swing component for holding other components, used to organize and manage the layout of GUI elements.
- KeyEvent: Used for handling keyboard input, such as implementing character movement using arrow keys.
- KeyListener: Used for handling keyboard input events, such as key presses and releases, enabling interaction with the game world through keyboard controls.
- Rectangle: Represents a rectangular area in a coordinate space, often used for collision detection and spatial calculations in graphics programming.
- URL: Specifies the location of sound files, used to load audio data for playing in the game.

Biggest Challenge:

Refactoring our maze game posed the biggest challenge, as much of the original code didn't function correctly after merging various components. We had to revise and reorganize the code, breaking it into smaller, more manageable sections and optimizing data structures and algorithms for better performance. Additionally, we focused on enhancing the user interface for improved usability.