CMPT276 Phase 2 Report: Group 4

Meeting Summaries

October 24th:

- Discussed the sequential diagram.

October 25th:

- Delved into understanding UML diagrams.
- Worked on task distribution among team members.
- Determined the contents to be completed by the mid-deadline.
- Discussed about whether to use JSON libraries to store player's record.

October 31st:

- Final check before halfway deadline
- Testing on all software packages to ensure seamless integration when combined.
- Identified codes that require repair and tested each objects appearance on game window.
- Determined the specific objectives and priorities for the latter half of Phase 2.
- Solicited feedback from each team member regarding their experiences, challenges faced, and suggestions for future enhancements

November 4th:

- enemyAvailable_pos is working now
- When creating player, the tiles are being removed. Need to be fixed.
- Discussed what we've accomplished by halfway deadline.
- Talked about the challenges we have encountered and how we solved it.

November 7th:

- Want the whole game logic to be done on Friday. Everything should show on the window. Need rewards and walls.
- Focus on the game menu after friday.

Strategy Overview

- We segmented the UML diagram into distinct packages, such as "player", "rewards", and "enemy".
- Each team member has been assigned a specific package for implementation.
- Inter-package communication is facilitated via the recordUsedPlace class, which maintains information about other classes.

Design Adjustments

- Incorporated additional classes for keyboard input and game window functionality.

Midway Deliverables (Nov. 1st)

Chengfeng Xiao:

- Design the Reward class.
    - Features: Collection of rewards like candies with potential for specific bonus effects in future updates. Rewards can appear and disappear during gameplay.

Li-Yu Wu:

- Construct the Enemy class.
- Enemy's initial position
    - Add a function that generate randomly initial position for ghost and in recordUsedPlace class
    - Types: Stationary enemy (spider) and moving enemy (ghost).
    - Behavior of ghost: Pursue the player if they come within a specified range (4x4 grid).
    - *Standard ghosts cannot traverse walls, but there's a consideration for an advanced type that can. Will implement in future if time permits.
- Provide graphical assets for the player and rewards.

Nick Smart:

- Develop the Tile class that encompasses all space types on the game board.

Tianyi Tang:

- Update and submit the UML diagram incorporating the gamePanel functionalities, pop-up window, and keyboardInput implementations.
- Implement the main character's controls via the keyboard.
    - Incorporate the Keyboard class.
    - Utilize the Observer design structure to update the player or game controls.
    - Ensure character movement adheres to the WASD directions and remains within the designated range.
    - Introduce the Room class with variables determining the maximum and minimum x/y coordinates, and a method playerAvailableTiles() which invokes recordUsedPlace.

Goals for Final Deadline

- Pausing Page
    - Grey transparen png
    - google
- Main Menu
    - Start Button
    - Display 3 houses representing different difficulties.
    - Player move to the horse to enter the game.
    - Frist build the three button can be click

- Map design
    - Predesigned obstacle layout.
    - One florrplan per difficulty. Can do 2 if time permits.
- Enemy position
    - Place randomly but not closed to player
- End game criteria:
    - When the enemy catch player should end the game.
    - When a player's score is negative
    - The player get all rewards and goes to the door
- The game only ends when you close the window.
- UI
    - start button in the main nemu
    - Image and interaction

Deliverables for Final Deadline (November 12th)

Chengfeng Xiao:

- Reward class

Li-Yu Wu:

- **More enemy testing.**
    - **Fix game logic: enemy should be able to anywhere player go**
- **Add game level: easy, hard. medium**
- Design game menu
- Popip windows and UI
- menu background
    - pop up window class
    - Create a interface should appear and grey out background
        - Show total score
        - List how many rewards, spiders
    - Draw brown tiles and upload.
- Also show the total score on the game menu

Nick Smart:

- Create door, wall, floor, and tombstone images and make sure the objects work and can be shown on screen for room implementations.
- UI - start menu, main menu, and end menu with interactable buttons (will try to complete if i finish above fast enough)

Tianyi Tang:

- Create main menu
- switch the main menu and game loop
- Implement more accessible function in player
- Finished the remove available function in RecordUsedPlace
- Player store value

- Add more function to GameManager

Documentation:

- Javadoc comments for all classes and methods
- Game Features:
- Introduce a class for tiles to display in-game images, detailing player interactions with tiles, such as wall collisions and door functionalities.
- Bonus reward, the pumpkin, should be able to appear randomly and stay for a fixed length of time and disappear.
- If time permits, we will add a special feature to this bonus reward. When the main character picks up the pumpkin, it will act as a shield, making the character invisible to ghosts. Have to add an image of dog wearing a pumpkin head to illustrate.
- Finalize and polish the User Interface (UI) along with all associated graphical assets.
- Incorporate immersive music and sound effects to enhance player experience.
- Design and implement a fully functional menu, complete with responsive buttons for game navigation.

Optional Enhancements:

- Explore the feasibility of incorporating a game-saving feature, allowing players to resume their progress.

Gameplay Mechanics:

- Store both bonus and regular rewards. After each level, compute the total score based on the rewards collected.
- Finalize the intricacies of various game levels. Introduce innovative elements to the game map. Implement a "Pause" feature, allowing players to halt gameplay midway. During the pause state, the game background should be grayed out, and a distinct "Paused" UI should be displayed to the player.

External Libraries Used

- Gui…and reason why we choose the libraries
- 1. **java.awt.Graphics**: This is a part of Java's abstract window toolkit (AWT) and is used for drawing shapes, text, and images onto the screen. It is crucial for rendering the game's graphics, such as the game characters, the maze, and the score display.

- 2. **javax.swing.JFrame**: JFrame is part of the Swing library, which provides a set of lightweight components for building graphical user interfaces (GUIs). We use JFrame to create the main game window, set its size, and handle close operations.

- 3. **MouseListener and MouseMotionListener**: These interfaces are used to handle mouse events. While our game is controlled by keyboard, players also use mouse listeners for navigating menus.

- 4. **KeyListener**: This interface is used to respond to keyboard events. It is essential for capturing the player's input to control the character's movement.

How We Enhance Quality of Code

- Good communication, clear class diagram
- Use create packages to put classes that have strong relationship
- Make class only contain the functions that responsible for one aspect to avoid the god class
- Use the inheritance to avoid hard coding and give the flexibility to the future extension
- Most object relative to the game are created by factory, so subclass can decide which instance it want to create
- Applied singleton to class which only need have one instance and its functions will be frequently used by other classes
- Draw a sequence graph to make sure the only necessary data will exchange between classes
- When the function are public and will cause big effect of game (like end of game), there will be a judging to check does whether the big change should applied to game not just directly run. This can prevent function from being misused

Challenges We Emcountered

- The first challenge we faced in our Java project was the asynchronous progress of work among team members, each responsible for a different package. Due to the varying work paces, there were instances where the attributes across different packages were not transmitted in a timely manner. This resulted in a delay in integration and required additional time for synchronization and verification.

- The second challenge arose during the integration of different packages, where we encountered numerous bugs attributable to the diverse coding styles of team members. Addressing these issues proved to be time-consuming and required a significant amount of effort to standardize the code and ensure consistency across the project.

- The third challenge pertained to our initial lack of understanding regarding the generation and operation of the game window. While we were proficient in expressing the methods and attributes of different packages, we were initially clueless about how to create and run the game window. It was only after extensive research and consultation of various online resources that we were able to identify the appropriate methods to generate the game, gaining valuable insights in the process.

Adjustments and Modifications to the Initial Design

- In our UML graph, we don't consider how to show the object on the screen and how to create an endless loop for continuously updating the player and enemy moving. Therefore, we don't consider these functions in our UML graph. In our phrase 2, we implement these functions by using customer design class that extend the JPanel and put these classes in one Package called WindowAndInput
- In the UML graph, we just assume there is one room in our game, which means the number of enemy, reward, wall and obstacle will always keep the same and the room will neve change. In phrase 2, we decide we are going to have multiple difficulty levels and we need to generate different rooms and different numbers of enemy, reward, and obstacle based on difficult player selection. So we added the BasicConfig, MediumConfig, HardConfig class to store information in different difficulty information and use the GameConfig class to get this information to base on player selection. These class contain in the Logic Package
- In the first design, we think once the program is running, the we should directly make people control the player and player. However, when we write code, we find out that the game will take about a few seconds to load all resources in the game and it can't make people directly play the game. Because of that, we decide after the program start, it will show the main menu and loading panel to give the program time to load all resoucre. This means, we need to implement the switch between panels which is not in our original design and we put this duty to the GameManger. For now GameManager will not only focus on checking the game end or not, it will also control the switch between panel
- In the room package we add a new Class called tile to make the room have floor. This change affect the factory and initialization to provide another function to generate the tile
- In our characterAvaliablePosition interface, we add a new function called takenPlace. The main purpose of this function is to check if it allows us to place the other object, like reward, enemy, obstacle or player on it. We also delete the getEnemyAviable function, because we realize that the enemy can't go through the wall and obstacle
- In the RecordUsedPlace class, we added lots of new functions to make this class more powerful. For example, we use the add playerGetReward and playerMeetEnemy function to make the player can easily find out whether it collects a reward or is caught by the enemy. We also use recordUsedPlace to store all objects which are showing on our game. So when panel drawing the object image in screen, it can use the recordUsedPlace to access all object without having to search everywhere