Phase 2 Game Implementation Report

Group 7: Usman Aziz, Ryan Chong, Nicole Malku, Curtis Pu

Overall Approach to Game Implementation

Our approach to implementing our game was to divide and conquer. It was each member's first time developing a game in Java, so we all tried our hand at almost every aspect of game development. Most of our communication occurred in our group Discord server. There we handed out tasks to work on daily. Although each member had been given a main task to implement, we decided to dabble in every other area of development as well to gain a richer understanding about the process of constructing a game. Our goal for the halfway deadline was to implement the classes made in our Phase 1 UML diagram, and also create 6 game maps for each level. Afterwards, we began implementing the logic behind each parent class, and extending subclasses. We maintained communication in our team through messages in our Discord server, and continuously developed our game until completion. Each member did their own tasks, aided other members with any questions sent via the Discord, and overall contributed to the creation of Alien Escape!

Adjustments and Modifications to Initial Design

The initial design, class diagrams and use cases outlined the basic structure of the game. However, modifications were made during implementation to the UML defined originally for the game implementation. Originally the UML lacked several essential classes such as GameScreen, MapManager, AssetSetter, Sound and UI. Additionally we had envisioned providing players with the functionality to customize their astronaut suits by changing colors, but unfortunately due to time constraints this feature had to be shelved during the development process in order to prioritize core elements of the game to provide a robust and functional experience.

Management and Division of Roles and Responsibilities

Our goal in role management was to assign everybody a main task, but to additionally work on a handful of various different tasks in the development process. The main roles were assigned as follows: Nicole handled game initialization, the design and partial implementation of the game maps and title screens, and creating and implementing original music and sounds into the game. Curtis implemented a pause screen, title screen and pathfinder class. Usman was responsible for handling the collision between different entities and UI related functionality. Ryan implemented enemies, portals, fixed pathfinding, bonus rewards and lots of sounds. Below is a detailed breakdown of the work each member contributed.

Nicole:

- Created the initial game screen with size settings, and made a game loop running at 60 frames per second.
- Initialized the update(), run(), and draw() methods used in the game loop.
- Created classes to handle keyboard input, to allow user controlled movement.
- Implemented the drawing of the Main Character to the Screen.
- Implemented Player movement based on desired speed.
- Designed half of the pixel art used for the player movement animations (left and right animations.
- Created the Map class, and it's subclass MapManager
- Designed 6 maps for each level in the game, and helped with integrating them into game play.
- Devised the idea to use a file with just 1's and 0's, in order to represent walkable areas and barriers, respectively, for each map. I made the 0 and 1's file for the 1st map.
- Redesigned the map drawing from being drawn tile by tile, to instead using the previously designed game maps, allowing for much nicer aesthetics.
    - Had to additionally solve and debug the lag caused by this change.
- Debugged object collision and object placement. Rewards are now only accessible to the Main Character, and cannot be consumed by monsters
- Successfully implemented reward consumption by the Main Character, as well as other object interactions.
- Created and implemented 2 original soundtracks for the title screen and the main game play.
- Found royalty-free sounds to be used for object consumption, as well as created custom sounds for option selections from the title screen.
- Created 3 pieces of art to be used for player health status.
- Implemented the player health status to change according to the player's life.
- Redesigned the title screen functionality to use the title screen image I had previously designed, instead of being just a blank screen with text.
    - Additionally solved and debugged the lag issues associated with this change.
- Created a function to display the player's score on the screen
- Debugged all sound issues when the option "Restart" was chosen in the loss screen.
- Redesigned the pause feature into a popup options screen.
    - Implemented features to:
        - Change the music volume, change the sound effect volume, go to a screen that teaches users how to play, an exit game button, and an option to go back to the game/unpause.

- Redesigned the loss screen aesthetics.
- Created and implemented custom sound to play after the game finished, either by a loss or by a win.

Curtis:
- Implemented regular reward class to get object image
- Created UI class for displaying UI element such as pause screen and game over screen
- Implemented pause feature in key “P” or “ESC”, once player pressed pause key, gamestate is changed
- Implemented title screen prototype, which was later updated by Nicole
- Implemented game over screen when player health is <= 0, player can choose “retry” or “quit”
- Created AssetSetter class for setting object in an array
- Created Node and Pathfinder class
- Created different gameState such as titleState and playState
- Debugged player’s position and health when enter “retry” in game over screen

Usman:
- Implemented the mapManager, which is responsible for managing various maps.
- Changed a map image into an array of tiles to improve game functionality.
- Implemented the UI class responsible for managing graphical elements of the game including but not limited to fonts and images.
    - Implemented various screen scenarios, such as pause screen, winstate, time, rewards collected.
    - The draw() method dynamically renders different game states (pause, play, win, loss, e.t.c.)
- Implemented the collisionChecker class, responsible for the hit between the character and the game environment (enemies, rewards, barriers)
    - The class utilizes gameScreen and encapsulates collision-related logic for efficient gameplay implementation.
    - Handles dynamic speed adjustment, computes grid positions of the entities boundaries, and based on entities direction checks for tiles with collison properties.
- Converted map images made by Nicole into text binary files for easier implementation and modification of tiles for game environment for map (2 - 6).
- Implemented the use of getResource method, rather than hardcoding the paths for resource files
    - Increasing portability, and ensuring independence of specific user machine directory structure.

Ryan:

- Created object package and basic object class such as BonusReward, PowerUp and RegularReward.
- Added punishment class and implemented timer.
- Created entity class, player and enemy children.
- Created Main and GameScreen that draws the game screen.
- Created gray astronaut frames.
- Fixed objects spawning not in map grid.
- Added alien class.
- Implemented player and alien collision in collisionChecker class.
- Implemented hitsound when a player collides with an alien.
- Fixed player update() to not switch images when no key is pressed.
- Implemented portal class to allow players to teleport to different map areas.
    - Implemented PortalHandler to check the portal status, if a player reaches the portal, changes the player coordinate.
- Implemented transition between maps to let player advances once they collect all regular rewards on their current level.
- Debugged music for not stopping after the player died.
- Added player start position for all maps as well as set all objects.
- Made the player transition between levels after touching the escape rocket.
- Created bonus reward using a timer & random
- Created bonus reward interaction with player
- Debugged pathfinder class that alien is now tracking player
- Fixed bug causing instant loss upon retry
- Added sounds for remaining objects
- Handled player loss scenarios
- Adjusted map collision (collision tiles)
- Made alien walk randomly when not able to reach player
- Created Alien images

External Libraries

We chose not to use any external libraries.

Code Quality Enhancement Measures

In order to elevate the quality of code, our team strategically employed the use of multiple classes, fostering a modular and organized structure. Wrapper methods were used to encapsulate complex operation, promoting code readability and maintainability. Diverse functions, each tailored to a specific task, simplified troubleshooting and debugging processes, ensuring seamless and coherent execution of the overall codebase.

Challenges Faced

A common issue faced by our group was with the git merging functionality, dealing with conflicting changes in parallel branches, lead to many merge conflicts that demanded careful resolution, this complicated the integration process a lot for our group. Furthermore, consistently not pulling the latest version lead to many merge conflicts for outdated code. Each member adapted to the habit of pulling the latest version and committing the smallest changes made to the code, which somewhat reduced the problem for us.

Another issue faced was the lag associated with drawing each game level's map. It slowed performance down, and caused lots of frustration. We had to keep switching back and forth, deciding whether we wanted to favor aesthetics or speed. After lots of frustration, we figured out it was the act of resizing of the images that was causing the game to lag. Manually resizing the images before adding them to the game fixed this issue immediately, allowing for both high quality aesthetics and speed.