Phase 2 Report

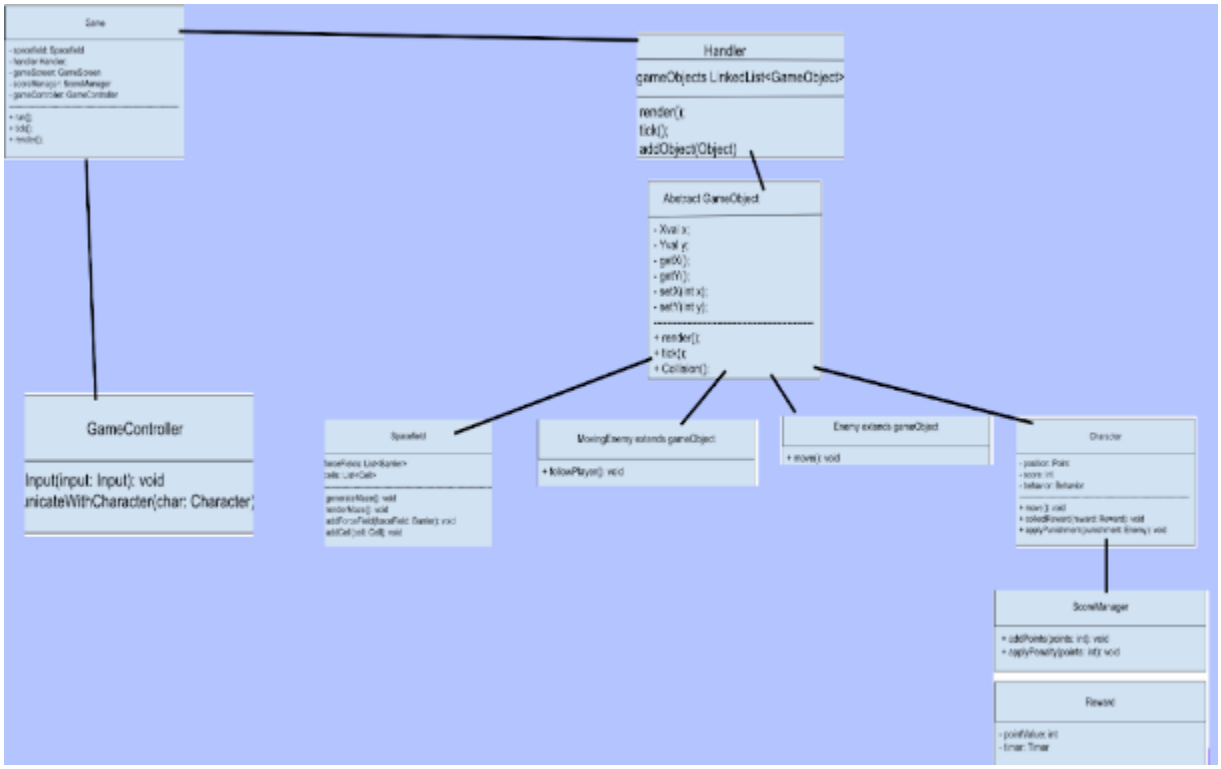
Introduction

This report details the second phase of the development process for "Pixel Pioneers," a 2D space adventure game. It outlines the implementation approach, design modifications, project management strategies, technical choices, quality enhancement efforts, and challenges encountered.

Overall Approach to Implementing the Game

The implementation of "Pixel Pioneers" focused on utilizing a game loop that invokes tick() and render() methods. The tick() method, called 60 times per second, updates the game state, including movement and collision, while the render() method draws game objects onto the window. The game architecture is built upon a robust class hierarchy, with a key focus on the Handler class, which manages all game objects and their interactions.

Shown below is our UML Class diagram



In All classes in the diagram were inputted as shown. Some naming changes were **GameController→ KeyInput** to be more general for future usage, StarField→SpaceField, **MovingEnemy→BasicEnemy** ,**Character→Player**. ScoreManager and Rewards implementation were moved into the HUD class.

The Most important Class is the **Handler**. Handler stores a list of all Game Objects and is used to run their tick and render methods. As of the halfway point of phase 2 advanced graphics have not been implemented yet and all graphics are rectangles and strings displayed using Java.awt external library which is a very general GUI Library. The Window Class is implemented using Java.swing for no other reason than ease of use.

Adjustments and Modifications to the Initial Design

- The UML class diagrams and use cases from Phase 1 provided a blueprint for development. However, practical adjustments were made:
- Renamed classes for clarity and future scalability: GameController to KeyInput, StarField to SpaceField, MovingEnemy to BasicEnemy, and Character to Player.
- Integrated ScoreManager and Rewards into the HUD class to consolidate game status management.
- These changes streamlined the codebase and aligned class functionalities with their real-world game counterparts.

Use Cases

1. Main title Screen: When the game is ran the user is presented with a main menu screen displaying a “Play”, “Help” and Exit button clicking these buttons with the mouse does exactly as you would expect. Clicking Play Starts the Game. Help brings you to a screen that tells you how to play and Exit shuts down the game.
2. Move Character: The Player character is controlled by the WASD keys. To move up left down and right respectively. If the player hits the edge of the screen it will not go through. If the Player hits a space field it will not go through.
3. Reward System: the longer the Player stays alive the higher his score. Bonus rewards to collect have not currently been implemented
4. Enemy: Currently only the basic enemy has been implemented the basic enemy bounces off walls and space fields and has no sentience. If it collides with the player the player will lose health. Smart enemies that follow the player will be implemented in the next part.
5. Win Condition: Currently there is no win condition for the player other than a personal goal of achieving the highest score possible. As the player reaches higher levels they encounter more and more obstacles. An end goal will be implemented in the next phase.

External Libraries

Two primary Java libraries were employed:

- **Java.awt:** Chosen for its general-purpose GUI capabilities, allowing for the creation of custom graphics and user interface components.

- **Java.swing:** Selected for its ease of use in creating window-based applications, particularly for the Window class responsible for the game's display.
- These libraries are well-documented and supported, offering a balance between functionality and ease of use for rapid development.

Challenges Faced
The development process was not without its challenges:

- **Collision Detection:** Crafting a reliable system for detecting and responding to collisions was intricate, demanding meticulous attention to detail.
- **Integration:** Ensuring seamless integration of various game components, from user input handling to game object management, posed significant hurdles.

Conclusion

To enhance code quality We used standard OOP practices to keep classes concise and powerful. The hardest Part this phase was probably getting everyone in the group on the same page as to how a video game is actually programmed in terms of things like displaying graphics and object interaction. There was an issue when setting up the maven support and an upstream branch was added. The Master branch should be the default branch but we are not sure how to set it as so without operating permissions so when marking please clone from the Master branch.