CMPT276 - PHASE 2 REPORT – GROUP 9

Our overall approach to implementing the "**City Escape Game**" involved careful planning, structured design, and efficient teamwork. The game's goal is straightforward: as the player, you must escape the city within a given timeframe, avoiding capture by the cops and pitfalls like potholes, while being mindful of spikes that deduct a minute from the timer. Strategic pickups, like the nitro, add an extra minute to help you succeed. To win, you must collect all diamonds and reach the marked exit.

Project Implementation Process

We began Phase 2 by translating our initial design from Phase 1 into an actual game, focusing on effective software design principles. This project represented our first attempt to create a fully functional game, complete with both front and back ends. We started with intensive brainstorming sessions to finalize the mechanics of the game on a granular level.

The map layout became foundational to our design, structured as a 2D array of numbers to represent different game elements. This array layout helped us systematically incorporate items like roads, grass, buildings, characters, rewards, and obstacles, which created a city-like environment for the game. We then curated images to enhance the game's visual appeal, ensuring that elements like roads, grass, houses, characters, and rewards were visually consistent with the city theme.

To display the game items and images according to the map blueprint, we developed a method to render everything accurately on our 2D map. Next, we implemented object-oriented principles by creating super classes for various entities (such as rewards, hurdles, and characters) and then subclassing them into specific entities like nitro, diamond, spikes, potholes, cops, and the thief. Each of these subclasses had unique effects on gameplay, aligning with best practices like strong cohesion, loose coupling, and modularity.

Adjustments to the Initial Design

While adhering to the Phase 1 design was a priority, we found some changes necessary to refine gameplay. Our original design had several elements that, upon implementation, required modification:

1. **Time Mechanic**: Initially, we planned to give players unlimited time, scoring based on completion time. However, implementing a time constraint added excitement and urgency, which enhanced the game experience.

2. **Nitro Mechanic**: Our initial use case suggested that collecting nitro would increase player speed. Implementing this mechanic turned out to be complex, so we modified nitro to instead grant an extra minute, complementing the time-based gameplay.

3. **Spike Mechanic**: Originally, spikes were supposed to slow down the player's movement speed. However, to simplify gameplay and support the time-based mechanics, we modified spikes to reduce the timer by one minute instead.

4. **Exit Choice**: We initially included multiple exits in the game, requiring players to find the correct one. However, this feature didn't significantly impact gameplay, so we simplified it to a single marked exit.

5. **Difficulty Selection**: Although planned as a feature, time constraints led us to postpone implementing difficulty levels until a later phase.

6. **UML Diagram Changes**: Certain classes required adjustments to fit our evolving needs.:

a.) Terrain Handler: The TerrainHandler class manages the rendering and display images of all items on the map, including rewards, hurdles, buildings, and characters. It takes the 2D array from the map file and translates it into visual elements displayed to the player, ensuring that each item is correctly positioned and rendered based on the map layout.

b.) Thief Controller: The ThiefController class handles the player's movement and controls within the game. Acting as the bridge between player input and the game's response, it listens for keyboard inputs and translates them into movements (up, down, left, right) for the thief character on the map. This allows the player to navigate obstacles and evade cops smoothly.

c.) Object Manager: The ObjectManager class is responsible for creating and managing lists of rewards and hurdles based on their locations specified in the map file. By reading the file, it dynamically creates and positions diamonds, nitros, potholes, and spikes throughout the game

Roles and Responsibilities

Our team divided responsibilities to ensure each member's strengths contributed to a well-rounded game. Here's an outline of our roles:

Vansh:

• Implemented tile rendering based on the map, developed the "TerrainHandler" and "GameScreenPanel."

• Coordinated with the backend team to ensure seamless integration with the UI.

• Designed and implemented UI elements, including the HUD components, menus, and in-game displays.

• Optimized graphics rendering for smooth performance and visual consistency.

• Created screens for game outcomes (win/lose) and drafted half of the Phase 2 report.

Sanika:

• Focused on backend development, working closely with the frontend team.

• Implemented logic for thief movement, key listeners, and the main class for game initialization.

• Developed super and subclass hierarchies for all rewards, hurdles, and game items, ensuring each had distinct attributes and behaviours.

• Helped in seamless integration with frontend and providing functions that can be invoked by UI to get the required data.

Ehsan:

• Designed the main screen panel for the game.

• Created the complete 2D map.txt file, detailing object positions and tile layout for seamless map rendering.

• Sourced and designed pixelated images for enhanced visual appeal.

• Completed report compilation, formatting, and wrote the other half of the report.

Andy:

• Developed cop movement, AI behaviour, super and subclass hierarchies for all characters, also implemented pathfinding to enable enemy pursuit.

• Designed cop spawning behaviour and located character and item images.

• Contributed image citations and ensured that assets met project requirements.

• Assisted in debugging the cop movement mechanics and optimizing performance.

External Libraries Utilized in Development

The first library we utilized was **java.awt.image.BufferedImage**. This library was instrumental in managing images for game elements, allowing us to load, manipulate, and render images efficiently for tiles and objects within the game. Its capability ensured that all images were displayed correctly on the screen, enhancing the visual appeal of our game. Another essential library was **java.awt.Graphics2D**. This library played a crucial role in rendering graphics, enabling us to draw various components such as characters, objects, and backgrounds on the game panel. Its advanced capabilities provided us with improved quality for rendering shapes, text, and images, thereby elevating the overall graphical experience of our game. We also employed **javax.swing.JPanel**, which was vital for creating the graphical user interface (GUI) of our game. Serving as the main canvas, it allowed us to organize and customize the layout of visual elements, manage user input, and seamlessly render game graphics. This library facilitated the interaction between the user and the game, making it a key component in our project. These libraries were chosen for their robustness, widespread use in Java applications, and the flexibility they offered in rendering graphics and managing images efficiently.

Strategies for Improving Code Quality

To enhance code quality, we implemented several key strategies. We established coding standards for consistency, including naming conventions and thorough documentation. Regular code reviews fostered collaboration, allowing team members to discuss improvements and share insights, which iteratively enhanced our code quality. We utilized Git for version control, enabling effective change tracking and seamless collaboration on different tasks without conflicts. A modular design approach allowed us to break down complex functionalities into reusable components, improving maintainability and reducing code duplication. We also engaged in regular refactoring sessions to optimize code structure and readability, ensuring clarity and performance. Additionally, integrating Maven for dependency management helped maintain consistent library versions, minimizing compatibility issues.

Key Challenges Faced in Phase 2

During this phase, we encountered several significant challenges. Coordinating meetings that fit everyone's schedules was difficult, but we recognized the value of in-person discussions for more effective decision-making about game implementation. Additionally, many team members faced a learning curve due to limited experience with Java and game development concepts. To address this, we organized coding sessions to help familiarize ourselves with the chosen libraries and frameworks, requiring extra time and effort. Collision management proved complex, as implementing detection and defining game termination conditions necessitated extensive discussions to establish the required logic. Designing an effective game loop that integrated player interactions with rewards, enemies, and scoring mechanisms also presented challenges, demanding careful planning and iteration to ensure seamless communication between these elements.

Conclusion

In conclusion, Phase 2 marked a significant milestone in our "City Escape Game" development journey. Building on the design blueprint from Phase 1, we created a functional 2D game prototype by integrating Java code with a user interface. This involved crafting a dynamic city environment populated with obstacles, rewards, and enemies, while refining mechanics to enhance player experience. Key aspects like collision detection and time-based challenges added complexity and excitement to the gameplay loop. Our effective use of Java libraries, combined with teamwork and rigorous research, allowed us to navigate both foundational and detailed integration tasks. As we look forward to Phase 3, we will focus on refining our implementation through thorough testing and debugging to ensure the final product meets our design principles and gameplay quality standards.