CMPT 276: Project Phase 2 Report

Kyle Deliyannides
Patrick Shaw
Chris Pinca
Bassim Ghasemzadeh

1. Describe your overall approach to implementing the game.

Our approach to implementing the game resembled the Agile method. We worked as a small group and held regular sprints to track progress and ensure that we were meeting our goals. During these sprints, we held both in-person and virtual meetings to discuss progress and plan the next steps.

We made use of design patterns such as Object-Oriented inheritance and polymorphism to reduce redundancy in our code and increase efficiency. For example, we created a parent Screen class that was responsible for creating different screens to be displayed during the game, such as the title screen, pause screen, and game over screens. This class was then extended by child classes to create unique screens for each situation. Additionally, items that can be added to the grid use a Drawable parent class which is anything that can be displayed, they are then inherited down adding unique functionality to each layer. The grid consists of Board Elements which are themselves extended Drawable, these are Roads and Obstructions. Then, Roads can contain Items which are also Drawable. And the Vehicle class (extended by Cop Car and Food Truck) are also Drawable, this allows for movement attributes to be shared by all Vehicles.

Creation of the game grid was designed to be flexible. We created a board element factory as well as a map layout reader which takes a csv file and translates them into elements on the grid. This allows for the ability to easily change the grid layout as well as add new items or change or remove existing items quickly. This takes advantage of the factory design pattern for BoardElement creation. Sprite loading is done through a single file which pre-loads all the grid sprites and then sends them out as requested. This allowed for the ability to entirely change the system from a mess of variables and switch statements to a system which simply uses a hash map. This switch was done near the end of the game's completion and did not cause a single issue elsewhere in the game, due to the modular design.

The scoreboard system utilizes a system resembling the observer design pattern. This sends updates to the scoreboard which then go back and gather score data from the Food Truck. During the refactoring phase this system could be simplified a little bit with less interdependence. But for now, this serves as a decent starting point for a modularly designed system.

The moving enemy utilizes a homemade breadth-first search algorithm implementation which is designed to be reliable and scalable to more than just one instance of the Cop Car.

2. State and justify the adjustments and modifications to the initial design of the project.

The original game requirements plan, mockup and use cases were almost spot on to what we ended up with. We stuck true to these plans as they were very well throughout and ended up not needing much deviation. The only difference would having PotHoles and Construction Zones just being one item, a pothole, as having both these things seemed redundant. Additionally, the scoreboard design changed to be a bit taller and contain more information than originally anticipated. However, our original UML diagram was heavily deviated from as it ended up being unfeasible and we needed much more classes. The Entity class was displaced with a Drawable, similar to before just with the inclusion of the Draw() function. We removed both the Obstacle and Reward abstractions as they were unnecessary, instead we gave each item an enum corresponding to which score value they should update. The vehicle abstraction was kept similar except for the removal of the EnemyController as this seemed better implemented by the Game class itself. The GameState was replaced with a Game Class and was much different as the scores were split between a scoreboard and the FoodTruck instead. The Grid stayed similar to what was originally planned. However, many classes were also added such as a modular menu system, a sprite loader, and a csv to map creator. All these modifications were the result of using an agile development method as we weren't quite sure about the technical feasibility when designing our original UML diagram.

3. Explain the management process of this phase and the division of roles and responsibilities.

Our management process can be broken down into four parts:

1. **Divide and Conquer.** This approach involved breaking down the project into smaller, more manageable tasks. Each team member was assigned specific deliverables to work on based on their strengths and expertise.
2. **Group Meetings:** Regular group meetings were held to discuss progress, share ideas, and make decisions. These meetings helped ensure that everyone was on the same page and working towards the same goal.
3. **In-Person Sprints:** In-person sprints were used to encourage efficiency and to help team members stay focused on their tasks. This approach helped to ensure that everyone was making progress and meeting deadlines.
4. **Constant Communication:** Communication was key throughout the management process. The team used an iMessage group chat to keep in touch and share updates throughout the day, while Discord video calls with screen sharing were used to discuss specific issues and work through challenges. Along with in person meetings during key deadlines and for major ideation phases.

The division of tasks was divided as follows:

- Kyle was in charge of creating the Grid with Roads, Obstructions, and interactable items, the vehicle movement, as well as the core game logic such as setting up the game screen and checking for and implementing win/loss scenarios, setting up the game tick timer, as well as the sprite loader and map layout creation. Alongside the major structure of the whole system. As well as creating some sprites.
- Bassim focused on implementing smoother player movement and animation, the breadth-first search algorithm of the cop car (moving enemy) as well as refactoring design and coming up with graphics and co designing the sprite loader alongside Kyle. As well as creating some sprites. Bassim contributed to many bug fixes and quality improvements alongside Kyle throughout this phase.
- Chris was responsible for the creation of the Secondary UI screens alongside their graphical components, these include designing the interfaces for the Title Screen, Pause Screen, Win and Loss Screens. As well as creating some sprites.
- Patrick focused on the scoreboard implementation and design. As well as creating some sprites.

4. List external libraries you used, for instance for the GUI and briefly justify the reasons for choosing the libraries.

- **javax.swing.*** - This library was used for creating and displaying JFrame and JLabel components on the screen. It was also used for creating LineBorders around certain elements in the game. We chose to use this library because it is easy to use and has plenty of tutorials and documentation available online. Additionally we used the Timer functionality within swing to create the game tick timer, as it was straightforward to set up.
- **java.awt.*** - This library was used for creating button inputs and images. It was also used for setting GridBagConstraints, Insets, and Font properties for various components in the game. We chose to use this library because it provides a lot of flexibility and control over the layout and appearance of GUI components. Additionally, this allowed for accepting keyboard input. This library also utilizes the ImageIcon class to load the sprites, as this supports gifs for animations. Finally, this library allows for easily drawing images to the screen in a grid using its Graphics2D object.
- **java.io.*** - This library was used for handling InputStream and IOExceptions. It was used to load image files into the various SecondaryUI Screen. We chose to use this library because it provides a simple and efficient way to read and write data to and from files, while allowing for easy resizing before converting them to an ImageIcon.

5. Describe the measures you took to enhance the quality of your code:

Communication between group members. Constant communication allowed us to eliminate redundant code: This measure helped to ensure that the code was clean and efficient, without any unnecessary duplication, and that team members were not accidentally working on the same system which would cause hiccups in the code quality produced due to conflicting code being merged with one person's idea overpowering the others.

Refactoring throughout. We made sure to go back and refactor things as we go once we realize the design may pose issues down the road. This helped prevent refactoring classes or packages that are already near complete, and allowed us to iron out bugs in earlier stages.

Git Branches: Each team member worked in their own branches while designing the features, allowing us to merge features safely. We agreed to look over each other's code before merging to ensure it was always of high-quality.

OOP design implementing inheritance and polymorphism: The use of object-oriented programming concepts such as inheritance and polymorphism helped to make the code more organized, maintainable, and extensible. It also helped to reduce the complexity of the code, by reusing functions so that if changes are made to one it will update across the application.

- **Example:** Creation of a parent Screen class responsible for creating screens to be displayed for any situation: This measure helped to reduce duplication of code and made it more efficient to reuse code. The Screen class made all screens but the unique screens pass it the specific image files and listeners to retain unique functionality.
- **Example:** Creation of a Game Over Screen class that extends Screen and two child classes that extend the Game Over Screen class: This measure helped to more effectively reuse and organize code. It also made it easier to add new types of Game Over screens in the future, without having to rewrite a lot of code.
- **Example:** Creation of a Frame class that is responsible for creating and displaying the JFrames and displays correctly: This measure helped to eliminate the need to hard code and manually create new JFrames wherever they were needed. The use of an enum to specify which screen to create and display also removed the possibility of errors in specifying which screen to display.
- **Example:** All items which are drawn as sprites in the game extend a common Drawable abstract class which contains the code to display them on the Graphics2D grid. This draw() function is then overridden by some sub classes to add more functionality such as displaying the Item (also a Drawable) on the Road which it is contained in.

- **Example:** Vehicle movement is done through inheritance as the general idea for both the food truck and cop classes movement is the same. However, extensions have to be made pertaining to path fining or picking up items off the board/damage to potholes and speed traps.

6. **Discuss the biggest challenges you faced during this phase.**

- **Scale of the project:** The size of the project was a significant challenge, as it required a lot of work to develop a functional game. Managing all the different components of the game, such as the UI, game mechanics, and scoring system, was a daunting task. Figuring out how to piece all these pieces together and solving merge conflicts while doing so proved a difficult, but rewarding, task for our group.
- **Learning how to make a game:** For all members of the team, this was their first time making a game from scratch, which meant that there was a steep learning curve. Understanding the different components of the game, such as collision detection, scoring, and movement, was a significant challenge. Creating the grid system with smooth movement proved to be very difficult.
- **Getting JFrame and JPanel to display correctly and elegantly:** This was a technical challenge that required a lot of trial and error. Getting all the images, scores, values, time, and buttons to display correctly on the screen in an appealing and intuitive manner was a significant challenge. Getting the CardLayout for the pause and game screen transition was taxing.
- **Changing and updating requirements:** As development went along, there were changes and updates to the requirements, which required the team to adapt and make changes to the code. This required careful communication and coordination to ensure that everyone was on the same page. Through a well managed agile development cycle we were able to overcome this challenge.
- **Making design choices:** Deciding on the overall design and aesthetic of the game was a significant challenge. This required careful consideration of factors such as user experience, game mechanics, and visual appeal. Additionally, designing the class hierarchy so it was extensible was a task that took lots of revision in the early stages before the system became too complex to easily refactor. In the end we came out with an extensible and maintainable package class design hierarchy with an efficient grid, creation, and sprite loading system.
- **Getting enough sleep**: As with any intense project, getting enough sleep was essential to maintain productivity and prevent burnout. However, this was a challenge for some members of the team, especially during periods of heavy work. At one point we worked from about 2pm to 2am on a Saturday!
- **Breadth-first search algorithm:** Creating a homemade implementation of a breadth first search algorithm was very challenging and required extensive debugging. This caused a lot of stress from Bassim as he was the one tasked with this difficult task.