CMPT276: Phase 2 Report
Group 25

Overall Approach to Implementation

Our approach was reminiscent of an Agile process model consisting of two sprints – one corresponding to the halfway deadline and another corresponding to the final deadline. We created a Discord server to coordinate our progress during sprints and facilitate communication.

During the first sprint, our sprint goals were structured so that many basic mechanics of our game could be completed and demonstrated without yet combining them into a full game and fitting them into a unified theme. Since we had no prior experience making games in java, our sprint goals were to finish implementing the features demonstrated from parts 1 to 8 in this video tutorial series:
https://www.youtube.com/playlist?list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq
Each group member was assigned a set of parts from this tutorial series to implement. By the end of the first sprint, our project contained a player that could move via WASD keyboard inputs on a map. The map contained tiles, some of which could block the player. There were also objects scattered across the map that demonstrated basic functionality – key objects that could be picked up to unlock door objects, along with a chest object that displayed a victory screen when picked up.

In the second sprint, since our general project structure was established, we no longer needed to follow a guide. We aimed to adapt existing functionalities and create new functionalities to meet all project specifications, add additional quality of life features, and to make our game components all fit into a cohesive theme. We decided to set the following sprint goals and indicated dependencies between our goals with (dX):

1. Add a score system and display the score on the top right
2. Adapt the keys into keycards by creating a new sprite and assign the keycards a score
3. Add randomly spawning battery objects that increase score (d1)
4. Add EMP objects with a predetermined location that decrease score (d1)
5. Add security guards that follow the player
6. Add game over state triggered by negative score or security guard (d1, d4, d5)
7. Add game over menu with a restart option (d6)
8. Add start menu
9. Add an exit
10. Modify the door so that it requires and consumes 3 keycards instead of 1 key
11. Add game win menu with restart option (d9)
12. Add images for player, keycard, battery, door, EMP, security guard, exit
13. Decide an on initial map configuration (d4, d5, d9)
14. Add pause menu

We each chose goals that we wanted to work on, created separate branches, and worked on our changes independently. Once someone completed a goal, they notified the group that they

had pushed a change, received feedback and made adjustments, then communicated which goal they would work on next in our Discord server. After completing all of our goals and making final adjustments, we had finished our game.

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

Our project satisfies all the behavior defined in our use cases from phase 1 with the exception of the pause menu behavior and the exclusion of a starting and ending cutscene. The pause menu in our implementation is not accessible via a visible clickable button as defined in our use cases, but instead by pressing p on the keyboard. We made this change because we had little space on the screen to fit a clickable pause button and the rest of our game did not utilize screen clicks as an input method. We also stated in our use cases that the player could check the game controls via the pause menu. However we decided to exclude this functionality to keep our pause menu uncluttered and because our game is relatively simple so many of the controls can be discovered through experimentation. Finally, we mentioned in our use cases that there would be short starting and ending scenes in the game. However, we excluded these scenes because we wanted to focus on other more important functionalities and the amount of design and planning required to implement scenes would not be proportional to the benefit to the user.

We made many adjustments to the internal design of our project compared to our initial UML class diagram. These changes involved renaming certain classes and changing workflows.

Game Manager

- Renamed to GamePanel. This was the name used in the video guide that we adapted in order to reduce confusion when following along with the guide.
- Uses an array instead of a list to store SuperObjects (items). An array limits the amount of items able to exist. This limit was useful to avoid adding too many items, which could impact performance.

Asset Setter

- This class modularized the functionality of initializing the SuperObjects in the GamePanel, increasing separation of concerns

Guard Manager

- Added this class to store and update each guard. Instead of the GamePanel storing and managing the guards, the GamePanel stored an instance of a GuardManager class. This provided a layer of abstraction and took the responsibility of Guard management off of the GamePanel class.

Deleted Clock Class

- We felt it would be better to have the GamePanel directly manage the game time, since the GamePanel could directly interface with the game components it contained for every tick. This resulted in two other changes:
    - An ObjectSpawner class was created to manage the randomly spawning and despawning batteries, modularizing the spawn management into this class.

    - Pausing and resuming was handled by the KeyHandler, which could directly change the game state of the GamePanel class in response to the ‘p’ button being pressed.

Key Handler

- Added this class to manage the detection of keyboard inputs and store them to be read by the player, or to allow the user to navigate through menus. This change modularized keyboard input handling. Modularizing this functionality simplified code in the Player class

Restructure the Map Manager into TileManager and delete Crate class

- The map manager was renamed to TileManager in order to reduce confusion when following along with the guide
- Instead of having crate objects on the map, we restructured the TileManager to hold Tile objects in an 2D array representing the map tiles. These tile objects could have a variable, collisionOn, used to check if a certain tile had collision properties. Tiles with collision turned on effectively became our new crates. This change was intended to simplify interactions between the player and the map tiles.

Collision Checker

- Since the crate class was deleted, a new CollisionChecker class was created to modularize the function of checking collisions with map tiles by interfacing with the Player class and TileManager class
- The CollisionChecker also modularized the detection of collisions between the player and SuperObjects

Removed Map Entity Class

- This class was initially conceptualized as a way to provide a common interface to draw all visible assets on the screen by storing all assets into a single data structure. However, we realized that the order assets were drawn in was important. For example, the player should be drawn on top of tiles and objects. To better handle drawing priority, we replaced the Map Entity class with a paintComponents method in the GamePanel that drew all assets with the correct priority.

UI Class

- This class renders menus, score displays, and other UI elements. We realized that we did not account for this functionality when designing our class diagram and added this class to perform UI rendering.

Changed Exit class to be a subclass of SuperObject

- Exit could be made a subclass of SuperObject, which defined a common interface for objects that could be picked up by the player. When the player picked up the Exit object, it would change the game state to win instead of changing the score
    - This allowed the collision system for objects to be reused for the Exit class, reducing unnecessary code.

We also renamed various other classes, methods, and fields. This was done either to make the names more descriptive or to reduce confusion when following along with the video tutorials used in our first sprint.

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

As stated in our overall approach, we used a management style reminiscent of an Agile process model. Alex was in a management role and in charge of setting our sprint goals and ensuring we stuck closely to fulfilling the project specifications defined in phase 1. Each member of our group chose a subset of our sprint goals to complete during each of our two sprints, and pushed to the repository when they completed a goal.

Responsibilities for sprint 1:

Alex (parts 1,2,3 of the video series):

- Set up an initial pom.xml file
- Create a window display
- Set up player movement with WASD
- Create a player walking animation

Brandon (part 4 of the video series):

- Create a tile system to render the floor tiles of the game

Aaron (part 5 of the video series):

- Create a camera system that follows the player
- Modify the tile system to render different tiles in response to player movement

Nathan (parts 6,7,8 of the video series)

- Create a collision system that blocks the player on wall tiles
- Create an object system that renders and lets the player pick up objects
- Create a UI system that displays messages
- Create a state system to finish the game

Responsibilities for sprint 2:

Alex (goals 1,2,3,4,10,13):

- Create a score system
- Create keycard, battery, and EMP objects that affect the score
- Set the initial map configuration

Brandon (goal 5):

- Create guards that follow the player and trigger a game over state when a collision is detected

Aaron (goals 6,7,12)

- Handle the game-over scenario and create a game over menu
- Set the final sprites for the player and guards

Nathan (goals 8,9,11,14)

- Create start, win, and pause menus
- Adapt the chest object into an exit

List external libraries you used, for instance for the GUI and briefly justify the reason(s) for choosing the libraries

We only used libraries that were part of the Java standard library in our code. These lightweight libraries were recommended by the video tutorial series we consulted during the

development process. Our application did not need advanced functionalities that were unable to be provided by libraries from Java's standard library. Only using libraries from the standard library increases platform independence and ensures these libraries will be well-supported. We used the following libraries for these purposes:

- javax.swing - used to create a window for our application
- javax.imageio - used to load image data
- java.awt - used to display images and text, and to handle keyboard input events

Describe the measures you took to enhance the quality of your code

We adhered to common clean code conventions such as appropriate indentation, capitalization, and line-break separation, as well as writing all class attributes ahead of all methods. We also added Javadoc comments for all our classes and methods, and also included comments inside methods and around attributes to clarify the behavior and intention of our code. We also renamed variables or classes to better reflect their functionality, and restructured or cleaned up sections of code if we felt they were disorganized.

Discuss the biggest challenges you faced during this phase

The biggest challenge we faced during this phase was understanding how to get started. None of our group members had worked with Java UI libraries before or had any Java game development experience, and were uncertain in what order features should be implemented. However, following the implementation structure of a video tutorial series for the first sprint helped to clarify the most important structural features of a game in Java and gave us an opportunity to build our confidence. In the second sprint, we had developed sufficient confidence and understanding of the structure of our project and no longer needed to consult a tutorial.

Another challenge we faced during our first sprint was ensuring all our code worked together and compiled without errors. Since we each worked on our sections separately, we had to deal with merge conflicts and compatibility issues between our code. Nathan made a final pass over our code and resolved these compatibility issues to ensure our code worked by the end of our first sprint. However, in the second sprint we learned from this challenge, pulled more frequently, and identified dependencies and waited for them to be completed before working on dependent code sections. This significantly reduced compatibility issues with our code.

Understanding the structure of our code was another challenge we faced. Since we made heavy modifications to our internal project structure compared to our UML class diagram in phase 1, consulting the diagram was not particularly useful to better understand our actual project structure. This meant we had to ensure our code was written cleanly and actively communicate changes or ask for help when needed to ensure everyone learned and understood our project structure.