Phase II Report

Project Details:

Project: 2D maze game in Java - Survive in the end

Development team: CMPT 276 - Group16

Team Members:
- Risa Kawagoe,
- Rongsheng Qian,
- Anika Sheikh,
- Sibei Zhou

Development Tools:
- intelliJ IDE with JDK
- GitLab
- Maven

Project Deadline: March 19th 2022

Overall Approach to implementing the game:

Before our implementation of the game, we had several meetings discussing our designs and UML from phase I, ensuring every member of the team was aligned with our design and goals for the project. Upon clarifications of our design plans, we also discussed a unified method for version control to be consistent with each other's code and to avoid large merge conflicts.

In the meeting, we also came to an agreement to meet twice a week as a team to catch up other team members on our work and used git branches, however, we also had met outside of those hours in case of bugs that urgently needed to be solved. Since meeting in person is not always the best depending on each members' situation or schedule, we tried to have in-person meetings only when we had the necessity to go in-person such as showing diagrams and sharing information that required physical presence or when it was too inconvenient to do it online otherwise.

We decided to divide the implementation process into smaller and manageable tasks in a way that they align with each feature of our game. This was done to lessen interdependence on each other's code giving us more time to progress on our feature independently. This also created smaller if not no merge conflicts as most people were working with different classes.

When developing, we first started by implementing the interface and class constructors initially needed for the features. We also made some To-do comments before implementing. This gives the entire team a good idea of what needs to be done in case we needed to switch tasks or needed help with any bugs. Each of us made different branches when working on our features separately. Upon completion of those tasks, we merged it back to a unified branch that contains all our work, which then we merged to master after final revision. After our main features were done, each of us tested overall performance for basic requirements of the program and looked for any basic errors or bugs.

We did this individually to ensure increased coverage on code. Lastly, we looked over our own and our team member's code to ensure quality and good or any lack of comments.

Changes to UML:

- Title screen buttons originally: start, reload, exit Changed to: start game, change level, exit
    - This was done to decrease complexity of the game given the time constraint
- Did not use Wall Class
    - Since our character can partially collide with the wall (i.e., head can go in the wall), it was not enough to just check for walls. We had to check for collisions with certain parts of the character and the wall. Therefore, we added a collisionChecker to check for such cases.
- Added new classes needed to enhance game quality and meet certain requirements
    - Music.java added background music and sound effects
    - Tile.java for importing tile images to create maze map
    - TimerClock.java for implementing timer inside the game
    - inputKey.java for detecting required keyboard inputs
    - checkCollision.java for detecting collisions between the main character and walls, also between zombies and walls

Management Process:

First thing we did is to divide the entire implementation process into parts so that we can assign responsibilities to each member. However, after a few days when we started implementing the program and writing the code, we realized that we did not divide the task into parts where each of the tasks does not interfere with other parts.

We divided the implementation task into parts: opening screen, display game map, character display, game screen frame(ph, vaccine, time), animation and GUI design.

For each part(each person's role) we created a new branch before merging into the main phase2 branch(set_up_phase2). After merging separate branches for partial development/implementation we merged them back into the main branch(master).

In addition to discord as our main communication tool, we used Google doc to share and synchronize our information regarding the project. In the Google doc, we included deadlines given by the professor as well as deadlines that we set for ourselves such as basic class implementation deadline, main frame of the game implementation deadline, basic debugging deadline, interface implementation deadline, etc. so that every member is aware of the development timeline.

The following shows the specific division between each team member:

- Risa Kawagoe:
    - Implement title, change level, tutorial, background story, win/lose screen
    - Implement the program's management of switching between different game states

    - Add music and sound effects
- Rongsheng Qian:
    - Implement Zombie character
    - Implement vaccine and HP feature
    - Implement winning/losing condition of the game
- Anika Sheikh:
    - Implement and draw maze map
    - Add collision detection for main character
    - Add timer in game
- Sibei Zhou:
    - Create all GUI for game
    - Create animation (Main character, Zombies, Mystery characters)
    - Implement Main Character

Besides the tasks above, all members took part in debugging and refactoring of code.

External Libraries used:

- java.swing (https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html)
    - the basic library for graphical interface, lightweight
    - most common, therefore, easier to find resources for it
- java.time (https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html)
    - Used to start and stop timer in TimerClock class which implements timer for the game
    - used in DynamicCharacter.java for moving the character in the game
- java.awt(https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html)
    - used for buffered Image to load GUI images and icons in game
    - most common, therefore, easier to find resources for it
- javax.imageio(https://docs.oracle.com/javase/8/docs/api/javax/imageio/ImageIO.html)
    - Used to read GUI images and icons from picture files into java code
- Javax.sound.sampled(https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/package-summary.html)
    - Used to import and play music and sound effects in the game

Measurements to enhance the quality of code:

We had several meetings to select unified code and documentation style. This helped maintain consistency throughout our files, which later also helped faster and more efficient debugging. We also made sure to add sufficient comments including To-do comments which helped us be aligned with each other's code and to be more organized with our code. To ensure consistency in code, we also made sure to use universal language for variable and function names, such as button instead of btn. Furthermore, we had two weekly meetings discussing our developing stage. This helped us refactor any chunk of code that might be out of place.

Biggest Challenges:

Since the main features of this game are very interdependent, initially we faced the difficulty of organizing the entire project and working on the project as a group and not as individuals which is what we were used to from our typical course assignments. Therefore, we had to meet several times after we started implementing to overcome the lack of organization.

After we started implementing/coding, we started to realize tasks that were missing in the designing/planning phase. For example, the UML from phase1 only contained the actual game itself(maze). We realized that the actual game is not entirely the game, instead a game requires a title screen

Given the different course load of our group from other courses, we also had trouble with estimating and scheduling the project development. This was difficult mostly for individuals whose parts were dependent on each other.

Each member had a different working environment (laptop, operating system, etc) so it was hard to unify the environment the program ran on. In the end, we decided to all use IntelliJ IDEA to run our program, bettering the consistency of our code and pom file.

In the beginning, most of us were not used to using the version control system(git) so we tended to update all the changes to the main branch. We decided to create a unified system on updating code on git which we followed to avoid large merge conflicts or any loss of work.

Ambiguity in the specification caused uncertainty in what we were trying to implement, more specifically, coding. We discussed frequently in discord on any confusion we had to clear this up, and we also tested each other’s code functionality to avoid this.

Furthermore there were several issues that we did not notice until we tried to play the demo version of our game. One of the things we noticed once we started playing the game was the lack of instructions that stands out to users. When we had some of our friends test out the game, some users found it confusing on which keys to use to move the character. Although we had put this instruction at the starting screen, it was not eye catching to the users leading to user confusions. This made us realize the difference between developer’s and user’s point of view when playing the game.