Phase 2 Report - Group #18

Overall approach to implementing the game

We approached implementing the game by assigning different sections to different members to get an idea of how best to implement them, then communicated any difficulties or changes in the plan with the group. We created classes for the objects contained in our UML diagram, but many needed to be added/changed/replaced when it came to actually implementing the code.

In the end, we build the code modularly based on the capabilities of the game. For example, we used borrowed placeholder sprites and implemented the player and movement first, followed by creating a background map, implementing collision with the background map, and then started working on items. We maintained a list of remaining tasks to be completed, scheduled different working times, and communicated which portions we were working on/completing at any given time in order to avoid conflicts.

We later created sprites and substituted them for the placeholder ones we had been using previously.

Adjustments and modifications to initial project design

While we had initially made java files for the classes in our UML diagram, we ended up re-making them to fit the design of the game once we had started using it. We had assigned different members to the class creation and the game structure creation and this resulted in some compatibility issues between the two. In the end, we created a game panel/screen (implementing JPanel) and took a more top-down approach as opposed to the bottom-up approach of making individual classes (eg. ‘key’) first. We built the rest of the game around this panel, starting with parent classes and manager classes that keep track of instances of the individual objects. The items came after, and some of the organization of the classes was modified as well. Elements such as traps and walls were originally planned to be generic ‘piece’ elements, but during implementation we found that traps would be easier to implement as collectible items and walls could simply be background tiles with a player collision check on.

Optional elements such as weapons and health restores were pushed down our priority list as we attempted to include all of the required elements first. While settings such as difficulty weren’t implemented as we had originally planned, we were still able to create a pause screen as well as end screens for either successfully completing the game or having the player lose all of their health. Other maps/levels as mentioned in phase 1 have not been implemented yet but could be added at a later date.

The names of some classes were also modified as their purposes were tweaked to fit the new system we were building - squares became tiles, pieces became entities and items, and most of the game is implemented in its own class as opposed to through map.

Management process, division of roles and responsibilities

We created a Discord server to communicate amongst ourselves throughout the project. Initially, as mentioned previously, we assigned different chunks of the project to different members: Aidan was in charge of creating the classes from the UML diagram, George was in charge of dealing with user inputs, David was in charge of implementing the structure for the game and map, and Peter was in charge of setting up Maven, keeping the files organized, and incorporating graphics.

In reality, this was not how the tasks were completed. George realized that individually working on these sections would make combining them difficult, and he was able to set up the basic structure of the game for us to build upon (screen, map, tiles, etc). As mentioned earlier, once this structure was established we maintained a to-do list with remaining tasks. We organized schedules and allowed each member to pick up and work on any remaining task, as long as we mentioned to the chat who was working on what and when anything was completed or pushed to the GitLab. Peter made significant progress implementing classes and expanding the project, David did the majority of the UI development, and Aidan created the sprites while also working on adding assorted functionalities such as traps and the game-end screens (victory & game over). This is only a rough summary of each member's tasks since the design became a much more collaborative process in the later half of phase 2, and many sections were developed with two or more members in a voice call doing pair programming.

We had met after classes several times during phase 2, but began communicating more commonly and thoroughly in the latter half. We held a meeting online and reorganized ourselves moving forward, resulting in much more effective teamwork.

External libraries used and why

We used JPanel to create the panel within which we built the game. Java's Abstract Window Toolkit (AWT) was the source of most of our other external code. We used BufferedImage (along with ImageIO, which was not in AWT) to read and display the png sprites and Graphics2D to render the sprites and manage colours and text. Rectangle was used to create hitboxes to determine when the player collides with a wall, item, or enemy. KeyEvent and KeyListener were used to read user input.

Measures taken to enhance code quality

We used parent classes and subclasses to simplify the classes lower down on the hierarchy and decrease repetition. Manager classes were also used to separate the use of each class from its implementation and keep the code more organized. As mentioned above, pair (or group) programming was used at times to catch errors as they were produced, and the self-driven method of allowing all members to work on whatever was needed during their free time allowed all members to interact with many different aspects of the game, leading to more thoroughly reviewed code (and more knowledgeable group members). The use of clear function, attribute and class naming also increases the readability and maintainability of the code, alongside the comments used.

Biggest challenges faced during this phase

We faced several challenges during this phase including communication (specifically early in the phase), overwriting code with GitLab, and deciding which features to include/omit. In the first week or so of phase 2, minimal work was accomplished and what *was* accomplished was not effectively communicated to the group, leading to the aforementioned restart. After we met as a group for a couple of hours to review our new plan, the entire group communicated effectively and was very active in our group chat and on our online calls.

Early in our rework, we had some issues pushing code to GitLab as we had some code conflicts, but after working together in a video call we were able to resolve our issues and re-acquire the code that was overwritten.

The question of which features to include was an ongoing discussion, with different team members favouring different features. As a group we decided to prioritize the requirements from the phase 1 instructions and will consider adding optional functionalities during later phases or after the project's submission.