Group-12 Phase 2 Report

Authors: Wilson, Daniel, Nicholas, Yuxi

Implementation Approach

Our approach to this project was to have our members working on separate branches, and to push to master once the branch was working. Each member was designated tasks, divided up by importance and effort (see **Management Process** below). Our goal was to implement the basic classes during the first week, and the more complex classes during the second week. Essentially, we wanted to build and test all of our classes independently and then combine them near the end of the development process to create the final build of the project. We tried to keep the dependencies between classes minimal to keep the code clean, independent and understandable.

Adjustments to Initial Design

Throughout the development and coding process, we came across various changes to our initial UML class diagram and to our overall plan. Although the methods within our original classes mostly stayed the same, we added a DisplayManager and GameKeyListener class. We initially handled display in GameMain with the function update, but decided to create the displayManager class to reduce clutter from GameMain.This class handled all the GUI and UI elements of our project. This allowed us to display the board, board entities and the state of the game (Menu, Pause, Game, Win, Lose). The second class we added was the GameKeyListener class, which handled keyboard inputs from the user to update player movements. Our initial design had GameMain handle the inputs, but we decided GameMain should not be handling keyInputs and it helped to reduce overall confusion. The “TickTimer” class was not as specifically defined as we had previously thought, and as a result we had to overhaul the design and determine the exact behaviour of the class. We settled on making “TickTimer” into a runnable class that would create a new thread and then call an update functions from the main class on an interval. This mirrored the behaviour of “Frames Per Second”.

Quality of Code

We enhanced the code quality by creating JavaDocs and comments for communicating with teammates through messages and online meetings on Discord. Methods that needed to be finished were marked with a “TODO” comment. All non-explicit methods have JavaDocs describing the purpose of the method, and any parameters or return items. All classes also have a Javadoc describing the purpose of each class, and its authors. All members also coded in Eclipse to ensure there weren't any unnecessary conflicts between IDEs. To ensure that all classes worked properly, informal tests were run on all of the methods and any dependencies between classes were thoroughly tested to make sure there would be no bugs. With the remaining time we had, we attempted to locate and fix any bugs that occured while playtesting.

Biggest Challenges

The biggest challenge we faced was the time constraint of two weeks. A lot of our members were busy leading up to the deadline but every member still tried their best to meet every week. Due to the time constraints, we didn’t have enough time to implement the shooting function or the function to update the HUD for the main character. Having to work with git at first was daunting. Learning to not destroy branches from merges took some time to learn. Merge conflicts were a headache to deal with, but we have become proficient with the software. Furthermore, this was the first Java program for all our group members. Learning its syntax and structure consumed a lot of time during the first week of coding. Implementing the design patterns in class also took a bit more work. Furthermore, implementing multi threading was quite challenging. We overcame most of the issues stemming from it. Miscommunication between teammates was an issue near the beginning of the project, but we became better at being explicit very quick.

External Libraries

We did not end up using any third-party external libraries during our coding process. However, we did use libraries:

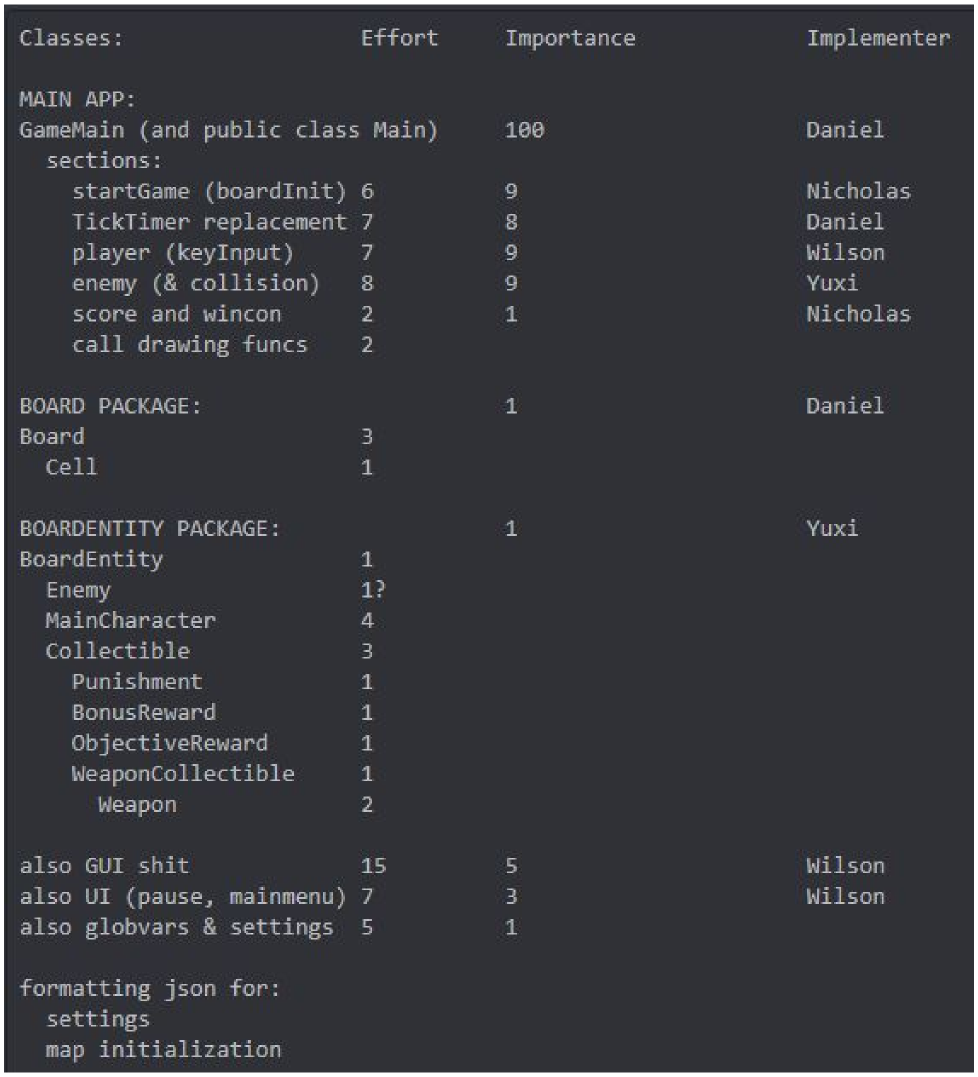
```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
Import java.awt.Font;

import javax.swing.JFrame;
import javax.swing.JPanel;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
```

We used the Graphics/Graphics2D and JFrame/JPanel libraries in the class displayManager. It allowed us to create the window for our game and provided graphics

for the menu, game, win and lose states and for the board entities and heads up display. Color was used to distinguish the character from the enemies, objective rewards, punishments and bonus rewards. Dimensions was used for creating the board entities. Font was used to display text on the various game state screens.
The KeyAdapter/KeyEvent/KeyListener libraries are used in the class keyInput to take in keyboard inputs from the user. This allows the user to move the main character, and to pause and to start/resume the game.
We found these libraries to be essential to our program

Management Process

Credit to Nicholas



```
Classes:                 Effort     Importance         Implementer

MAIN APP:
GameMain (and public class Main)    100                Daniel
  sections:
    startGame (boardInit) 6         9                  Nicholas
    TickTimer replacement 7         8                  Daniel
    player (keyInput)     7         9                  Wilson
    enemy (& collision)   8         9                  Yuxi
    score and wincon      2         1                  Nicholas
    call drawing funcs    2

BOARD PACKAGE:                      1                  Daniel
Board                    3
  Cell                   1

BOARDENTITY PACKAGE:                1                  Yuxi
BoardEntity              1
  Enemy                  1?
  MainCharacter          4
  Collectible            3
    Punishment           1
    BonusReward          1
    ObjectiveReward      1
    WeaponCollectible    1
      Weapon             2

also GUI shit            15         5                  Wilson
also UI (pause, mainmenu) 7         3                  Wilson
also globvars & settings  5         1

formatting json for:
  settings
  map initialization
```

Our initial management process followed the table above. We divided the methods and classes into tasks, ranked by its effort to implement and its importance to the game. The tasks were then divided into our strengths and personal preferences. Every class has the author/authors listed at the top for those who contributed to the code. Although we initially set meetings three times a week, as the deadline approached we met everyday.