Reflection SWEN20003 Project 2

Armaan Dhaliwal-Mcleod 837674

I made many changes to my class design from project 2A. I realised that my previous design of all the game objects inheriting from the Sprite class would not allow my project to be delegated and encapsulated properly. After looking at Eleanor's sample solution for 2A, I changed my design to include abstract classes such as *Moveable* and *Pushable.* This allowed the sprites to be group based on their roles in the game, since some sprites can move, can be pushed, or just stay still. I also created an *Enemy* abstract class to further group the enemy units which can kill the player. This was also added because all the enemies can kill the player on contact, so it made sense to create an abstract class for them. I also removed my overuse of static variables from 2A, and decided to instead pass *World* as an argument to other classes, instead of referencing *World.someMethod()* every time I needed to access a method or variable out of the World class.

The difficulties I came across in this project were a result of not spending enough time planning my class design before coding. This led to a lot of code sitting in my World class initially, and delegation and encapsulation were not evident. This led to a lot of code being deleted, or completely scrapped because it either was not delegated to the correct class, or it made no sense in terms of the design. It was difficult to spend this much time on designing classes, since most of us were so used to jumping into writing code, and providing solutions that merely "did the job". From what I noticed, this was the main difficulty I saw from my myself and my peers.

Other difficulties I ran into was using explicit frames, instead delta in my timer operations. This made calculating the correct timing of blocks/explosions difficult, and using delta was the easier option. This was a problem when trying to slide the ice block every 0.25 seconds, because calculating frames for that was more difficult and less accurate. Another issue I had was making the undo method work properly. Initially, I did not have uniform stack sizes of the sprite histories, and this led to incorrect undoing of the player and the blocks. I soon later realised that the stacks had to have the same sizes, so when you pop histories off, then the game state will be correct when undoing. Another difficulty I had was refraining from using too many static variables. I soon realised later that this just leads to a "global state" program, and my java code was just C code with a bunch of global variables. I also had difficulty choosing the correct data structure to store my sprites in. Originally, I had a 3D array of sprites, with the first two dimensions representing the map, and the third dimensions representing the levels of tiles. I realised that this just overcomplicates the code, and using a sprite array list is easier to use and manage.

The major piece of knowledge I can take away from this project is how important design is before coding. I never realised how much time you needed to spend on designing your classes beforehand, and this led to a lot of code chunked into classes where they didn't belong. This project really taught me how to think about how I want to do things, instead of just going in with both feet, and realising half way through that what you've done is not going to work. Additionally, this project has also taught me a lot of game development principles, which I really enjoyed and I hope to continue to do more projects along the lines of this one in the future.

In the future, I would not underestimate how much planning I need to do beforehand, especially in bigger software projects. I would also spend more time on designing UML diagrams, as I believe they do come helpful when designing projects, because you can see the relationships between different classes. I would also spend more time choosing the data structures I want to store data in. These were the two major issues I had with the project, and I hope to improve upon them in the future, since projects are only going to get harder and more complicated from here.