`Project 2` Reflection

Naufal Fikri Setiawan - 844027

This was the first project that I did (both school-related and not) that requires planning ahead before actually creating the project, which is a significant deviation from my standard 'code-on-the-go-and-let-it-be-broken-and-fix-it-later' style of programming. However - this planning phase actually gave me time to think "how should I implement `insert feature here`" which is a good thing.

Not much has changed since I submitted project 2A (assuming we're only talking about the base game), I designed project 2A with extensibility in mind such that "how do I extend this program by only adding as few lines as possible whenever I want to create a new `insert thing here`." This led me to the brilliant idea of creating interfaces to handle behaviors and events, (which in my opinion is not the best way to handle events), that is by making the sprite invoke a method if the sprite itself is implementing a certain interface `CollisionBehavior`, `TimedBehavior`, etc. This sounded like a smart idea until I learned that I can just implement `Observable` pattern in the lecture. However I am pretty satisfied by how I implemented it because this way I don't have to override `update()` every time I create a new thing.

There were many things that were mentioned in the later lectures that I would implement to the project if I get to do it again. The main changes I would like to do is to implement event-listening and the observer pattern properly (fun fact: I've been trying to find shortcuts to do the project by googling `event listening in Java` but all the StackOverflow answers regarding this topic is only about GUI, weeks later I figured out that it's called the "observer pattern" rather than the "listener pattern"). I had known about event-based programming for a while (due to my past exposure in C#, probably) but I had never known how to implement it myself especially in Java. (All I know for the entire time I've been programming is, if I want something to happen in an event in my program, all I have to do is click at the GUI button in the GUI editor and it will automatically create an `onButtonClick(Object sender, EventArgs e)` method for me that I can fill in with my own code and that method will automatically be invoked if I click the button - without having any understanding of how it works behind the scenes in the first place). The way I implemented "event listening" in this is pretty "hackish" and I think should be improved.

In this project I also learned that some design decisions may be seen as really glorious until we run to just one problem with it. For instance, I intended to implement undo the most trivial way possible: to create a copy of the world on each player step and revert the world back to a previous version when undo is invoked. However, this requires me to implement deep copying and copy all the sprites and requires me to implement a `copy` method on the `Sprite`. However, since my implementation of `Sprite` is rich of attributes, deep copying is really tedious and I had to implement undo using a coordinate history instead. I find this a fair tradeoff due to how my implementation of `Sprite` turns out to be really extensible - however, I learned about design tradeoffs right here.

Still, even though its not close to perfect, I am pretty satisfied with how this project turns out to be due to my opportunity to apply the things I've learned in class and my ability to extend the project by this much. I am pretty sure I've spent more time being delighted that a feature I implemented is working rather than doing the actual programming itself. Object oriented software allows me to see programs as "related forms of data connecting and interacting to each other in a more organized way" rather than just a bunch of code trying to solve a problem - and most importantly this helps me debug my code in a more structured way rather than panicking as if it was a C program. Because when a part of a program doesn't work we can simply debug the class that is related to the behavior of the not-working properly object.

To sum it up, object-oriented principles taught me how to see and design software as abstractions of interactions between objects, allowing us to see and debug software in a really organized way in comparison to functional programming where the data is simply a cluster-mess of composition of thousands of functions. I'd rather code in Java or C# because C literally has no class. While there are few difficulties that I encountered in working on this, these difficulties taught me the tradeoffs in design and how to improve implementations of objects in my game. Obstacles in software development carry a lesson behind them, and as programmers, we have to `catch` them whenever we fail to `try` something.