Alex Ayala
Software Engineering
10/17/18

Chips Challenge Reflection and UML Changes

Reflection:

For my Chip's Challenge implementation, the user can use the arrow keys to move Chip around or press the escape key to quit the game. There are collectable chips and colored keys that Chip can pick up. There are also wall tiles, colored key walls which unlock if Chip has a corresponding key, and a tile that blocks Chip unless he has all of the chips on the level. There is also a portal tile that will end the level and start the next one. I implemented loading levels by using text files and adding tiles based on corresponding characters in the text file. I made the ChipsChallengeMap a singleton class since I didn't think there was a good reason to ever have more than one map at a time since you can and should only play one level at a time. Secondly, I used Collectible and Unlockable abstract classes since there were some similarities and small differences between Key and CollectibleChip and between KeyWall and CollectibleChipBlocker respectively. Specifically, the small differences came in how to display a collection message and how to be unlocked respectively. I also used these to help implement the strategy design pattern. I chose to make the ChipsChallengeMap an observer of the Chip class because I also had the ChipsChallengeMap manage the loading of maps, maintaining the grid, and maintaining the unlockables and collectibles still active. As such, I needed the map to know when Chip moved so that it could see if it needed to update itself because of Chip's moves. To help with that, I had the Chip class keep track of whether or not the spot Chip tries to move to is blocked by an Unlockable, wall, or otherwise. This was stored in the blockingWallType attribute.

If I started from the beginning again, I would have probably been more intentional about conforming to the SOLID principles by doing things such as including more abstract classes or interfaces. The problem I always run into is that I tend to not think about how I could possibly ever reuse code and fall into the bad habit of just writing it well enough to make it work. I could have also encapsulated reusable components more. For example, I could've had a base stream reading class and had a subclass dedicated to reading files for a ChipsChallengeMap. However, I think I did a better job of making this implementation have less coupling than my implementation of Columbus. The other thing I would've probably done differently had I started again was include a more generic abstract tile class which could have handled the image and location details which were important for all tiles.

UML Changes:

The most significant changes to UML are that the CollectibleChip and CollectibleChipBlocker classes were added as I didn't implement them by the intermediate deliverable. Also, I added the Collectible and Unlockable abstract classes. This was so that I could implement the Strategy pattern and code less to an implementation. Also, ChipsChallenge no longer has a reference to a ChipsChallengeMap because I felt like it was unnecessary since I made ChipsChallengeMap a singleton class.