Report

1. Control Flow:

**A description of the control flow for the interaction of a player avatar and a bank square. Where in the code is the co-location of the two objects detected, and what happens from that point until the interaction is finished? Which functions of which objects are called and what do they do during the handling of this situation?**

-The move() function in StudentWorld.cpp iterates through the list of Actor pointers each tick to get each actor to doSomething(). During the PlayerAvatar’s doSomething(), the player shifts across the board according to the number of rolls it obtains. The player is in the walking stage from the time the die is rolled to the time its die roll reaches zero again.

-During the bank square’s doSomething() function, each bank square accesses a pointer to StudentWorld via the getWorld() function, and in turn accesses StudentWorld’s Peach and Yoshi pointers. From there, bank square accesses each avatar’s X and Y coordinates at that tick using getX() and getY(), and compares the coordinates with its own. If the coordinates are equal, co-location of the two objects is detected.

-Assuming a match in avatar’s and banksquare’s coordinates, If the avatar is in the walking state (known using Actor’s getStatus() function, the bank square calls the deposit() function which decrements the Avatar’s coins by at most 5, depending on whether the player avatar has less than or equal to 5 coins. The amount deducted is added to the central bank located in the StudentWorld class under the private variable m\_bank using getter and setter functions getBank() and setBank(int set). The bank square accesses these functions using the Student World pointer. The deposit sound is played.

-If the avatar is in the waiting state, the bank verifies if the Player Avatar is a newly landed player using the Player Avatar’s isBankNew() function. If so, the bank uses the withdraw() function, which increases player avatar’s coin count by m\_bank amount, and resets m\_bank to 0. The withdraw sound is played. Bank square then uses Player Avatar’s setBankNew(false); function to indicate that the Player Avatar is no longer newly landed (tracked by a boolean member variable in the PlayerAvatar class), the doSomething() function having already been performed for this turn.

-The interaction is finished and the iterator moves on to the next Actor\* in the m\_actorlist list.

2. What I didn’t finish:

-I did not have time to comprehensively test out all edge cases such as whether a player will be teleported again if teleported to another place. The cases that I discovered I didn’t have time to test will be discussed in my assumption section.

-I did not have time to verify if my players/baddies are in the right orientation/walk direction when teleported.

-Did not have time to condense my code, and so there is code smell.

3. Assumptions/Ambiguities:

-A fork will not be any square that activates while player is in the walking stage. This is because the fork by definition activates when the player is in the walking stage and causes the player to stop or stall (but still be walking) on the square for more than a tick. If the fork square were a Star/Bank square, it would cause continual executions of the square’s function that plays when a player passes over and stalls there.

-It was not completely clear how a player reacts to being teleported onto the start square during the event square’s swap function when the other player has not yet moved even its first round. At the start of the game, the player does not treat the starting corner as a fork. When teleported back, the player treats it as a fork as there are 2 pathways and the user is free to decide which way to go. It is assumed so as there are 2 pathways from the perpsective of the corner square, and the player is prompted for an action (with either direction being valid) because it did not necessarily come from the direction that is now an alternate pathway.

-When a bowser lands on a directional square and converts it to a dropping square while a player is standing on the dropping square, the program continues to treat the effect of the space on the player as that of a directional square instead of registering that the player is now at a fork. The spec did not specify if the latter or former case is recommended. Hence I implemented the former, since the player’s direction is set the moment the player passes over the direction square before it was destroyed.

-I don’t know what happens when a player is teleported from an event square to another event square. I don’t know if it may be teleported again, so I just assumed it may be. When teleported, the player may be teleported again. However, when swapped or given a vortex, the player will not be swapped again or given infinte vortexes.

-