1. There are two primary forms of interaction between the bank square and the players. One is when the player is walking over the bank square, and the other is when the player lands on the bank square. The control flow of the bank square consists of the bank square checking if a player currently has the same position as that bank square. If the bank square does not share a position with a player nothing happens. If the bank square shares a position with a player, it then checks whether that player is currently walking or waiting at that point. If the player is walking, it then increments the world’s central bank account by 5, and decrements the player’s coins by 5. If the player is waiting at that point, it gives all the coins in the central bank to the player, and sets the central bank’s value to 0.

The co-location is detected inside the bank square, in its doSomething() function. All the bank squares’ doSomething() functions are called when the vector containing all actors needed for the game are iterated through.

All the bank squares’ doSomething() functions use several functions in order to implement the functionality needed to complete its process. First a function getWorld() was used to get a pointer to the StudentWorld class, and within the student world init() function I initialized all the actors needed for the game. Based on having a pointer to the StudentWorld class, we could get a pointer to the array of pointers to the two players using the getAvatar(index) function to access the players. The players class which defines both peach and yoshi, was inherited from actors, which was inherited from GraphObject. And therefore by being able to generate a pointer to the players' class object we can also access the different functions of GraphObject needed to be able to execute the desired result of the bank Square’s doSomething() function. From graphObject I used the getX(), getY() functions. By using getX() and getY() I could determine whether one of the players was at the same location as the bank square. I also defined my own functions in the player class such as setFirstAttempt(), getStatus(), setStatus, getFirstAttempt(), getCoins() to give the player certain functionalities that I wanted it to have. setFirstAttempt() and getFirstAttempt() were used to set and retrieve a boolean value that would determine if the object was at that specific location for the first time, and when one of the player’s was waiting at that object it wouldn’t cause the result to process many times. getCoins() and setCoins() were used to control the player’s coins, and getStatus() and setStatus() were used to indicate that the player was either in a walking or waiting state.

1. I believe that I did complete all the functionality as specified by the specifications. Of course, there still may be bugs, but I haven’t seen any current known bugs.
2. One assumption I made was that it was not specified whether when the player is teleported by the event square if it would cause the square it landed onto to activate. So I assumed that the square landed on would process as if the player had just landed there, and was in the waiting state. In the spec it wasn’t specified on the particular sound conditions when a vortex collided with a baddie. So I assumed that if a baddie was hit by a vortex, it would play the hit by vortex sound, as specified by the spec, but it wouldn’t play the teleporting sound(assumed part).