1. A description of the control flow for the interaction of the Player and a goodie. 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?

A screen shot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

Both goodies are derived from a class called goodie, goodie is derived from base actor class. In the dosomething() of goodie, after I detect if the goodie is alive or not, I detect if the goodie is in the same block as of player. I use a non-virtual bool function called onPlayer that is in the actor base class and takes x & y coordinates. I input the goodie’s x & y using getX() and getY().

I have a pointer to studentworld in actor class, a **function1** that returns the address of the pointer to student world in actor class, a pointer to player in studentworld, a **function2** that returns the address of the pointer to player in studentworld, and a bool function isAt (Actor\* ptr, **int** x, **int** y) **const** that checks if the ptr passed in is at the coordination (x, y). So, in onPlayer, if I call isAt using student world pointer and pass in address of player (**function2**) via **function1**, isAt will return true if the goodie is in the same block as the player, otherwise false.

So if the goodie is on player, I called powerUp() that is pure-virtual in goodie. I implement powerUp() in each of the two goodies. For example, in extra-life-goodie, I called increaseScore (50) and incLives() using **function1**; in garlic-goodie, I called increase (25) in the same manner, and I also called to incBurps (5) via **function1 and function2** as incBurps () is a non-virtual function I implemented in player that update burps accordingly. Basically, the dosomething() of goodie will call the most up-to-date powerUp() function. After powerup, I set the goodie to death, which is a non-virtual void function in actor that turns a private bool member variable m\_isDead from false to true. So that dead actors will be deleted at the end of this move. At the end, I play the sound got-goodie via **function1**.

1. A list of all functionality that you failed to finish as well as known bugs in your classes, e.g. “I didn’t implement the Koopa’s freeze logic.” or “My Bonfire code doesn’t actually remove Barrels from the maze.”

**I think I fulfill all functionality**.

But for the koopa, when the koopa freezes the player, koopa immediately moves to the next grid, so the time it stays on the grid that it freezes the player is shorter than normal moveing rate.

1. A list of other design decisions and assumptions you made; e.g., “It was not specifiedwhat to do in situation X, so this is what I decided to do.”

* One fun fact I found was that you don’t need to move the player to the same floor as kong stands on. As long as the distance is close enough, a jump below kong may make you pass the level.
* In actor, I add a series of characteristic-determination function that helps determine if something is a floor, isa ladder, is fire proof, can be kill by burp…these are used in student world to figure out the type of class I want to use.
* In student world, I add functions to reflect/change condition of member variables, return member player pointer, and other functions to simplify code writing and avoid redundancy. Inside the private, I add functions that update display text in move(). I made them private because these functions are not used in other places.
* In student world, I also add functions that can add new burps and barrels into the actor pointer vector, which are not added in init (). These two can be used when appropriate.
* I also add a private member variable to detect if the current level is finished, which is initialized to false. Once it is detected to be true, I need to reset it to false, otherwise subsequent levels will automatically be finished.