Ali Alrubh
Final Project
8/8/2016

Final Project:

How to play:

The game will start with the player and 1 swan. After every 30 steps the player makes a new swan will appear. To move use w,s,d,a . To quit the game, enter q. To use an apple, enter e. To go to the next floor step on L or X and press u. The maximum number of floors is 3. And the floor name must be floor_#.txt

Reflection:

The final project is a maze where you can move the player around and avoid obstacles and swans. The symbol P is the player and S is the swans. When the swan touches the player, the player will be sent to the start position (E). The swan will not affect the player who consumed an apple. The player needs to reach the L or X in the map to finish the game.

The program consists of 4 essential classes. A game class, and an Actor, which is an abstract class that player class and swan class inherit from. The Actor class has a virtual move function that will move the actors around the maze. It also has some variable like the position of each actor. The player inherits the move function from player. It takes command from the user to move the player around the maze. W for up, S for down, D for right, A for left. To move up 1 will be subtracted from the row of the position of the actor. To move down 1 will be added to the row. To move left and right 1 will be added or subtracted from the column position of the actor. The swan move algorithm is similar to the player algorithm, but the swans will ignore the keys and apples that lie on the maze floor. The player has more command such as u to use the ladder or E to eat an apple to avoid getting caught by the swans.

The game class consists of the functions needed to read the floor and print it. The maze will be read from a text file. Then, it will be saved in a Dynamic 2D array. The maze will start with one swan only along the player. The game will be organized

on a function called loop, which is the main function of the game class. It will call the move functions of the player and swans and it will also consist of the code that will spawn the swans. The command from the users will also be taken on the loop function. Finally, the maze function will organize how many floors do the player wants and link the floors with the rounds.

Error handling and testing:

- When the player and the swan meet, the player will be sent to the starting position. I wrote the send code on the player move function. The result of that is when the player meets a swan it will work, but when a swan meets a player it does not. To fix that I wrote a code in the swan move function that will move the player to the starting position.
- When new swan spawn after the player made 30 steps. I used if statement to check if the position where the swan will spawn was empty which result a core dumped error sometimes. To fix that, I replaced the if statement with a while loop.
- To move the swans. I use a 2 nested for loops to find the position of the swan, then use another for loop to move them. This code only moves the first swan that the first two loops find. To fix that, I moved the swan move function inside the first two loops.
- Before checking the input. I get some errors when I input commands. The errors were fixed after doing an input check.
- ./maze 1
  No errors
- ./maze 2
  No errors
- ./maze 3
  No errors