Ali Alrubh	Assignment 4 & Final Project design	7/30/2016

Assignment 4: The required is to build a maze game with multiple floors and move the player to get out of the maze

P is the player, S: is swan (The enemy), K: Key, D: Door, A: Apple, L: Ladder, E: starting point, X: Exit (Ending point)

The program will take input from the user.

input: # of levels, w: up, a: left, s: down, d: right

e: eat apple, u: use ladder, q: quit.

output: the output will be the maze with any updates happens on it (player or swan moving, taken keys and apples, open doors)

1st part of the assignment: (Read Floors and Prompt the user)

* Reading Floors:

- The floor txt file begin with 2 numbers. These numbers will be rows and columns.
- A 2D Dynamic char** array will be created with the row and columns in the floor.txt file.
- each element will be stored in the array except the 1st 2 numbers
- input check:

a loop will loop through the element of the arrays to check the characters are valid or not.

- After checking the input the maze will be printed

* Prompting the user:

- Ask the user how many floors to use.

Open the floors in order and print them in order after checking for invalid characters.

There will be a game class where the game is occuring. There will also be a floor class where the floors array will be saved. And Finally there will be an abstract Actor class with 1 pure virtual function for moving the characters. A player and a swan classes will inherit from the Actor class.

- Player Class:

Based on the floors design, the player must start at 'E', which is the Entrance. to do that loop through the array until the position of E is found. Then spawn the player in the position next to E. The floor will be designed in a way that 'E' has an empty space next to it.

To make the player move, find the location of the player in the array P[row][col], then:

To move up enter w → P[row+1][col]
to move down enter s → P[row-1][col]
To move left enter a → P[row][col-1]
To move right enter d → P[row][col+1]
before making the move the new position will be checked, if there is # the player position will not be updated and if there is D and the player does not have a key, the position will not be updated.

There will be a counter to count every movement the player makes to determine when to spawn a swan and when the consumed apple effect wears off.

There will be a variable for apples and keys, when the player location be the same as one of them, the item will be removed and stored to the inventory. if the inventory is full, the player will be unable to take them.

- Swan Class:

The swan movement algorithm is similar to the player but it will be different a little.

After the player move ment counter reaches 30, it will be reseted and a swan will spawn in a random empty location in the maze. the swan movement mechanics is similar to the player but it will be organized. it will be designed to move 5 steps right and 5 steps left or 5 steps up or 5 steps down.

* The player and swan will both inherit their movement from an abstract base class called Actor. and the movement function will be a pure virtual function.

* This is for the player class: when the player is one step around the Ladder 'L' or the exit 'X' from any side u is used to proceed to the next level.

Game class:

In the game class, there will be a big while loop that will organiz the player and the swans movement functions. it will connect all the clases of the program together. the loop will continue looping until the player finish the game or enter q which will break the loop and end the game.

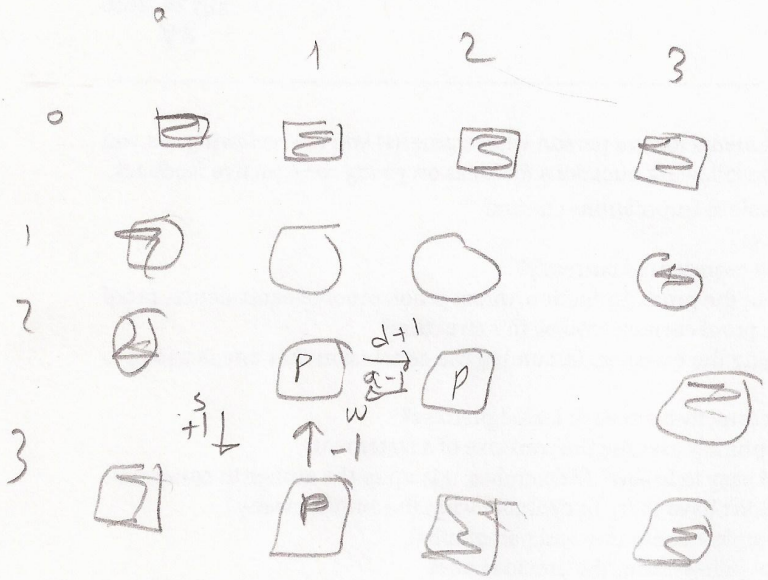
* when the player reaches the ladder and uses it, the current maze will be deleted and the player will proceed to the next maze. all the counters will reset

The program interference:

Enter the number of floors:

Play until win or quit.

Movement Mechanics



[3][1]

w [3-1][1] = [2][1]

[2][1]

s [2+1][1] = [3][1]

[2][1]

d [2][1+1] = [2][2]

[2][2]

a [2][2-1] = [2][1]