Khoa Phan
CS162, Final Project
June 11, 2017

<u>Program Description</u>

This project is a text-based game that was given the freedom of design. There were specific requirements such as needing at least four pointer variables that linked to other spaces. They can either link a different object or NULL. Another requirement was to have at least six spaces of at least three different types. These spaces must be derived from an abstract space class. A container of some sort, like an inventory, was needed to carry items that became part of the solution for the space. A time limit must be defined as the game should not go on forever.

For this design, I chose create an escape-room idea. This is where the user has to solve a few tasks in order to move on to the next space. Once the space's task has been completed the next door will open. The eventual goal is to escape, or leave the rooms until there are no more rooms before you run out of energy. There are circumstances in which energy can be lost quickly if the user does not avoid them.

Here is the list of rooms within this game and the unique traits for each one:

Room	Trait
Starting Room	*exception room as it starts as an empty space with four doors. It serves as the starting 'hub'.
Power Room	Find a battery to open the door to the power The power must be turned on in order to advance to the next room
Arrow Room	The door closes as user enters room. Machines shoot arrows across. Getting hit will reduce energy. These machines must be destroyed before the user can advance. A WRENCH can be found to break them.
Spike Room	Spikes raise from the ground causing damage if the user is nearby. Collecting a crowbar will drop a key that the user must retrieve to open the next door.
Zombie Room	4 columns with switches must be activated in order to unlock the door. Zombies roam the room randomly and if the user gets too close, they will attack.
Wall Room	The walls in this room close in on the user. If the user has enough energy, they can easily escape.

<u>Plan for Class Objects</u>

There exist an abstract Room class that each room will inherit from. It will contain variables that apply to all rooms or are needed to link other rooms together.

Notable variables include:

> cols, rows, Door structure with 4 doors (dNorth, dEast, dSouth, dWest), Player p with directionals (pNorth, pEast, pSouth, pWest), key, and door locks (powerStatus, spikeLock, zombieLock).

Notable functions are:

Void control();, void interact();, void printRoom();, int traverseRoom(); virtual void action();, Virtual void SetConnectedRooms();.

Each Room has unique actions within shown in the chart above. They will all the use the same base map, but will also create objects within. Each room points to a different set of rooms. This is where setting them allows them to be linked. Traversing a room needs to be unique to the room, just as the setting connected rooms are in order to go to the correct space. Interaction is also a big point, as each room has different interactions depending on the space. Information for the class can be seen in the chart above.

Class: Player

The player class contains all the information about the user's character. They hold the variables that identify location within the map space. An inventory is given to the user to hold the items that will be picked up. The last important variable is the energy. The user is given energy that reduces for every step and action that they take. The only thing that does not reduce it is checking the inventory.

Class: Inventory
Inventory holds the items the user picks up.

Class: Items
This is the base class for the items that will be picked up. It holds a name for the item, an icon and if they have the item or not.

Item Type	Description/Use
Battery	This is used to power the doors that lead to the power switch within the Power Room.
Crowbar	Picked up in the Spike Room. This is used to activate the key drop action within the room.
Key	Dropped in the Spike Room. This is used to open the north door within the starting area.
Wrench	Picked up in the Arrow Room. This is used to break the arrow shooting machines.

Class: Zombie
This class creates a zombie within the Zombie Room. It has unique functions such as roaming around at random. This is done by applying a rand function that is then used within a conditional if statement. For example, if the random number returned is 1, the zombie moves north. If it then returns 3 after the next move, it goes south. Since a new number is generated, it may or may not have a different directional path.

All movement will be handled via input by the user. Using 'W', 'A', 'S', 'D' as movement, the 'E' to interact and 'I' to check the inventory, the user will be able to control the character. One option for the user is to input a sequence of commands then hitting enter to apply it. For example, if the user enters just 'e', the user will move 1 space. If 'eeee' is entered, the user will move 4 spaces. This may speed up the movement but also has a drawback. Since every command, minus checking inventory, uses up energy, the user can lose energy without actually moving. Walking into a wall 10 times will result in 10 energy used. This design was actually on accident due to issues with validation for the input, but was kept as I had liked the result.

Testing Plan

Test	Input	Output
1	Create a room, set connected rooms	Room is created and correctly points to other rooms
2	printRoom() is called	The screen displays the room map
3	After control is called, action() is called	The user moves and the room's specific action occurs
4	User enters a directional command, control() is called.	User moves in the direction of command
5	After control is called, action() is called	The user moves and the room's specific action occurs
6	Interact() is called near a battery	Battery is picked up
7	Interact() is called near wrench	Wrench is picked up
8	Interact() is called near machine without wrench	Responds by saying it can't be broken
9	Interact() is called near machine with wrench	Machine is destroyed
10	User steps in front of arrow	User loses energy, arrow resets at machine location
11	All machines are broken	Door unlocks
12	As user moves, raiseSpikes() are called	Spikes appear after X amount of moves
13	Zombie moves as user moves, moveZombie() is called	Zombie moves at random according to the number that is randomly chosen per move
14	User interacts with switches in Zombie Room	Switch is turned on or states its been flipped if it has already been turned on
15	User interacts with all 4 switches	Door unlocks
16	User moves in Wall Room	Walls get closer for every movement or interaction taken
17	User enters north door	User is taken to the north pointer room
18	User enters south door	User is taken to the south pointer room
19	User enters east door	User is taken to the east pointer room
20	User enters west door	User is taken to the west pointer room
21	User runs out of energy	Game ends with losing message
22	User escapes from last room	Game ends with winning message

Reflection:

With the freedom to design a game like this, I had great expectation. By playing the example games from previous work, I was able to determine how I wanted to approach it. When thinking of text-based, inputting commands will be done per entry vs a real-time movement. This led me to decide a simple escape-room based off phones games I have played in the past would work.

This project had turned out to be more than I had expected. The first objective I had that took the longest was getting movement correct. This is because going near an edge, or a wall, would result in moving out of bounds and deleting the wall respectively. This lead to frustration as I had not realized it for a while. Once this is done, I was able to use this as a premise for all my movements and interactions. This was pushed onto the zombie as it had the same type of movements but was controlled by the computer. Interaction had the same result, as it was based off the checking of the direction based off player location. These set the tone for the rest of my project.

Again, like most projects in this class, I used some code I had learned from in past projects. None were taken directly, but ideas were modified to suit the new code. Langton’s ant eventually had me solving my movements and the die gave me an opportunity to create “AI”. But the biggest takeaway I had from this project was the fact that I had successfully used pointers and did not have any memory leaks on the first compile. This was a huge turn around for me because I would spend at least a day trying to fix leaks. I spent more time fixing the errors that kept popping up due to the compilers I used being different than the schools. This project was a warm welcome of what is to come and allowed me to expand my knowledge significantly in addition to making me think hard.