**CS 32 Project 3 Report**

**ARYAMAN LADHA | 805-299-802**

**The name of your recursive goblin movement function:**

findShortestPath

the name of the file it's implemented in: Monster.cpp

**A description of the design of your program. For each of your classes, indicate its purpose, what behaviours it implements, and how it relates to other classes:**

**Note: In my program, I use a x and y coordinate system instead of a row and column system. For reference, an x coordinate corresponds to a column, while a y coordinate corresponds to a row.**

I have the following classes in my program:

1.Player: the controllable player of the game

2.Monster: abstract base class which has the classes Bogeyman, Snakewoman, Dragon and Goblin inherit from it.

3.Object: abstract base class which has the classes Weapon and Scroll inherit from it.

4.ladderorIdol : a class that represents either the staircase or the golden idol on a level

5.Dungeon: the class that stores the two dimensional array on which the game is played, and contains all the game’s monsters as well as all the objects.

6. Game: this is the class highest in the class hierarchy and issues all commands to play and control a game.

7. Struct : Room : This is used to create the various rooms in the maze

The player object contains ints for the player’s x(column) and y(row) coordinates on the dungeon, as well as values for all the player stats: health, maxHP, Armour, Dexterity, Strength, etc. In addition, the Player also has a pointer to the dungeon it’s currently on. It also has a list of pointers to objects, which represent the items in its inventory, and a myWeapon pointer that points to the Weapon the Player is currently wielding.

All of the player’s moves are written as methods inside the Player class, namely:

1. Move
2. Pick up an Object
3. attack a Monster: The monster dying and dropping an object is accounted for in here, and so is putting a monster to sleep
4. See the Inventory
5. Wield a Weapon
6. Read a Scroll : The effects of reading a scroll are accounted for in here
7. Cheat

When any of these methods are called, they return a string communicating what happened on a given player move.

Note: attack is called within Move, whenever a Player attempts to move on a spot occupied by a monster

The Monster abstract base class contains ints for a monster’s x(column) and y(row) coordinates on the dungeon, as well as values for all the monster stats: health, maxHP, Armour, Dexterity, Strength, etc. A monster also has a pointer to its weapon, as well a string that indicates the name of the monster: either “Snakewoman”, “Goblin”, “Bogeyman” or “Dragon”. It also has an int for the goblin smell range.

Every Monster action is written as a method, namely:

1. attackplayer: Putting the player to sleep is accounted for in here
2. Move(This function is pure virtual, since every monster has a unique way of moving)

Note: attackPlayer is called within Move, whenever a Monster is directly next to a Player.

When any of these methods are called, they return a string communicating what happened on a given monster move.

The classes Bogeyman, Dragon, Goblin and Snakewoman do not have member variables of their own except for the “Monster part” or the base class. They all have implementations for the Move function.

The Object abstract base class contains ints for an Object’s x(column) and y(row) coordinates on the dungeon, ints for the object’s damage and dexterity, (all weapons have a damage and dexterity, for scrolls these are -1,-1).

It also contains a string for the name of the object, eg “magic fangs of sleep” or “scroll of teleportation”. The object base class has the following nontrivial methods:

1. effect: This returns a string based on what the object’s action is. This is pure virtual , since it differs for Weapons and Scrolls.

The Weapon class does not contain any member variables. It’s implementation of effect returns strings like “swings mace at” or “slashes short sword at” depending on what the Weapon’s name is.

The Scroll class does not contain any member variables. It’s implementation of effect returns strings like “Your muscles bulge” or “You feel your body wrenched in space and time” depending on what the Scroll’s name is.

The LadderorIdol class, which represents either the staircase or a golden idol, contain ints for its x(column) and y(row) coordinates on the dungeon.

The Dungeon class contains an char maze[18][70] that is updated by the monsters and player’s as they move to different positions in the maze. It contains an int for the current level, as well as a ladderorIdol object. The position of this object on the maze is marked as ‘>” for levels 0-3 and ‘&’ for level 4.

The Dungeon class also contains a list of pointers to Objects, representing the various Weapons or scrolls that are randomly placed on in the Dungeon.

It also contains a list of pointers to Monster’s representing the various monsters on a given level. It also has an int for the Goblin smell range.

The Dungeon class has the following nontrivial method:

1.display

It sets the position of the ladder/idol in the maze array to > or &.

It then iterates through the Objects list and sets the position of each object (in the maze array) to ) or ?.

It then iterates through the monster list and sets the position of each monster (in the maze array) to B, D, G, or S.

Finally, it sets the position of the player in the maze array to @

It prints out the array, along with the player stats.

The Game class is the overall controller of the program. It contains a pointer to the Player, a Pointer to the current Dungeon (which is later passed onto the Player object), an integer for the Goblin smell range, and a string list that stores the strings each monster returns when given an opportunity to move.

The Game class has the following nontrivial methods:

1. takePlayerTurn(const char x, string& whatHappened)

Depending on what the character x is, takeplayerturn calls the Player pointer’s methods. Eg. if the character is c, it calls the player’s cheat, and updates whatHappened to the string returned by cheat.

1. takeMonstersTurn()

This iterates through the Dungeon’s List of Monsters, calling move on each of them. It pushes back the string returned by each Monster’s move to the string list.

3.playerDescend()

This deletes the current dungeon, creates a new dungeon with a level one greater than the old dungeon, updates the Player’s dungeon pointer to point to this new dungeon, and sets the player’s position to a random spot in this new dungeon that doesn’t have a wall or a monster on it.

4.play()

This takes an input from the user. While the input isn’t q it plays the game by calling:  
1. takePlayerTurn

2.TakeMonstersTurn

3. The dungeon’s display

4. printing out the player’s action message, as well as every string from the monster string list.

5.Taking another input from the user

It keeps doing this until the player is dead, or the player picks up the idol.

When this happens, it prints out “Press q to exit”. Only pressing q will allow the user to exit play().

The user can also exit play() anywhere in the middle of the game by pressing q (provided player is awake)

The room struct contains the following values:

a TopLeftx int for the room’s top left corner’s x coordinate

a TopLefty int for the room’s top left corner’s y coordinate

ints for the depth and width of each room

bottomrightx and bottomrighty ints for the room’s bottom right corner’s x and y coordinates

midx and midy ints for the room’s middle x and y coordinates

**A documentation of non-trivial algorithms. Use pseudocode where it helps clarify the presentation.**

Player::Player(Dungeon\*dp){

Find random x and y coordinates in the dungeon that do not have a wall or a monster on them.

Set the player’s Xcoord and Ycoord to these values

set mDungeon equal to dp

set the player’s stats to the default values

use myWeapon to dynamically allocate a Weapon(short sword)

add myWeapon to the Player’s inventory

}

string Player::move(char Direction){

if the player is about to move on a monster, call attack(), and return the string that attack returns

else

if the Direction the player intends to move on is empty(not occupied by another monster or a wall)

set the player’s current position to ‘ ’ in the maze array of mDungeon,

change the player’s Xcoord and Ycoord depending on the Direction of movement

set the player’s new position to ‘@’ in the maze array of mDungeon.

}

Player::~Player(){

delete every dynamically allocated Object in the Player’s Inventory(except for the one that the player is currently wielding)

call delete on the myWeapon pointer

call delete on the mDungeon pointer

}

string Player::PickUp(){

if player is on the last level and is standing on the idol, return “Congratulations, you won”

if the inventory size is full, return “You can’t pick that up”

else,

Identify the object in the Dungeon Inventory that you’re standing on

add that object to player’s inventory and remove it from the Dungeon inventory

return the appropriate message- either:

“You pick up a” followed by the Weapon’s name, or

“You pick up a scroll called” followed by the Scroll’s name

}

void Player::seeInventory(){

clear the Screen

iterate through the player’s Inventory, printing out the name of every item in it

}

string Player::wieldWeapon(){

call See Inventory

get a character from the user

Let x = 96-character (so that a character a corresponds to 1)

find the xth item in the Player’s inventory.

If no such item exists, return the empty string

else if the item is a scroll, return “You can’t wield a scroll

else

set the Player’s current weapon equal to that item in the list, and return “You are wielding”+the Weapon’s name.

}

string Player::readScroll(){

call See Inventory

get a character from the user

Let x = 96-character (so that a character a corresponds to 1)

find the xth item in the Player’s inventory.

If no such item exists, return the empty string

else if the item is a weapon, return “You can’t read a weapon

else

if the scroll is a scroll of transportation, set the player’s coordinates to some random position in the maze that doesn’t have a monster or a wall. Update the maze array to reflect this.

if the scroll is a scroll of health, dexterity, strength or armor, increase the player’s corresponding stats by a predetermined value.

delete that scroll from the heap, and erase the corresponding pointer from the Player’s inventory

return the string “You read a scroll called” + The Object’s name + The Object’s effect

}

bool Player::isAbouttoMoveonMonster(char Direction) const{

if,

by going one step in the Direction specified, the player’s coordinates match the coordinates of some monster in mDungeon’s ListofMonsters, return true

else

return false

}

string Player::attack(char Direction){

Identify the monster in mDungeon’s ListofMonsters that the Player would be on if it moved one step in the given Direction

use the formula to compute attacker Points, defender points and damage points.

Use the formula to calculate if the player hit’s the monster.

If the player hits the monster

reduce the monster’s HP by damagePoints

if the HP is less than zero, or the monster is dead

if there’s no object on that spot, then with some probability, the monster will drop a weapon/scroll there. (the type depends on the monster)

If it drops a weapon/scroll , add it to the Dungeon Inventory

set the monster’s coordinates in the maze to “ ”

delete the monster object from the heap and erase its pointer from mDungeon’s list of monsters

return the corresponding string

if the monster is still alive,

if the player has put the monster to sleep,

set the monster’s sleeptime, and return the approporiate string.

else,

return the corresponding string

if the player misses the monster,

return the appropriate string

}

Room::Room(){

topLeftx is a random integer from (1,68)

topLefty is a random integer from (1,16)

width is a random integer from (5,12)

depth is a random integer from (3,6)

the other values are set accordingly

}

Dungeon::Dungeon( int n, int smell): set the level to n, create a ladderorIdol object and set the Goblin smell distance to smell

{

initially, set every cell in the maze to #

then pick a random number of rooms from (4,6)

keep generating rooms until the number of rooms matches the number required, and all of them are in bounds, and none of them overlap.

store all of the valid rooms in a vector

iterate through the vector, and draw each room, by setting every cell in the maze that a room should be on to “ ”

sort the vector of rooms from left to right(the first room will have the least topLeftx)

iterate through the list of rooms again.

connect the midpoints of room 1 to the midpoints of room 2, either by going up and right or down and right, and setting every visited cell to “ ”

do the same for room 2 and room 3, etc.

These represent the corridors.

set the positions of the staircase/idol to some empty spot in the maze

dynamically allocate 2 to 3 random weapons/scrolls and set their positions to valid cells in the maze, and add them to the Dungeon Inventory

depending on the level, create(dynamically allocated) a random number of Snakewomen/Goblins/Dragons/Bogeymen, set their positions to valid cells in the maze, and add them to the ListofMonsters

}

Dungeon::~Dungeon(){

call delete on all the dynamically allocated Objects in the Dungeon Inventory

call delete on all the dynamically allocated monsters in the Dungeon’s ListofMonsters

}

void Dungeon::display(Player\*p){

set the position of the staircase/idol in the maze array to > or &.

iterate through DungeonInventory and set the position of each object (in the maze array) to ) if it’s a Weapon or ? if it’s a scroll

iterate through the ListofMonsters and set the position of each monster (in the maze array) to B(for Bogeymen) or D(for Dragons) or G(for Goblins) or S(for Snakewomen)

set the position of the player in the maze array to @

print out the maze array,

print out player’s stats.

}

string Monster::attackPlayer(Dungeon\*dp, Player\*p){

use the formula to compute attacker Points, defender points and damage points.

Use the formula to calculate if the monster hits the player.

If the monster hits the player

reduce the player’s HP by damagePoints

if the HP is less than zero, or the player is dead

return the corresponding string

if the player is still alive,

if the monster has put the Player to sleep,

set the Player’s sleeptime and return the appropriate string.

else,

return the corresponding string

if the monster misses the Player,

return the appropriate string.

}

bool canSmell(int x, int y, int a, int b, const int distance){

if by going some number of steps to the right or left and then some number of steps up or down, you can reach the Player’s coordinates, or

if by going some number of steps up or down and then some number of steps right ot left, you can reach the player’s coordinates, such that the total number of steps is less than distance,

return true

else

return false

}

**a list of any known bugs, features not implemented, or serious inefficiencies.**

**-**Although I wrote my findMinimumPath function, I couldn’t get it working correctly and was getting bad access errors. So, I decided not to call it inside Goblin::Move so as to not affect the rest of my program.