**Brian Chang Han-Wei**

**UID: 405-142-813**

**CS32 - Project 3 - Report**

**The name of your recursive goblin movement function and the name of the file it's implemented in, or else a statement that your goblin movement function is not recursive. (Make this the first thing, to help the grader find the function.)**

I did not create a recursive function for goblins to pathfind.

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

**Game Class:**

Serves as the base for the game itself to be run, takes input from the user with the getCharacter function and communicates with the game. Game class only really has m\_dungeon as a member variable which acts like the world in which the game is played. I coded this with a series of switch statements to call the corresponding function, given the input.

**Dungeon Class:**

This is where the magic happens. Holds the positions of all of our objects, monsters, and the player. Objects and Monsters are stored as pointers within a monsterVector and objectVector respectively. The dungeon is where I store the functions to create and spawn any actor and non-actor object, furthermore, the printDungeon function is what actually creates the visual representation of the game for the player. The Dungeon also determines how many monsters and objects to spawn when instantiated. TL/DR: Locations and instantiation.

**Actor Class:**

This is the guts of the program. Here is where I define my actors such as the player and monsters. All of their stats, interactions and behaviors are stored within this class. Most notably, the move functions for all actors, the smell functions of the monsters, and the combat function are created within this class. This class also serves to implement the inventory, descend floors, wield, pickUp, stat incrementer, and stat decrementer functions of the game. TL/DR: Act Definitions and interactions with both actors/non-actors.

**Object Class:**

The cherry on top. This class serves to contain all the data of objects that are called within the game itself. Anything you can step on will be defined here, including my will to live. All the swords and scrolls as well as their stats are meticulously coded in this class. The stair and golden idol objects also use the object constructor for instantiation. TL/DR: Non-Actors

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

Play function which keeps the game running as long as player isn’t dead and player hasn’t won.

void Game::play()

{

char move = 0;

while (move != 'q')

{

// If player has won, stop program

if (m\_dungeon->getPlayer()->hasWon())

{

return;

}

// Display the board

m\_dungeon->printDungeon();

// If player is dead

if (m\_dungeon->getPlayer()->isDead())

{

char move = 0;

while (1)

{

move = getCharacter();

if (move == 'q')

{

return;

}

continue;

}

}

// Get the current input by user

move = getCharacter();

// Process input

switch(move) {

case ARROW\_UP:

case ARROW\_LEFT:

case ARROW\_RIGHT:

case ARROW\_DOWN:

m\_dungeon->setCurrentMove(move);

m\_dungeon->playMove();

break;

case OBJECT\_PICK\_UP:

m\_dungeon->getPlayer()->pickUp();

break;

case WIELD\_WEAPON:

m\_dungeon->getPlayer()->wieldWeapon();

case READ\_SCROLL:

m\_dungeon->getPlayer()->readScroll();

case INVENTORY:

m\_dungeon->getPlayer()->openInventory();

break;

case DESCEND\_STAIR:

m\_dungeon->getPlayer()->descend();

break;

case QUIT:

return;

case CHEAT:

m\_dungeon->getPlayer()->cheat();

break;

default:

// TODO: error handling?

continue;

}

clearScreen();

}

}

**The move function that allows us to control the player.**

void Player::takeTurn()

{

if (isDead()) {return;} // Player cannot make any moves if it's dead.

randomRegen(); //10% HP regen chance function.

if (isAsleep())

{

reduceSleepTime(1);

return;

}

// Get the input move

char move = getDungeon()->getCurrentMove();

switch (move) {

//Move Up

case Game::MOVE\_TYPE::UP:

if (getDungeon()->isMob(getRowPos() -1 , getColPos())) // If monster, attack it.

{

Monster\* mob = nullptr;

getDungeon()->getMob(getRowPos() -1 , getColPos(), mob);

attack(this, mob);

}

else if (getDungeon()->actorPosValid(getRowPos() - 1, getColPos())) // If not monster, just move (if valid)

{

clearLastPositionOnMap();

setPosition(getRowPos() - 1, getColPos());

}

break;

//Move Down

case Game::MOVE\_TYPE::DOWN:

if (getDungeon()->isMob(getRowPos() +1 , getColPos())) // If monster, attack it.

{

Monster\* mob = nullptr;

getDungeon()->getMob(getRowPos() +1 , getColPos(), mob);

attack(this, mob);

}

else if (getDungeon()->actorPosValid(getRowPos() + 1, getColPos())) // If not monster, just move (if valid)

{

clearLastPositionOnMap();

setPosition(getRowPos() + 1, getColPos());

}

break;

//Move Left

case Game::MOVE\_TYPE::LEFT:

if (getDungeon()->isMob(getRowPos(), getColPos() - 1)) // If monster, attack it.

{

Monster\* mob = nullptr;

getDungeon()->getMob(getRowPos(), getColPos() - 1, mob);

attack(this, mob);

}

else if (getDungeon()->actorPosValid(getRowPos(), getColPos() - 1))

{

clearLastPositionOnMap();

setPosition(getRowPos(), getColPos() - 1);

}

break;

//Move Right

case Game::MOVE\_TYPE::RIGHT:

if (getDungeon()->isMob(getRowPos(), getColPos() + 1)) // If monster, attack it.

{

Monster\* mob = nullptr;

getDungeon()->getMob(getRowPos(), getColPos() + 1, mob);

attack(this, mob);

}

else if (getDungeon()->actorPosValid(getRowPos(), getColPos() + 1))

{

clearLastPositionOnMap();

setPosition(getRowPos(), getColPos() + 1);

}

break;

default:

break;

}

}

**pickUp function that allows the player to store objects in their inventory and displays text upon acquisition.**

void Player::pickUp() //pick up and object you're standing on with 'g'

{

int playRow = this->getRowPos();

int playCol = this->getColPos();

if (getDungeon()->getLevel() == 4)

{

if (getDungeon()->isIdol(playRow, playCol))

{

//Winning condition.

winGame();

return;

}

}

if (standingOnObject())

{

if (inventory.size() >= 25)

{

cout << "Your knapsack is full; you can't pick that up." << endl;

return;

}

Object\* targetItem = nullptr;

getDungeon()->getObj(playRow, playCol, targetItem); //get object

inventory.push\_back(targetItem); //store object

Weapon\* weaponTest = dynamic\_cast<Weapon\*>(targetItem);

if (weaponTest != nullptr)

getDungeon()->getTextVector().push\_back("You pick up a " + targetItem->getName());

else

getDungeon()->getTextVector().push\_back("You pick up a scroll called " + targetItem->getName());

getDungeon()->eraseObject(targetItem);

}

else

return;

}

**Chase function for bogeybuddies, snakewomen, and unfortunately, Goblins.**

void Monster::chasePlayer(Player\* player, Dungeon\* dungeon)

{

int monRow = getRowPos();

int monCol = getColPos();

int playerRow = player->getRowPos();

int playerCol = player->getColPos();

int deltaRow = abs(monRow - playerRow);

int deltaCol = abs(monCol - playerCol);

int curDist = deltaRow + deltaCol; //how far is the monster from player currently?

// Check moving down

if (abs(monRow + 1 - playerRow) + deltaCol < curDist

&& getDungeon()->actorPosValid(monRow + 1, monCol)

&& !(monRow + 1 == playerRow && monCol == playerCol))

{

clearLastPositionOnMap();

setPosition(monRow + 1, monCol);

}

// Check moving up

else if (abs(monRow - 1 - playerRow) + deltaCol < curDist

&& getDungeon()->actorPosValid(monRow - 1, monCol)

&& !(monRow - 1 == playerRow && monCol == playerCol))

{

clearLastPositionOnMap();

setPosition(monRow - 1, monCol);

}

// Check moving left

else if (deltaRow + abs(monCol - 1 - playerCol) < curDist

&& getDungeon()->actorPosValid(monRow, monCol - 1)

&& !(monRow == playerRow && monCol - 1 == playerCol))

{

clearLastPositionOnMap();

setPosition(monRow, monCol - 1);

}

// Check moving right

else if (deltaRow + abs(monCol + 1 - playerCol) < curDist

&& getDungeon()->actorPosValid(monRow , monCol + 1)

&& !(monRow == playerRow && monCol + 1 == playerCol))

{

clearLastPositionOnMap();

setPosition(monRow, monCol + 1);

}

}

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

1. My goblins are dumb, they will not find the best possible route to the player and only navigate by trying to reduce the deltaRow and deltaCol squares.
2. I did not create a dungeon with rooms connected by corridors. Instead, I created the dungeon as one big sandbox with all the actors and non-actors.