1. Goblin recursion

*Write: 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.)*

2. Class explanation

*Write: 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.*

There are 6 general class in total. The main file is trivial: it starts the game by calling the game’s constructor and its play function. The global.h file stores all the game design needed global constants such as the size of the playground and the number of levels (difficulty) for this game. The utility file provides basic functions and algorithms such as generating a possibility or a random number with user specified range. The Game file is the main room for game to run. It is where the game got constructed by creating a dungeon and assigning a dungeon to a player. The Game also has a play function that triggers the game to run and, without the user typing quit or the player winning/losing the game, keeps calling the go function. The go function keeps asking the player to make its move by typing a character on the keyboard if it is not put to sleep by the monsters. The Game’s descend function is called when the player decides to go to the next level (when he can). It generates a new dungeon that links back to the previous player pointer instead of generating a new one in order to preserve all the information of the original player using a copy constructor (written in Actor.h file). In the Game class we have member functions that get access to the player and the dungeon. The Actor class includes all the characters in the game including the player and the monsters. In the actor class (virtual, cannot create a pure actor), there is constructors for all the possible subclasses of the actors such as player, goblin, dragon, etc. Each player is created with all the basic points (hit, armor, etc) and their weapon (if they do have one) equipped. In the Actor’s class, we specify functions that the player needs to call to wield a weapon or read a scroll. Next, we have the Dungeon class that initializes all the walls, rooms and corridors. It will then initialize the player along with all the pre-dropped objects and monster on the map. The Dungeon class also has a display function which is very important in that it will places all the subjects to their designated coordinate and display them on the window. Other than that, the display function is also responsible in displaying all the fighting history (such as “player slashes short sword at …, etc). There is a fight function within the Dungeon class that takes to characters and make them fight each other. Any time a fight function is called, there is a checkDeath function to check if the defender is put to death and does subsequent move accordingly. There is another important feature that is implemented in the Dungeon class which is the monstermove function. Just like the play and go function implemented in Game, the monstermove function implemented in Dungeon does basically the same thing as the player except their move is not decided by the user’s input. Based on the current process of the game and the status of the monster/player, each monster will make subsequent moves (such as attack the player or move to follow, etc) if they can. Last but not the least, we have an Object class that stores all the important objects and their data/features. The objects include all the scrolls and weapons along with how they are constructed and initialized.

3. Non-trivial function explanation

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

1. Game::go

**Keep prompting**

{

**if** (user need to type something)

{

Get a char from input;

}

**else**

{

Automatically enters a default input

}

Get the player from the dungeon

**if** (m\_player does not exit)

exit(1);

**else** **if**(player is put to sleep and cannot move)

{

Player’s sleep time -1

Player does nothing

**return** "";

}

**else**

{

**if** (there is a 1 in 10 probability that the player gains one hit point)

{

Player get one hit point (not exceeding its max hit point)

}

**Get player’s position;**

**switch** (input)

{

**case** move up

if not valid;

**return** "";

**else**

move up

**break**;

**case** move right

**if** not valid;

**return** "";

**else**

move right;

**break**;

**case** move down

**if** (not valid)

**return** "";

**else** move down;

**break**;

**case** move left

**if** (not valid)

**return** "";

**else**

move left;

**break**;

**case** '>':

**if** (player is on a stair (“>”))

{

Descend to next level;

**return** "";

}

If not on a stair

**return** "";

**break**;

**case** 'g':

**case** 'G':

**if**( player is on the golden idol)

{

Set player status to win;

**return** "";

**break**;

}

**if**(the player is on some other objects)

{

Pick that object;

}

**return** "";

**break**;

**case** 'i':

**case** 'I':

show inventory

**return** "";

**case** 'w':

**case** 'W':

show inventory;

wield that weapon if the player has one;

**return** "w";

**case** 'r':

**case** 'R':

show inventory;

read that scroll if the player has one;

**return** "";

**case** 'q':

**case** 'Q':

**return** "q", indicating quit;

**break**;

**case** 'c':

**case** 'C':

**cheat;**

**return** "";

**break**;

**default**:

**return** "";

**break**;

}

Check what is under the player if it makes that move;

**switch** (what is currently under the player)

{

**case** ' ': nothing here, go ahead;

{

Player move;

**return** "";

**break**;

}

**case** ')':

**case** '?': here is an object;

{

Move over;

**Allow pick up if inventory not full;**

**return** "";

**break**;

}

**case** '&': here is a golden idol;

{

Player can move over;

**return** "";

**break**;

}

**case** '>': here is a stair

{

Player move over;

Allow descend move from player;

**return** "";

**break**;

}

**case** 'S':

**case** 'D':

**case** 'G':

**case** 'B': here is a monster

{

//////////////Player turn///////////////

Let player fight monster;

**if** (mon != **nullptr**)

{

**if** (monster dead)

{

Allow object to be dropped;

Set current fighting monster status to dead;

}

**else**

{

Set current fighting monster status to not dead;

}

}

//////////////End player turn////////

**return** "m";

**break**;

}

}

}

}

}

2. **void** Game::play()

{

Default msg (at first) = “”;

**while**(player is not dead or wining the game)

{

Display the playground

Get the returning msg from the go;

**if**(msg == "")

{

Player Go;

Monster move;

}

**if**(msg == "w")

{

Don’t get char from the player for the next round;

player go;

}

**if**(msg == "q")

{

Ready to quit;

**break**;

}

**if**(msg == "m")

{

Don’t get char from the player for the next round;

Player go;

}

}

**if**(player win)

{

Display the wining message;

cout<<"You pick up the golden idol"<<endl;

cout<<"Congratulations, you won!"<< endl;

cout<<"Press q to exit game."<< endl;

**for** (;;)

{

**Get char from the user**

**if** (input == 'q' || input == 'Q')

quit the game if a q/Q is entered;

}

}

**if**(player is dead)

{

Displaying the death message;

cout<<"Press q to exit game."<< endl;

**for** (;;)

{

**Get char from the user**

**if** (input == 'q'|| input == 'Q')

quit the game if a q/Q is entered;

}

}

**if**(player ready to quit)

{

Quit the game;

}

}

3. **void** player::wieldWeapon(**char** input)

{

Check size of inventory;

**Convert input to index;**

**if** (input within the range: there exist such thing in the inventory)

{

Add that to the inventory;

**if** (not a weapon)

{

Fail to wield;

}

**else**

{

Make it the current weapon;

}

}

}

(reading scroll is similar to wielding weapon)

4. Dungeon::Dungeon(**int** l, player\* p)

:level(l)

{

Set all the default member Boolean to false;

**Place walls around the boundary;**

vector<center> centerv;

**create random number of buildings;**

**for** each building

{

**Create random number of levels;**

**Check how big a room can be;**

**for** (**each room)**

{

**Set a center**

**Set a boundary;**

Push the center to a stack;

**Draw the room;**

}

};

**for** (**each two rooms)**

{

Draw corridors connecting the centers of the two rooms;

}

**if**(level 0)

{

**Calculate player and stair position;**

**Randomly place some objects;**

**Push the created objects to scroll or weapon stacks accordingly;**

Randomly place some monsters

**if** (level < 2)

{

**Two kinds of monsters to be placed;**

}

**else** **if** (level < 3)

{

Three kinds of monsters to be placed;

}

**else**

{

**Four kinds of monsters to be placed;**

}

}

}

**Else**

{

**Calculate player and golden idol position;**

**Randomly place some objects;**

**Push the created objects to scroll or weapon stacks accordingly;**

Randomly place some monsters

**if** (level < 2)

{

**Two kinds of monsters to be placed;**

}

**else** **if** (level < 3)

{

Three kinds of monsters to be placed;

}

**else**

{

**Four kinds of monsters to be placed;**

}

}

}

5. **void** Dungeon::display(string msg)

{

clearScreen();

**place walls around the boundary;**

**get player coordinate;**

**if** (not final level)

{

**Place player and stair;**

}

**else**

{

**Place player and idol;**

};

**if** (monster dropped weapon)

{

**Get coordinate;**

**if** (m"Snakewoman")

{

**if**(can be dropped)

{

**if** (with 1/3 probability)

{

Drop ((**new** weapon("magic fangs")))

}

}

}

**if**("Goblin")

{

**if**(can be dropped)

{

**if** (there is (1/3) probability)

{

drop

{

push((**new** weapon("magic fangs")));

or drop

{

push((**new** weapon("magic axes")));

}

}

}

}

**if**("Dragon")

{

**if**(can be dropped)

{

**Drop one of the following:**

push((**new** scroll("scroll of strength")));

push((**new** scroll("scroll of enhance health")));

push((**new** scroll("scroll of enhance dexterity")));

push((**new** scroll("scroll of teleportation")));

push((**new** scroll("scroll of enhance armor")));

}

}

}

**if**("Bogeyman")

{

**if**(can be dropped)

{

**if** (with (1/10) probability)

{

push((**new** weapon("magic axes")));

}

}

}

setDrop(**false**);

}

**for**(**all objects on the map)**

{

**if** (player there)

{

Show as player

}

**else**

{

**Drop it on the map;**

}

}

**for**(**all the monsters)**

{

Drop them on the map;

}

If player read teleportation

Transfer it to a random valid spot;

**Print out the maps;**

Print the player’s data;

**if** (player wield a weapon)

{

Display wield;

}

**if**(player pick up things)

{

Display pick up;

}

**if**(player read scroll)

{

If not valid

{

Display failed to read msg;

}

**else**

{

Display read msg;

}

}

**if**(player fight monster)

{

Display fighting message;

}

**if**(monster fight player)

{

Display fighting message;

}

}

}

4. Debug

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

a. Too many Boolean statement in the dungeon and actor class that could be done more efficiently by writing player fight in dungeon instead of in Game class.

b. My armor can only glow blue. In the actual game, the armor can glow silver.