CS 7B - Fall 2018 - Assignment 4: Exploring Wumpus (Linked Rooms). Due 12/13/18

In this project we'll build, investigate and augment the the game "Hunt the Wumpus," as described in exercise 12 or PPP2, Chapter 18.

> "Hunt the Wumpus" (or just "Wump") is a simple (non-graphical) computer game originally invented by Gregory Yob. The basic premise is that a rather smelly monster lives in a dark cave consisting of connected rooms. Your job is to slay the wumpus using bow and arrow. In addition to the wumpus, the cave has two hazards: bottomless pits and giant bats. If you enter a room with a bottomless pit, it's the end of the game for you. If you enter a room with a bat, the bat picks you up and drops you into another room. If you enter the room with the wumpus or he enters yours, he eats you. When you enter a room you will be told if a hazard is nearby:
>
> - "I smell the wumpus": It;s in an adjoining room.
> - "I feel a breeze": One of the adjoining rooms is a bottomless pit.
> - "I hear a bat": A giant bat is in an adjoining room.
>
> For your convenience, rooms are numbered. Every room is connected by tunnels to three other rooms. When entering a room, you are told something like "You are in room 12; there are tunnels to rooms 1, 13, and 4; move or shoot?" Possible answers are m13 ("Move to room 13") and s13–4–3 ("Shoot an arrow through rooms 13, 4, and 3"). The range of an arrow is three rooms. At the start of the game, you have five arrows. The snag about shooting is that it wakes up the wumpus and he moves to a room adjoining the one he was in—that could be your room.
>
> Probably the trickiest part of the exercise is to make the cave by selecting which rooms are connected with which other rooms. You'll probably want to use a random number generator (e.g., `rand()` or mt19937) to make different runs of the program use different caves and to move around the bats and the wumpus. Hint: Be sure to have a way to produce a debug output of the state of the cave.

Here's some starter code you will want to base the project on:

```
1
#pragma once
3
#include <cmath>
5 #include <cstdlib>
#include <ctime>
7 #include <iostream>
#include <vector>
9 #include <string>
#include <sstream>
11 #include <limits>
using namespace std;
13
namespace Wumpus {
15
// to serve as return value from functions that can kill player or Wumpus
17 enum Game_state {
    plr_shot, no_arrows, wmp_dead, running
19 };

21 // a room in the cave
struct Room {
23     Room();
    Room(int n, Room* nxt0, Room* nxt1, Room* nxt2);
25     int label;
    //vector<Room*> next;
27     Room** next = new Room*[3];
```

```
    bool has_pit;
    bool has_bat;
};

// to check if two rooms are connected
///bool contains(vector<Room*> v, Room* x);
bool contains(Room** v, Room* x);

struct Player {
    Player(Room* n);
    Room* location;
    int n_arrows;
};

struct Wumpus {
    Wumpus(Room* r);
    Room* location;
};

class Cave {
public:
    Cave();
    Room* get_player_loc();
    Room* get_wumpus_loc();
    bool move_player(int lbl);
    void hazard_warnings();
    void room_description();
    void bat_flight();
    void wake_wumpus();
    Game_state shoot_arrow();
private:
    Room* rooms;
    //vector<Room> rooms;
    Player plr;
    Wumpus wmp;
    Room* lbl_to_ptr(int lbl);  // convert label to Room ptr
};

void print_instructions();

// test if route for arrow is valid
bool route_valid(vector<Room*> r);

}   // namespace Wumpus

// helper function to recover from failing to get an int from cin
void skip_to_int();

// read until cin provides an integer
int get_int();

// get integer in given range, prompt with greeting, print sorry if invalid
int get_int(int low, int high, const string& greeting, const string& sorry);

//------------------------------------------------------------------
namespace Wumpus {

// ------------------------------------------------------------------
// Construct a room
Room::Room() : label(0), has_pit(false), has_bat(false) { }

// ------------------------------------------------------------------
// Construct a room with pointers to three other rooms
Room::Room(int n, Room* nxt0, Room* nxt1, Room* nxt2)
    : label(n), next(new Room*[3]), has_pit(false), has_bat(false) {
```

```
93      next[0] = nxt0;
        next[1] = nxt1;
95      next[2] = nxt2;
}
97
// ------------------------------------------------------------------------
99 //Check to see if a room is connected to another room
bool contains(Room** v, Room* x) {
101         for (int i = 0; i<20; ++i) { ///v.size(); ++i) {
             if (v[i] == x) return true;
103     }
        return false;
105 }

107 // ------------------------------------------------------------------------
// Construct a Player with a location and a quiver of arrows
109 Player::Player(Room* n) : location(n), n_arrows(5) { }

111 // ------------------------------------------------------------------------
// Construct a Wumpus with a location
113 Wumpus::Wumpus(Room* r) : location(r) { }

115 // ------------------------------------------------------------------------
// Shuffle the rooms
117 void fisher_Yates_shuffle(int* ia) { //vector<int>& vi) {
        int r{0}, temp;
119     for(int i = 0; i < 18; ++i) { //vi.size()-2; i++) {
            r = i+rand()%(20-i); //vi.size()-i);
121         temp = ia[i];
            ia[i] = ia[r];
123         ia[r] = temp;
        }
125 }

127 // ------------------------------------------------------------------------
// Construct the Cave with rooms
129 Cave::Cave() : rooms(new Room[20]), plr(0), wmp(0) {
        srand(time(0));
131
        int* labels  = new int[20];
133     for(int i = 0; i < 20; ++i) labels[i] = i+1;
        fisher_Yates_shuffle(labels);
135
        // set up tunnel system
137     rooms[0] = Room(labels[0],&rooms[1],&rooms[4],&rooms[5]);
        rooms[1] = Room(labels[1],&rooms[0],&rooms[2],&rooms[7]);
139     rooms[2] = Room(labels[2],&rooms[1],&rooms[3],&rooms[9]);
        rooms[3] = Room(labels[3],&rooms[2],&rooms[4],&rooms[11]);
141     rooms[4] = Room(labels[4],&rooms[0],&rooms[3],&rooms[13]);
        rooms[5] = Room(labels[5],&rooms[0],&rooms[6],&rooms[14]);
143     rooms[6] = Room(labels[6],&rooms[5],&rooms[7],&rooms[15]);
        rooms[7] = Room(labels[7],&rooms[1],&rooms[6],&rooms[8]);
145     rooms[8] = Room(labels[8],&rooms[7],&rooms[9],&rooms[16]);
        rooms[9] = Room(labels[9],&rooms[2],&rooms[8],&rooms[10]);
147     rooms[10] = Room(labels[10],&rooms[9],&rooms[11],&rooms[17]);
        rooms[11] = Room(labels[11],&rooms[3],&rooms[10],&rooms[12]);
149     rooms[12] = Room(labels[12],&rooms[11],&rooms[13],&rooms[18]);
        rooms[13] = Room(labels[13],&rooms[4],&rooms[12],&rooms[14]);
151     rooms[14] = Room(labels[14],&rooms[5],&rooms[13],&rooms[19]);
        rooms[15] = Room(labels[15],&rooms[6],&rooms[16],&rooms[19]);
153     rooms[16] = Room(labels[16],&rooms[8],&rooms[15],&rooms[17]);
        rooms[17] = Room(labels[17],&rooms[10],&rooms[16],&rooms[18]);
155     rooms[18] = Room(labels[18],&rooms[12],&rooms[17],&rooms[19]);
        rooms[19] = Room(labels[19],&rooms[14],&rooms[15],&rooms[18]);
157
```

```
    // assign player and hazards: randomly shuffle vector with room numbers,
    // then select the first six and assign to player, pit, bats, and wumpus
    for (int i = 0; i<20; ++i)
        room_no.push_back(i);*/
    int* room_no = new int[20];
    fisher_Yates_shuffle(room_no);
    plr.location = &rooms[room_no[0]];
    rooms[room_no[1]].has_pit = true;
    rooms[room_no[2]].has_pit = true;
    rooms[room_no[3]].has_bat = true;
    rooms[room_no[4]].has_bat = true;
    wmp.location = &rooms[room_no[5]];

    // output for debugging
    //cout << "Pits: " << rooms[room_no[1]].label << " and "
    //    << rooms[room_no[2]].label << "\n";
    //cout << "Bats: " << rooms[room_no[3]].label << " and "
    //    << rooms[room_no[4]].label << "\n";
    //cout << "Wumpus: " << rooms[room_no[5]].label << "\n";
}

// ------------------------------------------------------------------------

// if possible, move player to room with label lbl
bool Cave::move_player(int lbl) {
    Room* n = lbl_to_ptr(lbl);
    // see if label existed and move is legal
    if (n==0) {
        cout << "Not possible\n";
        return false;
    }
    else if (contains(plr.location->next,n)) {
        plr.location = n;
        return true;
    }
    else
        cout << "Not possible\n";
    return false;
}

// ------------------------------------------------------------------------

// check for Wumpus, pits and bats in neighboring rooms
void Cave::hazard_warnings() {
    if (plr.location->next[0] == wmp.location ||
        plr.location->next[1] == wmp.location ||
        plr.location->next[2] == wmp.location)
        cout << "I smell a Wumpus!\n";
    if (plr.location->next[0]->has_pit ||
        plr.location->next[1]->has_pit ||
        plr.location->next[2]->has_pit)
        cout << "I feel a draft!\n";
    if (plr.location->next[0]->has_bat ||
        plr.location->next[1]->has_bat ||
        plr.location->next[2]->has_bat)
        cout << "Bats nearby!\n";
}

// ------------------------------------------------------------------------

// give neighboring rooms
void Cave::room_description() {
    cout << "You are in room " << plr.location->label << "\n";
    cout << "Tunnels lead to " << plr.location->next[0]->label << ' '
        << plr.location->next[1]->label << ' '
```

```
223         << plr.location->next[2]->label << "\n";
}

// ------------------------------------------------------------------------

Room* Cave::get_player_loc() { return plr.location; }

// ------------------------------------------------------------------------

void Cave::bat_flight() { plr.location = &rooms[rand()%20]; }

// ------------------------------------------------------------------------

// wake Wumpus: stays in same room (P=0.25), move to neighbouring room (P=0.75)
void Cave::wake_wumpus() {
    int n = rand()%4;///randint(4);
    if (n==3) return;
    wmp.location = wmp.location->next[n];

    // output for debugging
    //cout << "Wumpus is at " << wmp.location->label << "\n";
}

// ------------------------------------------------------------------------

Game_state Cave::shoot_arrow() {
    // get valid route of arrow
    int n = get_int(1,5,"No. of rooms","Must be");
    vector<Room*> route(n);
    while (true) {
        for (int i = 0; i<n; ++i) {
            ostringstream os;
            os << "Room #" << i+1;
            route[i] = lbl_to_ptr(get_int(1,20,os.str(),"Must be"));
        }
        if (route_valid(route)) break;
    }

    // for each room, check if tunnel exists, choose random tunnel if not
    Room* arrow_loc = get_player_loc();
    for (int i = 0; i<route.size(); ++i) {
        if (contains(arrow_loc->next,route[i]))
            arrow_loc = route[i];
        else
            arrow_loc = arrow_loc->next[rand()%3]; ///randint(3)];    // no connection
        //cout << arrow_loc->label << ' ';  // output for debugging

        // check if arrow in room with player or wumpus (die/win)
        if (arrow_loc==get_wumpus_loc()) return wmp_dead;
        if (arrow_loc==get_player_loc()) return plr_shot;
    }
    cout << "\n";

    // reduce number of arrows (if equals zero --> die)
    --plr.n_arrows;
    if (plr.n_arrows==0) return no_arrows;
    cout << "\n" << plr.n_arrows << " arrows left\n";
    return running;
}

// ------------------------------------------------------------------------

Room* Cave::get_wumpus_loc() { return wmp.location; }

// ------------------------------------------------------------------------
```

```
289 // return pointer to room with label lbl, and 0 if no such room exists
    Room* Cave::lbl_to_ptr(int lbl) {
291     for (int i = 0; i<20; ++i)
            if (rooms[i].label==lbl) return &rooms[i];
293     return 0;
    }
295
    // ------------------------------------------------------------------------
297
    void print_instructions() {
299     cout << "Welcome to 'Kara Hunts the Wumpus'!\n\n"
            << "The Wumpus lives in a cave of 20 rooms. Each room\n"
301         << "has 3 tunnels leading to other rooms. (Look at a\n"
            << "dodecahedron to see how this works - if you don't know\n"
303         << "what a dodecahedron is, ask someone.)\n\n"
            << "Hazards:\n"
305         << "Bottomless pits - two rooms have bottomless pits in them.\n"
            << "If you go there, you fall into the pit (and lose!)\n"
307         << "Super bats - two other rooms have super bats. If you\n"
            << "go there, a bat grabs you and takes you to some other\n"
309         << "room at random (which might be troublesome).\n\n"
            << "Wumpus:\n"
311         << "The Wumpus is not bothered by the hazards (he has sucker\n"
            << "feet and is too big for a bat to lift). Usually\n"
313         << "he is asleep. Two things wake him up: your entering\n"
            << "his room or your shooting an arrow.\n"
315         << "If the Wumpus wakes, he moves (75% chance) one room\n"
            << "or stays still (25% chance). After that, if he is where you\n"
317         << "are, he eats you up (and you lose)!\n\n"
            << "You:\n"
319         << "Each turn you may move or shoot a crooked arrow.\n"
            << "Moving: you can go one room (through one tunnel).\n"
321         << "Arrows: you have 5 arrows. You lose when you run out.\n"
            << "Each arrow can go from 1 to 5 rooms. You aim by telling\n"
323         << "the computer the room numbers you want the arrow to go to.\n"
            << "If the arrow can't go that way (i.e., no tunnel) it moves\n"
325         << "at random to the next room.\n"
            << "If the arrow hits the Wumpus, you win.\n"
327         << "If the arrow hits you, you lose.\n\n"
            << "Warnings: when you are one room away from a hazard,\n"
329         << "the computer says:\n"
            << "Wumpus - I smell a Wumpus!\n"
331         << "Bat - 'Bats nearby!'\n"
            << "Pit - 'I feel a draft!'\n\n";
333 }

335 // ------------------------------------------------------------------------

337 // test if route for arrow is valid
    bool route_valid(vector<Room*> r) {
339     if (r.size()<1 || r.size()>5) {
            cout << "Route not valid, start again\n";
341         return false;
        }
343     for (int i = 2; i<r.size(); ++i) {
            if (r[i] == r[i-2]) {
345             cout << "Arrows aren't THAT crooked - try again\n";
                return false;
347         }
        }
349     return true;
    }
351
    // ------------------------------------------------------------------------
```

```
353
}    // namespace Wumpus
355
// ------------------------------------------------------------------------
357
// helper function to recover from failing to get an int from cin
359 void skip_to_int() {
    if (cin.fail()) {
361     cin.clear();
        char ch;
363     while (cin>>ch) {
            if (isdigit(ch) || ch=='-') {
365             cin.unget();
                return;
367         }
        }
369 }
    cout << "no input";
371 ///runtime_error("No input");
}
373
// ------------------------------------------------------------------------
375
// read until cin provides an integer
377 int get_int() {
    int n = 0;
379 while (true) {
        if (cin>>n) {
381         cin.ignore(numeric_limits<streamsize>::max(),'\n');
            return n;
383     }
        cout << "Not valid\n";
385     skip_to_int();
    }
387 }
389 // ------------------------------------------------------------------------
391 // get integer in given range, prompt with greeting, print sorry if invalid
int get_int(int low, int high, const string& greeting, const string& sorry) {
393 cout << greeting << " (" << low << '-' << high << ")\n";

395 while (true) {
        int n = get_int();
397     if (low<=n && n<=high) return n;
        cout << sorry << " (" << low << '-' << high << ")\n";
399 }
}
401
// Write wumpus_loop (to do!)
403
int main()
405 try {
    cout << "Instructions (Y-N)?\n";
407 char ch;
    cin >> ch;
409 if (ch!='n') print_instructions();
    cin.ignore(numeric_limits<streamsize>::max(),'\n');
411 wumpus_loop();
}
413 catch (exception& e) {
    cerr << "exception: " << e.what() << endl;
415 }
catch (...) {
417 cerr << "exception\n";
```

```
}

/**
You are in room 4
Tunnels lead to 2 3 1
Shoot or move (S-M)?
m
Where to?
1

I smell a Wumpus!
I feel a draft!
Bats nearby!
You are in room 1
Tunnels lead to 4 6 17
Shoot or move (S-M)?
m
Where to?
4
...oops! Bumped a Wumpus!
Zap - super bat snatch! Elsewhereville for you!
Tsk tsk tsk - Wumpus got you!
Ha ha ha - you lose!
Play again?y




Instructions (Y-N)?
n
10 4 9 16 18 15 1 8 11 7 14 17 19 2 5 6 12 20 13 3
You are in room 11
Tunnels lead to 8 7 12
Shoot or move (S-M)?
m
Where to?
8

I feel a draft!
You are in room 8
Tunnels lead to 4 1 11
Shoot or move (S-M)?
n
Shoot or move (S-M)?
m
Where to?
11

You are in room 11
Tunnels lead to 8 7 12
Shoot or move (S-M)?
m
Where to?
12

You are in room 12
Tunnels lead to 11 6 20
Shoot or move (S-M)?
m
Where to?
6

I feel a draft!
You are in room 6
```

```
Tunnels lead to 1 12 3
Shoot or move (S-M)?
m
Where to?
3

I feel a draft!
You are in room 3
Tunnels lead to 5 6 13
Shoot or move (S-M)?
m
Where to?
6

I feel a draft!
You are in room 6
Tunnels lead to 1 12 3
Shoot or move (S-M)?
m
Where to?
12

You are in room 12
Tunnels lead to 11 6 20
Shoot or move (S-M)?
m
Where to?
20

You are in room 20
Tunnels lead to 14 12 13
Shoot or move (S-M)?
m
Where to?
13

I smell a Wumpus!
You are in room 13
Tunnels lead to 19 20 3
Shoot or move (S-M)?
s
No. of rooms (1-5)
1
Room #1 (1-20)
19
Aha! You got the Wumpus!
Hee hee hee - the Wumpus'll getcha next time!
Play again?
*/
```

To make the game more interesting:

```
\item Add another creature of your own invention with properties the player would find
    interesting.
\item Add SFML graphics.
\item Add rooms.
\item Add weapons.
```