Program 4

Due Mar 1 by 11:59pm **Points** 60 **Submitting** a file upload **File Types** tar
Available Feb 17 at 12am - Mar 6 at 11:59pm 19 days

This assignment was locked Mar 6 at 11:59pm.

Implementing Hunt the Wumpus

Motivation

The goal of this assignment is to start working with polymorphism and C++ template classes from the STL (standard template library). Follow the directions below and be sure to submit a TAR file containing only your .h, .cpp, and Makefile.

Introduction

Hunt the Wumpus is a game set inside the cave of the Wumpus, a scary, gold-hoarding monster who likes to eat people who disturb its sleep. The player's goal is to guide an adventurer to kill the Wumpus, find its hidden gold, and escape alive. The Wumpus lives in a large cave of rooms arranged in a grid, where each room has four tunnels leading to the rooms to the north, east, south, and west.

The adventurer starts the game in a random empty room in the Wumpus' cave. Her starting position is also the location of the escape rope that she must use to escape after she's completed her task.

Each room may be empty, or it may contain one (and only one) of four "events" (all described below): two kinds of dangerous hazards, the terrifying Wumpus, and the gold treasure. When the adventurer is in a room that's adjacent to one containing an "event", the player will receive a percept (a message) to inform them about the event the adventurer is close to.

If the adventurer perishes while searching for the Wumpus, the player should be presented with the option to start the game over with the same cave configuration, start the game over with a new random cave configuration, or quit the game entirely.

The player wins the game if they can safely guide the adventurer through the cave to kill the Wumpus, pick up the gold, and make it back to the escape rope unharmed!
Note: If you enter the cave, retrieve the gold, and escape without killing the Wumpus, that's okay too. The player still wins.

Game elements

The adventurer

Each turn you may take one of two actions to guide the adventurer:

- **Move**: You can move through a tunnel to an adjacent room.
- **Fire an Arrow**: The adventurer begins the game with three arrows. As long as the adventurer still has at least one arrow, the player can choose to fire one in any direction (i.e. north, south, east, or west). The arrow continues flying in the same direction until it hits a wall or travels through three rooms. If the arrow enters the Wumpus' room, it pierces the Wumpus' heart and kills the monster. As noted above, if you want to design the game such that the player can win by safely retrieving the gold (and not killing the Wumpus), that's okay too.

The Wumpus

Usually, the Wumpus is peacefully asleep in its cave. Two things wake the Wumpus up, however: the adventurer entering its room or shooting an arrow and missing it. If the Wumpus wakes up while in the same room as the adventurer, the Wumpus will angrily eat the adventurer, ending the game. If the Wumpus just wakes up from hearing an arrow being fired, though, there's a 75% chance it will move to a random empty room in the cave to avoid being found.

Hazards

There are two kinds of hazards in the Wumpus' cave:

- **Bottomless pits**: Two of the cave's rooms have bottomless pits in them. If the adventurer enters a room with a bottomless pit, she falls in and dies, and the player loses.
- **Super bats**: Two rooms have super bats. If the adventurer enters a room that contains super bats, an angry bat grabs her and takes her to some other room at random (which could be dangerous!).

The gold

The gold is a treasure sitting in a room that contains neither a hazard nor the Wumpus. If the adventurer is in a room containing the gold, she automatically picks it up and takes it with her.

Percepts

When the adventurer is in a room directly adjacent to to a room containing an "event", the player receives the following messages:

- **Wumpus**: "You smell a terrible stench."
- **Super bats**: "You hear wings flapping."
- **Bottomless pit**: "You feel a breeze."
- **Gold**: "You see a glimmer nearby."

Notice that there's no percept for the escape rope! That means the player will have to remember where they started and find their way back to that tile after they've completed their task.

As an example of how the percepts work, if the adventurer is standing in an empty room with the Wumpus one room to the North, the Gold one room to the East, and Bats *two* rooms to the South, they would receive the following messages at the beginning of their turn:

```
You smell a terrible stench.
You see a glimmer nearby.
```

Remember, the percepts don't tell you where the hazard or gold is, just that it's somewhere close!

Program requirements

- Your program should allow the user to play Hunt the Wumpus, as described above.
- The Wumpus' cave is represented by a square grid. The size of the grid (i.e. the number of rooms on one side of the square) should be specified **as a command line parameter to your program**. Caves smaller than four rooms on a side aren't allowed. You should visualize the grid to allow the user to play the game. In particular, you should display the grid, and indicate within the grid which room the player is in. An example visualization of a 4x4 grid might look like this, where the `*` character represents the location of the adventurer:

```
+---+---+---+---+
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
|   |   | * |   |
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
+---+---+---+---+
```

- Your program **must also accept a second command line parameter** which will be either "true" or "false". If the parameter is set to false (e.g. **./wumpus 4 false**) then the program will run as normal. If the second command line value is specified as "true" then your game must operate in debug mode.
- When your program in operating in debug mode, the player's map will show a "cheat view" with locations marked for each of the following: wumpus, bats, bottomless pits, gold, player, escape rope.

- To navigate the cave system, the player must be able to type "w" (north), "a" (west), "s" (south), or "d" (east). In order to fire an arrow, a space (" ") should be used, followed by either "w", "a", "s", or "d" to indicate the direction. For example, the user would enter " d" to fire an arrow towards the east.
- Your code must have the following classes: `Room`, `Event`, `Wumpus`, `Bats`, `Pit`, and `Gold`. You may create more classes if they would be helpful.
- The `Event` class **must be abstract** , and the `Wumpus`, `Bats`, `Pit`, and `Gold` classes should all be derived from `Event`. Remember, any event does something when the adventurer enters the same room as the event, and will display a message when the adventurer is nearby. Your design of the `Event` class should reflect this. For example, your `Event` class might have a `percept()` method that is called whenever the adventurer is in a room adjacent to the event, where the `Wumpus`, `Bats`, `Pit`, and `Gold` classes implement their own specific versions of the `percept()` method. Similarly, your `Event` class might have an `encounter()` method that is called when the adventurer enters the same room as the event, with the individual event classes implementing their own specific versions of `encounter()`.
- **You must use the `Event` classes polymorphically.** In other words, your `Room` class may only contain a member of the `Event` class but not members of the `Wumpus`, `Bats`, `Pit`, or `Gold` classes.
- Each `Room` contains *at most* one `Event`, but it's possible that it contains no `Event`. The design of your `Room` class should reflect this.
- The grid representing your cave should be implemented using the `std::vector` class.
- Your program may not have any memory leaks. I strongly recommend that you use Valgrind to occasionally test your program as you develop it (even when everything seems to be working).
- The Big 3 must be implemented as appropriate.
- Your program must be factored into interface, implementation, and application. Specifically, you should have one header file and one implementation file for each class, and you should have a single application file containing your `main()` function. You should also include a Makefile that specifies compilation for your program.
- Lack of correct coding style will incur an automatic 6 point deduction. You must follow the spirit of the assignment.

Hints

- Polymorphism only works when you are working with references or pointers. If you use the value of an object directly, it may not select the correct member function.
- Hunt the Wumpus is a game all about hiding information from the player, which might make it hard to debug! Your life will be easier if you implement the debugging mode first and then finish your implementation of the final version

Extra Credit

In addition to the requirements above, you may earn extra credit as follows:

- (6 points) Implement an AI class that plays the game for you. This AI class should use the same interface to the game that the player does. That is, it should use percepts to learn about the world and make decisions.
- (4 points) Implement the WASD controls so that the user can simply press the desired key (to move locations) without needing to press "Enter" afterwards.

Implementation Details

As with the previous program, you have additional freedom to design your own implementation. There are certain baseline requirements as specified in this document. However, the exact member functions and member variables are up to you.

Be sure to design your program on paper. Use scratch paper with flow charts or pseudo-code. In particular, think of how your program will flow. What will the constructors do? How will you pass objects and variables between different classes? Taking the time to do a thorough program design can save you hours of frustrating debugging time!

On the topic of debuggers, I strongly recommend that you choose a debugger and get familiar with its operation (GDB, Visual Studio, etc). As your programs get more complex it becomes even more useful to instantly know the value of all your variables and memory contents.

Programming Style/Comments

In your implementation, make sure that you include a file header for every .cpp or .h file. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Be sure to review the style guidelines for this class **(https://web.engr.oregonstate.edu/~goinsj/resources/general/cpp_style_guideline.pdf)** and try to follow them. i.e. don't align everything on the left or put everything on one line!

```
/*********************************************************
** Program: wumpus.cpp
** Author: Your Name
** Date: 02/28/2020
** Description:
** Input:
** Output:
*********************************************************/
```

When you compile your code, it is acceptable to use C++11 functionality in your program (override specifier, final specifier, etc). For ease of implementation, you may also download and use the example Makefile that is provided on the Lectures & Reading tab.

In order to submit your homework assignment, you must create a TAR archive that contains your .h, .cpp, and Makefile files. This tar file will be submitted here to Canvas. In order to create the tar file, use

the following command, replacing the "list_of_all_the_.h _and_.cpp_files" portion with your project files, separated by a space.

```
tar –cvf assign4.tar list_of_all_the_.h_and_.cpp_files Makefile
```

Hunt the Wumpus Rubric (1)

Criteria	Ratings	Pts
Program Header / Good indentation & Use of whitespace / Function Documentation At a minimum, header should contain author's name (1pt) and a description of the program. (1pt) Conditional blocks of code should always be indented. (2pts) Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose (3 pts). -1 pt for each function that is missing a header (up to 3 point penalty).		6.0 pts
Room class Implements Room class with correct members and functions		6.0 pts
Inheritance Implements abstract Event class (4 pts) and derived Wumpus (2 pts), Bats (2 pts), Pit (2 pts), and Gold (2 pts) classes		12.0 pts
Polymorphism uses Event classes polymorphically		6.0 pts
Use of vector uses std::vector to implement a grid of Room objects		6.0 pts
Shooting arrows allows the player to move through the cave and to shoot up to 3 arrows		6.0 pts
Gaming Functionalities Wumpus is killed if hit by an arrow and moves to a different room with 75% probability if woken up by an arrow that misses (3 pts) Adventurer dies if she falls in a pit and is carried to a different room if she encounters bats (3 pts) Game displays percepts based on adjacent events (3 pts) Adventurer wins by picking up gold, killing/not killing the Wumpus, and escaping (3 pts)		12.0 pts
Display and command line arguments (3 pts) Displays a square cave whose size is set by the first command line value (3 pts) Being able to run the program in debug mode specified by the second command line value		6.0 pts
Extra credit 1 (6 pts) Implement an AI class that plays the game		0.0 pts
Extra Credit 2 (4 pts) Implement the WASD controls so that the user can simply press the desired key (to move locations) without needing to press "Enter" afterwards		0.0 pts

Criteria	Ratings	Pts
Auto Deduction -30 Not utilizing inheritance & polymorphism to follow the spirit of the assignment -5 memory leaks (use valgrind and sliding scale for how many/bad it is), -5 no makefile or tar, -5 use of globals or libraries not introduced in class, -5 functions over 20 lines (sliding scale for how many/bad it is), -10 didn't separate files		0.0 pts
		Total Points: 60.0