CS 162 Assignment #4 Hunt the Wumpus

Design Document due: Sunday, 5/21/2023, 11:59 p.m. (Canvas)
Demo due: 6/9/2023 (without penalties)

Goals:
- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
- Practice Object-Oriented Programming
- Implement “Is-a” relationship using Inheritance
- Implement polymorphism and practice STL template by using vector
- Practice file separation and create Makefile
- Use functions to modularize code to increase readability and reduce repeated code

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 Zip 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.

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: you are not required to implement the “do-again” functionality, as it is one of the "extra credit options". (See below for detailed information)

---

**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.

**The Wumpus**

Usually, the Wumpus is peacefully asleep in its cave. Two things wake the Wumpus up, however: 1) the adventurer entering its room or 2) 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 confuses the adventurer, resulting in the adventurer will move in the opposite direction as the user input for the next 5 moves. For instance, if the user enters "w" (north), the adventurer will move to south; if the user enters "d" (east), the adventurer will move to west (which could be dangerous!).
  *Note: if the adventurer encounters another super bat during the 5 moves, the number will be reset to 5. In other words, the adventurer will not be confused for more than 5 moves.*

**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 (east, south, west, or north) 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 **rectangular** grid. The width and the length of the grid should be specified as initial inputs to your program. Caves smaller than 4 rooms on one side (either width or length) 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 5x4 grid might look like
this, where the * character represents the location of the adventurer:

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

- Your program must also accept a third input which will be either "true" or "false". If the
  input is false, then the program will run as normal. If the third input is specified as "true"
  then your game must operate in debug mode.

- When your program is 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, an "f" should be used, followed by either
  "w", "a", "s", or "d" to indicate the direction. For example, the user would enter "f" and "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 (i.e. it must contain purely virtual functions), 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 for the Event class should reflect this. For example, your Event class might have a percept() function 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() function. Similarly, your Event class might have an encounter() function 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 10 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:
• (5 points) Allow the user to play the game again. 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.

• (10 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.

• (20 points) Make a multi-level Cave. Implement a multi-level cave system by introducing a new class called "Cave" that contains a vector of 2-dimensional grids. Each grid will have the same length and width based on the user inputs. Additionally, the program should accept another input specifying the number of levels. In each level, there will still be two pits and two super bats, while the entire cave will have only one Gold and one Wumpus. Furthermore, each level will include a ladder up (^) and a ladder down (v) to facilitate movement between levels. The winning condition stays the same: the adventurer must find and kill the Wumpus, collect the gold, and return to the starting position (where the rope is) in the first level.

• (5 points) Implement the WASD controls so that the user can simply press the desired key (to move locations) without needing to press "Enter" afterwards. You may use Ncurses library if you are implementing this extra credit. More information about ncurses: http://jbwyatt.com/ncurses.html

Programming Style/Comments
In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying 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: 5/20/2023
** Description:
** Input:
** Output:
**
*/
Program Code – Due Sunday, 5/28/2023, 11:59pm on TEACH

Additional Implementation Requirements:
- Your user interface must provide clear instructions for the user and information about the data being presented.
- Your program must catch all required errors and recover from them.
- You are not allowed to use libraries that are not introduced in class, more specifically, you may not use the <algorithm> library in your program. Any searching or sorting functionality must be implemented "manually" in your implementation.
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
  - Make each function do one thing and one thing only.
  - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
  - Functions over 15 lines need justification in comments.
  - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed.
- You must not have any memory leaks or memory errors.
- Your program should not have any runtime error, e.g. segmentation fault.
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Compile and Submit your assignment

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your Makefile to include the proper flag. For example, consider the following approach (note the inclusion of -std=c++11):

```bash
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a zip file that contains your .h, .cpp, and Makefile files. This zip file will be submitted to TEACH. In order to create the zip file, use the following command:

```bash
zip assign4.zip <list of all .h and .cpp files> Makefile
```

Remember to sign up with a TA to demo your assignment. The deadline of demoing this assignment without penalties is 6/9/2023.