Due Date: Friday 19th October 2018

Weighting: 25% of your final mark for the unit

Submission Instructions:

A zip file containing you oject and the associated documentation files (design and reflection) must be ded to the Moodle site. Your code MUST be submitted as a Visual Studio proje of assessment and feedback.

Task Details:

This assignment consis gramming task. The purpose of this assignment is to have you design and in t-oriented program in C++, as well as reflect on your approach and design choices. The assignment comprises the following components:

- ✓ A diagram with annotation that describes your object-oriented design
- ✓ The completed program
- ✓ A 300 word reflection on your program
- ✓ A map of your game environment

Successful completion of the fundamentals of the task as described may obtain you up to a maximum of 80% of the total assignment marks. The last 20% of the mark will be allocated to additional functionality that you can design. The additional functionality should demonstrate advanced or more complex application of principles covered to date. It need not be large amounts of work but should demonstrate a willingness to explore new and advanced concepts. You MUST detail what you have done in an accompanying "readme" file, otherwise markers may not be aware of the extra work undertaken.

The assignment must be created and submitted as a Visual Studio 2017 project. You may complete the exercises in your preferred IDE, however you should create a Visual Studio project in order to submit. Your project folder must be identified by using your name and assignment number, such as ***YourNameA3***. The entire project folder must then be zipped up into one zip file for submission. The zip file **MUST** be named "FIT1048-A3-***YourAuthcateID***.zip". This zip file must be submitted via the Moodle assignment submission page.

Explicit assessment criteria are provided, however please note you will be assessed on the following broad criteria:

- ✓ Meeting functional requirements as described in the assignment description
- ✓ Demonstrating a solid understanding of C++ concepts, including good practice
- ✓ Demonstrating an understanding of specific C++ concepts relating to the assignment tasks, including object-oriented design and implementation and the use of Pointers
- ✓ Following the unit Programming Style Guide
- ✓ Creating solutions that are as efficient and extensible as possible
- ✓ Reflecting on the appropriateness of your implemented design Meeting functional requirements as described in the exercise description

NOTE! Your submitted program MUST compile and run. Any submission that does not compile will be awarded zero marks. This means you should continually compile and test your code as you do it, ensuring it compiles at every step of the way.

If you have any questions or concerns please contact Cheryl as soon as possible.

Assignment Task 3: Hunt the Wumpus

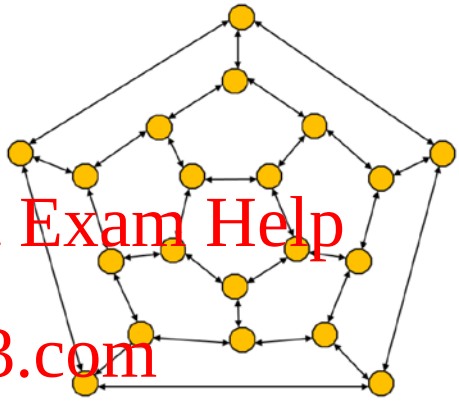
"Well met, young adventurer and welcome to the peaceful realm of Silven Valley. Actually, it used to be peaceful until the Wumpus, a vile creature, took up residence in caverns nearby. It has been terrorising the villages all along the valley for a long time now.

So, the elders from th[illegible]nded together to offer fame and fortune to the worthy adventurer who will ri[illegible]s dreaded monster! You have answered the call and have been told by th[illegible]eature is lurking in one of the cavern complexes in the surrounding hills.

All you have to do is [illegible]d slay them ... simple! Good luck!! You'll need it!"

Basic Game Play

The Wumpus lives in an underground cavern consisting of 20 smaller caves. Each cave has 3 tunnels leading from it to other caves. One example of a cavern complex is shown here (like a squashed dodecahedron).



Besides the Wumpus, there are several other hazards that could be fatal. These include two caves with ***bottomless pits*** – if you happen to go into them, you fall into the pit (and lose). Two other caves have ***super bats***. – if you stumble in them, you are immediately carried away by the bats and left in a random cave (would be annoying to say the least).

The *Wumpus*, on the other hand, is not bothered by these hazards (he has sucker feet and is too big for the bats to lift). It is usually asleep, only waking up if you blunder into its lair or if your shot goes astray (clattering weapons do echo quite loudly underground). If you do manage to wake a Wumpus there is a 75% chance that it will move to an adjoining cave or it may decide (25% chance) to go back to sleep where it is. If it moves to the same cave you are in … you become lunch (and you lose).

When entering the caverns, you have a lantern that has a limited amount of oil, and you are armed with a bow an 5 arrows. Each turn, you may move to an adjacent cave, using one of the 8 standard compass directions [N, S, E, W, NE, SE, SW, NW] (which also reduces your lantern oil) or shoot an arrow into an adjacent cave [SHOOT direction] (hopefully hitting the Wumpus and killing it).

When you enter a new cave, if you are one cave away from the Wumpus or a hazard, you will be given a clue indicating what is in one (or potentially more) of the caves leading from your current location but not the direction in which it lies.

- *Bottomless Pits*: they tend to be drafty, very deep and easy to fall into! They will just kill you outright, should you fall into one. So, watch your step!
- *Super Bats*: a colony of these bats makes an awful racket and are very protective of their domain. If you happen to wander into their cave, they'll swam you and carry you off to some random cave within the caverns.
- *Wumpus*: a nasty beast that smells really dreadful and sleeps a lot. If it's awake and you're in the same cave then you're its lunch … a truly horrible way to die. The Wumpus is not bothered by pits or bats so is free to roam anywhere in the caverns.

End game conditions:

- You win if you successfully shoot the Wumpus.
- You lose if your lantern runs out of oil, or you run out of arrows, or you fall into a pit, or you end up in the same cave as the Wumpus.

Note: The theme of the game can be any setting you like, so long as the main element equivalents (Wumpus and hazards, limited turns, and resources) and general game play as described above are present.

As with the first assignments, you should include a title and a brief description of the caves to make the game interesting and engage the player. You may use the techniques you developed in the first assignments to display this data.

Extra Functionality

The marking criteria indicates that you should make some individual additions to this in order to achieve the final 20% of the marks. What is up to you. You should aim to add some additional creative elements to the gameplay, as well as advanced object-oriented design elements or advanced use of pointers.

Some suggested features could be (but is certainly not limited to):

- The player can collect useful items to help them survive, either before the game starts (eg: select from a “shop”) or they find them as they explore the caverns. Items such as a rope to climb out of a pit, a whistle to confuse the bats, extra oil to fill the lantern, a map of the caves, etc.
- The Wumpus could move one cavern after a random number of turns taken by the player, so the player has to find the Wumpus before it moves too often.
- The arrows are magical crooked arrows which can be shot through 1 to 5 caves. You aim by typing SHOOT and a list of 1-5 directions in which you want the arrow to go (SHOOT N, E, S, S, W). If the arrow can't go that way (ie no tunnel) it moves at random to the next cave. If the arrow hits the Wumpus, you win. If the arrow hits you, you lose.
- Change some of the game conditions:
  - Instead of killing the Wumpus the player has to find a treasure and return to the village.
  - The player selects a skill level which change the number of caves (eg: easy = 15, tricky = 20, hard = 25), the length of time they have (how many turns will the lantern last?), the number of arrows they have, the number of hazards placed in the caves, what items they can buy and/or find, etc.
  - Create several different maps and randomly select one at the start of the game.

Program Design Diagram

In this final assignment you are able to design your program in any way you like. You must provide a diagram with annotation that shows the classes you have in your program and how they interact.

- It should contain any relevant notes that describe the classes.
- This diagram should use the UML notation that has been demonstrated in the labs.

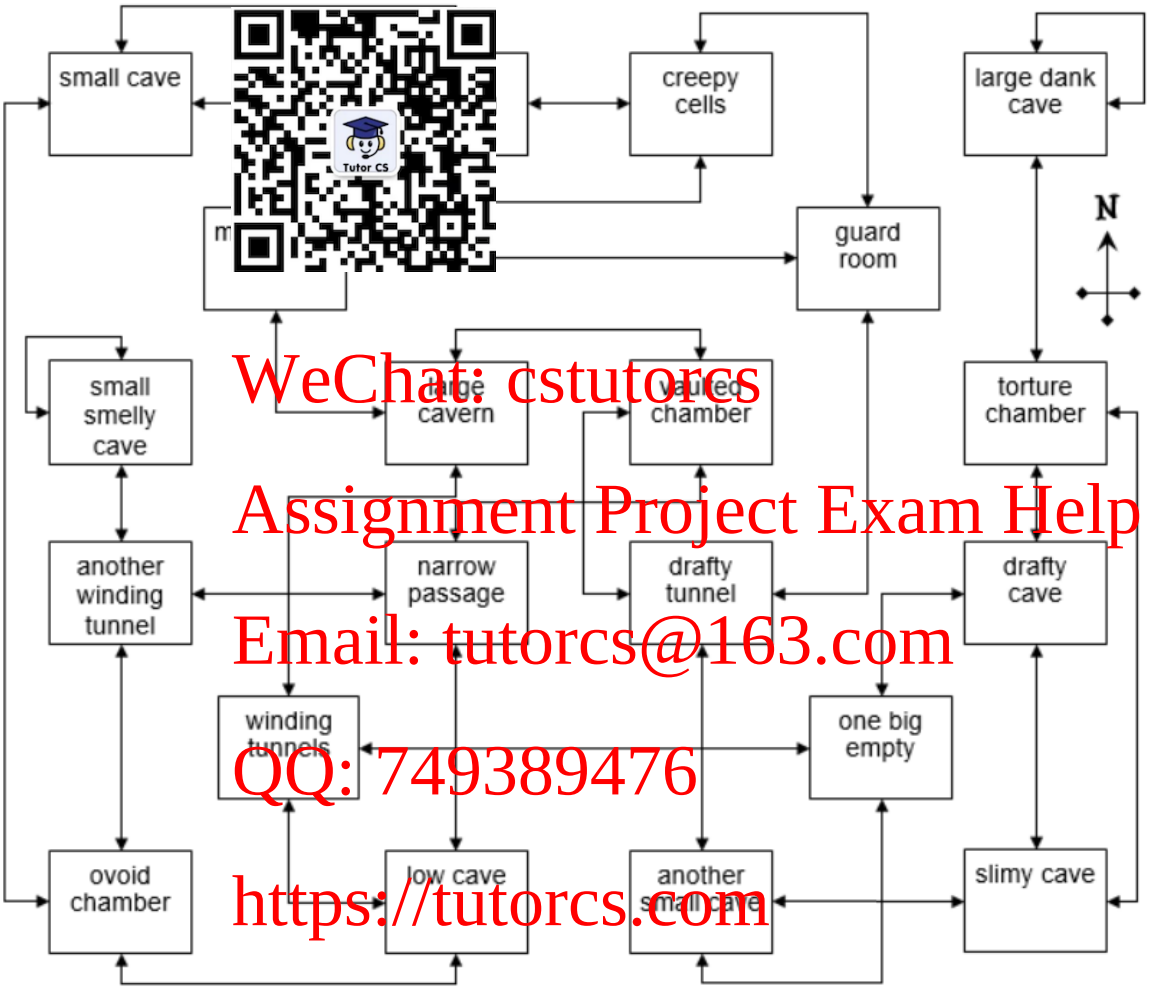
Program Reflection

You must provide a 300-word written reflection of your object-oriented design and how well you believe it was to implement. You should cover the following areas:

- Why you designed it the way you did.
- How well you were able to code it, highlighting any issues you found once you tried to implement your design.
- How you might change your design to make your solution easier to implement, more efficient, or better for code reuse.

Map of Your Game Environment

In order to make marking easier for your tutors, please include a map of your game environment. This can be hand drawn or created in another application (PowerPoint has a range of shapes and connectors that make mapping easy). See example below.



A sample map that you would include in your documentation

Suggested Development Process

Below is a suggested process you might use to develop your assignment piece by piece so that it is not too overwhelming. Feel free to follow this if you think it will help (or not!):

1) Design the program on paper… decide what classes you think you need and how they will relate to each other. Write down your thoughts on why you think this is the best design. You need to do this for the assignment anyway! Don't forget how you can use inheritance and pointers.
2) Write a simple class that you need and test it!
3) Write another simple class that you need and test it!
4) Do these two classes interact? If so, see if you can get basic interaction happening!
5) Repeat the process for each additional class or function… either add to existing classes or write new ones if you need to. Remember, just focus on the small specific task!

I hope this helps. Important thing is to focus on small sub-tasks one at a time and build up functionality that way.

For those of you who are curious, you can explore the following links:

Original article: https://www.atariarchives.org/bcc1/showpage.php?page=247

Wumpus 2 article: https://www.atariarchives.org/bcc2/showpage.php?page=244

Combined articles: https://www.atariarchives.org/morebasicgames/showpage.php?page=178

Assignment 3: Markin[illegible]00 marks in total]

- Does the program [illegible] Yes or No
  - **Zero marks wil[illegible] a non-compiling program.**

Class Design Diagram

- Does the diagram clearly show all the different classes used in the program and each class' attributes and methods? [5]
- Is rationale provided for the classes chosen and overall design? [5]

Functionality [35]

- Game set up, including setting up the player and creating a collection of hazards [5]
- Appropriate game dialog and player interaction [10]
- Appropriate information displayed when entering a cave (title, description, exit directions) [5]
- Correct clues are displayed when a hazard is one location away [5]
- Correct end game conditions applied [5]
- Appropriate display to the screen, user interface, and basic data validation [5]

Quality of Solution and Code [25]

- Has a well-designed OO program been implemented (i.e contains classes appropriate to the assignment brief)? [5]
- Does each class, as written, implement just the things required by the class (i.e the classes do not include things not specific to the class)? [5]
- Is there appropriate use of pointers in the program? [5]
- Does the program perform the functionality in an efficient and extensible manner? [5]
- Has the Programming Style Guide been followed, including documentation, code laid out properly in cpp and header files. etc.? [5]

Extra Functionality [20]

- Does the program addition demonstrate advanced application of programming concepts? [10]
- Does the program addition demonstrate functional creativity? [10]

Reflection [10]

- Discussion of motivations of the program design [3]
- Discussion of how well the design was to implement [3]
- Discussion of what they would do differently if they were to start it again [4]