C++

Treasure Caverns of Doom

This project will be in **TWO parts.** Both parts of the assessment are equally important for the successful completion of your project, so it is essential that you understand the requirements of both parts before you start.

You may complete the tasks in your preferred IDE; however, you **MUST** create a Visual Studio (Code) project in order to submit.

**Download if you don’t have it:** [**https://visualstudio.microsoft.com/vs/community/**](https://visualstudio.microsoft.com/vs/community/)



JUST STARTING OFF WITH **PROTOTYPE (PART A)**

YOU ARE **NOT** DOING **(PART B)** RIGHT NOW (UNLESS YOU WANT TO, WE’LL DISCUSS)

For now:

1. Use attached word document and complete a flowchart and all UML class diagrams.
2. Use attached visual studio project file to code in prototype according to your word document.

* **Treasure Caverns of Doom (Part A: Prototype and Documentation)**

**Due Date:** **(16/09/2022)**

**Marks:** This assignment will be marked out of 100 points.

This assignment is the first part of a larger project, which you will complete in Part B. This task consists of your project documentation and an initial prototype. The documentation must include a flowchart and all UML class diagrams of your programme and a test plan for your final project.

**The prototype must include the application class (Main) and the following four base classes:**

* + **Player:** stores data about the player such as their name, health, inventory, etc.
  + **Monster:** stores data about the cavern’s minions and the boss monster (inheritance applies here)
  + **Location:** stores data about a location such as its description, available exits, other things here, etc.
  + **Item:** stores data about the different items found in the caverns such as the artefacts, gold, potions, etc.

The Main class, for this part of the project, just focuses on the testing of each of your classes.

* **Treasure Caverns of Doom (Part B: Full Project Implementation)**

**Due Date:** N/A (we’ll discuss)

**Marks:** This assignment will be marked out of 100 points.

This assignment consists of the final implementation of your project, as outlined in your Project Documentation and prototype in Part A.

Your project must follow your project plan and must be submitted as a Visual Studio (Code) project, including all header and cpp files, and any appropriate text files to ensure the program compiles and runs.

This assignment consists of one Application file (Main) and all associated custom Class files.

**NOTE!** Your submitted files must be correctly identified submission that does not comply will receive and automatic

and submitted (as described above). Any **10% penalty** (applied after marking).

**NOTE!** Your submitted program **MUST compile and run**. Any submission that does not compile will automatically be awarded a **ZERO**. This means it is your responsibility to continually compile and test your code as you build it.

**Just warning to follow what’s being said and MAKE SURE IT COMPILES & RUNS BEFORE SENDING IT TO ME.**

**P.S Comments should be made to explain parts of code.**

***Submission Instructions:***

This project will be submitted in two parts: (1 for now, 2 if discussed)

…1

C++

**Check the other attached document   
“1. Prototype Documentation”**

**You will do Part A There.**

Treasure Caverns of Doom

* **Part A:** consists of your project documentation and a test prototype.

This document will include an outline of your **program structure** as a **flowchart** and **UML Class diagrams.**

**Your document should contain the project flowchart and the clearly identified UML diagrams of all your proposed Classes. You can use the *Insert > Table* option in Word to create the UML diagrams or you can use any other software, provided that the diagrams are included as *legible images* in your submitted document.**

Your project prototype must be submitted as a Visual Studio (Code) project, including all header and definition files, and any appropriate text files to ensure the program compiles and runs.

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

* + Details of a proposed project plan for the overall project.
  + Creating accurate and complete UML diagrams.
  + Applying a solid Object-Oriented Design (OOD) for the overall project.
  + Using appropriate naming conventions.
* **Part B:** consists of a reflection document and your final implementation of your game project. This document will contain a 500-1000 word written reflection of your object-oriented design and how well you believe it was implemented.

The assignment documentation must be created as a **single Word or PDF document**.

Your final project must follow your previous project documentation and be submitted as a Visual Studio project, including all header and definition files, and any appropriate text files to ensure the program compiles and runs.

*Explicit assessment criteria are provided, however please note you will also 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 OO design, the implementation and appropriate use of pointers, inheritance and polymorphism.
* A reasonably well-designed user interface displaying relevant details and appropriate feedback.
* Creating solutions that are as efficient and extensible as possible.
* Reflecting on the appropriateness of your implemented design and meeting functional requirements as described in the assignment description.

…2

C++

GAME DETAILS

Treasure Caverns of Doom

***Treasure Caverns of Doom (Part A)***

In this Project, you are to design an action adventure game, where your hero character will set out to find and collect ancient artefacts hidden in a complex of caverns. You must use your powers of observation and stealth to defeat the minions that guard the treasures that you must collect which are scattered throughout the caverns!

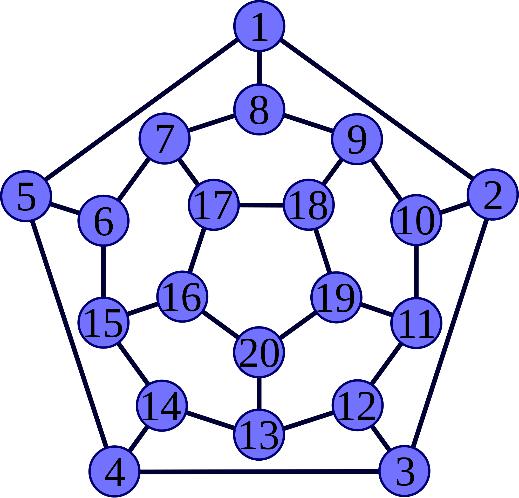
**For Part A of the project you will focus on documenting and prototyping the project with the four base classes required demonstrating your class design and basic testing to ensure the classes are ready for your final game implementation.** In Part B you will focus on developing the game setup and player interactivity demonstrating the implementation of the game interface design and the more complex aspects of the player’s interactions within the game environment.

*While enjoying the spoils of your latest adventure in the village tavern, a cagey old man approaches you and asks you this simple question: “Do you want to earn a lot of money for very little effort?”*

*Of course, your interest is piqued, “And how would I do that old man?”*

*“Simply go to the Treasure Caverns of Doom and retrieve the 5 artefacts hidden there by the great Wizard Balthazar many centuries ago. I have found a map to the location of the caverns, but my age prevents me from retrieving them myself. I will give you the map if I have your vow that you will return the artefacts to me. You may keep any other gold and treasures you find.”*

*You ask the man for more information about the caverns, discovering that the treasures are guarded by an assortment of monsters (one of which is rumoured to be a dragon), supposedly bound by Balthazar to guard the caverns until his return. Intrigued and ready for another adventure you agree to the old man’s terms, take the map, collect your gear and set out to find the Treasure Caverns of Doom …*

**

***Basic Game Play***

The Treasure Caverns are an underground cavern consisting of

20 smaller caves. Each cavern has 3 tunnels leading from it to other caves. One example of a cavern complex is shown here (like a squashed dodecahedron).

Throughout the caverns, the various minions of the Dragon (it was not just a rumour), roam its tunnels challenging anyone they meet to combat. If a minion is also guarding an artefact, it will defend it vigorously or die in the attempt.

You may also stumble upon small caches of treasure, presumably left by previous adventurers who did not survive long enough to return and claim it.

The Dragon is always guarding the most valuable artefact and will not be easily defeated. You will need to exploit some of the other artefacts you have found to aid you.

When entering the caverns, you have a map, a lantern that has a limited amount of oil which also sets the number of turns you have, you are armed with a sword, and have a backpack with some provisions. Each turn, you may move to an adjacent cave, or do one or more of the other actions available to you.

***Player Interactions***

When you enter a new cave, you may examine it, looking for a treasure cache OR engage in combat with a monster OR flee from the battle, taking a small amount of damage from the monster’s attack OR move on to the next cavern heading down one of the visible exits.

The Actions a player may take include **e[X]amine, [A]ttack, [F]lee, [N], [S], [E], [W], [P]rovisions, [T]ake, [D]rop, [M]ap, [H]elp** and **[Q]uit.** The letter [?] is the shortcut key the player can use to execute that command. You canread a full description of each command option in Part B of the brief.

***Artefacts and Other Items***

There are 5 artefacts that are placed in random locations at the start of the game. Your objective is to collect them all and return them to the old man in the village. Unfortunately, the artifacts are guarded by monsters that you must defeat before you can claim them – either minions of the Dragon or the Dragon itself.

Other items you may find in the caverns include potions, gems and coins that you can collect and sell when you return to the village.

…3

C++

Treasure Caverns of Doom

***Monsters and Combat***

There are a range of different minions that are either guarding an artefact or are wandering the tunnels always looking for a fight. If there are any wandering monsters, you had best be on guard should they try to attack you while you are searching.

Monsters, like the player, have attributes and skills that they can employ when engaging in combat. Each type will have the same skills but may have different attributes making them easier or harder to defeat. Combat is explained in detail in the main section of the brief.

***End game conditions***

* You win if you successfully collect all the artefacts and escape the caverns (return to the entrance).
* You lose if your lantern runs out of oil (turns), or you run out of health (combat). You should try to avoid these things!

***The Caverns***

You should include a title and a brief description of the caverns (locations) to make the game interesting and engaging for the player. You may also use the techniques you developed in the first assignment to load and display this data.

***Preparing the Treasure Caverns of Doom (Part A)***

***Project Documentation***

Having a clear plan for your project before you begin coding is essential for finishing a successful project on time and with minimal stress. So part of this assignment is defining what will be in your final project and what you need to do to actually develop it.

***Important:*** *You might want to check out the requirements for Part B to fully understand how to complete Part A. (But remember you are currently just working on Part A for this deal)*

The documentation you must submit will include the following:

* **Flowchart of the entire game structure**

In a nutshell, a flowchart is a visual representation of the game’s algorithm. It’s a useful tool that can help you see how your programme works. It makes it easier to focus on the flow of the game while developing your programme because you have already identified where all the game decisions and loops will be.

The symbols you’ll most likely need are:

**Symbol Title: what is it used for**

****

**Terminator:** Use at the beginning and end of a flow sequence. When you begin a sequence, add

the name of the sequence in the symbol (eg: display map).

**Process:** Use to describe the current step or action in the sequence (eg: get input, read data from text file, display data in varName, etc.).

**Decision:** Use when the programme must choose something usually in the form of a question (eg:



is Wumpus in Location? has arrows left? etc.). Evaluates to true or false.

**Pre-defined Process:** Use when you want to link to another flowchart. This is only necessary for

processes that require a finer level of detail (eg: “move Wumpus” would appear in the main flowchart then you could create another smaller chart describing what happens when you move the Wumpus). It’s not necessary for things like, get user input or display title.

Remember to keep your symbols concise and easy to understand by someone else. This is meant to be read by others so now is not the time to use secret codes that only you understand!

The level of detail required is “big picture” not “fine detail”. We don’t necessarily need to know all the fine details, as that is something we’ll see in your code, but we need to be able to understand how you are thinking and processing the various elements within your game.

**Tip:** Use the **Process** box to show things happening in a sequence and the **Pre-defined Process** box to identify a function that encapsulates a specific task using an appropriate description.

**Your flowchart should include:**

* The game setup (everything that happens before the game starts)

…4

C++

Treasure Caverns of Doom

* The player’s turn (the sequence of events that happen during a turn)
* Processing player input (the commands your player can use)
* Providing feedback to the player (in response to the player’s interactions)
* The end game conditions (include all win and lose conditions)
* Include the functionality of all your game classes – see Part B
* Additional Features included, if any – see Part B

Here is an example to get you started with and algorithm that you can convert into your flowchart:

1. The Game Setup
   * Read game rules from text file and display
   * Add the player – ask for the player’s name, set default variables
   * include all the other things that will happen during initialisation including
     + creating the hazards and the game world
     + initialising other game variables

As you can see, you only have to describe the actions the program will take, not the code. The idea here is to give you a starting point for when you start writing your code as you can use this as a checklist of the things you need to include.

* **Project Test Plan**

Part of any project development strategy is the inclusion of a project test plan. This document lays the foundations of how you plan to test the various elements within a project to ensure that it remains robust and does not crash when being used by others.

This this part of the assignment you are required to identify how you will test each individual class (as per your prototype) and the main game play elements (as per your final implementation). We will use this test plan when we are marking both your prototype and final project to ensure that you have done the required amount of testing.

We expect evidence of testing for the 4 base classes and the Dragon class, and the following 3 aspects of the game play – selecting options, combat including the player winning or losing, and the win or lose end game conditions. While the Dragon inherits from the Monster class (as per Part B) you must still include it as a class – you must still show how you test it but the inheritance and polymorphism features do not have to work.

Create a table in your document, using the example provided here as a guide, filling each section with appropriate information that is specifically relevant to your implementation of the game.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Function** | **Description** | **Expected Result** | **Actual Result** |
|  |  |  |  |  |
| Class Name | functionName() | What’s its purpose? | How did you test it? | What was the result? |
|  |  |  |  |  |
| add other classes here | |  |  |  |
|  |  |  |  |  |
| **#Scenario** | **Function** | **Description** | **Expected Result** | **Actual Result** |
|  |  |  |  |  |
| Selected [?] | functionName() | What’s its purpose? | How did you test it? | What was the result? |
|  |  |  |  |  |

add other functions here

You must include all your classes and all the related functions for the following game scenario criteria:

• **selecting an option:** include each one and the test done to ensure correct functionality

• **combat sequence:** include the player winning or losing and reaching the correct outcome

• **end game conditions:** include the player winning or losing the game with all possible outcomes

* **A description of “How to Play”**

This will be the information you display on the opening screen of your game to inform the player what they need to do in order to play and win the game. Do not just copy the game play description above, make your own description that includes the things unique to your interpretation of the game, such as items the player can find and use, any special features, etc.

You should also have a separate text file that the player can access at any time to display the various command options available and what they are used for. For example, each of the actions a player can use, maybe some hints on game play, etc.

***Note:*** This description must be saved as a text file and read into your game when the program begins. You can also use this as a convenient way to display “help” if the player asks for it.

…5

C++

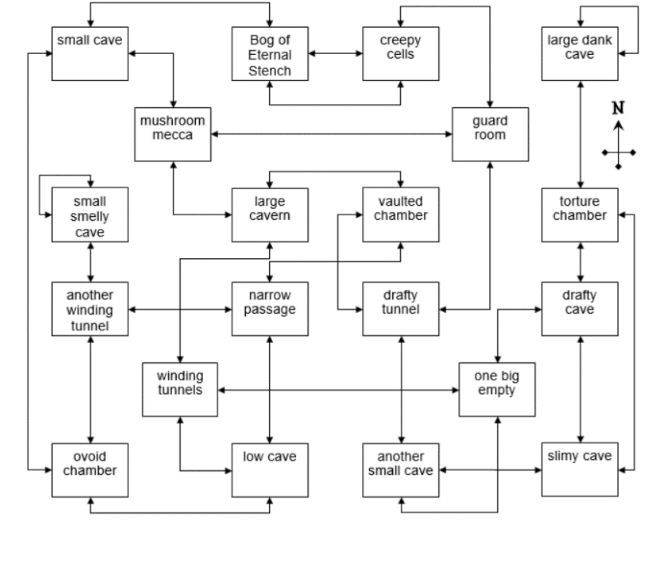
Treasure Caverns of Doom

* ***UML Diagrams***

UML diagrams are designed to make structuring your program easier. The general structure is shown here – see Part A brief for more details about classes.

You will need UML diagrams for each of the classes you include in your game – at least a Player (Character), Monster, Location, Item and the Main (application) class.

* ***Map of Your Game Environment***

****

In order to make marking easier for your tutors, please include a map of your game environment with location names. This can be hand-drawn or created in another application. If hand-drawn, please ensure that the map is clear and easy to read (not fuzzy images).

PowerPoint has a range of shapes and connectors that make mapping easy. This map was easily created in PowerPoint.

Remember the map requirements of:

* + each location having exactly 3 exits although the entrance can have a 4th exit leading out of the caverns for the player to escape
  + use a short descriptive name to help us identify where we are when marking your work

**ClassName**

list of attributes (variables)

list of behaviours (functions)

A sample map that you would include in your documentation

***Project Prototype :: What Needs to Work***

Your prototype is NOT the completed game. Its purpose is for you to demonstrate your initial development process of the basic custom Base Classes (Player, Monster, Location and Item and a Dragon) and to provide appropriate evidence that each has been thoroughly tested before you begin you final implementation.

The following are brief overviews of the 4 base classes you must include in your prototype, but please read the class descriptions under the Part B brief for more detailed information. Our expectation is that these classes are as close to fully formed as you can make them and should not require much additional development in Part B, other than a few small tweaks as required.

* ***Player Class***

This class holds the player’s details including their name, initial values for their health, attack, defence and luck attributes, and the equipment they are carrying (inventory).

* ***Monster Class***

This class holds the monster’s details including their name (eg: goblin, ogre, etc.), initial values for their level, health, attack, defence, damage and location attributes, and treasure they may be carrying (a small item or artefact).

The Dragon is a Boss monster which also has a special attack of breathing fire which it may use once during combat. This class inherits from the Monster class their base details but will have their own specific functionality but for the prototype you only need to create the class and include its special skill.

* ***Item Class***

This class holds the item’s details including its name, type (defined as enum for an artefact, potion, weapon, armour or item), its uses and location attributes, and any special effect(s) it may have.

The player must have a map, a sword, a lantern, and some provisions. The Backpack does not have to be a specific item but would be included in descriptions as what the player uses to carry items. All these things should be generated Items that are stored in the player’s inventory.

The monsters must also demonstrate that they can have a randomised item (any of the 5 types above).

…6

C++

Treasure Caverns of Doom

* ***Location Class***

This class holds the location’s details including its name, description, visible exits (as compass directions) and the destination of each exit – if in Location 1 the player could move to Locations 2 (SE), 5 (SW) or 8 (S), and see something if it is in the location (a monster or an item).

Your Locations should display relevant information to enable the user to be able to identify where they are on the map – its name and/or description and the visible exits as compass directions, and if a monster or item is in the location.

You DO NOT have to implement moving in a direction (N,S, E, W, etc. as described in the current location) as that will be completed in the final version.

* ***Main (application) Class***

For each class you must demonstrate that you can create a Class Object using both the default and an overloaded constructor with appropriate arguments and a have appropriate accessor and mutator functions for each attribute. All classes must include a **getDetails()** function that displays all the generated data of the class.

***Part A: Marking Criteria [100 marks in total]***

**Does the program compile and run?** Yes or No

***NOTE:*** *Your submitted program MUST compile and run. Submissions that do not compile will receive* ***ZERO****.*

Ensure you leave time to test your work thoroughly before you submit it.

**Project Documentation [50]**

* Flowchart Design **[15]**
  + Has used the correct flowchart symbols [1]
  + Has used appropriate descriptors in flowchart symbols [1]
  + Flowchart includes all game functionality (Parts A & B) [2]
  + Flowchart clearly shows the correct sequence of the game (setup, gameplay, end game) [4]
  + Pre-defined processes include all the required game functionality [5]
  + Processes are performed in a logical order [2]
* UML Class Diagrams **[10]**
  + Correct structure used (Name, Attributes, Behaviours) [1]
  + Included the correct designations for public [+] and private [-] data [1]
  + All variables and functions have meaningful names [1]
  + Included constructor(s), destructor, and appropriate access and mutator functions in all custom classes [4]
  + Included a class diagram for the Player, Monster, Location, and Item classes and the Main (application) [3]
* Project Test Plan **[20]**
  + Each class tests all constructors, accessor and mutator functions [12]
  + Appropriate testing procedures for selecting options, combat and end game conditions [8]
* Other Documentation **[5]**
  + The introduction to your game as a description (own words) [3]
  + Included a map with location names [2]

**Class Design [40]**

* Overall Class Implementation **[10]**
  + All classes have a correctly structured header and definition file [2]
  + All classes include appropriate constructor, destructor, accessor and mutator functions [4]
  + All classes include all the required attributes as given in the class descriptions [2]
  + All classes include the getDetails() function that displays all the class attributes appropriately [2]
* Individual Class Implementation **[12]**
  + The Player class initialises its attributes with random values [1]
  + The Player class adds the required items to the player’s inventory [1]

…7

C++

Treasure Caverns of Doom

* + The Monster class initialises its attributes with random values [1]
  + The Monster class adds a random treasure item, artefact or nothing [1]
  + The Dragon class initialises its attributes with random values [1]
  + The Dragon class adds includes the special skill of breathing fire [1]
  + The Item class initialises its attributes with values appropriate to each item [1]
  + The Location class initialises its attributes with values appropriate to each location [1]
  + The Location class displays all relevant data about the location as required by the brief [1]
  + The Location class displays the first location with the player in it [1]
  + The Location class displays a random location with a monster in it [1]
  + The Location class displays a random location with an item in it [1]
* Does each class have a testClassName() function that demonstrates the following: **[18]**
  + Create each class object using the default constructor and displays its details [4]
  + Create each class object using an overloaded constructor and displays its details [4]
  + The created class object can access each of its attributes and display it [4]
  + The created class object can modify all appropriate attributes and display the changes [4]
  + Demonstrates access to attributes of objects within other objects (eg: items in the inventory) [2]

**Quality of Solution and Code [10]**

* The code is well formatted and easy to read **[2]**
* Correct naming conventions have been applied **[4]**
  + Including all functions/methods and variables using meaningful names and applied camel case
* Appropriate commenting and code documentation **[4]**
  + Including file headers containing author’s name, date, purpose, one line comment before each functions, non-trivial code has additional inline comments

**END OF FIRST PART; READ THE REQUIREMENTS BELOW BUT DON’T DO SECOND PART UNLESS WE AGREE ON IT.**

…8

C++

Treasure Caverns of Doom

***Treasure Caverns of Doom (Part B)***

You are to implement the final version of the Treasure Caverns of Doom game you started in Part A by completing your Visual Studio Project using your project prototype as presented in your previous submission.

***Your completed Treasure Caverns of Doom game must demonstrate the following:***

You MUST implement your program using the following classes, as a minimum, but you may include more if it is appropriate for your game design:

* **Player class:** holds the player’s details including their name, initial values for their health, attack, defence and luck attributes, and the equipment they are carrying (inventory), as defined below:
  + **Name:** the player is asked for a name to use at the start of the game.
  + **Health:** is initialised using a random value from 4-12 then adding 12.
  + **Attack:** is initialised using a calculation based on the initial health (*health / 6*).
  + **Defence:** is initialised using a calculation based on the initial attack (*attack / 2*).
  + **Damage:** is calculated on the player’s *(attack / 3) + current weapon damage*.
  + **Luck:** is initialised using a random value from 2-6 then adding 6.
  + **Inventory:** the player’s backpack can hold a maximum of 12 items.

These attributes may be modified by weapons (attack and damage), armour (defence), potions (any), and artefacts the player may acquire throughout the game.

* **Monster class:** is the base class for the minions and the Dragon\* (boss) and holds the monster’s base details including their name, initial values for their level, health, attack, defence, damage and location attributes, and treasure they may be carrying (a small item or artefact).

Use this table to calculate the various attributes of the monsters you generate at the start of the game. The monster’s location should be randomly determined (excluding the entrance). There is a percentage (%) chance that the monster will also be carrying a random item or artefact which the player can collect if they defeat it.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Level** | **Health** | **Attack** | **Defence** | **Damage** | **Carrying Item** |
|  |  |  |  |  |  |  |
| Goblin | 1 | level x (2-3) | level + (1-3) | level + (1-2) | level | item = 40% |
|  |  |  |  |  |  |  |
| Hobgoblin | 2 | level x (3-5) | level + (2-4) | level + (1-3) | level + (1-2) | item = 50% |
|  |  |  |  |  |  |  |
| Ogre | 3 | level x (4-6) | level + (3-5) | level + (2-4) | level + (1-3) | artefact = 60% |
|  |  |  |  |  |  |  |
| Troll | 4 | level x (5-7) | level + (4-7) | level + (2-5) | level + (2-4) | artefact = 70% |
|  |  |  |  |  |  |  |
| Dragon\* | 5 | level x (6-8) | level + (6-10) | level + (6-12) | (claw) level + (4-6) | artefact = |
| (fire) level x (2-3) | 100% |
|  |  |  |  |  |
|  |  |  |  |  |  |  |

The Dragon is a Boss monster which also has a special attack of breathing fire which it may use once during combat. This class inherits from the Monster class its base details but will have their own specific functionality.

* **Item Class:** holds the item’s details including its name, type (defined as enum for an artefact, potion, weapon, armour or item), the number of uses it has, and location attributes, and any special effect(s) it may have.

This table provides some example of the items you can include in your game – you are free to add to this list. You can store the details of your items in a text file and store them in individual item objects if you wish.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Type** | **Uses** | **Item Notes** |
|  |  |  |  |
| Map | item | unlimited |  |
|  |  |  |  |
|  |  |  | Lights way losing 2-5 points with each move or search |
| Lantern | item | 100+ | action. Allow a realistic number of turns for the player to |
|  |  |  | complete the quest (anything < 50 is impossible to do). |
|  |  |  |  |
| Provisions | item | 2-4 | Each provision restores 5-10 **health** |
|  |  |  |  |
| Gems / Gold | item |  | 1-3 gems or 6-12 gold coins are carried or found |
|  |  |  |  |
| Sword | weapon |  | adds +2 damage to user’s current **damage** |
|  |  |  |  |
| Leather Armour | armour |  | adds +4 to the user’s **defence** |
|  |  |  |  |

…9

C++

Treasure Caverns of Doom

|  |  |  |  |
| --- | --- | --- | --- |
| Potion Flask | potion | 1-2 | random result when drunk, +/- 2-4 to one attribute |
|  |  |  |  |
| Krang’s Battle Axe\* | artefact |  | adds +8 damage to user’s current **damage** |
|  |  |  |  |
| Krang’s Chainmail\* | artefact |  | adds +12 to the user’s **defence** |
|  |  |  |  |
| Boots of Speed\* | artefact |  | adds +6 to the user’s **defence** |
|  |  |  |  |
| Elixir Maxima\* | artefact |  | if drunk by the player, they immediately lose the game |
|  |  |  |  |
| Balthazar’s Spell Tome\* | artefact |  | guarded by the Dragon |
|  |  |  |  |

The player must start the game with a map, a sword, a lantern, and some provisions. The Backpack does not have to be a specific item but would be included in descriptions as what the player uses to carry items. All these things should be generated Items that are stored in the player’s inventory.

The player must have the Krang’s Battle Axe and Chainmail, and the Boots of Speed if they are to successfully defeat the dragon. They have to have in their possession all 5 artefacts(\*) and escape the caverns (return to the entrance) to win the game.

You are permitted to include other items that suit your game if you have time to add and test the appropriately.

* **Location class:** holds the location’s details including its name, a short description, visible exits (as compass directions) and the destination of each exit – if in Location 1 the player could move to Locations 2 (SE), 5 (SW) or 8 (S), and if something is in the location (the player, a monster or an item).

Your main game interface will be displaying this data to the player in an uncluttered and easy-to- read manner, reflecting the current status of the location appropriately – if a monster or item is here, results of searching, and so on.

You may include other relevant attributes and behaviours to these classes, as identified in your project document.

* **Main (application):** holds the main() function and controls the overall flow of the game.
  + The **Player Class** must be able to do the following:
    - assign a name which is requested at the start of the game and used in the feedback given
    - assign the player’s initial attributes to the required values (as stated in the brief)
    - updates and displays the player’s statistics (stats) during the game as required
    - check and updates the appropriate player attribute when an item is collected (if any)
  + The **Location Class** must be able to do the following:
    - data must be loaded from a text file and stored appropriately at the start of the game
    - assign a name, description and initialise attributes (as stated in the brief)
    - access and display data about a monster or item if it is in the player’s current location
    - update the display data when an item or monster is removed from the player’s current location
    - display all appropriate information when the player enters it (name, description, exits, other information if required)
  + The **Monster** base Class and the inheriting **Dragon** class must be able to do the following:
    - assign the monster’s initial attributes of name, level, health, attack, defence, damage
      * be placed in a randomly assigned location
      * be randomly assign an item to carry based on their random chance if a random value from 1-100 ≤ the assigned chance value
    - the inheriting Class must do the following as stated in the brief under Monsters:
      * contain an appropriate description
      * be placed in a randomly assigned location
      * perform the required special action when activated during combat
  + The **Item Class** must do the following:
    - * assign the item’s initial attributes of name, type (enum), and number of uses (if applicable)
      * be placed in a randomly assigned location
      * displays its name or type when it appear in the current player location
      * the item is not usable when its uses are reduced to zero or less
  + The **Game Application** must do the following:
    - display the “how to play” information at the start of the game
    - initialise the player, locations, monsters and items with appropriate attributes/behaviours

…10

C++

Treasure Caverns of Doom

* + display an appropriate and uncluttered user interface providing relevant information to the player at all times
  + allow the player to explore the locations of the caverns using compass directions (N,S,E,W)
  + display the information about each location as it is entered (name, direction of exits, description, etc.)
  + terminate the game (player wins) when the player has collected all artefacts and returned to the entrance
  + terminate the game (player loses) when the player has run out of oil (no turns left) OR has run out of health
  + provide player stats at the end of the game (success or failure, inventory, etc.)
  + the player should be able to QUIT the game at any time
* ***Game Interface Design***

Having a clear and uncluttered main game interface is an integral part of this assignment. We are expecting to see, as a minimum, the following elements:

* + the title of the game
  + the location description
  + player status and options (including values for artefacts, gold/gems, health and turns left)
  + an appropriate prompt message
  + and NO MENUS using numbered options to make decisions

The Screen should also clear whenever the player enters a new location, so the player does not have to scroll through the display.

* ***Game Interactions***

The actual Player must be able to do the following:

* + **e[X]amine:** If no monsters are in the cavern, you may search for treasure using [X]. There is a 30% chance of finding a few gold coins or a small gem, a 20% chance of finding some other item (not an artefact), otherwise nothing is found. This will consume oil from your lantern, but it may prove to be time well spent, or not.
  + **[A]ttack:** If a monster is in the cavern you can choose to attack it using [A]. If you decide to attack, the combat sequence will activate, with you winning or losing at the end of the battle sequence.
  + **[F]lee:** If you choose not the engage in combat, you can flee using [F]. However, you will automatically take the minimum damage that monster would inflict in combat.
  + **[N], [S], [E], [W]:** If there is nothing in the cavern, perhaps after you have searched it, you can use one of the compass directions listed. You will not be able to go in a direction that is not listed. Moving also reduces your lantern oil by 2-5 points.
  + **[P]rovisions:** Allows you to eat one of your provisions which restores some of your health. If you have also collected potions you can drink one to gain its benefits or other effects!
  + **[T]ake and [D]rop:** Allows the player to add and remove things from their inventory.
  + **[M]ap, [H]elp and [Q]uit:** are all required actions that must be accessible at any time during the game. The map of the caverns and the help should be stored in separate text files and read and displayed when requested. The player should also be able to quit the game at any time (during or after a game).

Player input must be correctly processed, using basic validation to ensure that the programme doesn’t crash if an invalid value is entered. The letters in **[?]** are the shortcut keys used to execute the desired command.

The Player must also be able to display an ASCII version of the map using an appropriate input option. The map data must be read from a file and displayed appropriately. The Player must also control when they have finished looking at the map (eg: Press any key) and must return to the same location they were in they asked to see the map.

The tutor must be able to follow the Player’s movement using the map provided in your game (accessed by using [M]). The Player must also be able to quit at any time.

* ***Combat Rules***

Here are the rules of combat which you must implement when the player chooses to **[A]ttack** a monster in their current location. The combat sequence is as follows:

* + Calculate the player’s attack value as: their attack + a random value from 2-10
  + Calculate the monster’s attack value as: their attack + a random value from 2-10
  + If the player’s attack value is greater than the monster’s attack value
    - calculate damage inflicted as: player’s damage value minus monster’s defence
    - subtract points equal to the damage inflicted from the monster health
  + If the monster’s attack value is greater than the player’s attack value
    - calculate damage inflicted as: monster’s damage value minus player’s defence
    - subtract points equal to the damage inflicted from the player health
  + Repeat until either the player’s or the monster’s health reaches zero then the battle is immediately over

…11

C++

Treasure Caverns of Doom

* if the monster wins it’s “game over” for the player
* if the player wins and the monster is carrying an item or artefact display it in the location

The player may now take the item and/or continue playing the game, doing any other action(s) here or moving on.

You should now have more than enough information to complete the development of a fully functional project of the Treasure Caverns of Doom!

***Extra Functionality***

The marking criteria indicates that you should make some **individual additions** to this project in order to achieve the final 20% of your mark.

Following is a list of additional features you can include, with the maximum number of marks [x] you can earn for each one. You may implement one or more features from the list, but you will only score up to the maximum of 20% of the total assignment marks or 20 marks.

You should aim to add some of these additional creative elements to the gameplay, to enhance the presentation of your project.

* The player can select a skill level (eg: Rookie, Adventurer and Master Adventurer) which modifies the game

parameters. For example: the number turns the lantern lasts, the number of monsters placed in the caves, what items they can find, how much treasure they can carry, etc. **[2]**

* Implement an appropriate scoring system based on the game parameters – number of turns taken, a bonus for unused items collected, +/- points based on a win/loss at the skill level, etc. **[2]**
* The Dragon moves to an adjacent cavern, actively seeking the player, after a random number of turns taken by the player, so they have to avoid the Dragon until they have all the artefacts they need to defeat it. **[3]**
* Create 3 or more different maps, one of which is randomly selected and loaded at the start of the game. The correct one must be displayed when selecting the [M]ap action. You must still implement all other elements of the game as described in the brief AND include a copy of each map for the tutors. **[3]**
* Allow the game to be saved and restored at the player’s request. **[4]**
* The player must buy their starting items before the game starts (eg: select or buy from a “shop”). Items should

be suitable to your game: such as a range of weapons and armour, extra oil to fill the lantern, provisions to restore health, etc. The Player will also need to have a random amount of gold coins at the start. **[4]**

* Create all your items as derived classes using a base Item class. Each derived item will have a different abilities

– ordinary items (gems, coins, etc.) only add to the player’s inventory, weapons and armour have unique bonuses for attack and defence when used (the player has to wear new armour and arm a new weapon), potions have unique effects that can do more than change player attributes (teleport, find treasures, etc.), and artefacts can have multiple abilities. **[10]**

* Create 2-3 other roaming treasure hunters that the player has to defeat and/or avoid while exploring the game locations. These opponents can be generated before the game and placed randomly throughout the locations or they can be randomly generated as required (linked to a skill level, after a random number of turns, etc.). Each opponent should also have a special action that they will do when they encounter the player (fight, steal an item, talk, exchange items, etc.) based on the type of “character” they are. **[12]**

You certainly do not have to implement all of the above to earn marks for extra functionality. Just remember the maximum number of marks you can earn are given in [x] and is based on how well you implement the given task.

***Project Development Reflection Document***

You must also provide a 500-1000 word written reflection of your object-oriented design and project development process and how well you believe it was implemented.

* Discuss why you designed it the way you did.
  + Why did you create your classes that way?
  + How does this reflect an OO design or approach?
* Discuss how well you were able to code it.
  + Highlight any issues you found once you tried to implement your design.
  + How did you resolve these issues?
* If you were to do this project again:
  + Discuss how you might change your design to make your solution easier to implement, more efficient, or improve for better for code reuse.
  + Describe what things you have learned during the development of your project.

…12

C++

Treasure Caverns of Doom

***Part B: Marking Criteria [up to 100 marks in total]***

***NOTE:*** *Your submitted project must be correctly identified and submitted, otherwise it will receive an automatic* ***10% penalty*** *(applied after marking).*

**Does the program compile and run?** Yes or No

***NOTE!*** *Your submitted program MUST compile and run. Submissions that do not compile will receive* ***ZERO****.*

**Class Design [15]**

* Monster and Derived Class (Dragon)
  + All classes have an appropriate header files **[2]**
  + Base class modified to suit requirements for inheritance and polymorphism **[2]**
  + Required data members and member functions of each class use meaningful names **[2]**
  + Appropriate application of virtual functionality in each inheriting class **[3]**
  + The appropriate implementation of pointers to ensure polymorphic functions work correctly **[4]**
  + Contains only aspects that relate to the specific inheriting class (has no data members or member functions that are not directly related to that class) **[2]**

**Game Functionality [35]**

* Displays the “how to play” information at the start. **[1]**
* Initialises the player and game variables appropriately **[1]**
* Creates the locations, monsters and items using appropriate data **[3]**
* Implementation of a clear and uncluttered User Interface display **[3]**
* Successful implementation of the Player class functionality **[4]**
* Successful implementation of the Location class functionality **[4]**
* Successful implementation of the Item class functionality **[4]**
* Successful implementation of the Monster class functionality **[4]**
* Successful implementation of combat between the player and the monsters with correct consequences **[5]**
* Successful implementation of action processes and feedback displayed to the player **[4]**
* Appropriate end game conditions triggered **[2]**

**Evidence of Functionality Testing [10]**

* All the available options tested with invalid options handled appropriately **[3]**
* Combat sequence works as described in the brief with both win and lose results included **[4]**
* All possible end game scenarios are include and are selected when required **[3]**

**Extra Functionality [20]**

* Implement difficulty levels which may be selected by the player **[2]**
* Implement an appropriate scoring system based on game parameters **[2]**
* The Dragon could move after a random number of turns **[3]**
* Create and implement 3 or more different maps **[3]**
* Allow the game to be saved and restored at the player’s request **[4]**
* The player must buy their initial items from a shop before they play **[4]**
* Create all your items as derived classes using a base Item class with unique behaviours **[10]**
* Create and implement roaming opponents for the player to interact with **[12]**

**Quality of Solution and Code [10]**

* Does the program perform the functionality in an efficient and extensible manner? **[6]**
  + Code Architecture – algorithms, data types, control structures and use of libraries [3]
  + Coding Style – Clear logic, clarity of variable names, and readability [2]
  + Applied good programming principles and practices [1]
* Has the Programming Style Guide been followed appropriately? **[4]**
  + Appropriate commenting and code documentation [3]
  + Correct formatting of code within \*.h and \*.cpp files [1]

**Project Development Reflection Document [10]**

* Discussion of motivations for 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]**

…13