

COS30008: Data Structures and Patterns

Problem Set 1

Weighting: This assessment contributes 5% to the overall unit marks.

Learning Outcome(s) assessed: 1.

The focus is on being aware of commonly reoccurring programming tasks, how apply the best choice of data structures to quickly deliver them, and fully utilize the structures' accessibility/functionality.

This is a “reduced tasks” version of Problem Set 1. So, inside your report, you only need to supply the indicated items (ignore the Task Subsections from previous Problem Sets).

In each Problem Set, there is a number of Tasks that you need to read, understand and deliver. The **required deliverables** (e.g. diagram, description, code & proof and etc.) will be stated. Write a report to document these deliverables.

Name your report as “**COS30008_2024S2_PS1_<YourStudentID>**” and submit via Canvas as **PDF**.

C++ Aptitude and OOP Exercise

Computer games are a great example of complex programs, nigh impossible to design and develop without resorting to Object-Oriented Approaches. Regardless of whether you want to build a mobile app, desktop standalone application, or a multiplayer client, it's nature and content can be represented by UML diagrams so that other programmers understand what needs to be built.

Preparation

1. In this Problem Set, you will exercise your OOP mastery by first deciding on a genre that you like. Check here and pick a few to start with:

[Video Game Genres Everything You Need To Know | HP® Tech Takes](#)

[List of video game genres - Wikipedia](#)

2. Now that you have picked the genres you like, choose your favorite titles or look online for videos of example games in the same genres. Notice that there are similarities present in every game, such as:

- a) Someone representing you - the player. AKA the Player Character (PC)
- b) Other entities surrounding you in the game, who can be Friendly, Hostile or Passive. These are Non-Player Characters (NPC).
- c) There is a place involved. This has many names for this – Maps, Levels, Instances, etc. This is the “Environment”.
- d) Finally, there is the Mission. This describes the Victory Condition, Losing Condition and what you can do within the game. Can you identify these from within your examples?

In Tower Defense, MOBAs, and RTS game design, in-game entities are often referred to as “Units”. A Unit can belong to one of the Human or Computer players. A Unit can be spawned as one of the varieties of Unit Types available in a game (e.g. in StarCraft: Wraith, SCV, Ghost, Dragoon, Protoss Carrier, Hydralisk, etc.). Each Unit has its own internal data structure to contain its in-game Position, Health (Hitpoints), Energy, Shields, Movement Speed, Attack Speed, Attack Damage, Owner, Game State and more attributes. A player’s goal is to defeat all other opponent players by building their own units and commanding them to eliminate all of the opponent’s units.

For more background in Strategy Game Design:

https://en.wikipedia.org/wiki/Strategy_video_game

3. Now that you have reached this point, determine your own unique **Game Title**, **Genre**, **Entities** (NPCs and/or PC), **Environment** and **Mission**. Assume that your game will be taking the **Text-Based Perspective** (this can change but at your own risk).

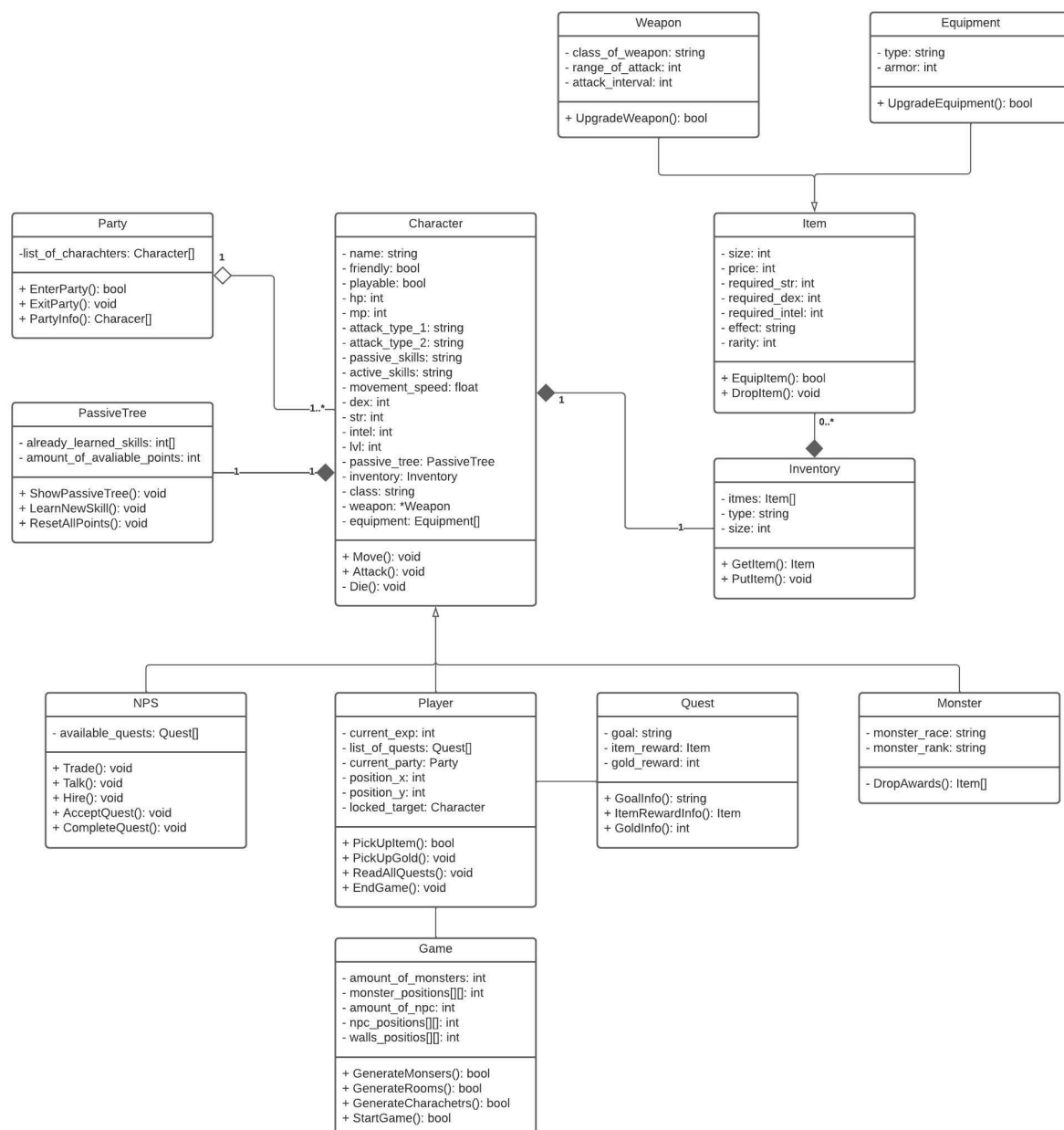
Now, you are ready to begin attempting the Problem Set.

Task 1 (1%): Conceptual Design of Game Content

Write a summary of your game idea (Title, Genre, Description of how does it work).

Draw a Class Diagram to describe the elements (entities) that are present within your game idea. Demonstrate your present understanding of Object-Oriented Design by incorporating concepts of **Abstraction, Attributes & Methods, Inheritance, and Polymorphism** into this diagram. It must have at least 3 different Derived Classes of the same Base Class (you can call this class "Entity" or similar).

The figure below shows an **EXAMPLE** of a Role-Playing game's model of the elements in it. You **do NOT** have to follow this. **Add Associations** to hint at how each class is related/interacts with the other:



Task 2 (1%): Base Class for all Units and Creating Derived Classes

Based on Task 1's class diagram, implement a C++ class to represent a generic "Entity" in your game. Create both the header and implementation files for this class (Entity.h and Entity.cpp) within a Visual Studio C++ Project.

Remember that "Entity" is any object that is present within your game. This can either be a Unit, Item, Terrain, or something else. Make sure that every Entity has these attributes:

- a) fID – unique identifier for this Entity
- b) fPosition – the Entity's position in the 2D game world (use Struct)
- c) fMaxHP – the Entity's total health
- d) fCurrentHP – the Entity's current health
- e) fAttack – the amount of damage this Entity gives in an attack

Next, add these Methods into the base class:

- a) Default Constructor – Creates an empty Entity. Initializes all members.
- b) Overloaded Constructor – Populate the attributes during Entity Object creation.
- c) Accessors for each attribute – Getters and Setters (Get_ and Set_).
- d) Destructor – Used to release memory upon Deletion of this Entity.

Then, practice how derived classes work:

- a) Demonstrate how you can apply OOP Inheritance by extending the Entity class into at least 3 different derived classes. (Example: you can derive one for a Zealot, Dragoon, Carrier, Nexus, Pylon, etc.).
- b) Make sure each derived class has unique attributes (fields and/or methods). This differentiates themselves from other Unit Types in terms of behavior.

In the Main function, test the functionality of your code by instantiating these different Entities and show the accessors are working correctly.

Paste the Full Source Code Text here, including comments to explain how it works.

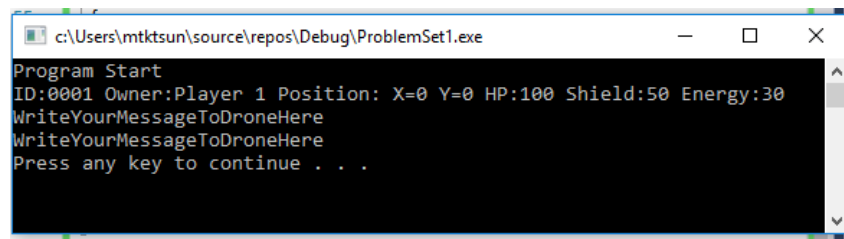
Include **Screenshots of the Command Prompt** to prove the functionality of your program.

Task 3 (1%): Unit Communications

Make a copy of the Task 2 program so you can continue developing Task 3 without overwriting it:

- Add a new private field (type string) called "fMessage". Its purpose is to remember the last message that is sent to it.
- Add a new public function into the Entity Class called "PrintStat". When called, this Entity will return a string that lists the contents of all its current fields.
- Create a Friend Operator Overload of the Insertion Operator (<<), making use of "PrintStat" to insert the field contents through "cout".
- Create a Friend Operator Overload of the Extraction Operator (>>), making use of accessor functions to save new values into the fields.
- Create testing code in the Main Function to show that this is working.

The output should look something like this:



```
c:\Users\mtktsun\source\repos\Debug\ProblemSet1.exe
Program Start
ID:0001 Owner:Player 1 Position: X=0 Y=0 HP:100 Shield:50 Energy:30
WriteYourMessageToDroneHere
WriteYourMessageToDroneHere
Press any key to continue . . .
```

Paste the Full Source Code Text here, including comments to explain how it works.

Include a **Screenshot of the Command Prompt** to prove it is operational.

Task 4 (1%): Parsing Commands

A spawned Unit cannot see or manipulate the attributes of other Units directly (thanks to encapsulation), but a Game Engine can communicate with them to change their parameters.

E.g. In a DOTA 2 game session, Sniper shoots Pudge. Programmatically, the Sniper Unit prompts the Game Engine to message the Pudge Unit: "damage 50". Internally, the Pudge Unit will process this message and determines that it is instructed to deduct his Current Health by 50 points.

Make a new copy of the Task 3 program, and continue modifying it:

- Change the Friend Overloaded Extraction Operator (>>) function so that the input message's content is parsed for <Command> and apply the <Value> changes to the appropriate attribute.

Example Commands:

Command Message	Effect
Damage <Value>	Deduct CurrentHP by <Value>
Heal <Value>	Add CurrentHP by <Value>
Move <Value> <Value>	Increments Pos.X and Pos.Y appropriately

Paste the Full Source Code Text here, including comments to explain how it works.

Include a **Screenshot of the Command Prompt** to prove it is operational.

Task 5 (1%): Offence is the Best Defense

Make a new copy of the Task 4 program, and continue modifying it:

Your final challenge is to make the following scenario possible.

There are 2 entities - "Ryu" and "Ken". Both have maximum HP of 100, and Attack of 30.

In Code: Ryu >> Ken;

Effect: Ken's fCurrentHP is now 70

If either Ryu's or Ken's fCurrentHP is 0 or below, the game ends.

Paste the Full Source Code Text here, including comments to explain how it works.

Include a **Screenshot of the Command Prompt** to prove it is operational.