Learning Summary Report

	Pass	Credit	Distinction	High Distinction
Self-Assessment (please tick)	✓	✓	✓	

Self-assessment Statement

	Included (please tick)
Learning Summary Report (This document)	✓
Semester Test with all corrections	✓(Last Opportunity)
C# or C++ Programming Reference Sheet	✓
Principles of OOP Report	✓
Reflection	✓
At least 50% of Pass Tasks signed off as Complete	✓

Minimum Pass Checklist

	Included (please tick)
All of the Pass requirements	✓
100% of Pass Tasks signed off as Complete	✓
Concept Map	✓
Evidence of credit tasks that have been completed	✓

Minimum Credit Checklist, in addition to Pass Checklist

	Included (please tick)
All of the Credit requirements	✓
100% of Credit Tasks signed off as Complete	✓
Code for your program that demonstrates good OO design *including the use of polymorphism, implementing an interface and managing collections of objects*.	
UML class diagrams and UML sequence diagrams that communicate the design your custom program	
Description of what your program does and screenshots of your program in action	

Minimum Distinction Checklist, in addition to Credit Checklist

	Included (please tick)
All of the Distinction requirements	
Research report and associated pieces	

Minimum High Distinction Checklist, in addition to Distinction Checklist

Introduction

This section summarises what I learned about Object Oriented Programming. It includes a justification of the assessment pieces included in the portfolio, details of the coverage of the unit learning outcomes, and a reflection on my learning.

Overview of Pieces Included

This report hard include my learning in this unit.

- Shape drawer- c# programming that can create a graphical shape drawing on screen
- Swinwarts Magic School – A C# program that cast spell onto console
- Case study adventure- C# programming to allow user to interact with item, travel.
- Planetary Rover – C++ programming that building an attach device with battery and able to operate
- Semester Test
- Distinction project – dice game

Coverage of the Intended Learning Outcomes

This section outlines how the pieces I have included demonstrate the depth of my understanding in relation to each of the unit's intended learning outcomes.

ILO 1: Object Oriented Principles

Explain the principles of the object oriented programming paradigm specifically including abstraction, encapsulation, inheritance and polymorphism.

The following pieces demonstrate my ability in relation to this ILO:

Encapsulation

Encapsulation ensure a good code modularity, make sure keeping code separated from the other, and limit the accessibility. Encapsulation is a way of <u>hiding the internal mechanisms and data structure of a software component behind a defined interface</u>. It is to ensure the user of an object on dependently on it interface, the implementation could be changing overtime. <u>Encapsulation is to encapsulate code and data feature within a class.</u> We can picture object as a capsule where it is a cohesive entity containing information hiding. It gives us the use of public interface and private implementation. Encapsulation is a packaging of the object as a class which contain private fields (Variable, things it know) and public method (things it can do), a private field known as the information needed to perform the functionality of the object it has all the logic behind the object but private field is kept protected where other object cannot access it directly but instead access it through a method call property. Property allows other object to access and modify. For example, in the Swin Game it requires a lot of encapsulation where as each shape is encapsulating into individual classes such as Circle Class, it contain radius (things it knows) as private field and its draw circle function (things it can do) as public method.

Abstraction

Abstraction is use as a planning method to plan a large programming solution where it is more oriented and simple to understand, it is to <u>hiding details to focus on fewer concept</u>. Abstraction method is to <u>break down</u> into a smaller object that does specific task. The detail of the task does not be needed but only the key feature from the object is used. For example, in SwinGame we <u>broke the program down to pieces</u> to a few classes such as Drawing class, Circle class, Line class and Rectangle Class, Drawing class does not know the parameter behind Rectangle class but know the key feature of Rectangle class to draw a rectangle, it is to <u>hiding details to focus on fewer concept.</u>

Inheritance

Inheritance is a relationship between classes, one class is a derived class where it can inherit all the feature from the base class. A derived class is a specialized version of the base class, or alternatively the base class is a more general version of the derive class. By using inheritance we can create a new derived class that inherit all the behaviour of its parent class (base class) and provide inheritance of all fundamental of base class. Derived class are able to extend the ability of base class by adding more method into the child class. For example, Circle class are the derived class of the base class Shape class, it have specific responsibility to draw a circle but it inherits the method and properties from their base class Shape class.

Polymorphism

Polymorphism enable derived class to be use as an individual base class with similar but different functionality. Polymorphism allows child class to override the method in parent class to do specific task. Polymorphism also allows sub casting of classes. Polymorphism is a type of overriding and overloading between derived class and base class. For example Circle and line class inherit the property of draw class but each of them override Draw method to draw their desire shape.

- Shape drawer- Circle class inherit the property of shape class
- Swinwarts Magic School – Able to inherit the property the cast spell class
- Pass Task 13 _Explain the basic of 4 principle OOP
- Planetary Rover – C++ programming that building an attach device with battery and able to operate.
- Semester Test – Uses the 4 principle

ILO 2: Language Syntax

use the class library System.Collections.Generic which allows me to use List<>, use Class Library system.Drawing.Color which allow me to use draw different colour in ShaperDrawer Program. I also like to use Random function. It is so much easier to use compared to C language where you need to seed the time. There are many more library that I have use such as System.IO, SwinGameSDK etc.

ILO 3: Writing Programs

C# are develop using the IDE Xamarin and the C++ are using Visual Studio, C++ are harder as we have only short time to finish the task with nothing thought to me.

In writing program, I've learn to use Nunit test to debug the program, and this is a good habit.

ILO 4: Object Oriented Design

I have demonstrate my ability in relation to this ILO by coming out with UML diagram and UML sequence. This allow me have an overall view to my program and let me have a good understanding and good logical thinking.

…

ILO 5: Program Quality

1) **Comment and Documentation**

Commenting is useful, it allow IDE and other tools to utilize them in different way for example when you call out some function they will have a description of the function where you've commented. Commenting also make the program better to edit improve or find out the problem. By documenting programmer will know what the program uses are without having to remember it in a pile of coding.

2) **Consistently Indentation**

When writing a code we should be best to consistently indent the code, it is a good practice to make the coding logic and style clear. If you are part of a team or if you are contributing code to a project, you should follow the existing style that is being used in that project. Indentation make the code clear and easy to read.

3) **Avoid Obvious Comment**

   Avoid obvious comment, overdone comment could be redundant, the usage of the code is simply obvious to know its functionality then there will be no need of commenting in every single line of code, instead use commenting on the functionality of the whole function.

4) **Code Grouping**

   Grouping code is the best way to make code more tidy, sometime we only uses a few line of code for certain task and it is best to group them and space it from other code, it makes the code looks tidier and easy to read.

5) **Avoid deep level**

   It is best not to make code hard to read by adding a lot of level in the code, such as a lot of if else cases inside each other.

6) **Naming**

   Naming should be a good practice in programming, naming a function and class would give a clear functionality and readability to the writer and also anyone who going to access the code.

7) **Coding Line's Length**

   It is good to limit the line length to a readable length, it is hard to read and diagnose or revise code when there are code which have a really long line

8) **Unit Testing**

   Unit testing is a good practice to diagnose the error, when there is a big project with different people working on different part unit testing will provide a good idea of coding error when combine or logically mistake. During working on the task I often find out unit testing shows a lot of result that I have not expected and able to find out the mistake of my logic, although the compile is successful but a small mistake have change the outcome. Unit test is a good way of testing before release or use of a program.

9) UML Diagram and UML Sequence

UML Diagram and UML Sequence enable me to have a clear view of logic before coding. It is useful during working of big project and complicated programming, it will help to clear things up.

Reflection

The most important things I learned:

I've learn a lot from this unit, the principle behind OOP and also the good programming practice, Good programming practice provide a good overview of the program make the thinking clear.

The things that helped me most were:

Internet is my best friend, any problem that I've faced can be solve online by finding discussion or similar problem that others have face.

I found the following topics particularly challenging:

C++ is very challenging, I've spend over few days to learn C++ from scratch, I've know nothing from start about C++.

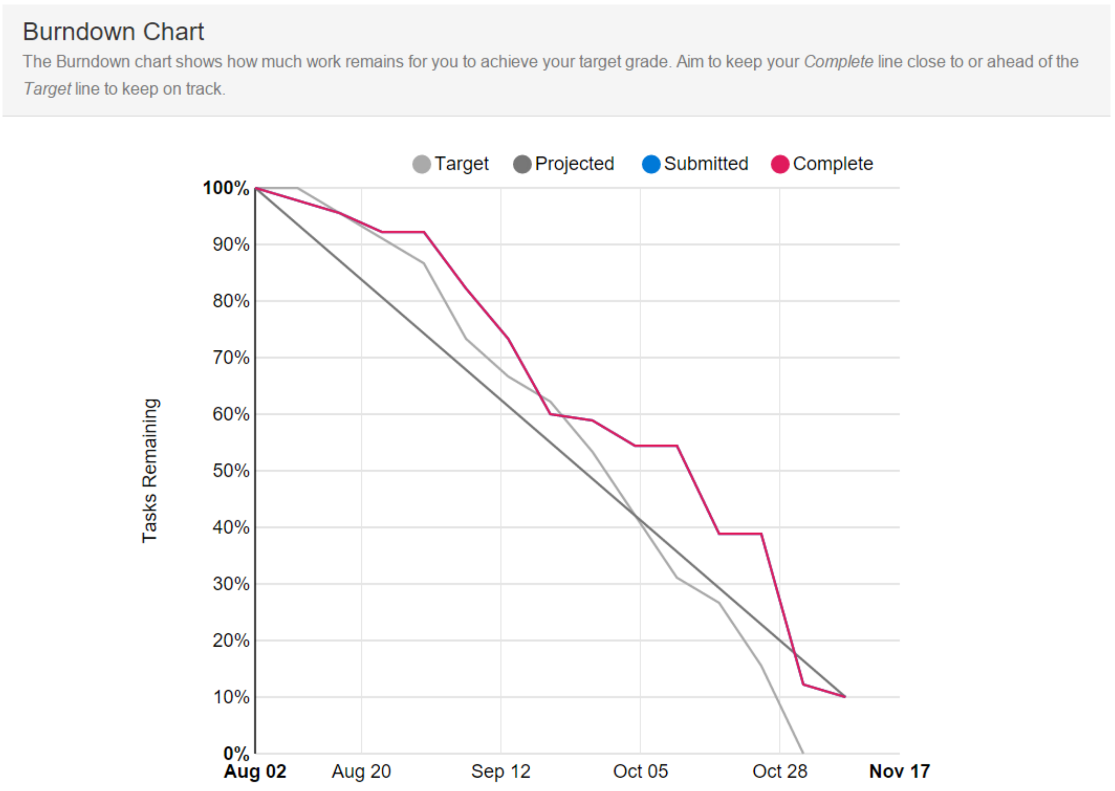
I found the following topics particularly interesting:

C# are interesting, I'm able to use different IDE to perform different coding feature.

I feel I learned these topics, concepts, and/or tools really well:

I feel that I've learn c# well, there are more IDE out there work flawlessly with C#

My progress in this unit was …:


Burndown Chart

The Burndown chart shows how much work remains for you to achieve your target grade. Aim to keep your *Complete* line close to or ahead of the *Target* line to keep on track.

[Include a screenshot of your progress graph from DoubtFire, and comment on what happened from your perspective… what does the graph say about how you approached the unit? (Login to Doubtfire to get your graph https://doubtfire.ict.swin.edu.au)]

All task are submitted on time except C++ which spend me a lot of time and the other study had hold me up against finishing these task. The uncompleted task will be the Distinction task and also this learning summary

This unit will help me in the future:

[How will the things you learned relate to the rest of your studies, and career. What have you learned that will be valuable for you in the future?]

The logical thinking will help me in my future study and problem solving, my programming skill will also improve.

Concluding Statement

In summary, I believe that I have clearly demonstrate that my portfolio is sufficient to be awarded a …. Distinction.

[