COS10009 – Introduction to Programming

Learning Summary Report

TA QUANG TUNG (104222196)

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment (please tick)				✓

Self-assessment Statement

	Included (please tick)
Learning Summary Report	✓
Test 1 and Test 2 are Compete in Ed	✓
All Pass level tasks completed (including tutorial tasks)	✓

Minimum Pass Checklist

	Included (please tick)
All Credit Tasks are Complete in Ed	✓

Minimum Credit Checklist, in addition to Pass Checklist

	Included (please tick)
Distinction tasks (other than Custom Program) are Complete	✓
Custom program meets Distinction criteria & Interview booked	✓
Design report has structure chart and screenshots of program	✓

Minimum Distinction Checklist, in addition to Credit Checklist

	Included (please tick)
HD Project included	✓
Custom project meets HD requirements	✓

Minimum High Distinction Checklist, in addition to Distinction Checklist

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Signature: 

Portfolio Overview

This portfolio includes work that demonstrates that I have achieve all Unit Learning Outcomes for COS10009 – Introduction to Programming to a **High Distinction** level.

I believe I should be awarded the High Distinction grade because I have successfully completed all tasks in my portfolio, including the HD maze creation and maze search tasks. My portfolio tasks exceed the minimum pass level, including the GUI music player which I completed at the top credit level (67).

All my portfolio code reflects good syntactic and stylistic conventions and follows structured programming practices which are appropriate functional decomposition, high cohesion, and low coupling. My custom code (extended music player) demonstrates a high level of logical complexity and is extensively documented to show its structure.

Additionally, I have gone beyond the materials presented in this unit. I applied the knowledge acquired in this unit to customize the Bezier curve generation function for task 3.3P. I also made extensive use of many of the untaught features of the Gosu library for my extended music player.

Reflection

The most important things I learnt:

This unit introduced me to a code testing method I did not know before – bottom-up testing. Before this unit, I often tested my code top-down by observing variables with *puts* and *print* statements. However, this approach becomes infeasible when the program becomes extremely large.

This unit also taught me the principle of high cohesion and low coupling. In order to achieve this, I need to thoughtfully plan out my program at the start so that everything would not be coupled up together. This also helps me later when I scale up my program – otherwise I might have to rewrite everything when I add a new feature.

Finally, I learned that to be a good programmer, you must be an independent learner. In the programming world, there are so many languages and libraries that no university course can fully cover all of them. Therefore, I will need to learn on my own and extrapolate the knowledge acquired while learning one language/library to many others.

The things that helped me most were:

The two most helpful things for me throughout this course were the Ruby API Reference (which comes bundled with the Ruby interpreter) and the official Gosu documentation. Since I primarily work with Ruby and Gosu, these documents allowed me to look up the functions and classes I needed to use. I also found the object-oriented and structured code comparison on Ed quite useful. I originally wrote much of my extended music player in object-oriented style and this material helped me convert my object-oriented code to purely structured code.

I found the following topics particularly challenging:

I found the concept of cohesion and coupling quite challenging when I first got introduced to them in week 8. I did not immediately understand how they were related and how high cohesion correlated with low coupling. However, I eventually realized that cohesive modules rarely interact with other modules, thus decreasing coupling.

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

I think I learned the Gosu library really well throughout the course. It took quite a bit of personal experimentation but in the end I managed to understand and utilize most of the Gosu library to develop my extended GUI music player. It made extensive use of the Gosu module as well as the Window, Image, and Song classes.

I still need to work on the following areas:

An area I still need to work on is deciding on the trade-off between stamp coupling and conciseness. Sometimes a function or procedure might require most but not all fields in a record. In these situations, I will have to decide whether to pass in the entire record (which risks adding in unneeded fields) or pass in just the fields I need (which requires typing out more parameters). I still find it hard to choose between the two at times.

This unit will help me in the future:

This unit has helped me hone some of the foundational programming skills required to take on larger projects: writing code that follows good syntactic and stylistic conventions for each language, writing code that is high in cohesion and low in coupling and applying the appropriate debugging techniques to identify problems in code.

Additionally, the knowledge I gained while working with Gosu in this unit will help me work with similar libraries. Gosu follows a traditional game loop consisting of handling click events, updating logic and rendering, which is also used by light-weight game libraries such as PyGame or Love2D.

The primary language for this unit is Ruby, which is an object-oriented language. Even though the focus of this unit is not OOP, I did manage to learn the three principles of OOP (polymorphism, inheritance, and encapsulation) while doing research on how classes and attribute accessors work in Ruby. This knowledge will be useful when I learn other object-oriented languages in the future.

If I did this unit again, I would do the following things differently:

Overall, I am quite happy with what I did during this unit. However, if I were to retake this unit, I would probably start preparing for my custom code project near the start of the semester. I feel like I did not have enough time to complete everything the way I wanted to for this unit.