COS30031 Games Programming

Learning Summary Report

Thomas Horsley - 103071494

Self-Assessment Details

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

	Pass (P)	Credit (C)	Distinction (D)	High Distinction (Low HD) | (High HD)	
Self-Assessment (please tick)			X		

Self-assessment Statement

	Included? (tick)
Learning Summary Report	X
Complete Pass (“core”) task work	X

Minimum Pass Checklist

	Included? (tick)
Additional non-core task work (or equivalent) in a private repository and accessible to staff account.	
Spike Extension Report (for spike extensions)	
Custom Project plan (for D and/or low HD), and/or High HD Research Plan document (optional)	X

Credit Checklist, in addition to Pass Checklist

	Included? (tick)
Custom Project Distinction Plan document, approved	X
All associated work (code, data etc.) available to staff (private repository), for non-trivial custom program(s) of own design	X
Custom Project “D” level documents, to document the program(s) (structure chart etc) including links to repository areas	X

Distinction Checklist, in addition to Credit Checklist

	Included? (tick)
Custom Project “HD” level documents, to document the program(s) (structure chart etc) including links to repository areas	

Low High Distinction Checklist, in addition to Distinction Checklist

	Included? (tick)
High Distinction Plan document, approved	
High Distinction Report document, which includes links to repository assets	
All associated work (code, data etc.) available to staff (private repository) for your research work	

High High Distinction (Research) Checklist, in addition to D/Low HD 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: Thomas Horsley

Introduction

This report summarises what I learnt in COS30031 Games Programming. It includes a self-assessment against the criteria described in the unit outline, a justification of the pieces included details of the coverage of the unit's intended learning outcomes, and a reflection on my learning.

Overview of Pieces Included

Core Work:

1. Spike - Git Repository Setup
2. Spike - C++ for Programmers
3. Spike - Gridworld
4. Lab - Debugging
5. Lab - Data Structure Basics
6. Spike - Performance Measurement
7. Spike - Game State Management
8. Spike - Game Data Structures
9. Lab - File Input Output
10. Spike - Game Graphs from Data
11. Spike - Command Pattern
12. Spike - Composite and Component Pattern
13. Lab - SDL2 Concepts
14. Spike - Soundboard
15. Spike - Sprites and Graphics
16. Spike - Messaging and Announcements
17. Spike - Collisions
18. Spike - Profiling Performance and Optimizations

Extension Work:

1. Custom Project Plan (D-HD)
2. Custom Project

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: Design

Discuss game engine components including architectures of components, selection of components for a particular game specification, the role and purpose of specific game engine components, and the relationship of components with underlying technologies.

Engine architecture and design was prevalent during the second half of the unit. The first example of engine design was in Task 13 – "Composite and Component Patterns". As I heavily overengineered this task, I've since come to realize that the patterns and ideas implemented here are used extensively within game engine architecture & design.

Task 19 – 'Messaging Systems' related to events and dispatching messages to the relevant components within the ECS implemented in Task 13. Polling and event structures were also seen in the SDL2 related Spikes, specifically "Sprites and Graphics" where input was used to randomly instance textures. Additionally, "Sprites and Graphics" introduced me to how a graphics renderer function and process data before presentation.

This (along with the "Soundboard" Spike), allowed me to design a rudimentary, code-based engine environment for my custom project. This engine uses an ECS designed for turn-based strategy styles games and supports rendering ASCII environments. Additionally, input systems have been developed using the concepts discussing Task 19 – 'Messaging Systems'.

ILO 2: Implementation

Create games that utilise and demonstrate game engine component functionality, including the implementation of components that encapsulate specific low-level APIs.

The second milestone for the unit was making a Zork-esque CLI game (titled Zorkish) utilizing the concepts and ideas discussed in the first portion of the semester. My version of Zorkish implements an ECS (Entity Component System) and input polling system implemented in the "Composite and Component Patterns" and "Messaging Systems" Spikes respectively. The ECS implemented within my Zorkish project was built such that it's extensible and performant enough to support hundreds of game entities containing components. The SDL2 Spikes ("Soundboard" and "Sprites and Graphics") served as an introduction to low level graphics API's, allowing me the tools to interact with GPUs for parallel computing during a game / engines' runtime.

My understanding of SDL2 and Entity Component Systems can be seen in my 'Custom Project' and 'Custom Project Plan'. The architecture seen here allows for rapid prototyping of ideas and quick iteration on decoupled, modular subsystems. Additionally, many of these components and systems implemented within my 'Custom Project' contain SDL data structures and Rendering operations specific to the Low-Level Graphics API.

ILO 3: Performance

Explain and illustrate the role of data structures and patterns in game programming and rationalise the selection of these for the development of a specified game scenario.

Performance and optimizations are important for games as if ignored, will most likely lead to an unplayable experience for the user. All the core runtime code (input → update → render) must have a maximum compute time less than the users desired framerate. Spikes 07 – 'Performance Measurement' and 24 – 'Profiling Performance and Optimizations' directly involved measuring the performance of functions and algorithms whilst effectively communicating my findings.

Task 07 – 'Performance Measurement' analyses and discusses the initial overhead and per-iteration cost of three unique data-structures frequently seen in games / game engines. From this I now understand the appropriate use-cases for multiple data-structures, the pros and cons associated with each data structure and how to conduct rudimentary performance testing on general data-structures. This task gave me insight into how the Standard Template Library architects their own data structures and allowed me to use these containers more effectively.

Expanding on this was Task 24 – 'Profiling, Performance & Optimizations', in which I learnt to measure and analyse functions and larger sets of code in the form of an AABB collision detection system. This task allowed me to research and understand the processes behind third-party / in-code profiling and program optimization. Whilst researching Task 15 – 'SDL2 Concepts', I was able to gain an understanding surrounding additional API specific render-flags and system calls which granted measurable performance improvements in the context of games.

These three tasks have heavily influenced all of my proceeding work. My 'Custom Project Plan' and 'Custom Project' display a marked distinction between my past use of data structures and my use of them today. I have a basic understanding of cache coherency and data abstraction and am able to construct custom data structures when necessary in order to optimize performance.

ILO 4: Maintenance

Explain and illustrate the role of data structures and patterns in game programming and rationalise the selection of these for the development of a specified game scenario.

The way data is stored and managed throughout the lifetime of a game will heavily influence the performance of the game as-well as the scalability and maintainability of its codebase. Task 06 – 'Data Structure Basics' introduced some core C++ data structures and useful Standard Template data structures such as Vectors, Stacks, Queues and Lists. Each of these structures were analysed and their useful features demonstrated. A second analysis was conducted in the following Task 07 – 'Performance Measurement' however this was under the lens of performance (discussed above).

Furthering my knowledge of data structures was Task 09 – 'Game Data Structures' and Task 11 – 'Game Data from Graphs'. The former of which applied Tasks 06 & 07 and had me to choose the appropriate solution for an inventory system before implementation. This led me to understand templated containers and generics in C++ as we'll as demystify any initial confusion with STL data structures. Task 11 allowed me to understand graph structures better and I learnt a great deal about data visualisation, container structure and relationships which is seen throughout future Zorkish related tasks.

Patterns were first covered in Task 08 – 'Game State Management' which introduced me to the State pattern as-well-as useful resources relating to game pattern architecture. Task 10 – 'File Input Output' saw me apply the factory pattern to instantiate game entities based of file data. This had me to develop an understanding of saving and loading concepts and proved useful in future Spikes pertaining to Entity Component Systems and Zorkish. In addition to world loading, the Factory pattern was applied in Tasks 12 – 'Command Pattern', 13 – 'Composite & Component Patterns' & 19 – 'Messaging and Blackboards'.

'Command Patterns' introduced me to the idea of game data abstraction, allowing multiple objects to interact with a data hub independently of each other helping decouple code. The Command pattern was applied in all future Zorkish tasks and can be seen in my D level custom project description in the EventDispatcher and System child-classes. 'Composite & Component Patterns' had me attempt to understand and implement and ECS (Entity Component System) architecture within Zorkish. I learnt that an ECS is a beast and not something I can implement properly within a week… I also learnt that ECS' are really cool and in larger and more modular frameworks allow for faster, more concise game system interactions. An ECS allows for system decoupling and for specific systems to only effect entities with the appropriate functionality as provided by its components. This allows for easier implementation of system-specific optimizations as the overall architecture is proven to be more robust than its inheritance-based counterpart. Finally, Spike 'Messaging & Announcements' introduced me to the Dispatch and Blackboard patterns. I implemented these abstractions by introducing Event Dispatcher and Input Handler objects which are responsible for taking the users input and dispatching an event notification to the relevant gameplay system.

This understanding on the maintainability of patterns and data structures allowed me to implement more complex hybridized patterns and structures within my 'Custom Project' and its plans. In my project, I've included a robust and specialized entity component system targeted towards Turn-Based Strategy games. This system implements Command Pattern and ECS concepts to quickly manipulate only relevant components given a context.

Reflection

The most important things I leant:

Rapid prototyping and concise, repeatable documentation. I learnt how to quickly construct ideas and concepts within C++, test those ideas over a range of criteria and provide a concise document detailing my findings for future reference either by myself or team-members. I learnt about the importance of concise demonstration, in that spikes and their reports are not to be overengineered as the results of doing otherwise costs resources. Finally, I learnt how to extract programmatic concepts from the work of others and apply their abstractions within the sphere of games programming.

The things that helped me most were:

The Spike structure. Given a pointer in the right direction and enough resources to get started researching. I found the overall process of a Spike was great for broadening all domains of my programming knowledge. I found the unrestrictive nature of spikes allowed me to develop my own programming style and realise the flaws in my own solutions. I find these mistakes still haunt my dreams and that's good, it means I'm not making them again anytime soon.

I found the following topics particularly challenging:

Combining ideas from multiple previous Spikes into one project was something I hadn't handled before and proved incredibly challenging on my first approaches. My spikes were designed without modularity in mind, and I thought refactoring wasn't going to be too difficult right? Frustrations were experienced during the 'Command Pattern' Task, where Zorkish had to be functional enough to implement the command pattern idea involving the merging of Tasks 8 – 11. In this case the Spike itself proved trivial opposed to the implementation of the game-world.

Once this issue was resolved however, I found myself taking the lessons learned into the next Spike (Composite and Component Patterns) and having a substantially easier time. Given this, the "Composite and Component Patterns" Spike was mismanaged regarding time. I didn't account for underlying datatype floating point errors and limitations prompting multiple refactors of the ECS before completion. The combination of these tasks took two weeks to complete due to the challenges faced and proved a major set-back for the desired outcome of my custom project.

I found the following topics particularly interesting:

Task 13 – 'Composite & Component Patterns' had me implement an ECS into Zorkish and since that task I've become interested in building a larger, more modular ECS. Possibly a project for the future. As a symptom, I've found myself as interested in the creation of game engines and their architectures than that of games programming.

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

I feel like Visual Studio has me at the peak of the Dunning-Kruger Effect. My understanding of VS has increased dramatically since the start of the semester. This can be seen in IDE and Linker heavy Spikes such as Task 15 – 'SDL2 Concepts' and Task 24 – 'Profiling, Performance & Optimizations'. However, even having over 2 years of consistent VS experience, I find myself encountering many misunderstood bugs and errors when diving into a cloned project so I can't let my head get too big. Same concept applies to Entity Component Systems, I've researched a lot of ECS solutions within the last few months and become quite familiar with them. However, this has only reaffirmed my understanding of how much there is to know regarding these software patterns and their various implementations.

Examples of a simple ECS can be found in Task 13 – 'Composite & Component Patterns', where I implemented a small, modular ECS for Zorkish utilizing character indexing and unique component hash-tables. More robust example can be seen in my 'Custom Project' and 'Custom Project Plan'.

I still need to work on the following areas:

I need to stop using raw pointers everywhere… they're quick and easy and definitely the answer sometimes but I found them to be a nasty C++ specific habit I've picked up. More generally, I felt my priorities and planning were not up to scratch for the unit. I'd consistently draw plans which weren't thought through enough in order to get to programming faster. This would often cost me time fixing errors and the initial plan was void. Additionally, I found myself prioritizing Spikes too heavily and spending far to much time on them, neglecting my custom project.

I'd like to gain a greater understanding of the rendering processes, graphical subsystems (such as lighting and shader calculations) and deep optimization techniques used in engines.

Additionally, my understanding of repositories and project management within the programming sphere is lacking and something I need to experience more.

My progress in this unit was …:

Whilst I worked and pushed to my COS30031 repo whenever I could (at least twice a week), I found it difficult to get to Lab classes during the start of the semester thanks to distance and work commitments. I didn't find this to affect the quality of my work as the Spikes and Labs were well defined and clear. However, once this was resolved I found my productivity boost as the extra input from Clinton cleared mental-blocks and confusion faster.

I feel as though the quality of my custom and research project suffered as a result of other commitments. Though not incomplete I recognize that much more could have been achieved if I'd been allowed the time to commit to completing the pass tasks sooner. However, the pass tasks proved invaluable to my understanding of games programming and software architecture and if they we're the only things I wanted to achieve for this unit I'd be happy.

This unit will help me in the future:

As of the time of writing, I plan to move into the game's development field. Whether it be engine design / development or games programming and design it depends, but in either case the knowledge gained from this unit has given me a look into the foundations of games / engine programming, the data structures and patterns behind them which have been established and used since their earliest iterations. At any-rate, I expect to be developing a portfolio soon and this unit has given me great insight into the processes behind that.

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

Given my general understanding of games / engine programming has improved substantially since the start of the unit, I feel as though I would do better structuring the order of tasks I would accomplish during the latter half of the semester. I felt this to be a non-issue during the first portion of the unit however once I started moving into unfamiliar territory, I would often drift down rabbit-holes of programming wizardry and fail to see the forest for the trees. With my current schedule this proved detrimental to the outcome of my unit as it caused each task to take at least a day or two longer than the last and by the end of semester I found myself in a time crunch. Therefore, if I were to do this unit again, I would find my pass tasks more focussed and minimal.

I found that once in crunch however, I'd been equipped with the skillset to rapidly build and demonstrated whatever I needed too with minimal 'feature fat'.

Conclusion

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