COS30031 Games Programming: Time Boxed Demonstration Activity

Instructions

This activity is designed to allow you to demonstrate your understanding of the Unit Learning Outcomes (ULOs). Questions are of an open form, and there may not be a single correct answer. It is up to you to demonstrate and support what you know in your responses.

- Select the best **one** or **two** questions that you feel will allow you to demonstrate your knowledge.
- Do not answer all questions. (There's no room anyway!)
- A maximum of the first **three** answers will be marked. Extra answers will be ignored.
- You can cross out answers you do not want marked.

Clearly write your name and student ID on every answer page.
Clearly label the question you are answering.
Clearly label each figure and include captions. Figures should be on the blank side of answer pages.
Answer in any order you wish.
You must submit both this document and all your answer pages at the end of the test.

You have 1 hour to answer the questions.

Assessment

Date	Name	Student ID (Write CLEARLY!)
1 / 4 / 2019	Sam Huffer	1 0 1 6 3 3 1 7 7

Markers will provide feedback in the matrix below regarding the depth of understanding you have demonstrated in each of your answers. (Simple numeric marks are not used.)

ULO's	Q	Q	Q
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			
2. **Implementation**: Create games that utilise and demonstrate game engine component functionality, including the implementation of components that encapsulate specific low-level APIs			
3. **Performance**: Identify performance bottlenecks by using profiling techniques and tools, and applying optimisation strategies to improve performance			
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			
Overall Comments			

Assessment Key

N/R or W	*Not Relevant (does not answer the question) or Wrong (incorrect) details*
Shallow	*Simple (relevant) details but not very deep (terms, concepts, process)*
Good	*Medium level of details (good descriptions, lists, combined ideas)*
Deep	*Strong relational knowledge (compare, analyse, contrast, relate)*
Very Deep	*Reflection, extended knowledge, generalisation (extrapolation), theorisation (ie. "Wow!")*

Open Answer Questions

Select and answer from the following set of open answer questions.

1. **Explain** and **discuss** the relationship between game software and real-time systems. *Suggestion: list the qualities of a real-time system and how they relate to the steps and stages in a game loop. Note – that was only a suggestion. Diagrams are encouraged.*

2. Select **three** of the game features listed below and **identify** the data structure you would use to store data for each. Clearly **justify** your choices.
   a. Player inventory in a first person shooter (e.g. Doom, Duke Nukem 3D, Halo)
   b. Player inventory in a role-playing game (e.g. Skyrim, Diablo)
   c. The game map in a tile-based strategy game (e.g. Chess, Civilization V)
   d. The game world in a 3D action game or shooter (e.g. Halo, Arkham Asylum)
   e. The upgrade tree/tech tree in a strategy game (e.g. Civilization, Starcraft 2)
   f. The shopping/recipe item mechanic in MoBA (e.g. DotA 2, League of Legends)

3. **Single player** games and **multiplayer network** games have different game architecture requirements. **Explain**, using diagrams to help support your answer, the different requirements of each game type with respect to the follow game engine features:
   a. The main game loop
   b. Communication between game subsystems
   c. Storage of game state and game world data.

4. Game states are a common feature of computer games, however the term can apply to many different aspects of game architecture and programming. **Explain** and **discuss**.

5. Data Structures and patterns are often useful in games programming. **Discuss** with respect to the work you have developed for spike work in the unit so far.

6. Based in part on the spike work you have developed for the unit so far, **present** a list of game engine features appropriate for the game genre of "text-based adventure", and **justify** why each feature should be included in your list.

7. **Explain** and **discuss** the relationship between low-level libraries, game libraries, game frameworks and game engines, using specific **examples** to support your answer.

8. The concept of patterns is very useful in software development, and applies not only to software architecture but also usability patterns. **List** at least **three** usability patterns, and for each **define** the pattern including any relevant positive or negative aspects. Use figures, if appropriate, to support your answers.

Tip: Remember that this activity is an "open answer" style opportunity for you to demonstrate your knowledge of the unit learning outcomes, and so referring to them (page 1) may help you. Refer to the instructions. A quick plan (key points?) before you try to answer in detail is a good idea.

Name: Sam Huffer ID: 101633177 Page 1

2. Game Features → Data Structures, & Why.

For a player inventory in an RPG (eg. Skyrim), a list ~~or dictionary~~ of items would be appropriate, ~~the~~ RPGs feature a wide variety of game objects the player can collect, ~~too many for dedic~~ and usually can collect more items than dedicated slots for items of that ~~type~~ would be worth the trouble. Adding all items to a list of items instead (or a list of a particular type of item) ~~of a dictionary of items if the designers only want players to have a single item of a with a particular~~ would be much easier. A dictionary could also be used if there was only one copy of any given item. But ~~most~~ Skyrim, for example, allows players to collect any number of a given type of item (e.g. the iron sword), making a dictionary unsuitable in ~~that~~ such cases.

For a first person shooter, the data structure that underlies the player's inventory would depend on the design of the game. An MMORPG FPS like Destiny would want to use a list (or several for each item type) to be able to handle the wide array of items. A single player FPS like Halo which has a more limited number of game objects the player can possess ~~can~~ both in terms of what they can carry and what's available to begin with could also use a list, ~~with~~ but ~~cheats when~~ an argument could also be made for dedicated fields of the type(s) of object the player would be allowed to acquire. For example, Halo CE could get away with two fields of type "weapon", and two int fields counting how many grenades (UNSC and Covenant) the player has, the actual grenade objects being pooled until they're needed. Setting up the player's inventory this way would be less extensible, but would eliminate needing to search through lists for game objects when instead those fields could directly (or via public properties) be accessed themselves.

The game map for a tile based game would be

Name: Sam Huffer ID: 101633177 Page 2

Use this side for Figures. Clearly Label and Caption each Figure. Refer to Figures in your answers.

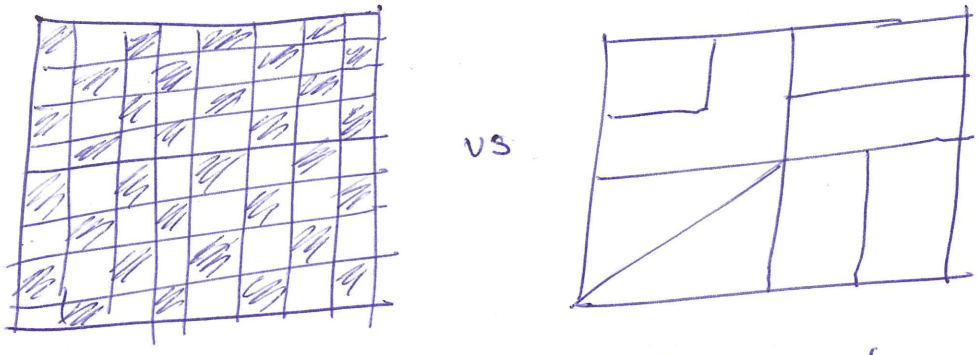

Fig 1: the tile grid of chess vs a hypothetical game with tiles of uneven sizes.

Brainstorming for

- stage -> dict(str) -> state
- input -> list(str)
- Cmds -> dict(cmd)
- mgrs -> singleton
- GO -> dict(ept) -> components
- Container(pt -> list(item)) -> composite
- Messages - dispatcher, observer pattern
- world - clean up commands

str -> list(str)

Dictionary(commands)

sta mngr
Cmd mngr
Str mngr
msg mngr
} singleton

input -> commands <-> acton game object } command pattern
output <- t list(str), world, player => messages

dispatcher
observer
subscribe/unsubscribe
messages
open container
unlock lock

Game Stages
(P state pattern
dict (Stages)

Game object
component pattern
dict of components

Container Component
List of items
} composite

would be best suited for a 2D array of tile objects - assuming tiles are arrangeable in a grid, like in Chess or my capstone project, Get the Fog Out! If tiles are of different sizes (and shapes, perhaps), then a grid representation in code might be unsuitable (fig. 1). Such a tile-based world would be better stored in a list of tiles, or a dictionary, where the arrangement on-screen would be irrelevant. Such tiles would likely have to keep track themselves of their positions, and how they connect to/relate to one another.

5. Data structures & patterns in spike work so far

For my spike work so far, I've had to use a number of design patterns, many of which are built in-code upon lists and dictionaries.

(vectors) Lists I've used for cases where I don't need to worry about elements being unique, such as for the player's inventory, or for the items in a container, or the player's input into the game. Lists of strings were also very useful for reading in game objects & components from text files.

(maps) Dictionaries I've tended to use more when I only want one of a particular thing to exist (e.g. commands, stages, and game objects (each has a unique id) components). If a particular key exists already, that tells me I shouldn't add another object of the corresponding type. It also allows for concise prevented access in-code

The different game scenes/stages (main menu, select world, gameplay, etc.) made use of the state pattern, plugging the appropriate stage into Game.cpp.

Game objects, I've gotten up to making them collections of components, using the component pattern. Container components storing items that may have container components also makes use of the composite pattern.

Several systems in Zorkish have had manager

Use this side for Figures. Clearly Label and Caption each Figure. Refer to Figures in your answers.

classes (e.g. stages, commands, messages, strings), each of which I used the singleton pattern for so that classes wouldn't require a field of that class to access them, but each could still access them, and everything could use the same instance.

Currently I am implementing the messaging system, the message manager I've put together is a dispatcher-style message manager, using the observer pattern where game objects subscribe to be able to accept messages. Once fully implemented, messages would make command parameters much cleaner.

Use this side for Figures. Clearly Label and Caption each Figure. Refer to Figures in your answers.

Name: _____ ID: _____ Page ____

Use this side for Figures. Clearly Label and Caption each Figure. Refer to Figures in your answers.