AI assessment KN 1

25.03.2022

Ryan Bassil
800406188

Contents

Task #1
Research and describe different programming terminologies and techniques

Question #1

List five basic syntax rules (coding grammar) of c#, and how does is it different to other programming languages (e.g., Python) that are commonly used in game development?

Syntax	C#	Java
Main Method	Can be parameter-less and can give default value to accessibility level if declared without.	Must have its parameters declared.
Constants	Uses the *const* keyword.	Uses the two words: *static final*.
Inheritance	Uses a colon to represent the class being inherited from. (*class A : B*)	Uses the word *extends* to represent the inheritance connection. (*class A extends B*)
Signed and Unsigned Primitive Integer Data Types	Must specify the version (signed or unsigned) variable being used, such as sbyte or byte, respectively.	It is not possible to declare an unsigned integer data type, though there are ways to utilise negative numbers.
Virtual Methods (Polymorphism & Overriding)	Must declare that a method is *virtual* otherwise it is not by default.	All methods are virtual and can be overridden by derived classes.

Question #2

What are the major architectural differences between Unity3D and another commonly used game engine such as Unreal Engine?

Unity3D has a "Project" window that you can drag and drop or create items within. These items are your "assets" and Unity does not categorise or separate them for you, you have full control over them.	With Unreal Engine there is a "Content Browser" that you drop or create items in, and they become your "content" for the project. Unreal then organises the items based on if they are 'Source' files with code or not. You still have the ability to

	control structure within this browser as in Unity.
Unity3D has tabs and windows to control how the user views their product. They have modes but is highly minimalistic sitting at roughly 4 modes such as play and edit. They have a "Scene" view to allow manipulation of game objects and other game assets within the scene.	Unreal Engine has a main view for manipulating game objects and it is called the "Viewport". Unreal has many modes for editing, scaling, painting, etc and is represented as a separate toolbar for easy access, whereas Unity only has one toolbar and is mostly for basic tools of movement and scaling objects. Unreal's other toolbar represents functions for running the engine.
Unity3D lends itself to artists and programmers to use by way of Blend Trees, Animator Controllers, as well as Game Objects and C# scripts. Though without code it would be impossible to make a functioning game in Unity3D, unless using an external tool/'plugin' that allowed one to do so.	Unreal Engine has the ability to be used by both programmers and non-programmers alike regardless of artistic ability by allowing the user to create functions and more using their system called "Blueprints". You could potentially make an entire game without coding anything using this unique system, all 'out of the box'.

Question #3

Find and explain five common programming principles and/or techniques that you can use to improve your code.

1. KISS – Keep It Simple, Stupid!
   In efforts to keep this explanation of KISS simple, I am writing only necessary information and non-complicated words to convey meaning. In programming, the less complex you can make your code, the easier it will be to modify, read, test, and maintain.
2. DRY – Don't Repeat Yourself
   When you find yourself writing the same code over and over again, write some code that does that code once and use it later, so that you can save space and make things more readable. Commonly, this could be an iteration of a list or array; so, you might code a method that takes parameters and executes that iteration in its body, then returns the values you need, rather than writing that iteration code repeatedly.
3. YAGNI – You Ain't Gonna Need It
   This principle is in the same vein as the KISS principle, it's all about avoiding complexity and time wasting. The idea is that you don't write code for

something that may or may not happen in the future, you write code to solve a problem. Don't write code to solve a problem that doesn't exist.

4. Documentation.
   This principle is the idea of making sure that no matter who reads your code, they will understand it without having to go to great lengths to decipher your code. Particularly useful with complicated concept implementation and abstract naming, documentation does a long way to reduce time in understanding the code. (I.E: comments)
5. SR – Single Responsibility (A principle in the SOLID principles acronym)
   Single Responsibility principle essentially means that any class written should have a single function/part in the functionality of the program. Everything within the class should relate narrowly to what the class's responsibility/function is.

Question #4

What is a software development life cycle (SDLC)? Describe the following SDLCs, and list their major pros and cons. Waterfall, Agile, Test Driven Development and DevOps.

Software Development Life Cycle is a process that is followed within software companies or organisations to develop high-quality software. It provides a well-structured plan or flow of stages of development that aid in organisation to produce high-quality, well-tested software that is ready for use.

SDLC	Outline	Pros	Cons
Waterfall model (Traditional methodology)	A model of development that endeavours to follow the "get it right the first time" philosophy, whereby the developer gathers as many details and requirements about the system needing to be developed as possible, and then follows through with developing the software until it meets all those requirements before being delivered back to the client.	• Thorough process of info gathering, development, and testing. • More personal and involved with the client initially through info gathering. • Cheap	• Time intensive • Does not allow for client's feedback through the project's development. • Not suitable for large or complex projects.
Agile Design	This methodology highly focuses on customer involvement in order to bring a solution to the problem needing	• High client involvement or engagement • Time efficient	• Expensive. Multiple iterations mean

	to be solved through rapid and multiple smaller iterations of a new model in order to reassure appropriate adaptation and satisfaction of customer requirements.	• Multiple opportunities to improve or fix the software. • Much higher bug catching success.	more work involved. • Time intensive. Each iteration increases the time to completion substantially.
Test Driven Development	Test Driven Development is a methodology that focuses on building and structuring the code around testing the code written. For every function or item of code written there is a test that is written after it. This methodology is similar to the Agile development but adds tests much more frequently through the cycle of the project and focuses on robustness of the code.	• Far less errors or bugs in the software than most other development methodologies. • More robust and extensible code. • Cheaper than lot of other methodologies.	• Harder to modify code due to specific tests being set up for each specific piece of code. • Code can get complex and difficult to read or understand due to the number and complexity of tests. • Implementation can suffer in favour of robust code.
DevOps	A methodology that is much larger than most other methodologies as its goal is to bring multiple 'departments' of a project into harmonisation and collaboration (where other methodologies have them separate and limited communication between them) to provide more rapid and cleaner codebase and software management. These departments are developers (who write the code) and the operators who deploy and manage the software.	• Cost efficient as there is more communication in a larger team, there is less need for major fixes or changes. • Time saving. A focus on automation allows for more time to be focused on development	• Organisation of the project is more difficult. • Security of the product is at greater risk due to the need of clear communication within a large/more involved team. • DevOps has a large emphasis on company and

	DevOps has a lot of emphasis on testing and delivery automation to streamline the development/production process of the project.	and other areas. • A more well-developed understanding of the project's progress and current standing.	development culture and this can take time to implement if is not already present as it is a very different methodology to all other SDLCs.

Task #2
Design and justify AI implementation

Question #1

Imagine you are in a AAA company creating a real-time strategy (RTS) game (E.g., StarCraft). In this game the player must build an army to protect a power generator. To train the army the player must gather resources from strategic locations.

Design an appropriate AI that the player will play against. Detail how the AI will react to different player actions and complete objectives in the gameplay scenario. Talk about how it would follow the principle that "an AI should never be trying to win, but to lose convincingly".

Detail the reasoning behind why you chose to design the AI this way and why you believe it is the best AI implementation for this game.

In this AAA company, the game being created is a sci-fi setting where the player can send military and strategic utility units (such as geographic engineers and scientists) to different planets around a singular solar system that they are placed in. The player's base (along with the sole power generator to save mankind) is located on a single planet within this solar system and the player's secondary goal is to 'capture' the entire solar system whilst their main goal is to prevent the last power generator from being destroyed.

AI thought design, from the start of the game:

- Start with bonus resources and combat strength.
- Evaluate player's military strength, send half of that strength at the start of the game to allow the player to experience combat and understand how the game functions.
- Gain resources of the AI's home planet at 1.25 times the amount of the player and focus on half base defences and half combat strength, unless the amount of in-game time exceeds a threshold, in which case create combat strength equal to 66% of the player's combat strength and assault their base.

- At the beginning of the game, the AI aim is to focus on building foundational protection for its base and sending a moderately challenging wave of enemies at the player at in-game time points or at player objective milestones.

AI thought design, from mid-game:

- After the player has built at least one more power generator and a couple days of in-game time has passed, the AI's aim is to be a little more focused on combat than its protective measures, to destroy the player's generator.
- If the player does not focus on combat power and instead hoards resources, the AI's aim is to instead focus on deploying mild enemy strength to most tactical resource areas. If these forces are cleared and the player does not create more militant forces within 2 in-game days, the next tactical resource point will have stronger enemy forces, and the next assault on the player will be 1.25x stronger than the last wave.
- If the player focuses on militant sources, the following enemy waves sent will be only 45% and 55% the strength of their combat strength to give them a sense of achievement and power. Following these waves, the AI will focus on protecting tactical resources needed for the creation of stronger military units, with moderate-hard difficulty strength, though the AI base will start lacking in progress and protective measures, being 25% weaker than that of the player's military, resource, and base buildings combined.
- If the player decides to utilise their military units to then assault the AI base, the AI will spawn one or two higher tier enemies among the existing allocated base-protection forces and focus on having 30% of the forces stand their ground within the base to fend from within the walls, and the rest (including upgraded units) to charge out to meet the player on the perimeter of the AI base. Likely, the player will win in this instance, but if not, the AI will destroy the player's military force at the scene but not advance on the player base until a randomly decided amount of days pass between 3-7 days and the player has at least 50% of the AI's military strength.

AI thought design, late-game:

- Following the advance on the player at 50% strength or a battle after the player has at least 3 or more power generators, if the AI loses the previous battle it will pause its military production and focus on base protection for 3 days. After this, it will send a wave of enemies to one or more tactical resource locations (preferably controlled by the player) at 75% of their military strength. If the AI wins this battle, it will then focus on military units and send waves semi-periodically at the player until the player manages to destroy the AI base or (if there is a timer) the timer runs out, at incremental strengths to a maximum threshold and utilising different strategies.
- If the AI wins the previous assault on the player with at least 3 or more generators, it will destroy 1 – 2 of their generators and 90% of the surrounding buildings unless its military strength is below 25% or achieved this objective. Following this it will retreat to focus 50-50 efforts on base protection and military forces for the next randomly generated number of in-game days between 3 and 5. It will then make periodic assaults on the player until the player manages to destroy the AI base.

Designing the AI to mostly be on the offensive will keep the players invested in learning different strategies that can counter the AI's strategies, and it will remind them of the goal of the game, which is essentially domination through military rule. While the AI will be on the offensive most of the time, it won't be strong enough for the most part to simply wipe the player out unless they make poor choices consistently, not until late game where the AI will actively try to focus on building military strength that has enough power with its strategies to wipe the player out if they don't play carefully. Using this AI design it follows the idea that the player thinks the AI is trying to win but ultimately behind the scenes it is just losing convincingly, giving the player the satisfaction of victory.

I believe that using mathematical approaches to the AI's responses with the player's resources and military units keeps the flow of the AI structure simpler than a subjective conceptual take on the player's movements and actions. It makes for easier repetition, coding, and upscaling.

Task #3

Create a simple AI MVP (Minimum Viable Product)

Question #1

Using a State machine, create a simple Artificial Intelligence (AI) with at least three distinct states. The AI must be aggressive in one state, and defensive in another, choosing the state depending on the situation. The third state can of any design you choose, but it must be distinct. Display the state the AI is in through the UI of your game. Project is in 2D, and is not related to task 2.
https://github.com/LunarBlacksmith/AI-Assignment---State-Machine

Task #4

Submission

Question #1

Submit your project on Moodle. Project's code needs clean and commented code with regions.

Documents should include documents in PDF Format. (In word, File -> Export -> PDF)
Remember to include your unity project files (APP) / Source code, a project build, and a GitHub link.