Assessment event 1 of 3: Knowledge

Criteria

Unit code and name

ICTPRG430 | Apply introductory object-oriented language skills

ICTPRG434 | Automate processes

ICTPRG439 | Use pre-existing components

ICTGAM423 | Apply artificial intelligence in game development

Qualification/Course code and name

ICT40120 CERT IV in Information Technology Game Development

Student details

Student name

Gavin Lampe

Student number

880644379

Version: 20230415

Date created: 15 April 2023

RTO Provider Number 90003 | CRICOS Provider Code: 00591E

This assessment can be found in the TAFE NSW Learning Bank.

Assessment instructions

Table 1 Assessment instructions

Assessment details	Instructions
Assessment event overview	The aim of this assessment event is to assess your knowledge and performance in introductory programming tasks using an object-oriented programming language including tool usage, documentation, debugging, and testing techniques. Write scripts to automate solutions by using basic scripting processes and application-specific scripting options. Identify, evaluate and incorporate pre-existing (re-use) components from a library, or other source, as part of a software project. Research, develop and implement artificial intelligence (AI) solutions in games. This assessment is in 3 parts: • Part 1: Programming Basics • Part 2: Reusing Components • Part 3: Artificial Intelligence And is supported by: • Assessment feedback **Note**: This assessment may contain links to external resources. Access to the long URL is provided via the Error: Reference source not found section located at the end of this document.
Unit assessment guide	Refer to the unit assessment guide (UAG) before attempting this assessment event. The UAG contains information including assessment requirements and how to achieve a satisfactory result.
Submission instructions	When you complete this assessment, submit it for marking: • keep a copy of all the electronic and hardcopy assessments you submit to TAFE NSW • make sure you have completed the assessment declaration before you submit.

Part 1: Programming Basics

1. List at least 5 different C# Syntax language rules.

C# Syntax
A line of code must end with a semicolon
An if statement's condition is placed inside parentheses '( )', and its contents placed inside curly braces '{ }' directly after the parentheses
The names of classes used pascal case (PascalCase), but public variables use camel case (camelCase) and private variables are the same but start with underscore '_' (_camelCase)
A condition of a switch statement starts with "case: _" where _ is the value being tested and each possible outcome is ended by using "break;"
A method must declare whether it is returning a value. If it is it must state the returning data type before the name of the method. If it is not the word 'void' must be used instead to declare nothing will be returned.

2. Which of the following are built-in C# data types, indicate **True** or **False**.

Data type	True or False
bool	True
bit	False
float	True
int	True
log	False

3. Read the statement/s carefully and indicate **True** or **False**.

Table 2 True or false

Statement	True or False
A struct in C# is a simple data type that can hold multiple values of different types together.	True

4. Read the following code and answer, what does the static keyword change about the numberOfWheels variable?

```
public class Car
{
    public static int numberOfWheels = 4;
}
```

Static means the variable is attached to the class rather than an object. This means that there can only be one of this variable so any object accessing or changing it will change it for every object, and even if you destroy all objects using this variable, it will remain because it is attached to the class *not* the object.

5. Match the Description with the keyword that c# uses.

Keywords	Answer	Description
while	B	A. Selection
for	B	B. Iteration
if	A	
foreach	B	
switch	A	
else	A	

6. Read the statement/s carefully and indicate **True** or **False**.

Table 3 True or false

Statement	True or False
C# code is run line by line, starting from the bottom and ending at the top.	False

7. Explain the difference between inheritance and polymorphism in C#

Inheritance, in real life, is when you inherit or what your parents own. This is also true in programming. When you create a class that inherits from another class, all the properties of the parent class are passed on to the subclass.

Polymorphism is the choosing the correct code to run depending on the situation. For example, if you have multiple animals that inherit from the class Animals, you may code multiple move options that change depending on which animal. For example, dog.AnimalMove() would run code to walk, and fish.AnimalMove() would run code to swim even though they are both using AnimalMove()

Part 2: Reusing Components

1. Read the statement/s carefully and indicate **True** or **False**.

Statement	True or False
Reusing components and libraries increases costs.	False

2. Does reusing components and libraries make writing your game easier, and why?

Yes because if we were not using components and libraries then we could have to program every little game function our selves from the physics engine to how the UI is displayed on screen to changing the colour of text. Instead of just writing one line of code we would have to program entire methods to handle what we now consider easy tasks.

3. What are the major limitations and costs of Unity's license?

The major limitations of Unity's licence are that it is free on a personal licence until you hit $100,000 per year in revenue for the last 12 months, and unless you pay for a Unity Plus or Unity Pro licence, you must have a splash screen when the game starts that shows the Unity logo

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

The first major difference is that Unity uses C# and Unreal uses C++ which is a more complex programming language. A second difference Unreal also has Blueprint which is a way of coding essentially using flowcharts, and while Unity also has Visual Scripting and has now integrated Bolt which is similar to Blueprint, it is not as advanced.
Another difference is the way things are setup in each game engine. In Unity everything is a GameObject or a component attached to a GameObject, whereas in Unity, they have pawns, actors, and components that are attached to actors. Actors in Unreal are similar

to GameObjects in Unity, but instead of a singular GameObject you can assign as as anything, pawns a type of actor that are specifically player and AI controlled. Unreal also has specific in-built classes for GameMode, GameState, and PlayerState among others where Unity does not; this makes Unreal easy to follow in terms of designing and planning the code of your game, but some developers find it more restrictive.

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

Programming principles	Improvement
Single Responsibility Principle	This principle states that a class should have just one responsibility so that only one type of change in the software will affect the class. It also makes it easier to collaborate because team members will not be working on the same script (generally), and it makes marge conflicts less likely to happen.
Open-Closed Principle	This principle states that a class should be open to extension but closed to modification. This means that we should be able to add new functionality to a class without needing to modify the existing code
Liskov Substitution Principle	This principle states that a subclass or child class should be able to pass as its superclass or parent class. So if a method is expecting an object that is the superclass, we should also be able to send it the subclass without getting any errors. Because a big part of OOP is inheritance and polymorphism, the LSP allows you to use polymorphism reliably without creating bugs.
Interface Segregation Principle	This principle states that interfaces should be separated into component interfaces so that the interfaces can be extended or have functions removed without having to add workarounds or extensively modify existing code.
Dependency Inversion Principle	This principle states that our classes should not depend on real classes and functions, but instead should depend on interfaces or abstract classes. This goes hand in hand with

	the Open-Closed Principle in that to extend a class without modifying it we can depend on an interface instead of the actual class itself. This makes coding in a team easier because it means less modification of existing code which means less chance of introducing bugs.

6. Match the following standards with their description.

Standards	Answer	Description
Ethical AI Principles	C	A. Responsible for classifying films, television programs, and video games that are released in Australia. Considers the ethical implications of game content when making classification decisions.
Australian Classification Board	A	B. Game developers should also consider platform-specific standards when using AI in their games. For example, Apple requires that apps that use AI should provide clear disclosures, have a privacy policy, and obtain user consent.
IGDA Developer Code of Ethics	D	C. Developers should consider the ethical implications of AI use in gaming and ensure that AI is used in a way that is consistent with ethical principles, such as fairness, transparency, and accountability.
Platform-specific standards	B	D. Sets out principles and standards for ethical game development, covering topics such as intellectual property, diversity and inclusion, and player safety and well-being.

7. In relation to programming, describe what an algorithm is, and how do you use algorithms when making games?

An algorithm is any series of instructions that perform a function or solve a problem. For example, a recipe to bake a cake is an algorithm and so is the code that makes your mouse move. A game is nothing more than a large number of algorithms working together to make your game work. So you cannot make a game or any software without using algorithms.

Part 3: Artificial Intelligence

1. List at least 4 path-finding algorithms

Path-finding algorithms
A*
IDA*
Greedy
Dijkstra

2. How do path-finding algorithms such a A* limit games created with them

1. The A* algorithm creates a list of every point that has been checked by not expanded on to decide which point to check next when finding the best path. Because of this, if there are many possible paths and the area is very large, it may limit the path size.

2. A* uses approximations to calculate the different available paths. This means it might not always find the shortest route. For example, shorter route with a lot of turns might be found to be longer than a longer one with less turns.

3. Pathfinding algorithms need a static environment where the path does not change. In a dynamic environment the pathfinding algorithm will need to keep being updated every time the environment changes. This can be a waste of resources if done often.

3. List and describe 5 major AI terms used within the games industry.

AI terminology	Description
NPC	Non-Player Character – The name given to agents that are controlled by AI and not the player.
Pathfinding	Finding a path between two points in a complex environment.
Flocking	A method of simulating the movement of a group of animals
Behaviour Tree	A flowchart like tree that models the behaviour of NPC
Machine Learning	Any AI algorithm that can be used to make NPCs learn from their environment and improve over time.

4. 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.

   Describe the design of an appropriate AI that the player will play against. Detail how the AI will react to different player actions and complete objectives in the game-play scenario.

For a game like StarCraft, the enemy is essentially trying to take the same actions as the player but in a more aggressive manner. Therefore the AI would have the same three basic objectives – gather resources, build a base and army, and combat

To do this we would need to create multiple scripts to handle the different AI components. For general AI game-play that covers all three objectives we would need to write scripts for:

- EnemyMovement, which would use a pathfinding algorithm to find the shortest path to the current goal and would use flocking to get all the individual enemy agents to move together in their respective units.
- EnemyDetection, which would be used in all objectives to allow the AI units to detect nearby player units that are within line-of-sight or hearing distance if that's a thing, and then relay that information to the EnemyStrategy script outlined below to get a course of action.

* Instead of writing the same thing over and over again, unless using pathfinding, all scripts I mention will use state machines and possibly behaviour trees for their decision making. The reasons for which will be outlined in the answer to the next question.

For the gathering of resources, if the player is gathering resources in a location close to AI agents, the AI would attack the player. The AI would also need to identify the best locations to gather resources without getting attacked by the player. This could be done using pathfinding to find the resources that take the least time to gather and prioritising the type of resources based on what is currently being built.

Some of the scripts we would need to write for resource gathering would be:

- ResourceCollection would use a pathfinding algorithm to find the resources and then the collection and carrying back to base of those resources.
- ResourceManagement would keep track of the resources, how they are spent, and decide which resources are currently most important for ResourceCollection to go for.

For base and army building, the AI would need to prioritise building of structures or building army units depending on the resources available and state of the game as well as in response to player actions. Which it prioritises would need to be play-tested because different armies have different strengths. Defensive measures might be prioritising creating weak units that are fast to make when being attacked or maybe building defensive structures like walls after a first attack from the player.

- EnemyProduction would be a script that decides what to build next. Should it build a building or an army unit, and which one. Using the known state of the player, the resources available, etc. to decide what can be built and then what should be built next.
- TownPlanning would be a script (maybe not by that name) that decides where to put the buildings, using things like how much area the building needs, does it have defences, and how important is it strategically to winning the game.

For combat, the AI would need to be more aggressive as the player cannot lose unless their power generator is destroyed. Therefore the AI would need to prioritise attacking over defence. For example, it would try building up a fast small army to attack quickly while the player is still focused on building and resource gathering. Using known information about the player's army and base it could prioritise sending different units. It would also need to strategically choose which buildings to attack first to increase its chances of reaching the power generator. It would also need to decide when to retreat if too many units or the overall health of units is low.

- EnemyDefence would be a script that handles the behaviour when the player is attacking. Agents would use methods like ray-casting for line-of-sight to determine which units can be seen and attacked. It would also decide which units should attack which of the player's units.
- EnemyAttack would use pathfinding algorithms to find the shortest paths to the base and to the most important structures to attack
- EnemyStrategy would decide what the best strategy is at the current time using all the information from the other scripts and analysing the player's buildings, base layout, and units. It would decide whether to attack or defend and which buildings or units to attack or defend.

5. 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.

There are two main reasons why I chose to use pathfinding, state machines, and behaviour trees for my AI.
The first reason is that I believe these are the minimum necessary components to make an effective AI for this type of game. For this, pathfinding is simply necessary for navigating a world with obstacles and objects that move and change, and state machines while good for responding to simple changes like health being low lack the complexity to respond in a dynamic and engaging way that the player won't learn quickly.

The second is that while it is time-consuming to create complex behaviour trees, it is at least within my possible skill-set whereas other AI techniques are way beyond my knowledge.

6. How do the strategies you have detailed effect the budget and timeline. Is the design technically feasible and suitable for players.

While I think the design is technically feasible and suitable for players, whether it can create a complex enough AI to engage them is another matter. The only choice I have made that would or at least *should* affect the timeline (and therefore budget) is the need to create complex behaviour trees. If the project planning has been done correctly for an RTS game, it would be expected that an AI would have to be written and that all the

different elements of the game would need to be included in the AI decision making. So pathfinding, flocking, and state-machines should already have been taken into account. If not, I don't think these would add to much to the timeline since they are pretty standard in game AI and don't take much to implement.

Student assessment declaration

This assessment is my original work and has not been:

- copied from any source without proper referencing
- written for me by any other person except where such collaboration has been approved by a teacher or assessor.

Student signature and date

Gavin Lampe - 30/06/2023

Reasonable adjustment

☐ Reasonable adjustment was in place for this assessment event.

If so, please provide details of any reasonable adjustment strategies that were implemented:

[Insert reasonable adjustment strategies]

Assessment outcome

☐ Satisfactory ☐ Unsatisfactory

Comments

[Insert comments]

Assessor name, signature and date

Student acknowledgement of assessment outcome

[Would you like to make any comments about this assessment?]

Student name, signature and date