COMPUTING ARTEFACT — AI

Version 5.0
Computing
COMP250

Dr Ed Powley

Introduction

In this assignment, you are required to **design** and **implement** a game component implementing one or more artificial intelligence (AI) techniques. The definition of "component" here is intentionally loose — it could be a subsystem within a game, or a standalone tech demo, or even a tool (such as an editor plugin) to help game designers to author AI behaviours or systems.

In the interests of managing workload, you may consider integrating your component into your GAM240 group game development project. However if this is not appropriate you may integrate it into another of your current or previous development projects, or into a standalone demo application. You may use any programming language and game engine you deem appropriate, including visual scripting languages such as Blueprints; this choice will probably be dictated by your choice of game.

*"To be enjoyable, an AI must put up a good fight but lose more often than win. It must make the player feel clever, sly, cunning, and powerful. It must make the player jump from his seat shouting, 'Take that, you little s**t!' "*

— Mat Buckland

Almost all types of games use AI in some capacity, and many genres rely on advanced AI techniques. Control of non-player characters is an important application of AI, and a wide-ranging one: enemy AI in a realistic stealth game is very different from in an arcade shooter, which in turn differs from a racing game. Other applications of AI can include adversaries in board, card or strategy games, procedural content generators, procedural narrative engines, assistive technologies, AI "directors", and many others. Your final product will be a portfolio piece, which you can use in future to demonstrate your mastery of these techniques.

The "Contracts" section at the end of this document gives some ideas for the type of component you might consider making for this assignment. This is not an exhaustive list, but is indicative of the scope and level of sophistication you should be aiming for.

This assignment is formed of several parts:

(A) **Write** a 2-page handout that will:
  (i) **outline** the concept of your component;
  (ii) **identify** the game into which your component will be integrated, and how it will fit into the overall game concept;
  (iii) **describe** the key requirements of your component;
  (iv) **state** the AI technique(s) that the component will demonstrate;
  (v) **justify** that the proposed component is feasible in scope.

(B) **Implement** your component.

(C) **Record** a 2–5 minute commentated video demonstration of your component that will:
  (i) **demonstrate** the component in action;
  (ii) **describe** the architecture of the component, with reference to source code as appropriate;
  (iii) **justify** the choices made in architecting the component.

Not all video games portray AI in a positive light. However, many games rely on AI to create a satisfying gameplay experience.

Assignment Setup

This assignment is a **programming** task. There is no template version control repository for this assignment; you should work within the existing repository for your game, or create a new repository on the Games Academy Git server as appropriate. Either way, remember to modify the .gitignore file (or equivalent on other version control systems) to exclude temporary build files from the repository.

Part A

Part A consists of a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. Answer the following questions to pass:

- What is the title and high concept of the game or demo into which your component will be integrated?
- What functionality will your component include?
- How does your component fit into the overall concept of the game or demo?
- What are the key requirements?
- Is the scope appropriate for the product development time-frame?

To complete Part A, prepare the handout using any word processing tool. Your handout may include images and/or links to online videos.

Show the handout to the **tutor** in the timetabled proposal review session in week 2 for immediate **informal feedback**.

Part B

Part B is a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Submission is timely;
(b) Enough work is available to conduct a meaningful review;
(c) A broadly appropriate review of a peer's work is submitted.

To complete Part B, record a 2–5 minute draft video demonstrating your component — a draft of your submission for Part C below. Upload your video to Microsoft Stream, and submit a link to the peer review section on LearningSpace prior to the timetabled peer review session.

You will receive immediate **informal feedback** from your **peers**.

Part C

Part C is a **single summative submission**. This work is **individual** and will be assessed on a **criterion-referenced** basis. Please refer to the marking rubric at the end of this document for further detail.

To complete Part C, record a 2–5 minute video demonstrating your component. The video should show the component in action, as well as some illustration of the underlying architecture. Optionally, the video may include a voiceover explaining the concept of the component, commentating the demonstration, and describing the architecture. Advanced video editing is not required — a screen-captured video recorded using OBS or similar software is fine.

Upload your video to Microsoft Stream, and submit a link to the submission area on LearningSpace. You will receive **formal feedback** three weeks after the final submission deadline.

Additional Guidance

As always, avoid underestimating the effort required to implement even simple software; always consider scope. From the proposal stage, you should consider very carefully what is feasible.

Your code will be assessed on **functional coherence**: how well the finished product corresponds to the user stories, and whether it has any obvious bugs. Correspondence to user stories runs both ways: implementing features that were not present in the design ("feature creep") is just as bad as neglecting to implement features.

Your code will also be assessed on **sophistication**. To succeed on a project of this size and complexity, you will need to make use of appropriate algorithms, data structures, libraries, and object oriented programming concepts. Appropriateness to the task at hand is key: you will **not** receive credit for complexity where something simpler would have sufficed. Likewise, if you are using an engine such as Unity or Unreal, you should make use of (and build upon) the AI functionality included therein; you will **not** receive extra credit for "rolling your own" without good reason to do so.

Use of **third-party code**, e.g. from online tutorials or asset packages, is permitted as long as you clearly identify which code is not your own, and clearly identify where it is from. Failure to give proper credit, whether deliberate or not, will result in investigation under the university's **Academic Misconduct** procedure.

Also bear in mind that you will **not** receive credit for any code that is not your own — it is important to make sure you change or build upon any third-party code in a substantial way, and are able to articulate how you did this. For example, simply following a tutorial to implement a particular AI system will not result in a high mark; however, using a tutorial as a starting point and then making significant contributions of your own will potentially yield much better results.

Maintainability is important in all programming projects, but doubly so when working in a team. Use **comments** liberally to improve code comprehension, and carefully choose the **names** for your files, classes, functions and variables. Use a well-established commenting convention for **high-level documentation**. The open-source tool Doxygen supports several such conventions. Also ensure that all code corresponds to a sensible and consistent **formatting style**: indentation, whitespace, placement of curly braces, etc. Hard-coded **literals** (numbers and strings) within the source should be avoided, with values instead defined as constants together in a single place. Where appropriate, values should be exposed as properties or variables in the Unity or Unreal editor so that they can easily be "**tinkered**" without changing the source code.

Maintainability is also important when using **visual scripting** systems such as **Blueprints**. Pay special attention to the **layout** of your Blueprints, which should be tidy and should make clear the flow of control and data. Use **grouping**, **macros**, **functions**, **routing nodes** etc. to achieve this. Blueprints which resemble bowls of spaghetti will not achieve high marks!

FAQ

- **What is the deadline for this assignment?**
  Falmouth University policy states that deadlines must only be specified on the MyFalmouth system.
- **What should I do to seek help?**
  Contact the module leader, via email or Microsoft Teams message. Contact details can be found on LearningSpace.
- **Is this a mistake?**
  If you have discovered an issue with the brief itself, the source files are

available at:

https://github.com/Falmouth-Games-Academy/bsc-assignment-briefs
.
Please make a pull request and comment accordingly.

Marking Rubric

Criterion	Weight	Near Pass	Adequate	Competent	Very Good	Excellent	Outstanding
Basic Competency Threshold	30%	At least one part is missing or is unsatisfactory.	Submission of all parts is timely and satisfactory. Enough work is available to hold a meaningful discussion. Clear evidence of programming knowledge and communication skills. No breaches of academic integrity.				
Functional coherence	20%	Few requirements have been implemented and/or the code fails to compile or run. Many obvious and serious bugs are detected.	Some requirements have been implemented. Some obvious bugs are detected.	Many requirements have been implemented. There is some evidence of feature creep. Few obvious bugs are detected.	Almost all requirements have been implemented. There is little evidence of feature creep. Some minor bugs are detected.	All requirements have been implemented. There is almost no evidence of feature creep. Some bugs, purely cosmetic and/or superficial in nature, are detected.	All requirements have been implemented. There is no evidence of feature creep. Few to no bugs are detected.
Sophistication of architecture	20%	Little insight into the appropriate use of programming constructs and data structures is evident. The program structure is poor or non-existant.	Some insight into the appropriate use of programming constructs and data structures is evident. The program structure is adequate.	Much insight into the appropriate use of programming constructs and data structures is evident. The program structure is appropriate.	Considerable insight into the appropriate use of programming constructs and data structures is evident. The program structure is effective. There is high cohesion and low coupling.	Significant insight into the appropriate use of programming constructs and data structures is evident. The program structure is very effective. There is high cohesion and low coupling.	Extensive insight into the appropriate use of programming constructs and data structures is evident. The program structure is extremely effective. There is very high cohesion and very low coupling.
Sophistication of AI techniques	30%	Lack of basic understanding of AI techniques is evident. AI techniques are unsuited to the task, and/or justification is absent or incoherent. Application of AI is trivial.	Understanding of basic AI techniques is demonstrated. AI techniques are suited to the task, with an attempt at justification. Application of AI is basic but effective.	Working knowledge of basic AI techniques is demonstrated. AI techniques are suited to the task, with adequate justification. Application of AI is somewhat sophisticated.	Working knowledge of standard AI techniques is demonstrated. AI techniques are suited to the task, with good justification. Application of AI is reasonably sophisticated.	Extensive knowledge of advanced AI techniques is demonstrated. AI techniques are highly suited to the task, with strong justification. Application of AI is sophisticated and complex.	Mastery of cutting-edge AI techniques is demonstrated. AI techniques are highly suited to the task, with very strong justification. Application of AI is highly sophisticated, complex, and state-of-the-art.

Appendix: Contracts

Choose **one** of the following contracts to implement in your component:

- Non-player character behaviour — note that this must include something more than the basic pathfinding and perception behaviours included in your chosen game engine
- Pathfinding with advanced movement, e.g. physics-based vehicle steering
- An authored AI behaviour system based on something other than finite state machines or behaviour trees, e.g. planning or utility-based AI
- A solver for a puzzle game
- An AI director or adaptive difficulty system
- A procedural content generation (PCG) system — either as an in-game system or as an editor tool
- A tool to allow designers to specify AI behaviours in editor
- Another AI component of your choice, subject to discussion with your tutor and demonstration that you have researched the feasibility of your chosen techniques