Assessment Guidelines

Games Development Pathway Assessment Guidelines

These guidelines must be followed when submitting assignments for the Game Development pathway. They ensure that your work is accessible for marking and reflects industry-standard practices such as version control, clear documentation, and collaborative development workflows.

These expectations are introduced during **launch week** and reinforced during **task briefings**. If you are unsure about any part of the submission process, contact the Course Lead Leader.

Submission Overview

All students **must submit a single file** as their Development Commentary to myUCA. You can choose one of the following formats:

Accepted Formats:

Format	**Description**
`.html`	A self-contained HTML file with embedded images, videos, and content.
`link.md`	A file containing a link to a publicly hosted Git repository with a `README.md`

Do Not Submit:

- `.doc`, `.docx`, `.odf`, `.pages`, or similar word processing files.
- Do not incorrectly name the file such as `123456_StudentName_UnitName.md.html`.
- Multiple scattered files or folders.
- Private repositories or restricted links.
- Builds hosted on cloud storage platforms such as **OneDrive**, **Dropbox**, or **Google Drive**.

Builds

- All game builds **must be publicly accessible** via an appropriate platform.
- Acceptable platforms include but not limited to:
  - **itch.io**
  - **GitHub Releases**
  - **Google Play** (for mobile builds, if applicable)
- Do **not** use cloud storage services like OneDrive, Dropbox, Google Drive, or iCloud to host builds. These often require login permissions or expire.

If Using GitHub (or any Git hosting):

- Your repository **must be public** — we cannot request access or open private links.
- Make sure your repository includes:
  - Working links to **builds**, **project files**, and **video demonstrations**

    - Working links to **builds**, **project files**, and **video demonstrations**.
    - A structured `README.md` that documents each task.

- Ensure **all assets and builds** are either included in the repository or clearly linked externally via an approved platform.
- Test your links before submitting — **no broken or permission-locked links**.

Development Commentary

What Is A Development Commentary?

Each unit on the Game Development pathway requires you to keep a Development Commentary, submitted as a single document. It is used to:

- Document your development process
- Reference required components (e.g. files, builds, videos)

Each unit's tasks must be documented in the same file, structured clearly for assessment.

How Do I Make A Development Commentary?

You must submit either:

- A self-contained `.html` file
- A public Git repository with a `README.md`

DO NOT submit `.doc`, `.docx`, `.odf`, `.pdf` or `.pages`. These can cause formatting issues across devices.

Markdown (Recommended)

Markdown is a lightweight markup language used for writing documentation, readme files, and developer messages (e.g., on GitHub, Discord, Slack).

We recommend VSCode for Markdown editing due to strong syntax highlighting and project support.

Google Docs

You may use Google Docs if you install:

- Code Blocks plugin
- Zotero extension

Docs can be exported to PDF or HTML, but use caution embedding media.

How Do I Manage References?

Zotero

Zotero is a free citation manager supporting Harvard referencing, with UCA styles available. It integrates with Google Docs and lets you generate citations by right-clicking a source.

Figures

Figures refer to **any non-body content** used to support your commentary. This includes:

- Images and screenshots
- Diagrams and flowcharts
- Code snippets
- Embedded videos and GIFs
- Graphs and data tables
- Blueprint visual scripting (via screenshot or embed)

Figures are used to visually communicate your work, support your explanations, or provide supplementary evidence. Figures are a powerful way to visually support your commentary without affecting your word count. They can be used to illustrate:

- External references (e.g., links to YouTube tutorials or dev talks)
- User testing data (tables, graphs, charts)
- Blueprint layouts and visual logic
- Code snippets or technical breakdowns
- UI mockups, scene layouts, or iteration history

> **Figures do not count toward your word count**, and there is no limit to how many figures you may use, as long as they are relevant and clearly labelled.

You are encouraged to use figures to explain specific things that would otherwise consume space in your word count. For example:

- A breakdown of survey responses as a table
- A screenshot walkthrough of a bug or fix
- A linked or embedded video explaining a setup or tutorial you followed

Figure Formatting Rules

To ensure clarity and accessibility, all figures must follow these formatting rules:

1. **Figures must be separated from body text.** Do **not** place your caption or label on the same line as the figure. It must appear **directly underneath** the figure on its own line.

2. **Each figure must be numbered and clearly labelled.** Use the format: `*Figure X: Description of figure contents*` You may italicise the description for clarity, but this is optional.

3. **Figure numbers must be sequential and consistent.** Label figures in the order they appear (e.g., Figure 1, Figure 2, Figure 3). Refer to them in your body text using this label (e.g., "as shown in Figure 3").

4. **Code snippets are considered figures.** Use proper formatting (e.g., triple backticks in Markdown or Code Blocks in Google Docs), and follow the same captioning and numbering system.

Figure Example

Correct usage of a figure in Markdown or HTML:

```
![Example](https://beforesandafters.com/wp-content/uploads/2021/05/Welcome-to-Unreal-Engine-5-Early-Access-11-16-screenshot.png)

*Figure 3: Unreal packaging menu interface.*
```

> Note how the image and its caption are **on separate lines**. The figure is clearly numbered and described, and could be referenced in the main text.

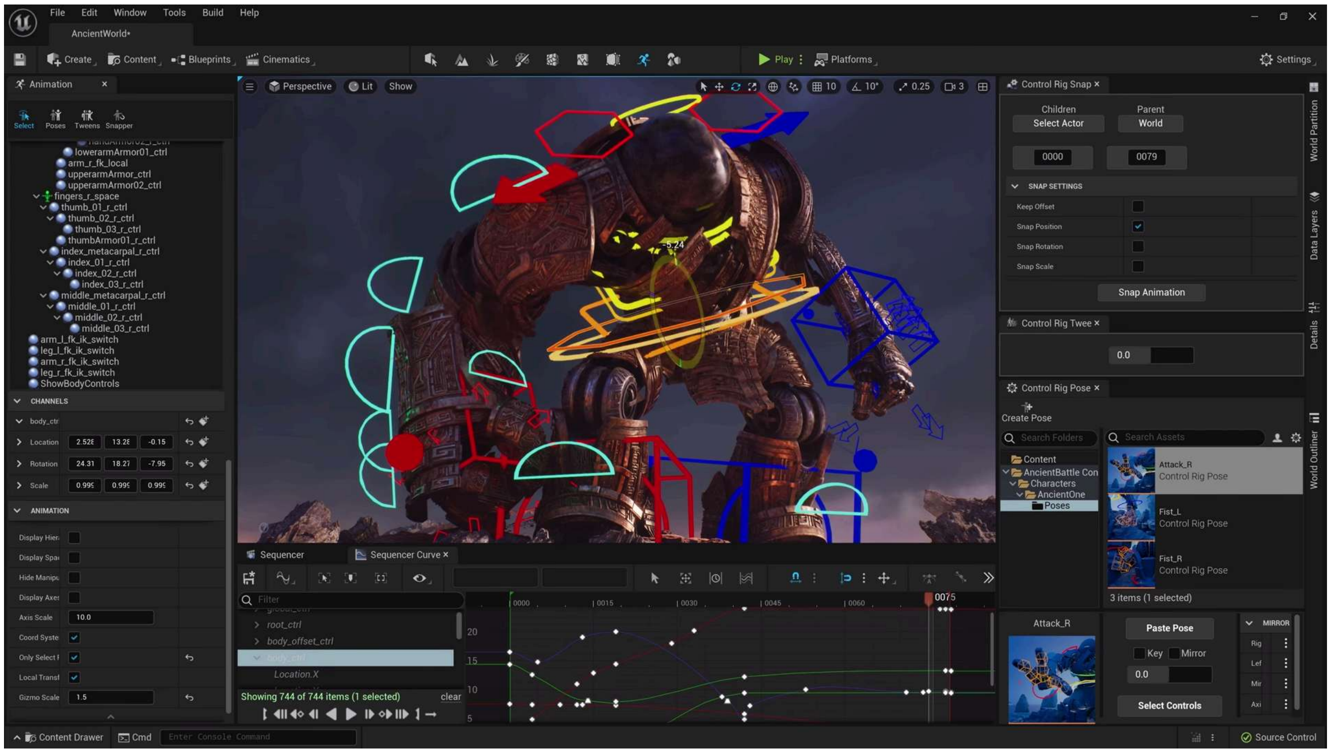

Figure 3: Unreal packaging menu interface.

Figure Misuse Examples (Do Not Do These)

Placing the label on the same line:

```
![Example](url) *Figure 3: This is incorrect formatting*
```

Forgetting to label the figure:

```
![Example](url)
```

Using a figure without referencing or explaining it:

```
![Chart](url)

(No explanation in surrounding text)
```

```
(No explanation in surrounding text)
```

Code

Include code snippets in your commentary. **Do not use screenshots of code.** Screenshots reduce legibility and prevent testing.

Code Snippet Example:

```
using UnityEngine;

public class HelloWorld : MonoBehaviour
{
    public void Start()
    {
        Debug.Log("Hello World!");
    }
}
```

Blueprints

Use BlueprintUE and embed your logic:

```
<iframe src="https://blueprintue.com/render/pboamojr/" style="width:100%;
height:560px; border:none;" allowfullscreen></iframe>
```

Word Counts

Each Development Commentary may have a target or required word count set in the assignment brief. To ensure fairness, you are allowed to go **±10%** around the stated word count without penalty.

For example, if the word count is 2,000 words, acceptable submissions range from **1,800 to 2,200 words**.

To ensure consistency, **only the core written content** will contribute to this count.

The following **DO NOT COUNT** toward the word count:

- **Figure captions or descriptions**
- **Headings and subheadings**
- **Embedded code snippets**
- **Embedded Blueprints**
- **Tables and Diagrams Containing Text**
- **Bibliographies or reference lists**
- **Asset declarations**

Bibliographies, Citations & Declared Assets

Inline Citations & Bibliography

All research and references must be properly cited using **UCA's Harvard referencing style**. This applies to both:

- **Inline citations** (within your commentary)
- **Final bibliography/reference list** (at the end of your document)

You can use tools like Zotero to manage your references and generate correctly formatted entries. Failure to cite sources may result in lower marks under the **Research** and **Professionalism** criteria and could constitute academic misconduct.

Example inline citation:

> This method is commonly used in game AI systems (Rollings & Adams, 2003).

Example bibliography entry:

> Rollings, A. and Adams, E. (2003) *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing.

Declared Assets

You must include a **Declared Assets** section in your Development Commentary if you used **any of the following**:

- Assets (e.g. 3D models, textures, sound effects) **not made by you**
- Code/scripts sourced elsewhere
- Text, diagrams, or documentation partially or fully generated with **AI tools**
- **AI-assisted images**, textures, or design elements

Clearly list what was used and where it came from. Be honest and transparent — declared assets **will not automatically reduce your grade** but failing to declare them may be considered plagiarism.

Example:

> **Declared Assets:**
>
> - "Sci-Fi Wall Texture" by Kenney.nl (CC0)
> - DialogueScript.csv created using ChatGPT 4.0
> - Idle.fbx downloaded from Mixamo

AI-generated material must always be acknowledged — this includes code suggestions, visual content, or text drafted or revised by tools like ChatGPT, GitHub Copilot, DALL·E, etc.

Documentation

What Is Project Documentation?

In addition to your Development Commentary, **you are expected to maintain project documentation**. This refers to the technical, creative, and procedural notes that accompany your project. It should explain:

This refers to the technical, creative, and procedural notes that accompany your project. It should explain:

- How your project works
- How others can build, run, or test it
- Key design or development decisions
- Instructions for users, testers, or collaborators

This is a standard expectation in industry and part of working in a team or delivering professional builds.

Writing Guidance

Effective documentation should be:

- **Clear** – use short, direct sentences
- **Structured** – break content into logical sections with headings and bullet points
- **Actionable** – provide step-by-step instructions or setup steps
- **Contextual** – explain the purpose or logic behind key systems or code

Use a **second-person tone** ("you") when giving instructions. Keep explanations concise, and use inline formatting for code and file paths like `GameManager.cs` or `Assets/Blueprints/`.

Example:

> You'll need to install Unreal Engine 5.3 and enable Lumen under Project Settings. The `LevelManager` blueprint uses a state machine to transition game phases.

Tools for Writing or Generating Documentation

You are encouraged to write documentation manually, but you can use tools to **support and speed up** the process. These include both traditional and AI-assisted options.

Tool / Method	Purpose	Notes
Markdown (`.md`) with AI Assistants (e.g. ChatGPT, Copilot)	Lightweight, editable documentation with optional AI support	*Recommended* – Combine `.md` files in Git with AI-assisted drafting. Must declare any AI usage.
Doxygen	Generate docs from code comments	Great for larger codebases with good inline comments
Unreal Blueprint Comments	Visual annotation	Helps markers and teammates understand logic visually
Other Generators (e.g. DocFX, Sphinx)	Language-specific documentation automation	More advanced, used in industry pipelines

> Always **review and edit** auto-generated or AI-assisted content to ensure accuracy and clarity. Any AI-generated documentation must be listed under your *Declared Assets*.

Where Should I Write Documentation?

Your project documentation must be stored in a **publicly accessible** location and linked in your Development Commentary. Common industry-standard options include:

Markdown Files (Recommended)

Use `.md` files inside your Git repository to keep your documentation organised and accessible. Common examples include:

- `README.md` — your development commentary
- `DOCS.md` - your documentation overview and links
- `INSTALL.md` — installation or setup instructions
- `USAGE.md` — interaction or testing instructions
- `KNOWN_ISSUES.md` — bugs, limitations, or workarounds
- `PLAYER.md` — brief explanations of key scripts and logic for specific namespaces or systems

Markdown is preferred for its **simplicity**, **version control compatibility**, and **widespread industry use**.

Confluence

Confluence is a collaborative documentation tool widely used in studios and companies. You may use a free version and provide a public view link if needed.

Notion

Notion is another common platform for documenting game and software projects. You may create a project dashboard, dev log, or technical breakdown in Notion — but it must be publicly viewable and linked clearly.

Unreal Blueprints

If you are using Unreal Engine Blueprints, **you must include comments** that explain the usage, purpose, and logic of your blueprint scripts.

> Think of blueprint comments as your in-editor documentation — they help tutors, teammates, and industry professionals understand your work at a glance.

Tips for Blueprint commenting:

- Use **comment boxes** to label sections of logic (e.g. "Player Input Handling" or "Damage Calculation")
- Add **brief descriptions** above custom functions, macros, or event graphs

Minimum Documentation Requirements

At a minimum, your project documentation **must include**:

- **Installation or setup instructions** – how to run or build the project
- **Required dependencies and tools** – e.g. Unreal version, Unity packages, plugins
- **Known issues or limitations** – bugs, unsupported features, or workarounds
- **Instructions for testing and interaction** – how a user or marker should test or interact with your

project

- **Basic code documentation** – short explanations of scripts, key systems, or logic used

> Think of this as your project's user manual or handoff document — someone unfamiliar with your work should be able to get started by reading it.

Project Files & Assets

All files must be submitted as accessible links in your Development Commentary. If the assessor cannot open your files, your work may be unaccessible.

Project Files (Unity & Unreal)

GitHub (Recommended)

GitHub is industry standard for version control. It allows collaboration, backups, and easier debugging. Not using Git will limit your support options.

- Git for Unity
- Git for Unreal
- GitHub LFS

> GitHub is free for students via GitHub Education

Perforce (Advanced Unreal Only)

Helix Core (Perforce) is used in AAA studios. Unreal projects using Perforce **must add the unit leader as a user**.

- Perforce for Unreal
- How to Add Users

Video Demonstrations

Use YouTube:

- **Unlisted:** Shareable without being public
- **Public:** Ideal for showcasing work in a portfolio

Embed example:

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/CWwc2FcoLC8" frameborder="0"
allowfullscreen></iframe>
```

Game Builds

itch.io (Recommended)

Use itch.io for public hosting, testing, and showcasing.

- Unreal to itch.io
- Unity to itch.io

GitHub

You may store builds in your repo (LFS recommended for large files).

Here is a new section you can insert into the **Games Development Pathway Assessment Guidelines**, titled **"Figures"**, written in the same formal, instructional tone as the rest of your document. It explains figure usage clearly, includes your example, and introduces the formatting rules students should follow:

Marking

How Will I Be Marked?

You will be assessed across four categories:

- **Professionalism** (20%)
- **Research** (30%)
- **Creativity** (20%)
- **Technical** (30%)

Each category is graded from 0 to 70+ using descriptors:

Grade Band	Description
0-24	Inadequate / Substandard
25-39	Weak / Underdeveloped
40-59	Satisfactory / Moderate
60-69	Strong / Well-executed
70-79	Excellent / Polished
80+	Professional Quality

Assessment Criteria

Professionalism