

PROGRAMMING AND TECHNICAL DESIGN FOR GAMES

Thom Kaczmarek*

University: University of the Arts London

College/School: London College of Communication / Screen School

Department/Program: Moving Image and Digital Arts / BA (Hons) Games Design

Level: Undergraduate (Level 4 – 1st year of 3-year-long BA)

Credits: 40 (UK) equivalent of 10 US Semester Credits or 15 US Quarter Credits

Length: 14 weeks

Medium: Face-to-Face with Blended/Hybrid provision

Unit (UK) / Course (USA) Keywords: Object Oriented Programming (OOP), Game Design Patterns, Finite State Machines, Gameplay Scripting, Game Design Documents, System Integration and Asset Pipelines, Profiling & Optimisation.

CATALOG DESCRIPTION

From the simplest casual game to the most complex open-world game, games provide obstacles to overcome, problems to solve, enemies to defeat, collaborators to struggle alongside, and even strangers to engage in conversation with. Game designers must understand how these very different experiences are represented as systems and rules, not just in the game but also in the documentation. By discussing games as rule-based systems (if-then statements) and the information they need to maintain (variables and states), students can design and develop in-game events and game object behaviours to guide player actions.

This unit/course introduces students to the fundamentals of game programming, focusing on how to think about, plan, and write code to implement a wide range of game mechanics. Using a hands-on, problem-solving approach, students will learn flexible programming skills that will enable them to develop games from technical Game Design Documentation (GDDs).

Students will be guided in using a contemporary Game Engine, Unity (Unity Technologies, 2026a), alongside an Integrated Development Environment (IDE), Visual Studio (Microsoft, 2026), to develop game environments and write code to direct the behaviour of game objects, support a range of player controls, and record and display player achievements on-screen. We use forms to assess functionality (bugs), usability (interaction), and the game's level of challenge (accessibility and balance).

COURSE PURPOSE AND OBJECTIVES

This unit/course is assessed holistically on a pass/fail basis against the marking criteria specified in the Learning Outcomes section. It teaches students the principles of programming and technical design for games, in the context of utilising a predetermined game engine to develop game projects within predefined limits.

This offers students the opportunity to study Game Design theory in a practical context. The knowledge and skills to conceptualise, design, and develop their own original game ideas are developed through teaching fundamental programming principles. This is done in a gamified, game-based format, enabling the creation of goal-oriented, challenging experiences constructed from rules that support player dialogue.

To help understand the whole game development life cycle, students need to understand not just game development jargon (vocabulary) and range of project constraints (that can be due to engine, programming

* Thom Kaczmarek is an award-winning gamedev (1997-), academician (2018-) and founder of www.gic.gd (2007-)

language, target platform, project scope or time available), but also to use scripting (in-engine programming) to prototype and problem-solve solutions. This will evidence desired features and functions as playable artefacts rather than just conceptual wishes.

Knowing how games are made enables students to perform deep-dive game analysis and break down individual systems that overlap and work together, which can be recreated one by one from a mechanics perspective rather than from an aesthetic perspective as experienced by a player (Hunicke et al., 2004). This deconstruction of existing game mechanics down to the level of programmable rules is achieved with the use of Finite State Machines to communicate the behaviours of the game objects that will respond to the usage of variables, if statements and functions, allowing for the creation of in-game events that respond to player input and, in turn, manipulate game assets.

This enables students to quickly move beyond the conceptual and documentation stages of the project (such as initial ideas, mood boards, and storyboards) to a playable prototype. Demonstrating the validity (or not) of their work, moving them away from the label of just the 'idea person'. It also enables them to see the meaningful difference between low-level engine programmers (which they are not, but will work with) and higher-level, abstract scripting within a game engine (which many apprenticeship, graduate, and junior-level roles require of them; for technical roles, it is a must-have). It also enables them to experience the Indie concept of wearing multiple hats and to find their strengths across transferable skills.

Learning Outcomes (LO)

At UAL, students' work is marked against up to 5 assessment criteria (UAL, 2019). They are being used to help guide students' learning, and they are as follows:

- Enquiry: active learning and reflection. It is how students explore, research and learn about their subject. Enquiry is central to all creative learning at UAL.
- Knowledge: how students inform their work, explore diverse cultures and connections and appreciate what they do in a wider context.
- Process: students' journey of learning. It is how they take risks and experiment, moreover, how they keep the creative momentum going – developing ideas from start to finish.
- Communication: telling the story of their learning and making. It is how they share their learning with an audience, presenting and explaining their work to different people.
- Realisation: the work students create and how this reveals what they have learned. It is a look back at the work produced and an evaluation of it.

For this assessment, the used criteria are **Realisation**, **Knowledge**, and **Communication** represented within the following Learning Outcomes (LO):

- LO1 Demonstrate the ability to build digital game mechanics by applying coding techniques from visual and text-based designs (**Realisation**)
 - The mapping of mechanics from the Game Design Document to the working digital prototype will evidence this.
 - The effectiveness of the prototype will be evaluated based on the features and mechanics implemented against the technical GDD.
 - Development of 12-second challenges following given templates
- LO2 Develop the game systems which include emergent qualities and respond to player achievement (**Knowledge**)
 - Emergent qualities will be evidenced through the application and balancing of the rules that relate to the presented game elements.
 - Player achievement will be evidenced by feedback from the game systems.

- LO3 Evidence of skills needed for effective learning and communication of ideas, arguments and criticism in visual form (**Communication**)
 - Visual feedback will be evidenced within the game's communication with the player, regarding any in-game actions.
 - The application of visual feedback to communicate the state of game objects in the final builds of submitted games will provide evidence of this.

COURSE CONTEXT

At University of the Arts London (UAL), courses (UK) / majors (USA) come with pre-packaged units (UK) / courses (USA), and students cannot pick and choose what they want to study to collect the required number of credits. UAL has been ranked second in the world for Art and Design in the QS World University Rankings® since 2018 (UAL, 2026a). Moreover, as an art institution, we do have a strong ‘auteur culture’ that results not only in 25% of our graduates being self-employed post-graduation but also in our university being the number 1 higher education institution for producing start-ups (UAL, 2026b). Additionally, approximately 20% of the student body identifies as disabled, neurodivergent (such as ADHD or dyslexia), or has a long-term health or mental health condition, impacting the priorities and delivery structure. Consequently, the submission structure and requirements for this unit/course must support our diverse learners while also meeting industry standards and expectations to maintain our reputation and ranking.

Our students are creatively motivated, often pursuing self-initiated projects (analogue/digital) for game jams, competitions, and commissions. They are technically curious but have uneven or limited prior experience and cannot explain the fundamentals. This does not, however, stop them from being ambitious, as many imagine themselves as studio founders or technical specialists. They are open to multiple career pathways but are still forming a professional identity as they are underexposed to collaborative, industry-standard practices, such as teamwork or industry documentation.

As such, we do not expect students to have any prior programming background, and we have taught programming as part of the first-year curriculum for decades (XNA, Flash, and now C#). To bring equity for those who have never used any game engine, and those who have, we put certain constraints in this unit/course on the Unity (Unity Technologies, 2026a):

- 2D games only (no 3D)
 - Scrolling as expansion and not default (focus on core mechanics, not content in empty worlds)
- Ensuring that all programming concepts are tangible
 - No abstract concepts such as Arrays, Lists, or Loops. Those concepts are introduced at a later stage.
- No built-in Physics
 - Students can write their own; they learn primitive sprite- and mesh-bound collisions.
 - Colliders are deliberately introduced at the very end of the unit (but only as 2D and without physics), so it is rare for students to use them in their submissions.

On the surface, those limitations can appear artificial; however, they streamline the learning process for students who have never been exposed to programming and digital game development, while remaining novel and challenging to those who were (self)taught, giving everyone structured pathways into technical aspects of game development through scaffolded materials and forced development of smaller and simpler games.

COURSE PEDAGOGY

'Show, do not tell' is our pedagogy. The focus is on delivering playable first, not just in student submissions but also in teaching materials. Rather than teaching programming for programming's sake, we start with the fundamentals and demonstrate how each is used to make a game. Consequently, every class is contextualised in terms of what students may have experienced as players. As we strive to be consciously inclusive with our students, we aim to complement our digital materials with non-digital activities that provide tactile and visual representations of every programming concept we deliver.

By the end of 14 weeks, students had over 100 presentations, over 100 game examples, and over 100 game mechanics demonstrated, all of which they could have chosen to engage with and explore more deeply. Alongside this, hundreds of smaller coding tasks and activities start as follow-along tutorials and progress to commented, functional code that they are asked to adapt and expand. This is in the style of what students could find on GitHub (GitHub Inc., 2026) or Stack Overflow (Stack Exchange Inc., 2026). All this material is bespoke, designed and developed by the teaching team over several years, and updated annually.

This unit/course is not part of the Bachelor of Science degree but of the Bachelor of Arts degree. It is not delivered in the Game Development but in the Game Design course. This makes us more closely aligned with Creative Computing than Computer Science. Consequently, we use coding primarily to inspire people not to be afraid of lifelong learning and technology, and, secondly, as a means to an end: a playable digital prototype without limitations imposed by genre-specific game engines or visual coding.

To be inclusive of international students (for whom English is not the first language), students with learning disabilities, or those with additional caregiving and work obligations, the entire year's teaching materials are made available from day one through a scaffolded experience within the Virtual Learning Environment, Moodle (Moodle, 2026). Each week has its own section with an overarching narrative that breaks down into grouped sub-topics, each with its own activities.

Furthermore, we do run student-focused surveys and queries for mechanics, interactions and systems not yet covered, which are called 'Live Coding' as after agreeing on the mechanic, for instance, bamboo cutting sequence from *Ghost of Yōtei* (Sucker Punch Productions, 2025), alongside students we find several reference videos, discuss what is just audiovisual juice (Cardoso, 2026, <https://tinyurl.com/mryus4m4>), which during reverse engineering is noise as we want to see what does really happen in terms of game design and game mechanics, and member of staff recreates those step by step, commenting and discussing their thinking process along the way, within all technical and coding limitations that are self-imposed for this unit. Consequently, any live coding materials delivered one year ago will be incorporated into the teaching materials for all subsequent years.

COURSE TEXTS, GAMES, SOFTWARE, AND HARDWARE

All our primary and secondary reading lists are available through the campus library, in both physical and electronic formats. Furthermore, we do utilise several materials that are freely available online from their authors, such as Chris Crawford's 'The Art of Computer Game Design' (1984, <https://tinyurl.com/37tpxved>), Robert Nystrom's 'Game Programming Patterns' (2014, <https://tinyurl.com/3ueywuen>), and Jesper Juul's articles (n.d., <https://jesperjuul.net/text/>). The reading list also recommends several YouTube channels and playlists, such as Indie Game Clinic (Baxter-Webb, n.d., <https://tinyurl.com/8daranak>), Game Maker Toolkit (Brown, n.d., <https://tinyurl.com/5n8wtvf3>), Coding Adventures (Lague, n.d., <https://tinyurl.com/pwdru5x8>), and Code Monkey (Cardoso, n.d., <https://tinyurl.com/5x66rtxn>), as a bridge between code and research/design.

The course has a dedicated High-Spec Windows PC classroom. The Unity (Unity Technologies, 2026a) game engine and Visual Studio (Microsoft, 2026) are being used because they offer free personal-use licenses that students can use at home, regardless of whether they have a Windows or Mac computer. Outside of scheduled teaching time, students have access to classes during extended evening hours or in

alternative spaces, such as the shared Digital Space (London College of Communication, n.d.); consequently, they do not need a powerful enough computer at home.

Furthermore, as part of this unit, we cover game accessibility (from input to design perspectives), in line with the SpecialEffect Dev Kit (SpecialEffect, 2022). The course has dozens of Xbox controllers, several accessible controllers (both Xbox and PlayStation), Nintendo Switch Devkits (require NDAs), Valve's hardware (Valve Corporation, 2025), and some mobile hardware on offer. Unity (Unity Technologies, 2026a) allows us to export to those and more. For a decade now, anyone can convert their home retail Xbox (One, Series X/S) into a devkit using the Developer Mode app (Crecente, 2016), and we have teaching materials and expertise to support this.

COURSE ASSIGNMENTS

The unit/course includes both formative (evaluation and feedback-only; ungraded) and summative (graded) assessments. This is an individual submission; students are expected to work independently. Although group work is part of in-class activities, this is done in a formative capacity.

Non-Digital

Every week, students work together in different groups on analogue activities, prototypes and game ideas that link game design theory, their in-game experience, and what is being taught in coding class. Those are critiqued and given formative feedback on the day, and allow for self-reflection and soft-building of social skills by forcing students to work together without risking their grades.

Digital formative (feedback only)

The formative parts consist of weekly (section- and subsection-based) submissions that help us understand students' learning progress and where they are stuck, informing in-person support and tutorials, as well as the creation of new exercises and activities to explain the problems they face from their unique perspectives. As touched upon in the Course Pedagogy section, those submissions can:

- Test students' understanding of the range of common in-engine errors and being able to fix them (broken project files, with purposeful in-code errors and mistakes)
- Require re-creation of pre-built playable activity showcased in class, evidencing students' ability to deconstruct and implement mechanics just by watching gameplay'
- Building a short, playable game that either
 - Provides a unique mechanic twist to the in-class template game
 - Create a game using a specific prompt (Cat & Mouse), a sprite sheet or a rudimentary mechanic (object drops and loops)
 - Re-creating specific game experience in a set time limit, either by following instructions provided (minimal pass), on your own with general guidance on what is expected (good) or with full coding and creative control (high achievers)
- Depending on the nature of the submission, the students either
 - Upload playable games that can be reuploaded if updated based on players' feedback.
 - Upload the project files for the teaching team to evaluate. However, after submission, students obtain access to solution sheets (which present various approaches to the same task), so they are limited to a single upload to limit cheating opportunities.

Digital summative (graded)

Building on the formative submissions, the summative parts consist of two parts in the form of three games – each requiring a playable Windows executable and access to project files (e.g., a Unity Package). Year 1 (Level 4), Block 1 (First 1.5 terms) units/courses are assessed on a UAL Pass/Fail basis only; the assessment is holistic (all requested parts must be submitted and completed to a satisfactory standard) to achieve a pass grade (which, by UK standards, is 40% completion).

- **Part 1: Game Design.** Game is a larger game project (with a 30-second game loop) that requires students to use the technical practices taught in the course (programming restrictions mentioned previously) to develop a specific 2D game.
 - This game must be playable, based on a Game Design Document (GDD), and use game assets supplied by the teaching team (the same game for all students).
 - By adding the appropriate code, the game must recreate the play experience and mechanics as documented in the supplied GDD, utilising only the provided assets.
 - When the game is functional, students will be able to add mechanics (such as instructions, resets, and a high-score table) to better communicate the nature of the challenge to the player.
- **Part 2: Three Twelve-Second challenges** consist of smaller original game projects. Working with the structure of the Game Design Template supplied by the teaching team, students are to design three (3) games, which present challenges to be completed in under twelve (12) seconds.
 - **Physical challenge** will be based on the player's input.
 - The player must learn to use variable levels of input to achieve the goal, relying on physical dexterity, reflexes, or hand-eye coordination.
 - **Mental challenge** will require some form of reasoning to achieve the goal.
 - The player will need to build and test conceptual models rather than rely on simple memorisation.
 - **Accessibility Challenge**
 - Merge keyboard and mouse inputs and map them to a controller (Left stick movement, right stick mouse, south button accept, east button cancel)
 - All share certain limitations, to name a few:
 - First, Physical OR Mental has mouse input. That is the mouse movement and the left mouse button for action.
 - The second one has keyboard input. Permitted keys are WSAD (mapped to the Arrow Keys) and the Space bar for action.
 - The input must be the same in-game and in the menu (consistency)
 - Running out of time is the only permitted loss condition.

The reasoning

The purpose of Game Design Game (Part 1) is to get a benchmark of students' understanding of the concepts taught (such as state changes) within a controlled game environment. Despite this restriction, the freedom to create in-game logic makes it easy to see how different students approach the problem in their coding style.

The way teaching materials are structured allows this bigger game to be created with full marks using only the first section of programming materials (however, the resulting code will be quite extensive, at around 1800 lines). However, every other week allows for a reduction by about a third, to under 180 lines of code by the 7th week. This is done on purpose, as we do not expect our students to be computer science-level programmers, but rather to script playable mechanics that either prove or disprove the (re)playability of their ideas.

As part one is quite limiting and uniform, the purpose of the smaller games in part two is to give students a certain level of creative freedom with their design skills and to test their understanding of coding. This works as self-reflection and evaluation, with students utilising their other skill sets and past experiences (art, narrative, programming) to expand on the game's simple premise and create something unique that can form part of their portfolio.

ARTIFICIAL INTELLIGENCE IN THE COURSE

University of the Arts London has an AI and Assessment policy (UAL, 2026c), developed by staff and students, which operates on a traffic-light system depending on the nature of the assessment.

It outright prohibits the (red) use of AI tools in areas that require students' reflection (crits, discussions) or skills assessment (written, studio, and practical work). However, it is context specific (orange) with students permitted to disclose how they used the tools, to what extent, and use it as support to draft and structure content (authored by students), be a dialogue partner for debate and ideation purposes or supporting a particular process (such as code debugging or translation). Permitted areas (green) are those where AI will explicitly form part of the assignment task and learning outcomes, including developing code, analysing or comparing content, researching, and seeking answers (in line with traditional search engines).

As AI features are now built into Microsoft products, Co-Pilot in Windows (Microsoft, 2023) and GitHub (GitHub Inc., 2026), Visual Studio (Microsoft, 2026), with IntelliCode (Microsoft, 2026), as well as in Unity: AI Assistant and Generators (Unity Technologies, 2026b), and Sentis (Unity Technologies, 2024), they are being leveraged as helpers for student learning rather than replacements. Online tutorials are easy to follow and engage with. Consequently, we try to teach our students to use them as advisors, but with the final decision and direction being made by humans.

Programmers' dirty secret for years was their ability to find solutions on Stack Overflow (Stack Exchange Inc., 2026), GitHub (GitHub Inc., 2026), and other repositories, which, thanks to AI tools, is now easier to access and use. Furthermore, as English is the lingua franca of global coding. The most popular non-English programming language is Easy Programming Language, which holds a 0.15% market share (TIOBE Software BV, 2026). The use of AI assistance offers equity and support to students with dyslexia, whose first language is not English, or both.

However, even with that assistance, any solutions obtained through AI tools must be disclosed and adapted to the style and scope of students' projects and to specific coding limitations. The introduction of said limitations was intended to help streamline and scaffold programming learning for non-technical students. However, these days it also trips up AI models, as it requires thinking outside the box and using more tangible concepts.

As our students are digital natives, they grew up with the assumption that the best answers are presented on the first page of Google results. Already more than 1 in 10 of Gen Z are starting with either Circle or Google Lens to search for a very specific answer (Taylor, 2025). This creates an issue, as human-like, structured answers from AI tools (which are now part of search engines) are taken at face value and not interrogated or critiqued for their accuracy or the specific source used to generate them. As such, ensuring that students engage meaningfully with those new technologies, use them as a critical friend, rather than offloading their thinking, is a current point of discussion.

COURSE ASSESSMENT

The goal of the Year 1 units is to help students adjust to the British HE system. 'Programming and Technical Design for Games' has only a pass-or-fail submission. The grade is holistic; all parts must be submitted and completed to a satisfactory, passing level before a final single-pass grade for the entire submission is awarded.

Part 1 Constitutes 55% of the overall grade, and Part 2 is 45% (15% per game). To pass, students must complete each game to at least 40% of its requirements.

As 40% of the grade for part 1 is given to the expansions (instructions, high-score table, manual game reset – not just scene reload), the basic version of this project alone cannot constitute a pass, as doing so would only give a total of 33% (60% of 55%).

This teaches students that to pass any future submission:

- All submission elements must be submitted
- Having a playable (even if buggy) build of the game is the minimum requirement
- Anything less will constitute a failure.
- Not engaging with formative tasks and teaching materials will cause you to fall behind and inadvertently impact the quality and complexity of your work

EXPANDED COURSE OUTLINE

The content that students engage with every week is bespoke and created by the teaching team, consisting of dedicated presentations, handouts, and analogue exercises, as well as custom-made activities (games, mechanics, etc.). Rather than using publicly available resources, we keep it in-house to provide proper scaffolding, avoid licensing issues, and ensure quality. This unit/course has two 4h classes per week (Wednesday half-class to enable offline activities) and a Friday whole-class session to work together on computers. Table 1 summarises this information. Format of the class is as follows:

- Friday
 - Presentation (1.5-2.5h)
 - Student coding exercise(s) (1-2h)
 - Evaluation of understanding (0.5h)
- Wednesday
 - Presentation (0.5-1h)
 - Non-digital group activities related to presentation materials (2-3h)
 - Presentation and evaluation of understanding (0.5-1h)

Materials are made available as grouped topics on a rolling weekly basis as they are taught in class or when students complete previously required tasks (we use Moodle Completion Tracking to automate this process). This helps avoid overloading neurodiverse students with too many activities, while also enabling everyone to work at their own pace (whether they are ahead or behind). Those materials are available to students throughout their course, allowing them to revisit pre-recorded materials and activities as refreshers.

Each subsection has a dedicated presentation (primary material) that introduces concepts in a short, comprehensive manner, delivered in person and available as pre-recorded videos with annotated slides.

Each presentation is supported by additional activities that are part of in-class and homework tasks:

- Obligatory:
 - Exercises include instructional handouts that teach students how to work within the Game Engine and IDE, which can be obligatory (Basic-Intermediate level). We advise students to complete the obligatory ones, as these are required to pass the unit.
 - Annotated presentation slides and pre-recording of presentation (it is not an in-class recording)
- Optional Context tasks include

- Exercises that are (Intermediate-Advanced) and are there to expand understanding or allow students to dive deep and pique their interest.
- Explanatory Games as well as Mechanics that demonstrate how obligatory exercises from this section can be used to either make something new, or expand on things taught in the past
 - Those tasks can be either follow-along handout tutorials OR
 - Playable and fully commented game projects that encourage experimentation and modification.

After the presentation, a secondary self-evaluation H5P test follows the primary materials, allowing students to assess their understanding of the subject matter without requiring them to answer publicly in class.

Every obligatory section ends with a formative Interactive Project submission that is either private (just for tutors to assess) or public (other students can see your game, play it and comment on it)

- This allows students to feel that they work not only as seen and feedback on by staff, but also seen and engaged by their peers, which builds rapport and works like an icebreaker
- The submission activity also breaks down to several difficulty levels (to help with scaffolding and challenge) and varies in task, but is in the same format:
 - Compiled an executable project for other students to play and evaluate
 - Access to project files (such as UnityPackages) for the teaching team to evaluate and provide feedback on areas where students have made coding mistakes.

Subsections often include optional formative games that students can engage with, create, or modify; these will not be listed in Assignments (only items evaluated by staff or peers will be). Those can be considered challenges or small jams.

There is also a series of Extracurricular (programming) activities and sections that are unlocked only if students choose to access them (the decision can always be reversed), allowing them to choose the complexity and amount of material made available to them.

Week	Section Topic	Sub-section topics	Assignments
1	1a – Introduction to University	1. Introduction to College 2. Introduction to Blended 3. Introduction to the course 4. Introduction to submissions 5. Introduction to GameDev history (from a programming perspective)	Formative quizzes ensure that students know how to evacuate, who to contact with specific queries, and their own background.
	1b – Technical introductions	1. Required Software: General (OneDrive, Adobe Suite, Office) and GameDev (Unity, Visual Studio). 2. How to organise projects in Unity 3. Introduction to Unity Graphic User Interface.	Formative submissions ensure that students have set up their software correctly on their home machines (and that everything is integrated).
2	2 – Inputs and Outputs	1. Introduction to Coding (high level) 2. Basic Objects and Code (game objects, Unity script structure, console debugging) 3. Variables and Structures (integers, floats, strings, comments, operations on variables)	In subsection 6, 'Keyboard Input', students are given the first set of games (4 different examples) and are asked to create their own game, which will be shared publicly with their peers. The game operates solely on variables, keyboard input, and console output. Despite those limitations, the projects range from Quizzes through escape-the-room experiences to narrative RPGs with multiple endings. At this point, students also decide whether they wish to have access to more (extracurricular) programming concepts and pursue the Developer path or remain purely design-

Week	Section Topic	Sub-section topics	Assignments
		4. Conditional (if) statements 5. Comparison Operators (<, >, ==, !=) 6. Keyboard Input (Input class) 7. Position and Movement (Vector 2, moving pre-existing game objects) 8. Projects and Executables (impact of Frame Rate, Screen resolutions and Aspect Ratios on compiled builds) 9. Code Errors	oriented with only a rudimentary understanding of code. Subsection 9 gives dozens of coding errors (unattached scripts, wrong variable types, typos) as a teacher-evaluated submission, which are recreations of the most common mistakes that students make – having them experience them first hand in a controlled environment and solve, resulted in a significant amount of repeat answers to the same queries in large class cohorts of up to 60 students.
3	3 – Basic Communication	1. Communication between Objects (Script and GameObject as variables, TextMesh) 2. Logical operators (AND && as well as OR) 3. Communication within the World (Space Wrapping and Clamping, as well as Scaling and Pivots)	Subsection 1 includes a formative ‘Text Submission’ to ensure students know not only how to recreate the logic of the tasks presented, but also how to display a mix of text and variables on screen. Subsection 3, among other things, asks students to recreate a specific game (Pull the Rope, which is their 16th game) and add any mechanic they can think of (some students pushed it as far as 1 v 1 basketball)
4	4 – Amazing Numbers	1. Numbers in games (Data type)	Subsection 1 is formative, in which students recreate simple random-noise mechanics (along one or two axes).

Week	Section Topic	Sub-section topics	Assignments
		conversions, Randomness) 2. Make it your own (Custom functions) 3. (Extra) Time in games (deltaTime) 4. Human-Computer Interactions (introduction to Artificial Smarts) 5. Connecting Objects (Linked Lists) 6. (Extra) Basic Inheritance (Game Objects within Unity Hierarchy) 7. (Extra) Basic rotations	Subsection 5 asks students to create a linked, circular menu where we can select objects by moving left or right. Linked lists also help introduce grids and embody fundamentals that later become part of Arrays and Search Algorithms. We also collect the first month's feedback here.
5	5 – Sprites and Colours	1. Sprites (from high-level concepts of visual communication, through technical Sprite Renderer and usage of Sprites in-engine) 2. Coding Colours and Transparency (Ability to impact how sprites look in real time, helps with visual communication)	Subsection 2 includes the 30th game, Swimmer, which starts simple (get the character from the left-hand side of the screen, flip its sprite, and display a win message) and progresses to very complex (4 CPU- or player-driven opponents, each with unique colour, timed endings, and a result table).

Week	Section Topic	Sub-section topics	Assignments
6	6 – Collisions and simpler coding	1. (Extra) What if? (if, else, else if; switch statements, branchless coding) 2. Collision course? (Bounds collisions)	Subsection 2 asks students to recreate a game (37th) provided by the teaching team as a playable build only. The tasks utilise previously learned colour coding to display three states for the NPC character and two for the player. The game gives you a goal, but it feels unattainable, further testing students' deductive capabilities through gameplay observation and system testing. This is also the week when students are given the Game Design Document for their summative submission.
7	7 – States Everywhere	1. States are everywhere (coding iteration of Finite State Machines) 2. Typo Resistance (introduction of Enumerators as superior to bool, string and integer states) 3. Managing the game (demonstration of how games use states as game managers)	Subsection 3 Formative submission requires students to recreate a playable test (object moves left-right on return to centre, changes direction to down-up and loops) with a superior state that allows its movement or forces it to stop.
8	8 – Beyond Keyboard	1. Input (Mouse position, buttons) 2. Positioning (World Space, Screen Space and Viewport) 3. Collisions (replacing the mouse cursor with in-game objects)	Subsection 3 asks students to develop a simple card game of matching pairs (65th available game project)

Week	Section Topic	Sub-section topics	Assignments
		4. Seeing the world differently (Accessibility through controllers and cross-platform development)	
9	9 – Animations	1. Intermediate Animation (Animator and Animation system)	None ('Introduction to Game Design' ends in week 10)
10	10 – (Recap 2) Assessment Practice	1. The good and the bad (recap of previous 9 sections with focus on the upcoming submission)	Submission week for 'Introduction to Game Design'
11	11 – (Recap 3) Term 1 Festive Season	1. Summary of teachings (Unity, Code and Design principles) 2. Recap Exercises (Activities for Christmas) 3. Christmas break (optional Christmas-themed games)	Formative exercises (teacher-assessed) ensure that students are ready for more advanced coding post-Christmas materials that are mostly optional.
4-week-long Christmas Break			
12	12a – Touch Point (Basic)	1. Introduction to Unity Colliders (Collider, Rigidbody with constraints, Materials)	None.

Week	Section Topic	Sub-section topics	Assignments
	12b – (Extra) Power of Strings and Loops	1. (Extra) Behind the strings (operations on strings of characters) 2. (Extra) If not loops, then what? (creating basic loops using iteration with if statements)	
13	13a – (Extra) Above and Beyond	1. (Extra) Code inheritance 2. (Extra) Menus 3. Ready for more? (encouragement to go back and change the Designer role to developer, to unlock additional coding tasks)	None.
	13b - (Extra) Time of Creation	1. (Extra) ‘Basics’ (adding and removing object components on the fly) 2. (Extra) ‘Dynamic Advancements’ (creation and linking of GameObjects with components via script)	
14	14a – (Extra) Advanced Rotations and Math	1. (Extra) Advanced Rotations	Submission week.

Week	Section Topic	Sub-section topics	Assignments
		(Rotation Quaternions) 2. Interpolating (LERP, Slerp) 3. Dynamic Visual Effects (Prefabs and Instantiation)	This is Global Game Jam Week, during which our first-year students can participate.
	14b – (Extra) Stepping up	1. Advanced Techniques (Blend Trees, Sprite Masks, While loops)	
2 weeklong reading weeks, with industry activities			

Table 1 Expanded Course Outline

COURSE BEST PRACTICES

This course was not built in a single burst of inspiration. It grew over decades, shaped by many educators and practitioners as our first graduates finished in 2003. That legacy is a reminder that you do not need to know everything to teach well. Start where you are. Lean into your strengths, and let your local expertise be the foundation you build on.

If programming is not your comfort zone, choose tools that do not demand it. Clickteam Fusion (Clickteam, 2024), for example, has supported educators since the mid-90s and remains a reliable 2D engine. It is the community I began with, and it has powered well-known titles like *Five Nights at Freddy's* (Cawthon, 2014) and *Baba Is You* (Teikari, 2019). We were always sharing how we made those games and made little examples for each other, and this is a culture I brought to the course.

If art is not your strength, focus on narrative, tone, and emotional clarity. *Thomas Was Alone* (Bithell Games, 2012) shows how compelling a game can be with simple visuals and strong storytelling. Furthermore, when you do need assets, there are plenty of CCo and affordable licensed bundles.

If you do not yet have an industry network, start small. Search LinkedIn for local creators: indie developers, board-game designers, escape-room builders, RPG writers. Reach out, invite them for a chat, visit them in their spaces, and let those early connections grow. Over time, these collaborators often become trusted partners or part-time industry specialists within your teaching team.

When I joined UAL in 2018, we were transitioning to Unity (Unity Technologies, 2026a) with no exercises or mechanics library. By the end of the year, we had a dozen. A year later, students were proposing their own ideas. The growth did not come from knowing everything; it came from playing to my strengths and starting with what I could do.

As I had to create all teaching materials from scratch, as an avid game designer, I did not want to teach coding for the sake of coding, but rather through the awe of discovery, so I pitched to my line manager what I wanted to do and why, and they gave me a chance.

I utilised game-based learning, such as playing the *Lightbot* series (SpriteBox LLC, 2020) or *RoboRally* (Garfield, 1994), to make programming instructions tactile and tangible. All games I designed were playable

from the engine and encouraged editing and learning through breaking them. To make the game development learning experience a game in itself, I introduced extras like live coding and easter eggs in presentations, rather than just handouts and tasks.

The Moodle and blended/hybrid structure came to me before COVID-19, when I observed students with work and other obligations, forced by extenuating circumstances to miss many classes, returning after weeks and trying to pick up where the class left off rather than where they had left off. As such, creating a system that would be inclusive, enable them to participate remotely, and track their own engagement (and for me to supervise and advise) became a matter of urgency, while the pandemic served as a baptism of fire for this approach's viability, it survived and won student nominated 'Outstanding Online Teaching & Learning Award', consequently I kept it and maintained it ever since.

FUTURE COURSE PLANS

'Programming and Technical Design for Games' is a brand-new unit that emerged from merging two halves of other units. This was done in response to student and industry feedback that called for clearer unit names that reflect what is directly taught within them, as past historical names, such as 'Introduction to Games Design' (Part 1) and 'Prototype Development' (Part 2), were rather generic. Consequently, the description merges what was taught with what was expected in the submission for both and will change to some degree before the course finalises its reapproval.

Ideally, I want it to become engine-agnostic rather than engine-centric, and to allow Godot (Godot Engine contributors, 2026) as a much more lightweight and open-source alternative to Unity (Unity Technologies, 2026a). Alternatively, to introduce a new engine/programming language every week that builds on previously learned material and requires students to make a small game: PICO-8 (Lexaloffle Games, 2025), Pygame (Pygame contributors, 2024), Clickteam Fusion (Clickteam, 2024), RPG Maker MZ (Gotcha Gotcha Games, 2020), Twine (Twinery, 2024). As programming and game development fundamentals are quite universal, being able to pivot between game engines and other tech tools is quite essential for any game developer.

With development of more open and modern platforms such as Analogue3D (Analogue Inc., 2025) it would be also interesting to be able to create niche side materials for extra credits/brownie points on developing games for older platforms to understand better restrictions in terms of CPU cycles, RAM Memory and available Polygons, which are often forgotten these days and lead to bad practices and game projects that are simply not optimised for good experience by average customer hardware. Those projects could also serve as interesting talking points on a student's CV, alongside potential additional business opportunities, as retro communities are quite willing to purchase new experiences for their beloved hardware.

Technical design is currently implicit in the taught materials. It is primarily addressed in optional extracurricular materials and activities, such as better organisation of the creation of in-editor tools, such as an in-game level editor, or allowing for balance changes based on external files. At the same time, game logic links the objects and gives them proper expected behaviours, without designers' involvement with code. Consequently, as noted in the title, it would need to be made more explicit. It may require its own requirements and formative submissions to test that specific skill set, which could be significantly dependent on Unity's (Unity Technologies, 2026a) built-in AI tools, which may help or hinder the currently existing project when used by non-technical team members, such as artists or level designers.

It would also be nice to reflect on this merger and see what has and has not worked, and why.

AI STATEMENT

As I am dyslexic and taught British English as a second language, I used Grammarly to ensure compatibility with APA 7 and grammar.

REFERENCES

- Analogue Inc. (2025). *Analogue 3D* [Hardware]. <https://www.analogue.co/3d>
- Baxter-Webb, J. [Indie Game Clinic]. (n.d.). *New to game dev? Start here* [YouTube playlist]. YouTube. <https://www.youtube.com/playlist?list=PL3n3ExgUASw7Mia-iygliv3nF4jHDNPQr>
- Bithell Games. (2012). *Thomas Was Alone* (PC version) [Video game]. Bithell Games. <https://www.bithellgames.com/thomas-was-alone>
- Brady, H., & Huntley, J. (2018). *Game programming for artists*. CRC Press.
- Brown, M. [Game Maker's Toolkit]. (n.d.). *Game Maker's Toolkit* [YouTube channel]. YouTube. <https://www.youtube.com/channel/UCqJ-Xo29CKyLTjn6z2XwYAw>
- Cardoso, A. (2026, January 15). *Improve your game feel! I had the chance to build this demo for the folks at Game Design Skills!* [Post]. LinkedIn. <https://www.linkedin.com/feed/update/urn:li:activity:7480500388954210304/>
- Cardoso, H. [Code Monkey]. (n.d.). *Code Monkey* [YouTube channel]. YouTube. <https://www.youtube.com/@CodeMonkeyUnity>
- Cawthon, S. (2014). *Five Nights at Freddy's* (PC version) [Video game]. Scottgames. https://store.steampowered.com/app/319510/Five_Nights_at_Freddys/
- Crawford, C. (1984). *The art of computer game design*. Osborne/McGraw-Hill. <https://dn721507.ca.archive.org/o/items/artofcomputergamoochri/artofcomputergamoochri.pdf>
- Clickteam. (2024). *Clickteam Fusion* (Version 2.5) [Computer software]. <https://clickteam.com>
- Crecente, B. (2016, March 30). Starting today, anyone can turn their Xbox One into a dev kit for free. *Polygon*. <https://www.polygon.com/2016/3/30/11318568/xbox-one-dev-kit>
- Felicia, P. (2020). *A quick guide to C# with Unity: Get started with C# in Unity in less than 60 minutes*. Self-published. http://learntocreategames.com/display_book_info/?id=12&/
- Garfield, R. (1994). *RoboRally* [Board game]. Wizards of the Coast.
- GitHub, Inc. (2026). *Change is constant. GitHub keeps you ahead*. <https://github.com>
- Godot Engine contributors. (2026). *Godot* (Version 4.7) [Computer software]. <https://godotengine.org/>
- Gotcha Gotcha Games. (2020). *RPG Maker MZ* [Computer software]. <https://rpgmakerweb.com>
- Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. *Proceedings of the AAAI Workshop on Challenges in Game AI*, 4(1), 1–5. <http://www.cs.northwestern.edu/~hunicke/MDA.pdf>
- Lague, S. (n.d.). *Coding adventures* [YouTube playlist]. YouTube. https://www.youtube.com/playlist?list=PLFt_AvWsXloehjAFLFspIPGaatzAwoouK
- Lexaloffle Games. (2025). *PICO-8* (Version 0.2.7) [Computer software]. <https://www.lexaloffle.com/pico-8.php>
- London College of Communication. (n.d.). *The Digital Space*. University of the Arts London. Retrieved July 7, 2026, from <https://www.arts.ac.uk/colleges/london-college-of-communication/student-life-at-lcc/facilities/the-digital-space>
- Microsoft. (2023). *Copilot in Windows* (Version 11) [Computer software]. <https://microsoft.com>
- Microsoft. (2026). *Visual Studio with IntelliCode* [Computer software]. <https://visualstudio.microsoft.com/>
- Moodle. (2026). *Moodle* (Version 4.5) [Computer software]. <https://moodle.org>
- Nystrom, R. (2014). *Game programming patterns*. Genever Benning. <https://gameprogrammingpatterns.com/>
- Pygame contributors. (2024). *Pygame* (Version 2.6.0) [Computer software]. <https://www.pygame.org/>
- Salen, K., & Zimmerman, E. (2004). *Rules of play: Game design fundamentals*. MIT Press. ProQuest Ebook Central.
- Schell, J. (2008). *The art of game design: A book of lenses*. Elsevier. ProQuest Ebook Central.
- SpecialEffect. (2022). *SpecialEffect DevKit* [Toolkit]. <https://specialeffectdevkit.info/>
- SpriteBox LLC. (2020). *Lightbot* (Mobile version) [Video game]. [https://lightbot.com /](https://lightbot.com/)

- Stack Exchange Inc. (2026). *Stack Overflow for Teams: Knowledge sharing for developers*. <https://stackoverflow.com>
- Sucker Punch Productions. (2025). *Ghost of Yōtei* [Video game]. Sony Interactive Entertainment.
- Taylor, D. (2025, July 23). Google Search Central APAC 2025: Everything from day 1. *Search Engine Journal*. <https://www.searchenginejournal.com/google-search-central-apac-2025-everything-day-one/551634/>
- Teikari, A. (2019). *Baba Is You* (PC version) [Video game]. Hempuli. <https://hempuli.com/baba/>
- Thirslund, A. [Brackeys]. (2026). *Brackeys playlists* [YouTube playlists]. YouTube. <https://www.youtube.com/@Brackeys/playlists>
- TIOBE Software BV. (2026). *TIOBE programming community index*. <https://www.tiobe.com/tiobe-index/>
- Twiney. (2024). *Twine* (Version 2.9.0) [Computer software]. <https://twinery.org>
- Unity Technologies. (2024). *Unity Sentis* [Computer software]. <https://docs.unity.com/en-us/ai>
- Unity Technologies. (2026a). *Unity* [Computer software]. <https://unity.com/>
- Unity Technologies. (2026b). *Unity AI assistant and generators* (Version 2.8) [Computer software]. <https://docs.unity3d.com/Packages/com.unity.ai.assistant@2.8/manual/index.html>
- University of the Arts London. (2019). *New assessment criteria*. Retrieved July 7, 2026, from <https://www.arts.ac.uk/students/stories/new-assessment-criteria3>
- University of the Arts London. (2026a). *UAL leads the world in undergraduate creative education for the 8th year running*. <https://www.arts.ac.uk/about-ual/press-office/stories/qs-world-rankings-2026>
- University of the Arts London. (2026b). *Industry and partners*. <https://www.arts.ac.uk/students/student-careers/industry-and-partners>
- University of the Arts London. (2026c). *AI and assessment*. <https://www.arts.ac.uk/about-ual/learning-and-teaching/digital-learning/ai-and-education/ai-and-assessment>
- Valve Corporation. (2025, November 12). *Steam hardware*. <https://store.steampowered.com/hardware/>