# Delivery / Session Plan

ICTGAM533 — Create complex 3-D interactive games

| | |
|---|---|
| **RTO** | [RTO name] |
| **Training package** | Information and Communications Technology |
| **Unit release** | 1 (current) |
| **Delivery mode** | online |
| **Document** | Session Plan · version 1.0 · review by [review date] |

## Session overview

| | |
|---|---|
| **Delivery mode** | online |
| **Total duration** | 16 weeks |
| **Amount of training** | 72 hours structured training (derived from volume of learning — confirm) |

Each element below is delivered as one or more sessions. Timings and dates are indicative and completed by the trainer against the group timetable.

## Sessions by element

### E1 — Prepare to create game

Indicative session time: ~964 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| Element 1: Prepare to create game | element intro | — | Preparation turns ideas into production work Briefs, assets, schedules and controls matter Good planning protects creativity, budget and quality This… |
| 1.1 Obtain the project brief | content | 1.1 | Start with the authorised brief Confirm scope, audience and platform Locate supporting production documents Do not rely on assumptions |
| 1.1 Gather required supporting documents | content | 1.1 | Briefs rarely stand alone GDD, technical specs and art direction guide work Check platform and marketplace requirements Find the documents before estimating |
| 1.1 Obtain organisational procedures | content | 1.1 | Procedures tell you how work must be done Version control and naming rules prevent rework Follow security, WHS and communication practices Use the studio… |
| 1.1 Validate and clarify requirements | content | 1.1 | Check documents against each other Identify gaps, conflicts and assumptions Ask targeted questions early Record decisions as production evidence |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 1.2 Identify asset categories | content | 1.2 | Assets are all production items used in the game List assets against creative and technical needs Include art, audio, code, UI and data Beta assets must... |
| 1.2 Translate creative requirements into assets | content | 1.2 | Creative direction drives asset choices Art style affects workload and performance Define quality level and acceptance criteria Assets must support the... |
| 1.2 Translate technical specifications into assets | content | 1.2 | Technical specs set asset limits Engines have capabilities and constraints Plan LODs, collision, shaders and memory Performance is an asset requirement |
| 1.2 Build an asset register | content | 1.2 | Use an asset register to control production Record owner, status, format and acceptance criteria Prioritise beta-critical assets Track dependencies and risks |
| 1.3 Discuss asset formats with required personnel | content | 1.3 | Formats affect whether assets can be used Talk to artists, programmers and technical leads Agree source and export formats Document decisions before... |
| 1.3 Discuss integration issues | content | 1.3 | Integration is where assets meet the engine Check scale, pivots, rigs, materials and data Resolve issues with the people who own them Use test imports early |
| 1.3 Agree technical budgets and engine constraints | content | 1.3 | Budgets protect performance Shader, polycount and memory limits must be discussed Engine features involve trade-offs Agree acceptance criteria with... |
| 1.3 Document integration decisions | content | 1.3 | Discussion must become production instructions Update the asset register and pipeline guide Record unresolved issues and owners Make the Definition of Done... |
| 1.4 Understand beta-version prototype purpose | content | 1.4 | A beta prototype is a quality gate Sequence work around what must be tested Prototype to prove game play, not every feature Define the beta success criteria |
| 1.4 Map dependencies and critical path | content | 1.4 | Sequence depends on task dependencies Critical path controls the earliest beta date High-risk systems must start early Avoid blocking downstream work |
| 1.4 Build the beta development sequence | content | 1.4 | Use milestones to structure the sequence Greybox, vertical slice, alpha and beta are different Integrate and test progressively Keep the game playable... |
| 1.4 Plan iteration and beta readiness | content | 1.4 | Game play needs iteration to find the fun Beta readiness needs clear gates Use feedback loops without losing scope Known exceptions must be controlled |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 1.5 Create a production schedule | content | 1.5 | A schedule converts sequence into dates Break work into tasks and milestones Estimate with the people doing the work Include dependencies, reviews and buffers |
| 1.5 Schedule asset production and integration | content | 1.5 | Assets need creation, review and integration time Dependencies must be visible Prioritise beta-critical items Do not leave integration to the end |
| 1.5 Schedule testing throughout production | content | 1.5 | Testing is part of production, not an afterthought Plan QA checkpoints and playtests Include bug fixing and retesting Use builds that match the test purpose |
| 1.5 Balance schedule, scope and quality | content | 1.5 | Schedules involve trade-offs Protect the critical path and beta goal Plan contingency without hiding risk Keep the schedule maintainable |
| 1.6 Define monitoring strategies | content | 1.6 | Monitoring compares progress with the schedule Use clear metrics and visible workflow Track tasks, milestones, risks and quality Monitoring must trigger action |
| 1.6 Monitor quality and Definition of Done | content | 1.6 | Progress is not just task completion Done means accepted against criteria Use QA checkpoints throughout the pipeline Track defects, rework and build health |
| 1.6 Manage variance, risk and critical path | content | 1.6 | Variance shows the gap between plan and reality Risks must be reviewed and acted on Critical path slippage threatens beta delivery Escalate early with options |
| 1.6 Communicate progress and update the schedule | content | 1.6 | Monitoring outputs must be communicated Use agreed reporting rhythms Update schedules when decisions change Keep evidence for production control |
| Knowledge check: Preparing to create game | knowledge check | 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 | Check your understanding Choose the best answer for each question Focus on workplace application Review the relevant slides if needed |
| Element summary: Prepare to create game | element summary | 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 | Preparation creates a controlled production baseline Assets must meet creative and technical requirements Beta sequencing, scheduling and monitoring reduce… |

**Assessment for this element:** E1 Knowledge Questions — Prepare to create game (knowledge questions); E1 Project — Plan the game production (project (submitted evidence)).

## E2 — Evaluate capability of game-engine software and tools

Indicative session time: ~688 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| Element 2: Evaluate capability of game-engine software and tools | element intro | — | Choose tools for the game, not the hype Compare engines, tools and constraints Match selection to play requirements Confirm decisions with the right people… |
| 2.1 Build a current view of industry-standard engines | content | 2.1 | Start with the market and project context Review engines, not just brand names Include commercial, open-source and specialist tools Check current licensing… |
| 2.1 Review the broader development toolchain | content | 2.1 | The engine is only part of production Review DCC, audio, version control and build tools Check integration, licensing and collaboration Match tools to team… |
| 2.1 Compare engine capabilities using evidence | content | 2.1 | Use criteria, not impressions Compare rendering, scripting and workflow Include performance and profiling tools Test claims against documentation Note… |
| 2.1 Review constraints, licences and industry conditions | content | 2.1 | Capability includes constraints Check hardware, licence and platform limits Consider privacy, copyright and classification Evaluate total production cost… |
| 2.2 Translate the game concept into requirements | content | 2.2 | Start from the specified game concept Define play, platform and production needs Separate must-have from nice-to-have Turn ideas into testable requirements… |
| 2.2 Assess play requirements against engine systems | content | 2.2 | Map gameplay to engine features Check physics, animation, camera and UI Assess scripting and debugging needs Consider multiplayer and AI early Prototype… |
| 2.2 Assess hardware constraints and performance fit | content | 2.2 | Hardware limits shape design choices Assess CPU, GPU, memory and storage Match fidelity to target platform Use profiling, not guesses Plan scalable quality… |
| 2.2 Assess asset integration and production pipeline | content | 2.2 | Assets must move cleanly into the engine Check formats, scale, materials and rigs Plan naming, folders and version control Test import and reimport early… |
| 2.3 Identify required personnel and their decision needs | content | 2.3 | Engine choice affects many roles Consult the right personnel early Use evidence each role can evaluate Resolve conflicting priorities Record decisions and… |
| 2.3 Discuss selection considerations clearly | content | 2.3 | Frame discussion around outcomes Explain trade-offs, not just preferences Use a structured agenda Capture questions and decisions Escalate unresolved risks |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 2.3 Confirm the choice will meet specified outcomes | content | 2.3 | Confirmation needs evidence Use proof of concept where risk is high Check outcomes against requirements Define acceptance criteria Record approval and… |
| 2.3 Manage disagreement, risk and change | content | 2.3 | Disagreement is normal in tool selection Use decision criteria to resolve conflict Document risks and mitigations Escalate issues that affect outcomes Keep… |
| 2.4 Make the final software and tool selection | content | 2.4 | Selection is a documented production decision Choose engine, version and core tools Link the choice to outcomes Define conditions and responsibilities… |
| 2.4 Select for creative, production and technical balance | content | 2.4 | Balance competing requirements Do not over-select for one discipline Protect schedule and maintainability Support the intended player experience Choose the… |
| 2.4 Lock in the asset and integration toolchain | content | 2.4 | Selection includes asset workflow rules Choose formats and import settings Define naming and folder conventions Plan version control for binary assets Test… |
| 2.4 Document, justify and prepare the selected environment | content | 2.4 | Document the final decision Justify against evidence and outcomes Set up the environment consistently Plan onboarding and validation Review if assumptions… |
| Knowledge check: evaluating engines and tools | knowledge check | 2.1, 2.2, 2.3, 2.4 | Test your engine selection judgement Choose the best answer for each question Focus on evidence, requirements and constraints |
| Element 2 summary | element summary | 2.1, 2.2, 2.3, 2.4 | Review a range of current engines and tools Assess against the specified game concept Consult required personnel using evidence Confirm outcomes, risks and… |

**Assessment for this element:** E2 Knowledge Questions — Evaluate game-engine software and tools (knowledge questions); E2 Project — Select the game engine and tools (project (submitted evidence)).

## E3 — Prepare required assets for a game

Indicative session time: ~688 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| Prepare required assets for a game | element intro | — | Shared environments prevent asset chaos Assets must match the engine and platform Create, acquire, customise, then test Good asset preparation protects… |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 3.1 Plan the shared asset environment | content | 3.1 | Start with the pipeline, not the folder Define source files, exports and engine files Separate working files from game-ready assets Document the rules... |
| 3.1 Choose version control for asset sharing | content | 3.1 | Game assets are often large binary files Use central repositories and file history Use locking where merge conflicts are likely Protect both code and... |
| 3.1 Organise folders, names and metadata | content | 3.1 | Use predictable folders Use consistent asset names Track status, owner and dependencies Make assets easy to find and safe to change |
| 3.1 Control access, reviews and project risk | content | 3.1 | Assign ownership and permissions Schedule asset reviews and integration tests Manage copyright, security and WHS risks Keep the shared environment auditable |
| 3.2 Read the game design into asset requirements | content | 3.2 | Start from gameplay and visual purpose Translate design into asset attributes Record measurable requirements Identify dependencies before production |
| 3.2 Review file formats and import attributes | content | 3.2 | Formats affect what data survives import Check mesh, UV, material and animation data Confirm units, axes and scale Use import presets where possible |
| 3.2 Review engine and hardware constraints | content | 3.2 | Engines enable features and impose limits Hardware affects memory, frame rate and input Review assets against the target platform Use measurement, not... |
| 3.2 Review assets through scheduled tests | content | 3.2 | Review in the engine, not only in art tools Use checklists and acceptance criteria Schedule testing across milestones Record defects and required changes |
| 3.3 Create visual assets in-house | content | 3.3 | Create from the asset brief Use blockout before final detail Keep source files and exports separate Check art, topology and gameplay purpose |
| 3.3 Acquire assets legally and strategically | content | 3.3 | Check licence before download or purchase Match style, scale and technical needs Record source, terms and attribution Budget time for adaptation and testing |
| 3.3 Prepare acquired assets for the project pipeline | content | 3.3 | Quarantine and inspect new assets Scan, organise and rename safely Convert only after preserving originals Test imports before approval |
| 3.3 Produce asset packages for integration | content | 3.3 | Deliver complete asset packages Include source, exports and review notes Commit changes with useful messages Make integration repeatable |
| 3.4 Customise scale, orientation and materials | content | 3.4 | Engine defaults are not universal Fix scale, pivots and axes Convert materials to project shaders Use presets to avoid repeated errors |
| 3.4 Optimise meshes, textures and memory | content | 3.4 | Optimisation depends on platform and camera Use LODs, compression and batching wisely Balance visual quality with performance Profile before and after changes |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 3.4 Configure collision, physics and interaction attributes | content | 3.4 | Visual shape is not gameplay shape Set collision for purpose and performance Add sockets, tags and components Test interactions in real gameplay scenes |
| 3.4 Validate customised assets in the chosen engine | content | 3.4 | Test in representative scenes Check builds, not only editor previews Track defects and approvals Update documentation when attributes change |
| Knowledge check: preparing game assets | knowledge check | 3.1, 3.2, 3.3, 3.4 | Check your understanding Choose the best answer for each question Focus on workplace asset preparation decisions |
| Element summary: prepare required assets | element summary | 3.1, 3.2, 3.3, 3.4 | Build a shared asset environment Review attributes against engine and platform Create or acquire assets with legal control Customise and validate assets… |

**Assessment for this element:** E3 Knowledge Questions — Prepare required assets (knowledge questions); E3 Project — Prepare and customise game assets (project (submitted evidence)).

## E4 — Use game-engine software and development tools

Indicative session time: ~551 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| Using the engine as a production tool | element intro | — | Build the playable prototype in-engine Connect code, assets and scenes Check gameplay against requirements Run the prototype as a presentation Confirm… |
| 4.1 Scripts, code and the engine workflow | content | 4.1 | Scripts turn assets into behaviour Engine code usually attaches to objects Use language and engine APIs correctly Plan before editing production files Keep… |
| 4.1 Creating and modifying scripts deliberately | content | 4.1 | Translate requirements into logic Use engine events and components Expose designer-tunable values Debug with console output and breakpoints Avoid… |
| 4.1 Combining assets into playable objects | content | 4.1 | Assets are files used by the project Import settings affect quality and performance Prefabs and blueprints support reuse Check scale, orientation, materials… |
| 4.1 Combining code and assets through reusable systems | content | 4.1 | Reusable objects reduce rework Connect scripts to assets through references Use version control and naming conventions Validate behaviour after every… |
| 4.2 Defining gameplay elements from requirements | content | 4.2 | Gameplay elements are playable systems Start from creative and technical requirements Convert the brief into acceptance checks Balance fun, clarity, scope… |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 4.2 Building core gameplay elements in-engine | content | 4.2 | Prototype the minimum playable loop Use engine systems for input, physics and UI Create feedback for player actions Check interactions in context Keep… |
| 4.2 Checking gameplay against technical constraints | content | 4.2 | Target hardware shapes design Profile before guessing Check frame rate, memory and input Optimise assets and scripts early Record risks and critical path items |
| 4.2 Quality checking gameplay elements | content | 4.2 | Check function, feel and fit Use test cases and playtest notes Compare results to acceptance checks Track defects and decisions Confirm the element is… |
| 4.3 Preparing to run the prototype as a presentation | content | 4.3 | A presentation build must be controlled Define the intended play sequence Use a clean scene and reliable start state Check assets, code and settings before… |
| 4.3 Testing the prototype in editor and build | content | 4.3 | Use play mode for fast iteration Use packaged builds for real validation Check console logs and runtime errors Test sequence start to finish Compare editor… |
| 4.3 Confirming sequences meet creative and production requirements | content | 4.3 | Check the sequence against the brief Review pacing, clarity and player feedback Confirm milestone scope Document known limitations Use stakeholder-ready… |
| 4.3 Confirming technical requirements and final readiness | content | 4.3 | Validate target platform and settings Check performance and asset integration Triage defects before presentation Freeze the build when ready Archive… |
| Knowledge check: engine tools and prototype readiness | knowledge check | 4.1, 4.2, 4.3 | Check your understanding Choose the best answer for each question Focus on workplace application Review the content if unsure |
| Element summary: using engine tools professionally | element summary | 4.1, 4.2, 4.3 | Create and modify scripts safely Combine assets into reusable systems Build and check gameplay elements Test editor and packaged versions Present a… |

**Assessment for this element:** E4 Knowledge Questions — Use game-engine software and development tools (knowledge questions); E4 Project — Build and run the game prototype (project (submitted evidence)).

## E5 — Evaluate game prototype

Indicative session time: ~964 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| Evaluate the game prototype | element intro | — | Prototype evaluation protects the project Feedback turns an idea into a playable product Evidence supports go, change or stop decisions Endorsement… |
| 5.1 Identify who must see the prototype | content | 5.1 | Required personnel are decision-makers and specialists Map each person to the feedback they can provide Prepare the build for their role and authority… |
| 5.1 Prepare the demonstration build | content | 5.1 | Show the intended gameplay loop State what is prototype and what is final Control technical risk before the session Use a clear demo script |
| 5.1 Run the demonstration professionally | content | 5.1 | Set context before asking for opinions Demonstrate, then let reviewers interact Separate observation from explanation Capture feedback in a traceable format |
| 5.1 Turn comments into usable feedback | content | 5.1 | Capture exact comments and evidence Classify feedback by type and severity Clarify ambiguous statements Protect confidentiality and IP |
| 5.2 Define evaluation criteria | content | 5.2 | Criteria must be agreed before judgement Include creative and user-friendly outcomes Link criteria to the brief and target player Use measurable evidence… |
| 5.2 Evaluate creativity | content | 5.2 | Creativity must serve the player experience Judge novelty, coherence and feasibility Compare against genre expectations Avoid confusing polish with creative… |
| 5.2 Evaluate user-friendliness | content | 5.2 | User-friendly means playable by the intended user Check goals, controls, feedback and recovery Observe behaviour, not only opinions Consider accessibility… |
| 5.2 Make the evaluation decision | content | 5.2 | Compare evidence against each criterion Identify pass, change, defer or stop decisions Consider engine and hardware constraints Document the reasoning |
| 5.3 Prepare for change discussions | content | 5.3 | Bring evidence, not just preferences Group feedback into change themes Separate must-have from nice-to-have Check impact on scope and schedule |
| 5.3 Negotiate required changes | content | 5.3 | Agree the problem before choosing the fix Compare options using criteria and constraints Record decisions and dissenting risks Maintain professional… |
| 5.3 Document agreed changes | content | 5.3 | Write changes as actionable work items Define owner, priority and acceptance criteria Update schedule, risks and dependencies Keep version history traceable |
| 5.3 Confirm shared agreement | content | 5.3 | Check that required personnel accept the plan Resolve conflicts before implementation Communicate decisions to the whole team Protect the milestone baseline |
| 5.4 Understand tests and user trials | content | 5.4 | Testing checks quality and function User trials observe real player experience Define goals before recruiting testers Prototype tests must be ethical and safe |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 5.4 Assist with test planning | content | 5.4 | Create scenarios and tasks Prepare builds, devices and data sheets Schedule around dependencies Assign roles for facilitation and observation |
| 5.4 Support test execution | content | 5.4 | Set up consistently for each participant Facilitate without coaching Record observations and defects accurately Escalate safety or technical issues quickly |
| 5.4 Manage test data and defects | content | 5.4 | Use consistent issue logging Attach evidence and reproduction steps Protect participant data Feed results back into the evaluation cycle |
| 5.5 Organise user trial feedback | content | 5.5 | Clean and categorise feedback before analysis Separate observations, comments and metrics Look for patterns across participants Link findings to evaluation... |
| 5.5 Interpret player feedback carefully | content | 5.5 | Players identify symptoms more than solutions Compare feedback with design intent Check sample limits and bias Use evidence to infer causes |
| 5.5 Evaluate technical and production implications | content | 5.5 | Feedback may reveal hidden technical risk Check engine, hardware and asset constraints Estimate impact before recommending changes Update risk and critical... |
| 5.5 Produce evaluation findings | content | 5.5 | Summarise findings against criteria Prioritise recommendations Identify what needs retesting Communicate confidence and limitations |
| 5.6 Understand endorsement | content | 5.6 | Endorsement is formal approval to proceed Required personnel must accept evidence and risks Approval may be conditional Record endorsement clearly |
| 5.6 Plan the transition to complete product | content | 5.6 | Prototype code and assets may not be production-ready Define production scope and milestones Resolve technical debt and pipeline risks Protect the creative core |
| 5.6 Build the production roadmap | content | 5.6 | Break the complete product into milestones Schedule development and testing together Manage dependencies and critical path Plan for target hardware and... |
| 5.6 Maintain endorsement through development | content | 5.6 | Use endorsement as a baseline Track conditions, risks and change requests Keep stakeholders informed Validate the complete product against the prototype promise |
| Knowledge check: evaluating a game prototype | knowledge check | 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 | Check your understanding Choose the best answer for each question Focus on evidence-based prototype evaluation |
| Element summary: evaluate game prototype | element summary | 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 | Demonstrate the prototype to required personnel Evaluate against creative, usability and technical criteria Agree and document required changes Use trials... |

**Assessment for this element:** E5 Knowledge Questions — Evaluate game prototype (knowledge questions); E5 Project — Evaluate and endorse the prototype (project (submitted evidence)).

## E6 — Transform prototype into final publication

Indicative session time: ~826 minutes of learner activity (indicative — from the volume-of-learning allocation; confirm against the group timetable).

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| From prototype to publication | element intro | — | Prototype evidence becomes release decisions Final builds must match the specification QA checks protect navigation, stability and player flow Sign-off and… |
| 6.1 Interpreting user trial evidence | content | 6.1 | Separate evidence from opinion Prioritise issues by player impact Record defects and enhancement requests Protect participant privacy and consent |
| 6.1 Converting feedback into change requests | content | 6.1 | Write clear change requests Link each change to evidence Check scope, schedule and risk Use version control before editing |
| 6.1 Implementing and retesting changes | content | 6.1 | Fix in small controlled increments Retest the original issue Run regression checks nearby Update documentation and build notes |
| 6.1 Closing user-trial changes | content | 6.1 | Confirm acceptance criteria Mark status accurately Keep evidence for sign-off Avoid unapproved feature creep |
| 6.2 Reading the specification before integration | content | 6.2 | Use the GDD as the integration baseline Check required systems, assets and platforms Confirm acceptance criteria Identify dependencies before merging |
| 6.2 Integrating assets and systems | content | 6.2 | Import assets in approved formats Connect scripts, prefabs and scenes Resolve dependencies and missing references Optimise for target hardware |
| 6.2 Integrating gameplay, UI and audio | content | 6.2 | Connect mechanics to player feedback Check UI, input and accessibility options Balance audio with gameplay events Validate save, progress and failure states |
| 6.2 Confirming feature completeness | content | 6.2 | Feature-complete means all required systems are present Alpha is not final polish Track gaps and dependencies Do not hide missing specification items |
| 6.3 Planning final navigation checks | content | 6.3 | Use the navigation design as the test source Check menus and in-game sequences Look for dead ends and softlocks Test on the target input devices |
| 6.3 Checking gameplay sequences | content | 6.3 | Verify trigger order and state changes Test checkpoints, objectives and cutscenes Probe for progression breaks Confirm recovery from failure states |

| Slide / topic | Type | PCs covered | Indicative content |
|---|---|---|---|
| 6.3 Final QA passes and defect triage | content | 6.3 | Run smoke, functional and regression tests Prioritise crashes and progression breaks Record build numbers and evidence Escalate release risks early |
| 6.3 Confirming conformity and readiness | content | 6.3 | Compare results to navigation design Update the test report List unresolved known issues Recommend proceed, fix or defer |
| 6.4 Understanding final sign-off | content | 6.4 | Sign-off is a formal release gate Identify who must approve Provide evidence, not assumptions Record decisions and conditions |
| 6.4 Preparing for the sign-off review | content | 6.4 | Freeze the candidate build Prepare release evidence Brief reviewers on scope and risks Use clear acceptance criteria |
| 6.4 Managing feedback during sign-off | content | 6.4 | Classify review comments Fix only approved blockers Document deferrals and conditions Retest after any sign-off change |
| 6.4 Recording final approval | content | 6.4 | Capture who approved what build Store sign-off evidence securely Communicate release status Do not release without authority |
| 6.5 Planning the final publication package | content | 6.5 | Follow organisational distribution procedure Package for the target platform Include required documentation Use versioned, approved release files |
| 6.5 Building for engines and platforms | content | 6.5 | Select the correct target platform Check engine packaging constraints Test the packaged build, not only the editor Resolve missing assets and plugins |
| 6.5 Documentation, licences and compliance materials | content | 6.5 | Include release notes and instructions Confirm asset licences and credits Prepare classification or store materials if needed Protect privacy and player data |
| 6.5 Distributing and archiving the final publication | content | 6.5 | Upload through approved channels Verify download, install and launch Archive the exact approved build Prepare for post-release support |
| Knowledge check: final publication readiness | knowledge check | 6.1, 6.2, 6.3, 6.4, 6.5 | Check your understanding Choose the best answer for each question Focus on workplace release decisions |
| Element summary: transforming prototype into publication | element summary | 6.1, 6.2, 6.3, 6.4, 6.5 | Use trial evidence to drive controlled changes Integrate against the approved specification Confirm navigation and sequences through QA Obtain traceable… |

**Assessment for this element:** E6 Knowledge Questions — Transform prototype into final publication (knowledge questions); E6 Project — Deliver the final publication (project (submitted evidence)).

DRAFT session plan — confirm timings and LMS resources before delivery.