International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 11 November-2026 | Impact Factor: 3.5



Escape the Museum: A Unity-Based 3D Project Game

TARUN HN[1] , NIKHIL ND[1] , MARUTHI BA[1] , DIWAKAR SV[1] , Prof. Ashwini Singh S[2]

[1]Department of Computer Science and Engineering,
Akshaya Institute of Technology, Tumkur
[2]Assistant Professor, Department of Computer Science and Engineering,
Akshaya Institute of Technology, Tumkur
Email: shobha.ash1@gmail.com

How to Cite this Article:
HN, T., ND, N., BA, M. & SV, D. (2026). Escape the Museum: A Unity-Based 3D Project Game. International Journal of Creative and Open Research in Engineering and Management, 02(11), 1-9. https://doi.org/10.55041/ijcope.v2i11.178

License:
This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.
© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.





https://doi.org/10.55041/ijcope.v2i11.178

***Abstract*—**Escape rooms ask players to observe, connect clues, and solve puzzles within a limited time in order to leave a locked space. This paper presents the design of Escape the Museum, a single-player 3D room-escape game developed in the Unity game engine, in which the player explores a series of museum rooms, collects items, solves puzzles, and unlocks the way out. The prototype brings together a first-person player controller, an object-interaction system, an inventory, a puzzle and door-lock manager, a game-state and timer module, and a user interface with audio feedback. The paper describes the architecture, implementation approach, and a repeatable test plan, and it states the limitations of the design. No measured frame rate, load time, or player-study results are claimed, because a controlled evaluation has not yet been completed.

Index Terms*—Escape room, serious game, Unity, game design, puzzle design, C#, first-person interaction.*

I. INTRODUCTION

An escape room places players inside a themed, locked space and asks them to find clues, combine objects, and solve linked puzzles before time runs out. Nicholson's survey of escape-room facilities shows how such spaces are built around story, puzzles, and teamwork [1], and a systematic review of escape rooms in education reports their growing use for learning and engagement [2]. Because a physical room is costly to build and cannot be played remotely, a digital version offers a convenient alternative that can be played on an ordinary computer.

Game design provides the vocabulary for such a system. The MDA framework separates a game into mechanics, dynamics, and aesthetics [3], and Schell describes game design as a set of lenses for checking whether an experience is clear, challenging, and rewarding [4], and Fullerton treats playtesting and iteration as central to good design [9]. Csikszentmihalyi's concept of flow suggests that challenge should match the player's skill, which is a useful guide when ordering puzzles from easy to hard [5]. Brown and Cairns describe immersion as a staged experience in which controls, attention, and atmosphere all matter [6]. Environmental storytelling and consistency are also recognized as important in escape-room design [7].

This project, Escape the Museum, is a prototype game built with the Unity engine [8]. The player is locked inside a museum and must move through its rooms, inspect exhibits, collect items, and solve puzzles to unlock each door until the exit is reached. A C# script layer handles player movement, interaction, inventory, puzzles, the countdown timer, and the user interface. The game is a learning and entertainment prototype; it does not claim to be a validated educational tool, and no player study has been carried out.

II. OBJECTIVES

- Build a 3D museum environment made of several connected rooms in the Unity engine.
- Provide a first-person player controller with movement, camera look, and collision with walls and exhibits.
- Allow the player to interact with objects, pick up items, and examine clues.
- Maintain an inventory in which collected items can be viewed, selected, and used.
- Implement puzzles whose solutions unlock doors, containers, or the next room.
- Track game state, including the current room, solved puzzles, and collected items.
- Provide a countdown timer and a win or lose condition for each play session.
- Offer an optional hint system so that a stuck player can continue.
- Provide a user interface with a main menu, an in-game display, a pause menu, and an end screen.
- Add sound effects and background audio to support the museum atmosphere.
- Keep puzzle and room settings configurable in the Unity inspector so that content can change without rewriting code.
- Define a repeatable test plan and state the limitations of the design.

III. REQUIREMENT ANALYSIS

Before implementation, the needs of the game were written as functional requirements (what the game must do) and non-functional requirements (how well it must do it). The quality terms follow the general vocabulary of ISO/IEC 25010 [10]. Table I lists them.

ID	Requirement	Type
FR1	The player can move and look around the museum.	Functional
FR2	The player can interact with objects within reach.	Functional
FR3	The player can pick up items and store them.	Functional
FR4	The player can use a selected item on a lock or puzzle.	Functional
FR5	The game checks puzzle solutions and gives feedback.	Functional
FR6	A door opens only when its condition is met.	Functional
FR7	A timer runs and the game ends in a win or a loss.	Functional
FR8	The player can request a hint.	Functional
FR9	The player can pause, resume, and restart.	Functional
NFR1	Prompts and messages must be clear and readable.	Usability
NFR2	The game should run smoothly on the test computer.	Performance
NFR3	Puzzle and room settings are editable without code changes.	Maintainability
NFR4	The player must never be stuck without a way to continue.	Reliability
NFR5	The game is delivered as a Windows build.	Portability

Table I. Functional and non-functional requirements.

Requirement NFR4 is important in escape games. If a needed item can be lost, destroyed, or left behind in a locked room, the game becomes impossible to finish. The design therefore keeps every key item in the



inventory until the end of the session and never removes an item that a later puzzle needs.

IV. RELATED WORK AND COMPARISON

Escape rooms began as live, physical attractions in which a team solves linked puzzles in a staged room [1]. Research on their educational use reports that they can promote engagement and teamwork, although the evidence is still developing [2]. Game-design texts treat the escape room as a combination of exploration, puzzle solving, and narrative [4], [9]. A digital version keeps the puzzle structure but changes how the space is built and played. Table II compares the two in general terms.

Aspect	Physical escape room	Digital (this project)
Set-up	Needs a real room, props, and staff.	Built once in the engine; no physical props.
Access	Visitor must travel to the venue.	Played on a computer.
Replay	Limited by staff and room resets.	Restarts instantly.
Teamwork	Strong face-to-face teamwork.	Single-player in this version.
Changes	Costly to rebuild puzzles.	Edit settings or scenes.
Touch	Real objects and locks.	Simulated interaction.

Table II. General comparison of physical and digital escape rooms.

The comparison is qualitative and describes general differences, not measured results. Escape the Museum sits on the digital side and trades physical touch and team play for easy access, instant restart, and flexible content.

V. SYSTEM REQUIREMENTS AND TOOLS

The prototype targets a Windows computer. The Unity engine supplies rendering, physics, audio, and the scene editor, while gameplay logic is written in C#. Three-dimensional models, textures, and sounds form the content of the museum. Table III lists the components.

Component	Purpose in the prototype
Unity engine	Scene editor, rendering, physics, audio, and build tools.
C# scripts	Player control, interaction, inventory, puzzles, and game state.
Unity physics (raycasts, colliders)	Detects walls and identifies the object the player is looking at.
Unity UI / TextMeshPro	Menus, heads-up display, inventory panel, and messages.
Audio sources and mixer	Background music, footsteps, and puzzle feedback sounds.
3D models and textures	Walls, exhibits, furniture, and props that form the museum.
Code editor (Visual Studio / VS Code)	Writing and debugging the C# scripts.

Table III. Components of Escape the Museum.

VI. PROPOSED SYSTEM ARCHITECTURE

The game follows a four-layer architecture, shown in Fig. 1. Each layer has one responsibility and talks mainly to the layer next to it, which keeps the code easy to test and extend.

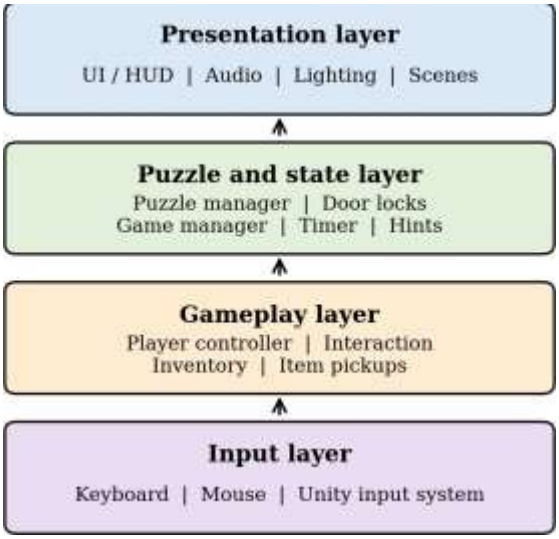

Fig. 1. Layered architecture of Escape the Museum.

A. Input layer

The input layer reads keyboard and mouse actions, such as movement keys, mouse look, the interact key, and the pause key. It performs no game logic and only passes the actions upward.

B. Gameplay layer

The gameplay layer turns input into actions in the world. The player controller moves the character and camera, the interaction module finds the object in front of the player, and the inventory stores collected items. Item pickups are separate objects that the interaction module can recognize.

C. Puzzle and state layer

This layer holds the rules of the game. The puzzle manager checks whether an attempted solution is correct, door-lock scripts decide whether a door may open, the game manager keeps track of the current room and progress, and the timer and hint modules support the win or lose condition.

D. Presentation layer

The presentation layer shows the result to the player: the user interface, audio feedback, lighting, and scene loading. Messages such as “locked” or “puzzle solved” are produced here after the state layer reports an outcome.

E. Data flow

A typical action passes through the layers in the following steps.

- The player presses a key or moves the mouse.
- The input layer passes the action to the player controller or interaction module.
- The interaction module casts a ray from the camera and identifies the object hit.
- The object is handled as a pickup, a door, or a puzzle element.
- The puzzle and state layer checks the rule and updates the game state.
- The presentation layer displays a message, plays a sound, and opens the door if allowed.



VII. GAME DESIGN

A. Game flow

A play session follows the flow in Fig. 2. The player starts from the main menu, enters the first room, explores to find items and clues, and works on the puzzle. When the puzzle is solved, the exit door unlocks and the player moves on. After the last room, the win screen is shown. At any moment, if the timer reaches zero, the lose screen is shown.

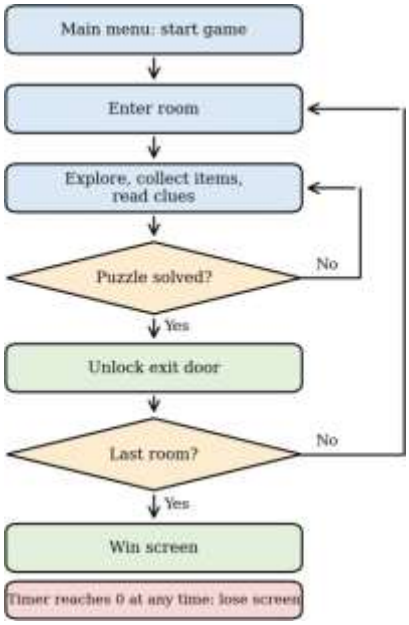

Fig. 2. Flow of a play session.

B. Game states

The game manager keeps the game in exactly one state at a time, which prevents conflicts such as the player moving while a menu is open. Table IV lists the states.

State	Meaning	Can move to
Menu	Main menu is shown.	Playing
Playing	Player explores; timer runs.	Paused, Win, Lose
Paused	Game frozen; pause menu shown.	Playing, Menu
Win	Exit reached in time.	Menu, Playing
Lose	Timer reached zero.	Menu, Playing

Table IV. Game states and allowed transitions.

C. Controls

A small, familiar set of controls keeps the learning cost low. Table V shows an example mapping that should be matched to the final build.

Action	Input (example)
Move	W, A, S, D keys
Look around	Mouse movement
Interact / pick up	E key or left mouse button
Open inventory	I key
Request hint	H key
Pause	Esc key

Table V. Example control scheme (edit to match the final game).

D. Puzzle design principles

The puzzles follow a few simple principles drawn from the game-design literature [4], [5], [9].

- Start easy. The first room teaches movement, pickup, and the use of a key before any harder puzzle appears.
- Increase difficulty gradually so that challenge keeps pace with player skill [5].
- Make every clue findable. A clue should be visible in the room, never hidden by a bug or by chance.
- Give clear feedback. A solved puzzle, a wrong attempt, and a locked door each give a different message and sound.
- Keep the story consistent. Clues should fit the museum theme so that the setting explains the puzzles [7].
- Avoid dead ends. No action may remove an item that is needed later.

E. Win and lose rules

The player wins by opening the final exit before the timer reaches zero. The player loses when the timer reaches zero first. In both cases the end screen lets the player return to the menu or restart from the first room.

VIII. IMPLEMENTATION

A. Player controller

The player is a first-person character. Movement is applied from keyboard input, and the camera rotates with mouse movement within a limited vertical angle. Colliders on walls and exhibits stop the player from passing through the scenery. Movement and look can be disabled while a menu or puzzle panel is open.

B. Interaction system

Each frame, a ray is cast from the center of the camera for a short reach distance. When the ray hits an object marked as interactive, a prompt appears on screen, and pressing the interact key calls that object's handler. Using one common interaction path means new items and puzzles can be added without changing the player script.

C. Inventory

Picked-up items are removed from the scene and added to an inventory list. The player can open the inventory, select an item, and use it on a door or puzzle slot. A key item carries an identifier, and a lock accepts only the item with the matching identifier.

D. Rooms and puzzles

The museum is divided into rooms that are connected by doors. Each room contains clues and one or more puzzles; solving them unlocks the exit to the next room. Table VI shows the room plan used by the prototype. Room names and puzzle types are examples that should be matched to the final build.

Room	Theme	Puzzle type (example)
1	Entrance hall	Find a key hidden among exhibits.
2	Gallery	Arrange items in the order shown by a clue.
3	Artifact room	Enter a code found in earlier rooms.
4	Final vault / exit	Combine collected items to open the exit.

Table VI. Example room plan (edit to match the final game).

E. Game manager, timer, and hints

A game manager object stores the current room, the list of solved puzzles, and the remaining time. When the timer reaches zero before the exit is opened, the game shows a lose screen; when the exit door opens, it shows a win screen. The hint module displays a short message for the current room when the player requests it, and the number of hints can be limited.



F. User interface and audio

The main menu lets the player start or quit the game. During play, the display shows the interaction prompt, the timer, and the selected inventory item. A pause menu allows resume, restart, and quit. Background music sets the mood of the museum, and short sound effects confirm pickups, locked doors, and solved puzzles.

G. Interaction logic

The handler below outlines how one interaction is processed.

```
on_interact():
  if not ray_hits(obj): return
  if obj is Pickup:
    inventory.add(obj); obj.hide()
  elif obj is Door:
    if obj.unlocked or inventory.has(obj.key):
      obj.open()
    else: ui.show("Locked")
  elif obj is PuzzleSlot:
    puzzle.try(obj, inventory.selected)
  if puzzle.room_solved():
    room.unlock_exit()
```

Algorithm 1. Outline of the interaction handler.

H. Configuration and build

Puzzle answers, key identifiers, the time limit, and the number of hints are exposed as fields in the Unity inspector, so a designer can change them without editing code. The finished project is built as a standalone Windows application from the Unity build settings.

I. Module summary

The scripts are kept small, each with one job. Table VII summarizes the main modules; the names are illustrative and should match the final project.

Module	Responsibility
PlayerController	Movement, camera look, and collision handling.
InteractionHandler	Casts the ray and calls the object's handler.
Inventory	Stores, lists, and selects collected items.
Pickup	Marks an item that can be collected.
DoorLock	Holds the key identifier and opens the door.
PuzzleManager	Checks solutions and tracks solved puzzles.
GameManager	Holds the game state, current room, and progress.
TimerController	Counts down and triggers the lose state.
HintManager	Shows the hint for the current room.
UIManager	Menus, prompts, timer display, and messages.
AudioManager	Plays music and sound effects.

Table VII. Main modules and their responsibilities.

J. Puzzle check and timer logic

Two further outlines show the puzzle check and the timer update.

```
try(slot, item):
  if slot.solved: return
  if item.id == slot.expected_id:
    slot.mark_solved()
    audio.play(OK); ui.show("Solved")
  else:
    audio.play(WRONG); ui.show("No fit")

update(dt):
  if state != Playing: return
  time_left = time_left - dt
  if time_left <= 0: game.lose()
```

Algorithm 2. Puzzle check and timer update.

K. Project organization

The Unity project is organized into folders for scenes, scripts, prefabs, materials, audio, and user-interface assets. Reusable objects such as doors, keys, and pickups are saved as prefabs, so a new room can be assembled by placing prefabs and setting their fields. Separate scenes can be used for the main menu and the museum, with the game manager carrying the state between them.

L. Performance considerations

A room-based game has a small, enclosed play area, so ordinary optimization practices suit it. These include baked lighting in place of many real-time lights, occlusion culling for rooms that cannot be seen, reuse of materials, compressed textures, and low-polygon props. The design aim is smooth play on the test computer. The final report should record which of these practices were actually applied and measure the result as described in Section IX.

IX. TESTING AND OBSERVATIONS

Because a controlled test procedure and player study have not yet been carried out, this section defines the test environment, test cases, and measurements, and reports no numerical results. Every item is marked pending; none is reported as passed. Results should be added only after each test has been run and observed in the game.

A. Test environment

Tests are to be run on a single Windows computer using the standalone build. The Unity version, operating system, processor, graphics card, and memory should be recorded with the results.

B. Test cases

ID	Procedure	Expected evidence	Status
T1	Move and look around in the first room.	Smooth movement and camera control.	Pending
T2	Walk into walls and exhibits.	Player is blocked by colliders.	Pending
T3	Pick up an item.	Item leaves scene and appears in inventory.	Pending
T4	Try a locked door without the key.	"Locked" message; door stays closed.	Pending
T5	Use the correct key on the door.	Door opens.	Pending
T6	Enter a wrong puzzle solution.	Solution rejected; state unchanged.	Pending
T7	Enter the correct puzzle solution.	Puzzle marked solved; exit unlocks.	Pending
T8	Let the timer reach zero.	Lose screen is shown.	Pending
T9	Open the final exit.	Win screen is shown.	Pending
T10	Pause, resume, and restart.	Game state and timer behave correctly.	Pending
T11	Move between rooms with items held.	Inventory and progress are preserved.	Pending

Table VIII. Planned functional test cases (status: pending).

C. Measurements

Table IX defines the measurements to be collected after the functional tests pass. The definitions allow results to be reproduced; no values are claimed.

Measure	Definition	How collected
Frame rate	Frames per second during play.	Unity stats / profiler.
Load time	Time to start the game and change rooms.	Stopwatch or timestamps.
Memory use	Memory used by the running build.	Profiler / task manager.
Completion time	Time a tester needs to escape.	Timed playtest.
Defects found	Bugs seen during playtests.	Tester log.

Table IX. Planned measurements (no values reported).



D. Recording observations

For each test, the tester should record the action taken, what the game did, and whether this matched the expected result. A failed test should be recorded with its cause, such as a missing collider, a wrong key identifier, or a display fault, rather than repeated until it passes.

E. Interpretation

A passing test shows only that the behavior occurred under that test condition. It does not show that the game is enjoyable, balanced, or suitable for every player, and results from one computer should not be generalized to others.

X. DISCUSSION

A. Value of the integrated design

The main value of the prototype is that exploration, item collection, puzzles, and a timed goal are combined in one playable experience. Because every interactive object follows the same interaction path, a designer can add a new room or puzzle by placing objects and setting fields rather than rewriting code.

B. Puzzle difficulty and pacing

Puzzle order affects how the game feels. Following the idea of flow [5], early rooms should be simple enough to teach the controls, and later rooms should demand more observation and combination of clues. The time limit and the hint option should be tuned together so that the game stays challenging without becoming frustrating.

C. Clarity of clues

In escape rooms, players quickly lose interest when they cannot tell what is interactive or what a clue means [7]. The interaction prompt and the short messages for locked doors are intended to reduce this confusion, although their effect has not yet been tested with players.

D. Digital and physical escape rooms

A digital room can be restarted instantly, played alone, and changed without building new props, but it lacks the physical touch and face-to-face teamwork of a real room [1], [2]. The prototype is therefore a single-player experience and does not attempt to replace physical escape rooms.

E. Threats to validity

Planned tests use one computer and a small set of testers, so results will describe that setup only. Frame rate and load time depend on the hardware, the graphics settings, and the size of the assets, and any future measurements should state these details.

F. Risk analysis

Table X lists the main risks identified for the prototype and a way to reduce each.

Risk	Effect	Mitigation
Needed item lost	Game cannot be finished.	Never remove key items; test full route.
Puzzle too hard	Player gives up.	Add hints; tune with playtests.
Unclear clue	Player is confused.	Test clue visibility and wording.
Missing collider	Player leaves the map.	Check all walls in test T2.
Low frame rate	Poor experience.	Optimize lights and assets.
Loud audio	Annoys the player.	Balance levels; volume setting.

Table X. Risks and mitigation.

XI. LIMITATIONS

- The game is single-player and has a limited number of rooms and puzzles.
- It has been designed for a desktop computer and has not been tested on other platforms.
- Puzzle difficulty and the time limit have not been tuned with real players.
- No accessibility options, such as subtitles or color-blind modes, are included in this version.
- Progress is not saved between sessions.
- No measured performance or player-study results are claimed in this paper.

XII. FUTURE ENHANCEMENTS

A. Evaluation

The first priority is to run the planned tests and then a small playtest, reporting completion time, hints used, and player feedback under a defined procedure.

B. More content

Additional rooms, more varied puzzles, and a stronger storyline can extend play time. Environmental storytelling can be improved by placing clues so that the setting itself explains the narrative [7].

C. Save system and settings

A save and load feature, adjustable difficulty, and settings for volume, controls, and graphics quality would improve usability.

D. Accessibility

Subtitles, adjustable text size, color-blind friendly clues, and alternative controls would allow more players to enjoy the game.

E. Multiplayer and VR

A cooperative multiplayer mode would bring the game closer to a real escape room, and a virtual-reality version could increase immersion [6]. Each would require new design and testing and should be described only after it has been implemented.

XIII. CONCLUSION

Escape the Museum presents a prototype design for a Unity-based room-escape game in which the player explores museum rooms, collects items, solves puzzles, and races a timer to reach the exit. The design is deliberately simple, and its limits are stated openly: it is a single-player prototype with a limited number of rooms and has not yet been measured or tested with players. Completing the planned tests, tuning puzzle difficulty, and adding more content are the next steps toward a polished game.

REFERENCES

[1] S. Nicholson, “Peeking behind the locked door: A survey of escape room facilities,” white paper, 2015. [Online]. Available: http://scottnicholson.com/pubs/erfacwhite.pdf

[2] A. Veldkamp, L. van de Grint, M. P. J. Knippels, and W. R. van Joolingen, “Escape education: A systematic review on escape rooms in education,” Educ. Res. Rev., vol. 31, Art. no. 100364, 2020.

[3] R. Hunicke, M. LeBlanc, and R. Zubek, “MDA: A formal approach to game design and game research,” in Proc. AAAI Workshop Challenges Game AI, 2004, pp. 1–5.

[4] J. Schell, The Art of Game Design: A Book of Lenses, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.

[5] M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience. New York, NY, USA: Harper & Row, 1990.

[6] E. Brown and P. Cairns, “A grounded investigation of game immersion,” in Proc. CHI Extended Abstracts Human Factors Comput. Syst., 2004, pp. 1297–1300.

[7] S. Nicholson, “Ask why: Creating a better player experience through environmental storytelling and consistency in escape room design,” in Proc. Meaningful Play, 2016.

[8] Unity Technologies, “Unity User Manual,” Unity Documentation. [Online]. Available: https://docs.unity3d.com/Manual/



[9] T. Fullerton, Game Design Workshop: A Playcentric Approach to Creating Innovative Games, 3rd ed. Boca Raton, FL, USA: CRC Press, 2014.

[10] Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models, ISO/IEC Standard 25010, 2011.