Artificial Intelligence

Coursework 02: Intelligent Agent Artefact

Student Details

Student ID: 25026764

Module: Artificial Intelligence

Assignment: Coursework 02

Submission Type: Individual submission approved by

module lecturer

Declaration

Module: CS6P05ES **Deadline: 04/27/2026**

Module Leader: Mr. Chameen Sandeepa **Student ID: 25026764**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

(signature:)____yasaspasindufernando@gmail.com_____ Date: 05/07/2026

This header sheet should be attached to the work you submit. No work will be accepted without it.

Extracts from University *Regulations* on Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples…

(i) Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.

(ii) Falsifying data in experimental results.

(iii) Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.

(iv) Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.

(v) Collusion to present joint work as the work solely of one individual.

(vi) Plagiarism, where the work or ideas of another are presented as the candidate's own.

(vii) Other conduct calculated to secure an advantage on assessment.

(viii) Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. "$e = mc^2$ (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

Acknowledgement

The author would like to acknowledge the guidance and support provided by the module lecturer during the development of this coursework artefact. The feedback and direction received during the project helped clarify the expectations for the AI artefact, documentation, and final preparation.

The author also acknowledges the use of online technical documentation, tutorials, Unity resources, and asset sources that supported the learning and development process. Relevant external sources are referenced in the report where appropriate.

This acknowledgement is included to recognise academic guidance, learning support, and external resources used during the preparation of the project.

Abstract

The coursework artefact, Scout vs Wolf: Intelligent Agent Survival Game, is a Unity 2D intelligent-agent game based on a scout survival scenario. The scout must stay alive by managing hunger, water, and energy while responding to predators, weather, night conditions, resources, shelter, exit availability, and rescue events. The artefact demonstrates perception, decision making, searching, memory, and state-based behaviour through C# scripts such as ScoutAI, ScoutBrain, ScoutPerception, ScoutMemory, and related predator logic. Real-world physics indicators are represented through range limits, distance checks, stat decay over time, speed modifiers, and collider-based interactions. The game supports alternative outcomes, including exit success, helicopter rescue, starvation, dehydration, exhaustion, and predator capture. As the scout's behaviour is calculated from runtime events and internal status rather than a fixed sequence, the project demonstrates non-scripted gameplay suitable for intelligent agent evaluation.

Table of Contents

List of Figures

List of Tables

Abbreviations

AI - Artificial Intelligence

BT - Behaviour Tree

FSM - Finite State Machine

P.E.A.S - Performance, Environment, Actuators, Sensors

NPC - Non-Player Character

UI - User Interface

UX - User Experience

HUD - Heads-Up Display

QA - Quality Assurance

2D - Two-Dimensional

C# - C Sharp

PNG - Portable Network Graphics

PDF - Portable Document Format

HD - High Definition

1. Introduction

Intelligent agents are systems that observe their environment, make decisions, and act towards a goal under changing conditions. In this project, the intelligent agent is represented by a scout in a 2D Unity survival environment. The scout does not simply move along a fixed animation path. Instead, it senses nearby threats and resources, updates internal survival needs, and chooses actions based on current priorities.

Games are useful for demonstrating artificial intelligence because agent behaviour can be seen directly. State changes, danger response, resource search, and environmental reactions are visible during gameplay, which makes the system easier to evaluate than a text-only demonstration.

The scout survival scenario is suitable for this module because it includes uncertainty, competing goals, and multiple possible outcomes. The scout may need food, water, shelter, rescue, or escape at different times. This report explains how those requirements were implemented in Unity and how the completed artefact demonstrates intelligent-agent behaviour.

This makes the project suitable for intelligent-agent coursework because the agent has a clear environment, measurable goals, limited sensing, and visible actions. The design allows the assessor to connect theory, such as perception and state transitions, with behaviour that can be observed during gameplay.

2. Project Motivation

Many teams selected chatbot-based projects for the coursework. This project took a different direction by focusing on a game-based intelligent agent. The reason for this choice was academic as well as practical: a game can visually demonstrate AI behaviour through movement, sensing, state changes, and direct interaction with the environment.

The author also has a long-term interest in game development, and this coursework provided a suitable opportunity to connect that interest with artificial intelligence concepts. The project was therefore designed not only as a playable 2D survival game, but also as a clear AI demonstration that could be assessed through visible behaviour.

From an AI perspective, the game format allowed the project to show perception limits, predator response, survival decisions, and alternative outcomes in a more concrete way. Instead of presenting a scripted sequence, the artefact aims to show how an agent can react to events as they occur during runtime.

A chatbot can demonstrate language processing, but it often hides the decision process behind text responses. This project selected a game-based artefact because movement, danger, search behaviour, and survival pressure can be seen on screen. That made it a better fit for showing an intelligent agent that observes a changing world and acts within it.

3. Problem Description

The game models a jungle survival problem where a scout is lost and must stay alive long enough to escape or be rescued. Food and water are limited, and the scout's hunger, water, and energy values decrease over time. These internal values create pressure and force the agent to make decisions rather than move randomly.

Predator entities such as the wolf and lion create direct danger through detection, chase, and catch behaviour. Environmental conditions, especially rain and night, also affect sensing and movement efficiency. This means the scout must act under uncertainty instead of assuming that all resources and threats are always known.

A key requirement is that the behaviour should not be fixed in advance. The scout must respond to events such as hunger, thirst, predator distance, weather changes, energy level, exit availability, and rescue availability. These events allow the same game system to produce different endings across different runs.

4. Aim and Objectives

Aim

To develop a Unity 2D intelligent-agent survival game that demonstrates practical AI concepts through dynamic, state-based behaviour in a changing environment.

Objectives

- To implement an autonomous scout agent that updates behaviour during gameplay.
- To implement state-based behaviour that responds to internal and external events.
- To implement perception using vision, hearing, and short-range resource detection.
- To implement decision making through priority-driven behaviour logic.
- To implement searching behaviour for food, water, shelter, and exit targets.
- To implement memory-based learning by experience for resource and danger locations.
- To include real-world physics indicators such as range limits, stat decay, speed modifiers, and collision checks.
- To support alternative outcomes, including exit success, rescue success, and multiple loss conditions.
- To produce technical documentation, diagrams, and test evidence aligned with coursework expectations.

5. Game Overview

Scout vs Wolf: Intelligent Agent Survival Game is a 2D Unity survival simulation focused on autonomous AI behaviour. The intended audience includes module assessors and students who need to evaluate how intelligent-agent concepts have been implemented in a practical artefact.

The main entities include the scout agent (ScoutAI and ScoutBrain), predator agents (WolfAI, LionAI, and WolfBrain), environmental and outcome control (GameManager), and resource pickups (ItemPickup). During gameplay, the scout explores the map, senses nearby resources and threats, manages hunger, water, and energy, and selects actions such as searching, fleeing, sheltering, or moving towards the exit.

The game includes more than one ending. Win conditions include reaching the exit trigger or completing helicopter rescue. Lose conditions include starvation, dehydration, exhaustion, wolf capture, and lion capture. This design supports AI assessment because the scout's behaviour is observable, state-driven, and affected by runtime events.

A separate Game User Manual is provided as Appendix F and as a supporting document. It explains how to launch the build, select game modes, use manual controls, understand the HUD, and interpret win/lose outcomes.

6. Requirement Analysis

6.1 Functional Requirements

Table 1 summarises the main functional requirements implemented in the game.

ID	Functional Requirement	Evidence in System
FR1	Manage scout hunger, water, and energy	ScoutAI.UpdateStats()
FR2	Restore stats through food and water pickups	ItemPickup.Consume()
FR3	Detect, chase, and catch through predator agents	WolfBrain, WolfAI, LionAI
FR4	Change observable behaviour mode at runtime based on events and thresholds	ScoutBrain.StateLabel, mirrored ScoutAI.State

FR5	Apply weather and night effects	GameManager, ScoutPerception, ScoutBrain
FR6	Support exit and rescue win paths	ScoutAI trigger logic, GameManager rescue logic
FR7	Handle all main loss conditions clearly	GameManager.LoseGame()
FR8	Provide UI feedback for status, environment, and outcomes	GameManager UI methods
FR9	Show explainable AI state and reason information	State labels, reason fields, active behaviour path
FR10	Allow replayable runs with alternative outcomes	Random weather/rescue events and branching behaviour

Table 1: Functional Requirements

6.2 Non-Functional Requirements

Table 2 presents the main non-functional requirements considered during development.

ID	**Non-Functional Requirement**	**Rationale / Evidence**
NFR1	Usability	Clear HUD states, warning messages, and game-over feedback
NFR2	Performance	Lightweight per-frame checks and custom behaviour-tree logic
NFR3	Maintainability	Separation between scout, predator, perception,

		memory, UI, and manager scripts
NFR4	Explainability	Human-readable state labels and reason strings
NFR5	Extensibility	New states, items, or enemy types can be added later
NFR6	Robustness	Stuck recovery, boundary clamping, and terminal state guarding
NFR7	Presentation quality	Environment overlays, labels, sounds, and result screens support demonstration

Table 2: Non-Functional Requirements

7. Research and Background

Intelligent-agent theory describes systems that map percepts to actions in pursuit of goals (Russell and Norvig, 2021). In this game, the scout receives percepts from the environment, evaluates them against survival goals, and selects actions such as searching, fleeing, or moving to a target. Games are also useful AI testbeds because they combine real-time decision making, interactive environments, and visible outcomes (Laird and van Lent, 2001).

State-based behaviour is common in game AI because it is readable and easy to test. Finite state machines provide clear state changes, while behaviour-tree structures are useful when priorities and branching decisions become more complex (Bourg and Seemann, 2004; Millington and Funge, 2016). This project uses a behaviour-tree style decision structure while still exposing clear state labels for assessment and debugging.

Perception in games is often simplified through vision ranges, hearing radii, distance thresholds, and trigger zones. Searching and movement may use heuristic steering in smaller

projects, while full graph-based algorithms such as A* are usually applied when obstacle-aware path planning is required (Buckland, 2004). The A* algorithm is a classic heuristic graph search method for route planning using path cost and an estimated remaining cost (Hart, Nilsson and Raphael, 1968). The current project uses practical search and steering rather than full A* pathfinding.

Unity was selected because it supports 2D gameplay, C# scripting, collision and trigger systems, UI integration, audio, and fast testing inside the editor and standalone builds.

8. Intelligent Agent Design

The primary intelligent agent is the scout. The design can be described using standard agent components:

- Agent: Scout (ScoutAI and ScoutBrain)
- Environment: 2D jungle map with resources, shelter, predators, rain, night, exit, and rescue events
- Sensors: Vision, hearing, resource smell, distance checks, and memory recall
- Actuators: Movement, target selection, behaviour-mode label updates, resource pickup, shelter entry, and trigger interaction
- Performance goals: Stay alive, avoid danger, maintain survival stats, and reach a valid win path

The behaviour engine is priority-based and behaviour-tree driven. Higher-risk behaviours can interrupt lower-priority behaviours because the tree is evaluated from the highest-priority branch to the lowest-priority branch each tick. For example, a nearby predator can pre-empt food searching, while a severe hunger or water crisis can override normal exploration. This makes the scout's choices easier to justify because each action is connected to a current need or threat.

The scout's behaviour is represented using readable state labels such as Explore, SearchFood, SearchWater, EscapeDanger, GoToExit, Win, and Lose. These labels are useful for debugging, reporting, diagrams, and viva explanation. However, the main decision-making process is not

implemented as a strict finite state machine. Instead, the core action selection is handled by a priority-based behaviour-tree structure, where selector and sequence nodes evaluate conditions such as predator danger, hunger, water level, energy, shelter need, and exit availability. Therefore, the project demonstrates state-based behaviour at the observable behaviour level, while the underlying implementation is better described as behaviour-tree driven.

At runtime, the scout first observes the environment through its perception and memory systems. It then compares this information with internal status values such as hunger, water, and energy. The selected action is therefore based on both what is happening in the world and what the scout currently needs to survive.

The action stage is expressed through movement, target choice, shelter use, escape behaviour, and win/lose trigger interaction. Because these actions are recalculated during the game, the scout can change direction when a predator appears, when a resource becomes important, or when the exit becomes a better option.

The system is autonomous because each update cycle recomputes the scout's action choice from current context. It is not a pre-recorded or video-like sequence, and it is not driven by a switch-based state loop. The same scene can produce different outcomes because predator position, weather, resources, stat values, exit safety, and rescue timing can change during runtime.

9. P.E.A.S Model

The P.E.A.S model formalises an intelligent-agent problem by identifying the Performance measure, Environment, Actuators, and Sensors. Table 3 applies this model to the scout agent.

Performance Measure	**Environment**	**Actuators**	**Sensors**
Stay alive, avoid capture, maintain hunger/water/energy, and achieve exit or rescue	Jungle map with food, water, shelter, exit, wolf, lion, rain, night, and rescue events	Movement steering, behaviour-mode updates, target selection, flee/search actions, shelter and trigger interaction	Vision cone, hearing radius, resource smell radius, distance checks, trigger detection, and remembered points

Table 3: P.E.A.S Model

This table shows that the artefact is structured as an intelligent-agent system. The scout senses its environment, chooses actions, and evaluates success through survival and escape outcomes.

9.1 Predator Agents P.E.A.S Model

The coursework focuses mainly on the scout, but the wolf and lion also behave as secondary autonomous agents. They patrol, detect the scout using vision and hearing logic, chase when detection is valid, and trigger a loss when a catch is allowed. Shelter rules can also prevent unfair catches near safe zones.

Table 4 summarises the shared predator-agent model. The lion uses LionAI, while the wolf pattern is mainly controlled through WolfAI and WolfBrain.

Performance Measure	**Environment**	**Actuators**	**Sensors**
Detect the scout, close distance,	Shared jungle map with scout	Movement, rotation, patrol/chase/investigate	Vision cone, hearing radius,

catch when in range, and respect shelter safety rules	position, shelter colliders, safe radii, and night modifiers	states, catch callback through GameManager.LoseGame()	distance to scout, line-of-sight style checks, and shelter proximity

Table 4: Predator Agents P.E.A.S Model

This secondary P.E.A.S model strengthens the multi-agent analysis because predator behaviour contributes directly to the scout's decision pressure.

10. Intelligence Traits Implemented

Table 5 maps the main intelligence traits to the game implementation. This section is important because it explains how the project meets the assignment requirement for intelligent-agent behaviour.

Trait	**Evidence in Game**	**Main Scripts**	**Assessment Relevance**
Perception	Limited vision, hearing, resource smell, and distance sensing	ScoutPerception, ScoutBrain	Shows partial observability instead of full map knowledge
Decision making	Priority branches respond to danger, stats, weather, exit, and rescue	ScoutBrain, BehaviorTree	Shows autonomous runtime choice
Searching / pathfinding	Visible target, memory target, emergency global probe, and exploration	ScoutBrain, SteeringAgent	Shows goal-directed navigation under uncertainty

Memory / learning	Remembered resource and danger points with decay and cleanup	ScoutMemory, ScoutBrain, ItemPickup	Shows experience based adaptation
Status-based behaviour	Hunger, water, energy, danger, rain, night, and rescue state influence decisions	ScoutAI, ScoutBrain, GameManager	Shows internal-state driven behaviour

Table 5: Intelligence Traits Mapped to Implementation

10.1 Perception

Perception means that the agent receives information from the environment before acting. In this project, perception is intentionally limited. The scout does not know the whole map at all times. Instead, it uses a vision cone, hearing radius, resource smell range, distance checks, and remembered locations.

In the game, perception appears when the scout detects food, water, predators, shelter, exit related targets only when conditions allow. Rain and night can reduce sensing effectiveness, which makes the environment less predictable and more realistic. This is supported mainly by ScoutPerception, with methods such as CanSee(), CanHear(), and CanDetectResource(). ScoutBrain then uses those sensed results when choosing the next behaviour.

This demonstrates that the scout successfully acts under partial observability. The agent must make decisions from available percepts rather than using complete global knowledge.

This also makes the behaviour more believable. For example, if a predator is outside the sensing range, the scout should not respond as if it has perfect knowledge. When the predator becomes close enough to detect, the decision logic can then update and trigger a danger response.

10.2 Decision Making

Decision making means selecting an action based on the current situation. The scout uses a priority-based behaviour structure so that urgent needs are handled before lower-priority actions. Predator danger, hunger, thirst, energy, weather, exit safety, and rescue availability all affect which branch is selected.

In implementation, ScoutBrain.BuildRoot() defines the main decision order, while condition and action methods control when states such as EscapeDanger, SearchFood, SearchWater, RecoverEnergy, GoToShelter, and GoToExit become active. The BehaviorTree structure ticks these branches during gameplay, so the scout's action can change when new events occur.

Consequently, the scout avoids following a single fixed route and makes dynamic choices. The agent selects behaviour at runtime according to priorities and changing evidence.

The priority structure also makes the decision process easier to explain. A low-risk situation may allow exploration, but a nearby predator can immediately make escape the most suitable behaviour. This gives the agent a clear reason for each visible state change.

10.3 Searching / Pathfinding

Searching means finding a useful target when the agent has a goal, such as locating food or water. In this project, searching uses a layered approach. First, the scout checks visible or directly detectable targets. If none are safe or available, it uses remembered targets from previous perception. In emergency conditions, it may perform a wider global probe. If no suitable target is found, it continues exploration.

Movement is carried out by SteeringAgent, which supports target movement, arrival behaviour, world boundary clamping, and recovery from stuck movement. This is heuristic search and steering rather than full A* pathfinding, which is acknowledged in the limitations section.

This illustrates how the scout performs goal-directed navigation during the simulation. It does not simply wander without purpose when resources are needed.

The search process is also useful for gameplay because it creates variation between runs. If food or water is visible, the scout can respond quickly. If not, remembered locations or wider

search behaviour give the agent a reasonable fallback without pretending that it has complete map knowledge.

10.4 Memory / Learning

Memory means that the agent can use information gained from earlier experience. In this game, ScoutMemory stores remembered resource and danger points, merges nearby points, applies decay, and removes stale resource locations when pickups are consumed.

In gameplay, memory helps the scout return to likely useful areas even when a resource is no longer directly visible. Danger memory also supports more cautious movement around places where threats were detected. This is learning by experience. It is not machine learning or reinforcement learning, but it still allows the scout to adapt based on previous observations.

Therefore, the agent's later choices are actively influenced by earlier encounters rather than relying solely on immediate perception.

The use of decay is important because old information should become less reliable over time. This prevents the scout from depending too strongly on a resource or danger location that may no longer be useful.

10.5 Status-Based Behaviour

Status-based behaviour means that the agent's internal condition affects decisions. The scout has hunger, water, and energy values, as well as danger and environment flags. These values are updated during gameplay and influence priority selection.

For example, low hunger can activate SearchFood, low water can activate SearchWater, and low energy can activate RecoverEnergy when immediate danger is acceptable. Predator proximity can activate EscapeDanger, while rain or night can encourage shelter behaviour if survival needs are not critical. These behaviours are supported by ScoutAI, ScoutBrain, and GameManager.

This clearly shows that the scout's behaviour is dynamically shaped by both its internal needs and external status. The state labels and reason text make these decisions visible and understandable during testing and demonstration.

This trait is important because survival is not only about reacting to enemies. The scout may be safe from predators but still need water, food, or energy recovery. These internal needs create realistic pressure and help the game show more than simple chase behaviour.

11. State-Based Behaviour

State-based behaviour in this report refers to observable behavioural modes, not a strict hand coded state-machine engine. The game demonstrates state-based behaviour because the scout visibly changes between modes such as exploration, food search, water search, escape, shelter, exit movement, rescue, win, and lose.

The implementation uses behaviour-tree logic to select these modes. ScoutBrain evaluates priority branches each tick, while ScoutAI mirrors the selected StateLabel into readable enum style output for the Inspector, debug UI, screenshots, diagrams, testing, and viva explanation. Therefore, the state labels make the behaviour understandable, but action selection itself is behaviour-tree driven.

Table 6 shows the main scout states, their triggers, and typical next states.

State	Trigger / Event	Behaviour	Typical Next State
Idle	Initial startup	Waits for first active decision	Explore
Explore	No urgent condition	Explores the map and samples targets	SearchFood, SearchWater, EscapeDanger, or GoToExit
SearchFood	Hunger is low or critical	Moves to food using sensed, remembered, or wider search targets	Explore or continue searching

SearchWater	Water is low or critical	Moves to water using sensed, remembered, or wider search targets	Explore or continue searching
RecoverEnergy	Energy is low and risk is acceptable	Holds or slows activity to recover before travelling	Explore or search branches
EscapeDanger	Wolf or lion is within danger range	Flees from the predator and increases separation	Explore
GoToShelter	Rain or night creates risk and no resource crisis is active	Moves to a shelter zone	Explore
GoToExit	Exit decision condition is satisfied	Moves towards the exit target	Win or Explore
HelicopterRescue	Rescue is pending or committed	Holds or moves as required by the rescue flow	Win
Win	Exit or rescue succeeds	Terminal success state	Terminal
Lose	Capture or stat depletion occurs	Terminal failure state	Terminal

Table 6: Scout State Transition Table

The simulation does not play like a fixed video. State changes occur because of events, including hunger, thirst, predator distance, weather, energy, exit availability, and rescue availability. The behaviour-tree structure controls priority, so a high-risk event such as predator danger can override lower-risk actions such as exploration.

For example, hunger dropping below a threshold can move the scout from Explore to SearchFood, while low water can activate SearchWater. If a wolf or lion moves into danger range, the scout can switch to EscapeDanger even if it was previously searching for a resource. Rain and night can make shelter more useful, and exit or rescue availability can create a route towards a win condition when survival conditions allow it.

As shown in Figure 1, the state labels make the AI behaviour easier to follow because each visible state has a reason and a transition condition.

12. Real-World Physics Simulation

The game includes practical physics-style indicators that make the simulation more believable and easier to assess. These are not advanced physics simulations, but they show that the agent operates with range limits, time-based effects, movement constraints, and collision rules.

Indicator	**Implementation**	**Why It Is Realistic**	**Script / Evidence**
Vision cone and angle	Range and angle checks	Agents cannot see in every direction at once	ScoutPerception.CanSee()
Hearing range	Radius-based hearing	Nearby threats may be detected even when not facing them	ScoutPerception.CanHear(), predator hearing logic
Distance-based danger and catch	Danger and catch ranges	Predator threat increases with proximity	ScoutBrain, WolfBrain
Survival decay over time	Hunger, water, and energy drain during play	Survival pressure increases over time	ScoutAI.UpdateStats()

Weather and fatigue speed effects	Rain, night, and low energy modify movement	Conditions affect movement performance	ScoutBrain.EffectiveSpeed(), ScoutAI
Collider-based pickup and capture	Trigger/collision callbacks	Physical contact produces resource or outcome effects	ItemPickup, predator handlers
World boundary clamping	Position is constrained to allowed bounds	Prevents impossible movement outside the map	SteeringAgent.ClampToWorld()

Table 7: Real-World Physics Indicators

These indicators support the claim that the AI behaviour is grounded in a playable game world rather than an abstract text-only decision system.

Rain, night, and low energy are treated as gameplay modifiers that affect the scout's behaviour during a run. These conditions support the survival simulation by influencing movement or sensing behaviour, increasing the usefulness of shelter, and creating additional decision pressure for the agent. Shelter areas also provide a safer zone during suitable conditions, while predator danger can still change the scout's priority if the situation becomes unsafe.

13. System Design and Diagrams

The system design is represented through five draw.io diagrams. Figures 1 to 4 focus on the scout and overall game flow, while Figure 5 focuses on predator agents. Editable sources are stored under documentation/diagrams/final/, and Google Drive copies are listed for submission access.

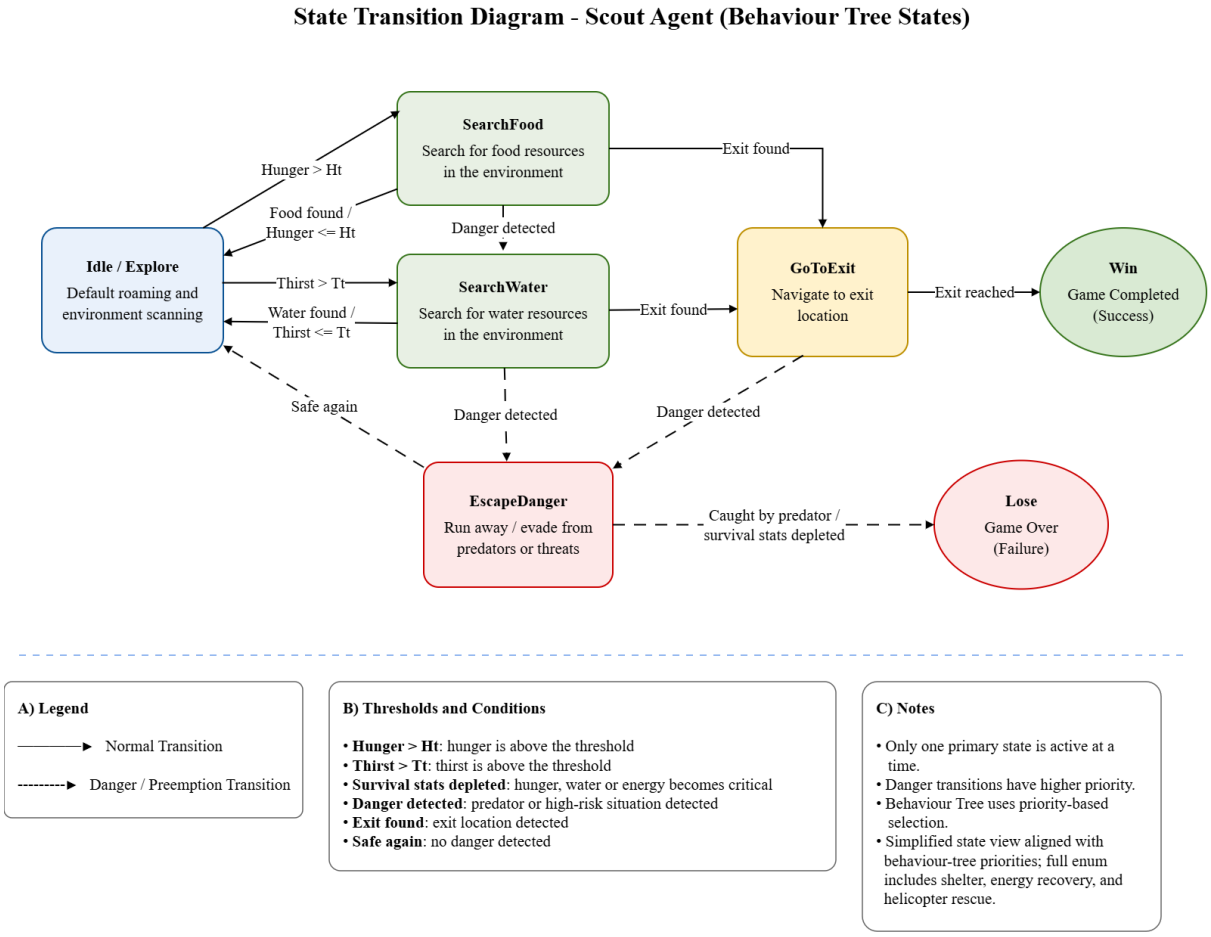

Figure 1: State Transition Diagram

Figure 1 shows the scout's behavioural modes and the main conceptual transitions between them. It supports the state-based explanation in Section 11 by showing how hunger, thirst, danger, weather, exit, and terminal outcomes affect behaviour. This diagram is a conceptual state transition view for explanation and assessment; it should not be read as proof of a literal finite-state-machine engine in the code.

- Google Drive: scout_state_transition.drawio https://drive.google.com/file/d/1tW5a2-bBY_4BPFv3yEJ9KEF_PGJ_xlfq/view?usp=sharing

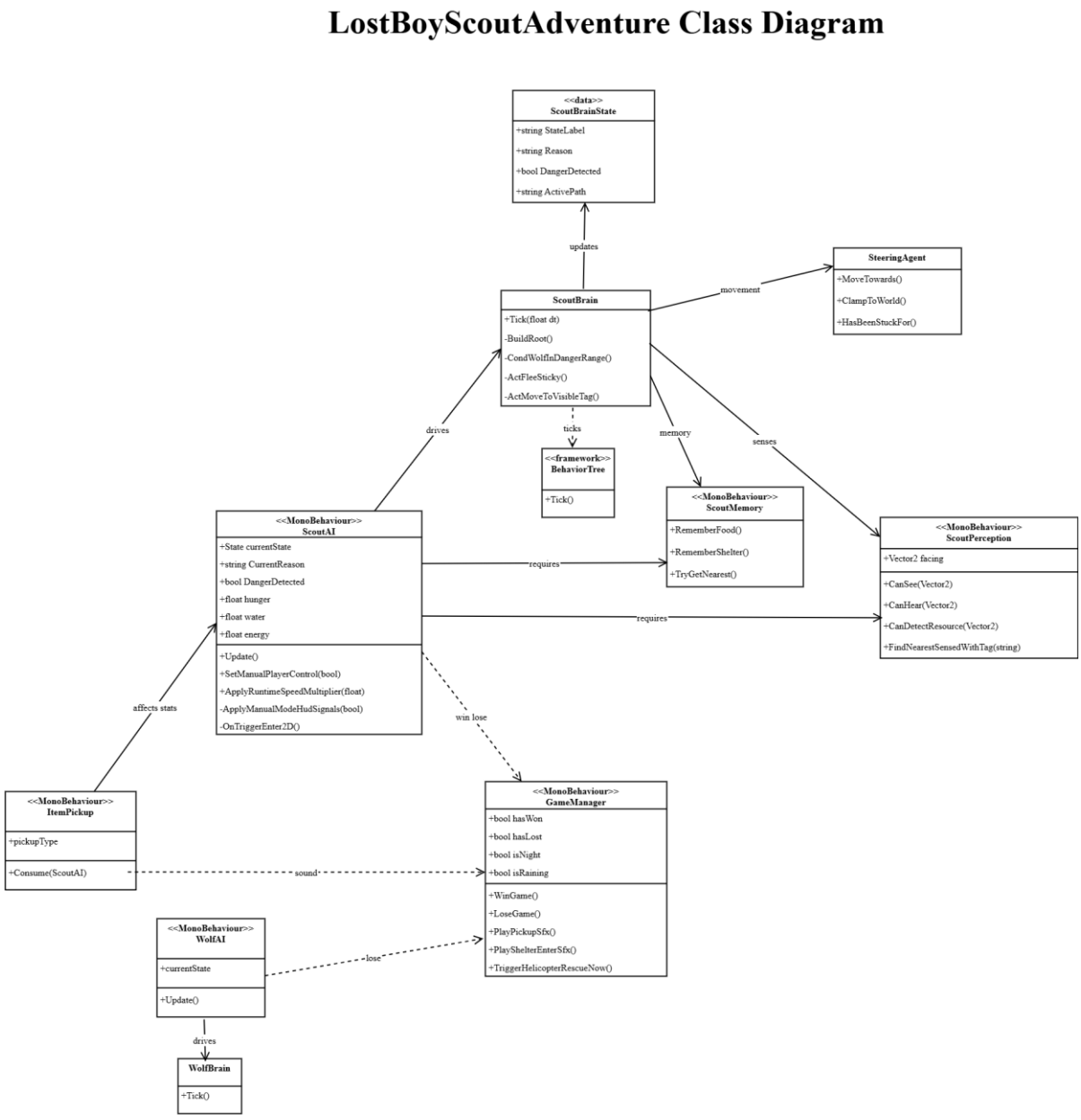

Figure 2: Class Diagram

Figure 2 illustrates the main class structure of the project, including ScoutAI, ScoutBrain, perception, memory, behaviour-tree support, predator scripts, GameManager, and pickups.

- Google Drive: class_diagram.drawio https://drive.google.com/file/d/1rpWSHait61Vp4hwvaJ7_vhOCAP2F91UR/view?usp=sharing

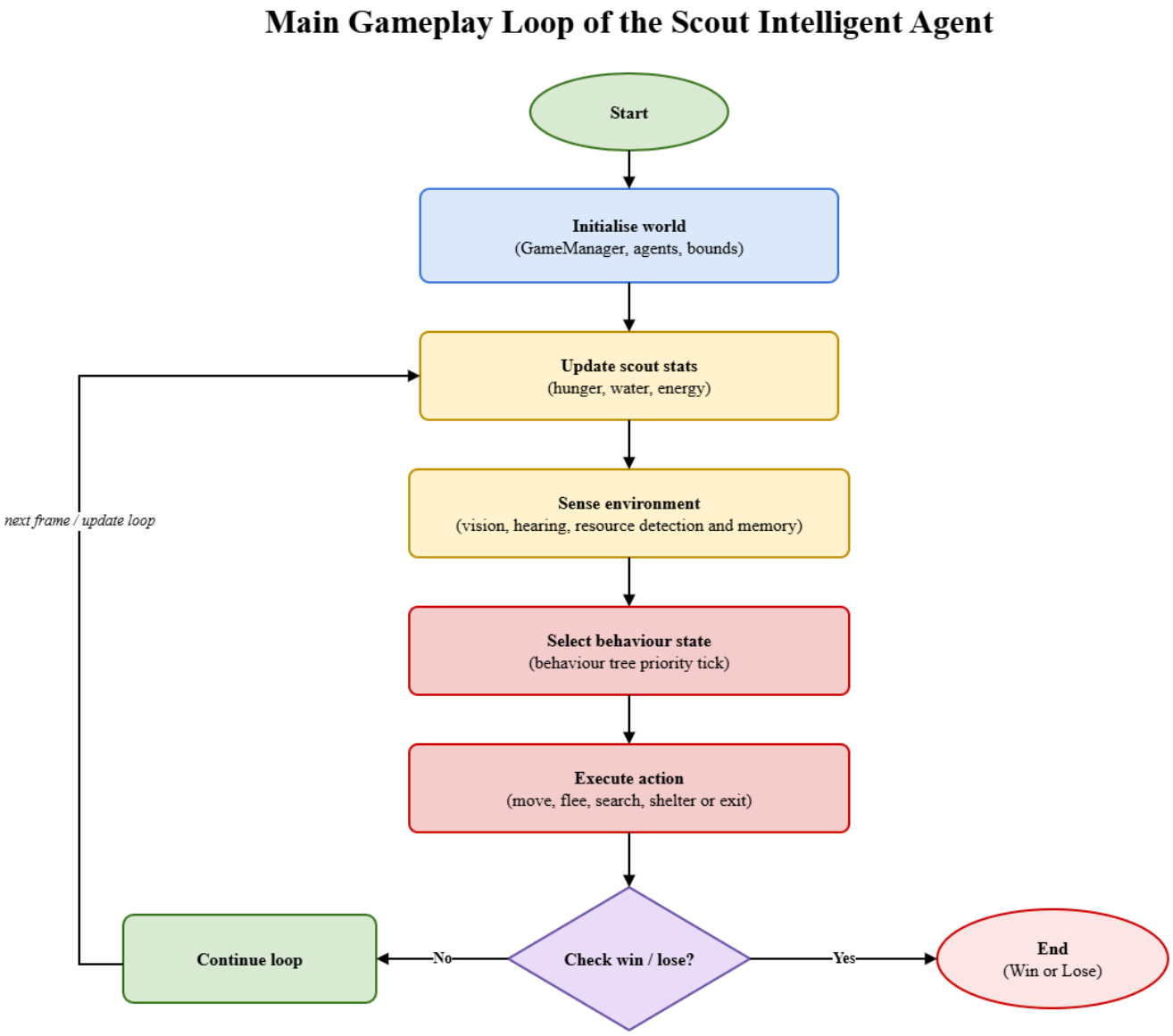

Figure 3: Game Flowchart

Figure 3 presents the high-level runtime loop. The game initialises, updates stats, senses the environment, runs the behaviour-tree decision logic, performs actions, checks win/lose conditions, and repeats until a terminal state is reached.

- Google Drive: scout_gameplay_loop_game_flowchart.drawio https://drive.google.com/file/d/15TmlYgeITaejQ1kMq2wng69BbECL4Pbr/view?usp=sharing

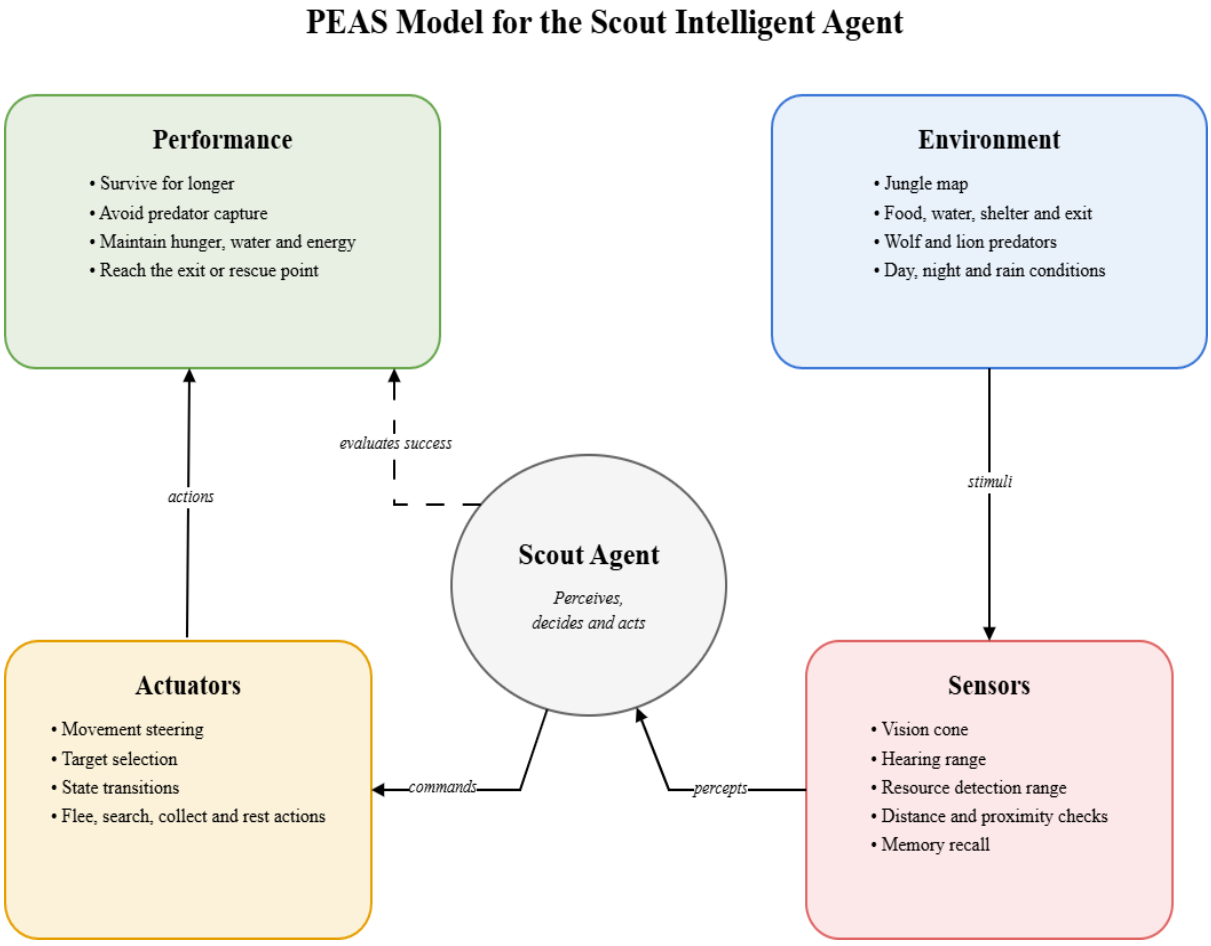

Figure 4: P.E.A.S Model

Figure 4 summarises the scout P.E.A.S model visually. It supports Table 3 by showing the scout's performance measure, environment, actuators, and sensors.

- Google Drive: scout_peas_model.drawio https://drive.google.com/file/d/1LmGUCKlwtq6zti6K2FUdJ1ImCSD1SCqV/view?usp=sharing

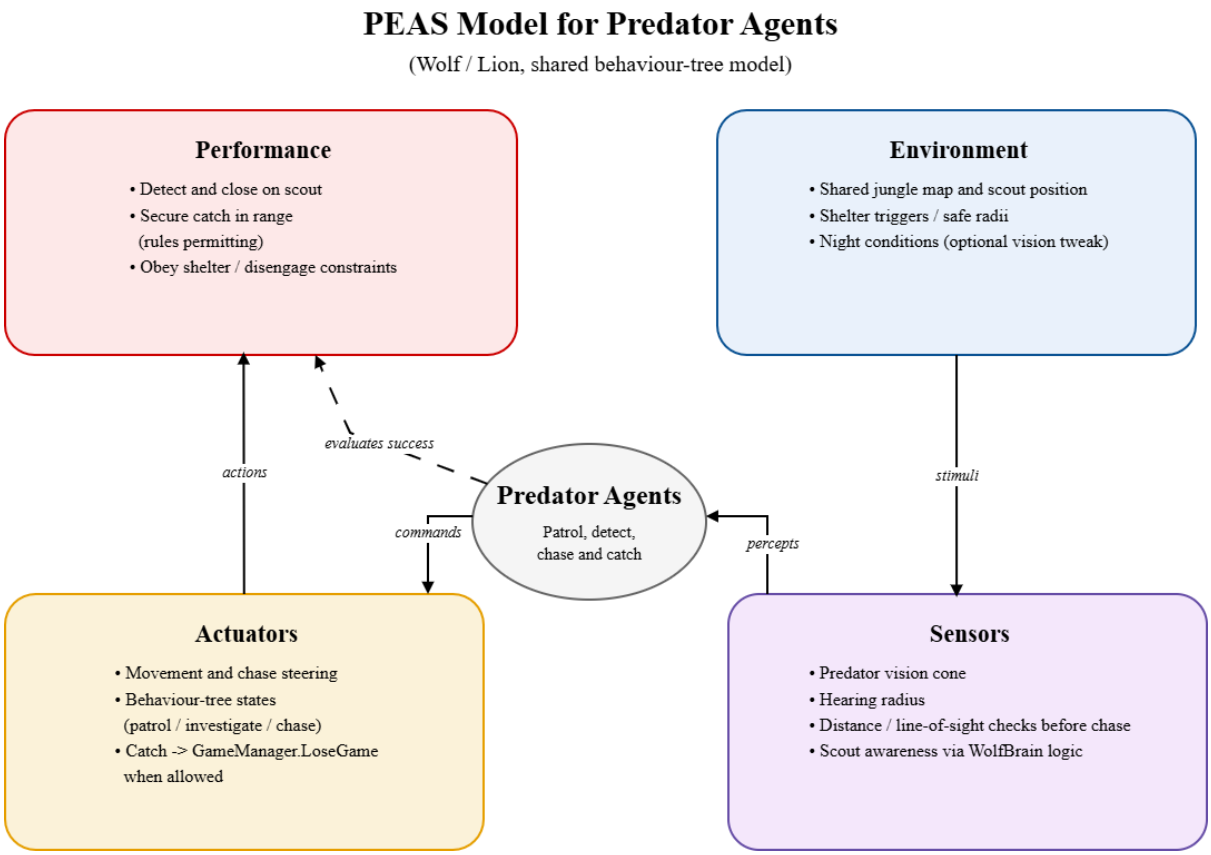

Figure 5: Predator Agents P.E.A.S Model

Figure 5 shows the predator-agent P.E.A.S model for the wolf and lion. It links to Table 4 and explains how adversarial agents contribute to the survival challenge.

- Google Drive: predator_agents_peas_model.drawio https://drive.google.com/file/d/1n-JIir-TUiyKf3wVgvKyNWWBi78pfefF/view?usp=sharing

14. Implementation

The main Unity scripts are summarised below. The descriptions focus on purpose, key responsibility, and AI contribution without adding unsupported features.

Script / Class	**Purpose and Responsibility**	**Key Methods / Areas**	**AI Contribution**
ScoutAI	Connects the scout GameObject to gameplay stats, triggers, HUD signals, update flow, and mirrored state labels	Update(), UpdateStats(), trigger handlers, label mapping	Provides the runtime body of the scout agent and exposes readable state output
ScoutBrain	Controls the main scout behaviour and priority decisions	BuildRoot(), condition methods, action methods	Core autonomous decision making
ScoutPerception	Models limited sensing for the scout	CanSee(), CanHear(), resource detection methods	Supports partial observability
ScoutMemory	Stores and updates remembered resource and danger points	Remember, TryGetNearest(), decay methods	Supports learning by experience
SteeringAgent	Handles target movement and movement stability	MoveTowards(), ApplyNudge(), ClampToWorld()	Converts decisions into movement
BehaviorTree	Provides behaviour-tree node execution through selectors, sequences,	Tick() and node classes	Core priority based decision framework

	conditions, and actions		
WolfAI	Connects the wolf GameObject to predator interactions	Update(), catch handlers	Adds active enemy pressure
WolfBrain	Controls predator detection, chase, investigate, roam, and catch behaviour	Priority branches for catch/chase/investigate/roam	Creates dynamic predator response
LionAI	Provides the secondary predator flow	Wolf-style predator behaviour	Adds another danger path
GameManager	Controls environment state, UI, weather, rescue, win, and lose results	Weather, rescue, win, and lose methods	Provides changing world context and outcomes
ItemPickup	Handles food and water resource consumption	Consume()	Links resource interaction to survival stats and memory cleanup
CheatCodeManager	Provides typed QA and demonstration shortcuts	Buffer matching, ActivateCheat, ResetAllCheats	Supports testing, but is not core AI
MainMenuPauseController	Handles menu, pause, help, and sound settings	ApplySelectedSoundOn()	Improves usability and presentation
ScoutActionLabelView	Displays floating scout state and reason text	BuildActionText()	Improves explainability

ShelterSafeLabel View	Shows shelter safety feedback	Trigger overlap behaviour	Supports player-facing clarity

Table 8: Core Script Responsibilities

ScoutAI acts as the Unity-facing facade of the scout. It keeps track of survival stats, update flow, trigger interaction, and signals that are needed by the HUD. It also mirrors the brain's StateLabel into a readable enum-style value for the Inspector and debug display. This class is important because the AI brain needs current gameplay data before it can make a useful decision.

ScoutBrain is the scout decision engine. It builds and runs the priority behaviour-tree structure, checks conditions, selects actions, and updates the active state label. Its contribution is central to the coursework because it turns perception, memory, internal status, and danger checks into autonomous behaviour without relying on a hand-written state transition loop.

ScoutPerception models what the scout can sense. It checks vision, hearing, resource detection, distance limits, and environment-related sensing changes. This prevents the scout from acting with unrealistic full knowledge of the map.

ScoutMemory stores information gained from previous perception. It records resource and danger points, applies decay, merges nearby entries, and supports nearest-target lookup. This gives the scout a simple learning-by-experience mechanism without claiming to use machine learning.

SteeringAgent converts selected targets into movement. It supports movement towards goals, arrival behaviour, boundary clamping, and recovery from stuck positions. This helps the scout's decisions appear as stable movement in the Unity scene rather than only as abstract state changes.

BehaviorTree provides the core behaviour-tree runtime used by the scout and predator logic. It defines node statuses, base nodes, selector nodes, sequence nodes, condition leaves, action leaves, inverter logic, cooldown support, and the tree Tick() process. This is useful because urgent behaviours, such as escaping a predator or catching the scout, can interrupt lower priority behaviours through priority ordering.

WolfAI, WolfBrain, and LionAI support the adversarial side of the simulation. Predator AI follows the same behaviour-tree pattern, where the wolf and lion behaviours are selected through priority branches such as catch, chase, investigate, and roam, with mode labels mapped for display and documentation. These scripts create active pressure, which forces the scout to make survival decisions rather than only search for resources.

GameManager controls the wider game context, including weather, rescue, UI feedback, win conditions, and lose conditions. It is not the scout brain, but it supplies important world events that influence the scout's behaviour. It also locks terminal outcomes so that win and lose states do not conflict.

ItemPickup handles food and water consumption. It connects physical pickup interaction with survival stat restoration and memory cleanup. This makes resource search meaningful because reaching an item changes the scout's future condition.

CheatCodeManager, MainMenuPauseController, ScoutActionLabelView, and ShelterSafeLabelView support testing, presentation, and explainability. They do not replace the autonomous AI logic, but they make the artefact easier to demonstrate and evaluate. In particular, the floating labels help show why the scout is currently searching, fleeing, sheltering, or moving towards an outcome.

ScoutBrain is the most important script for AI marking because it combines perception results, survival thresholds, memory, danger checks, and priority logic. ScoutAI provides the gameplay data that the brain acts on, while GameManager supplies world events and final outcomes. SteeringAgent then turns selected targets into movement.

14.1 Presentation, Audio, and QA Cheat Support

The project also includes presentation, audio, and typed cheat-code support. These features improve demonstration and testing, but they sit outside the main autonomous ScoutBrain logic unless a cheat explicitly changes stats, speed, weather, or rescue state.

Area	Implementation	Notes
Ambient loops	GameManager rain and night AudioSource components	Fade in/out and respect win/lose state
Pickups	GameManager.PlayPickupSfx()	Plays food or water one-shot sounds from ItemPickup
Win / lose stings	GameManager audio clips	Selects win or lose sound based on outcome
Shelter entry	GameManager.PlayShelterEnterSfx()	Plays once when the scout first enters a shelter trigger
Main menu	MainMenuPauseController	Optional looping menu clip
Global mute	MainMenuPauseController.ApplySelectedSoundOn()	Controls AudioListener.volume and AudioListener.pause

Table 9: Audio Integration and Asset Sourcing

Royalty-free audio assets, including Mixkit-style sound effects, were used where suitable for a student build. For example, the repository includes Assets/Audio/mixkit-shelterslide-2363.wav for shelter entry. Clips can be assigned through the Unity Inspector or left empty for silent fallback.

All visual assets and supporting images were either created manually using standard 2D editing software or sourced from royalty-free, open-license asset libraries suitable for academic projects. These tools were used to create or refine game-style visual assets for the coursework artefact. External audio and effect sources, including Mixkit and EventLabs where relevant, were used as supporting media resources. These sources are acknowledged to maintain transparency about asset preparation and external learning support.

Feature	**Script(s)**	**Purpose**
Floating action label above scout	ScoutActionLabelView	Displays state, danger flag, and reason text
Manual-mode danger and shelter sync	ScoutAI.ApplyManualModeHudSignals()	Keeps HUD feedback consistent when manual control is used
Shelter safe messaging	ShelterSafeLabelView, ScoutAI.IsScoutInsideShelter	Shows shelter safety status when overlapping shelter triggers
Rescue countdown text	GameManager state UI string	Shows rescue timing, such as rescue becoming possible after a countdown

Table 10: Player-Facing UI and Manual-Mode Consistency

The helicopter rescue feature is implemented as an alternative win path controlled through the game outcome logic. In the automatic rescue flow, helicopter availability is probability-based after the scout has survived for the required unlock period. The system checks rescue availability at timed intervals, so the helicopter does not always appear at the same moment in every run. When rescue is triggered, it does not give an immediate win. The scout must still satisfy the rescue conditions, such as reaching the rescue zone and remaining safe long enough for the rescue to complete. This supports replayability because the game can end through

different valid outcomes depending on the scout's condition, predator danger, weather state, and the active rescue flow.

Shelter trigger size and wolf/lion shelter safety radii were tuned so that the safe-zone behaviour is clear during demonstration. This affects collision and predator avoidance rules, not the behaviour-tree structure itself.

Code	**Effect**
food / water	Spawn a pickup near the scout if the template exists in the scene
energy	Set energy to 100
cocacola	Set water to 100
kfc	Set hunger to 100
eternal	Toggle maximum hunger, water, and energy each frame
unkill	Toggle blocking of enemy-catch loss reasons
speedup / run	Increase scout move-speed multiplier within limits
speeddown / slow	Decrease scout move-speed multiplier within limits
day, night, rain, clear	Force weather or time-of-day presentation
heli	Start the helicopter rescue flow

Table 11: Cheat Code Manager Examples

These cheat codes support repeatable testing and demonstrations. They should not be used as the main

15. Algorithms

15.1 Priority-Based Scout Decision-Making Algorithm

The scout decision algorithm represents the priority order of the behaviour-tree structure, not a switch(currentState) style state-machine loop. Each tick rechecks the highest-priority valid branch before lower-priority behaviours are allowed to run.

1. Update sensed objects, internal stats, and memory state.
2. If critical predator danger exists, execute flee behaviour.
3. Else if the exit decision is valid and safe enough, move to the exit.
4. Else if normal predator danger exists, execute flee behaviour.
5. Else if hunger or water is in crisis, perform urgent resource search.
6. Else if energy recovery is needed and risk is acceptable, recover energy.
7. Else if hunger or water is below proactive thresholds, search for resources.
8. Else if weather or night conditions encourage shelter and no crisis exists, move to shelter.
9. Else continue exploration.

15.2 Perception Algorithm

10. For each relevant target, calculate distance from the scout.
11. Check vision range and angle using CanSee().
12. Check hearing radius using CanHear().
13. For resources, check CanDetectResource() using vision or smell.
14. Return the nearest valid sensed target by type.

15.3 Resource Search Algorithm

15. Attempt the nearest safe visible or detectable target.
16. If unavailable, attempt the nearest safe remembered target.
17. If unavailable and the need is urgent, perform a wider safe probe.
18. If no useful target exists, continue scored exploration.

15.4 Predator Detection and Chase Algorithm

The predator algorithm also follows priority-based behaviour-tree selection rather than a strict state-machine transition table.

19. Measure distance from predator to scout.
20. Ignore chase if the scout is outside chase range.
21. Detect the scout if inside hearing radius.
22. Otherwise, apply vision range and cone-angle checks.
23. If detected, move towards the scout and refresh last-seen data.
24. If catch range is reached and safety rules allow it, trigger the lose result.

15.5 Win and Lose Evaluation Algorithm

25. If the scout reaches the exit trigger, set win.
26. If the rescue sequence completes successfully, set win.
27. If hunger, water, or energy reaches zero, set lose.
28. If a predator catch occurs, set lose.
29. Lock the terminal state to prevent conflicting outcomes.
30.

16. Testing

Testing was completed through manual playtesting with controlled setups and repeated runs. Scenarios were observed in the Unity Editor and in standalone Windows executable builds. The tests checked state labels, HUD feedback, resource behaviour, predator behaviour, weather effects, audio feedback, and final outcomes.

16.1 Standalone Build, Peer Testing, and Display Targets

After the core gameplay features were implemented, a standalone Windows build was produced using Unity Build Settings. The executable was smoke-tested outside the Unity Editor to confirm that the main menu, AI mode, manual mode, win/lose screens, audio toggle, and closing behaviour worked as expected.

The main development reference resolution was Full HD 1920 x 1080. Menus, HUD elements, and end screens were first checked at this resolution. Because peer testers used different machines and display scaling settings, the UI was also checked at 1280 x 720. This helped confirm that hunger, water, energy, rescue text, and outcome screens remained readable on smaller laptop displays and projector-style demos.

Repeated playtesting was used because the game can end in different ways depending on agent decisions and runtime events. Individual scenarios were set up to confirm specific behaviours, such as food search or predator escape, while longer runs were used to check whether alternative outcomes could occur naturally. This approach gave a more realistic view of the artefact than testing only one fixed route through the game.

Test ID	Test Area	Test Scenario	Input / Setup	Expected Result	Actual Result	Status
T01	Food search	Scout seeks food when hunger drops	Lower hunger and place food in detectable range	Scout enters SearchFood and consumes food	Behaviour observed as expected	Pass

T02	Water search	Scout seeks water when thirsty	Lower water and place water pickup	Scout enters SearchWater and restores water	Behaviour observed as expected	Pass
T03	Wolf capture	Wolf catches scout	Move scout into wolf catch range	Lose with wolf reason	Killed by Wolf result triggered	Pass
T04	Lion capture	Lion catches scout	Move scout into lion catch range	Lose with lion reason	Killed by Lion result triggered	Pass
T05	Escape danger	Scout escapes near predator	Place predator in danger range with open path	Scout enters EscapeDanger and distance increases	Flee branch activated and distance increased	Pass
T06	Exit win	Scout reaches exit	Keep scout alive and move into exit trigger	Win by exit	Win triggered correctly	Pass
T07	Helicopter rescue	Rescue path activates	Run long enough with rescue enabled	Rescue sequence completes and win appears	Rescue win observed on valid runs	Pass
T08	Hunger loss	Scout starves without food	Remove or avoid food pickups	Lose by starvation	Starvation result triggered	Pass

T09	Water loss	Scout dehydrates without water	Remove or avoid water pickups	Lose by dehydration	Dehydration result triggered	Pass
T10	Energy loss	Scout exhausts energy	Force sustained drain without recovery	Lose by exhaustion	Exhaustion result triggered	Pass
T11	Night/rain effects	Environment modifiers affect gameplay	Run through weather and night cycle	Sensing/speed changes and overlays appear	Effects observed as implemented	Pass
T12	Memory search	Scout uses remembered resource	Let scout detect a resource, then move away	Scout uses memory target later	Memory based search observed	Pass
T13	Alternative outcomes	Repeated full-run sessions	Play multiple runs	Different outcomes are possible	Exit, rescue, and loss variations observed	Pass
T14	Cheat stat refill	Typed QA codes restore stats	Type cocacola or kfc during a live round	Water or hunger restores to 100	Observed	Pass
T15	Shelter sound	Shelter entry sound plays once	Assign shelter clip and enter shelter trigger	One-shot sound plays without repeated spam	Observed	Pass

T16	Sound toggle	Sound setting changes audio state	Toggle Sound ON/OFF in menu	AudioListener reflects selected choice	Observed	Pass
T17	Windows executable	Build runs outside Unity	Launch executable and complete a play loop	Stable run matching Editor behaviour	Passed on developer PC	Pass
T18	Peer machines	Build tested on other machines	Share build with colleagues/testers	Playable without Unity-specific errors	Feedback incorporated where needed	Pass
T19	Display targets	1080p baseline and 720p validation	Check menus, HUD, and outcome screens	Layout remains readable without critical clipping	Matches intended layout	Pass

Table 12: Test Plan and Results

The results show that the key AI behaviours and gameplay outcomes were functioning in practical tests. The tests were manual rather than automated, so the evidence relies on observed behaviour, screenshots, and repeated playthroughs.

The Game User Manual also supports testing and demonstration because it explains the intended play flow, available controls, debug features, and cheat-code options used to verify game behaviour.

17. Evaluation Against Assignment Brief

Table 13 evaluates the artefact against the main coursework expectations.

Assessment Requirement	Evidence in Project	Strength
Suitable intelligent-agent idea	Scout survival agent in a dynamic jungle game	Strong
At least three intelligence traits	Perception, decision making, searching, memory, and status-based behaviour	Strong
Real-world physics indicator	Range limits, stat decay, speed modifiers, distance checks, and collisions	Strong
Events and observable state transitions	Hunger, thirst, predator distance, weather, energy, exit, and rescue trigger behaviour-mode changes	Strong
Overall functionality and appearance	Playable loop with HUD, menus, audio, win/lose screens, and executable build	Good
Technical documentation quality	Structured report, diagrams, tables, implementation notes, testing evidence, and appendices	Strong
Creativity and showcase value	Game-based AI demonstration rather than a common chatbot-style artefact	Strong

Table 13: Assignment Requirement Mapping

The project meets the core expectations of the assignment. The intelligent-agent idea is suitable because the scout has clear goals, limited perception, internal survival pressure, and event-driven decisions. The artefact demonstrates more than three intelligence traits, includes physics-style indicators, and provides visible state transitions. The game is also suitable for demonstration because assessors can observe the AI behaviour directly rather than only reading output text.

The project satisfies the state-based behaviour requirement through observable state and mode changes, even though the implementation uses a behaviour-tree style controller rather than a strict finite-state-machine controller. This distinction is important because the assignment can still assess state-based behaviour through the visible modes, diagrams, debug labels, and test evidence, while the actual action selection remains priority-based and reactive.

The project also links well to the marking criteria because the AI elements are not only described in theory. Perception is implemented through sensing limits, decision making is shown through priority branches, searching is visible through target selection and movement, and memory affects later choices. State transitions are supported by runtime events, while physics-style indicators such as distance, collision, speed, and stat decay make the behaviour more grounded. The result is a functional and creative artefact with enough technical evidence to support assessment.

18. Creativity and Showcase Value

The project has strong showcase value because it presents AI through an interactive 2D game rather than a static or text-only artefact. The scout survival scenario allows the assessor to see how perception, memory, searching, and decision making affect movement and outcome.

The inclusion of predators, weather, night effects, shelter, exit, rescue, and multiple loss conditions creates a richer demonstration environment. State labels and debug-style feedback also make the behaviour more explainable during a viva or walkthrough. This supports both the technical requirements of the coursework and the visual clarity expected from an AI artefact demonstration.

19. Individual Submission Approval and Communication

This coursework was initially started as a group artefact. During the final preparation stage, the author informed the module lecturer about the practical difficulties experienced with group progress and the limited time remaining before submission. Based on the lecturer's guidance, the author was permitted to continue and complete the coursework individually.

Following this guidance, the author proceeded independently and completed the remaining technical development, testing evidence, documentation, diagrams, user manual preparation, and final submission materials. This helped avoid further delay and allowed the artefact to be completed within the remaining time.

The final report, appendices, user manual, project repository, packaged build, and supporting individual development report provide the main evidence for the work completed.

Evidence Type	Purpose	Appendix Reference
Final report and appendices	Shows completed documentation, diagrams, testing, and evaluation evidence	Main report and appendices

Project repository and packaged build	Provides access to the Unity source project and playable build	Appendix E
Task planning / evidence record	Shows how the remaining work was organised during final preparation	Appendix D

Table 14: Individual Submission and Final Preparation Evidence

20. Individual Contribution

The full individual contribution report is attached as a supporting document at the end of this submission.

21. Challenges Faced

Several practical challenges were encountered during development:

- Avoiding fixed scripted behaviour while keeping the game predictable enough to test.
- Balancing predator danger so the game remained challenging but still fair.
- Keeping AI decisions explainable for assessment, screenshots, and viva discussion.
- Applying perception limits consistently across normal, rain, and night conditions.
- Handling Unity trigger and collider timing for pickups, shelter, and predator catch events.
- Tuning hunger, water, and energy drain so the game had suitable pacing.
- Testing the standalone build across different screen sizes and display settings.
- Keeping the written report aligned with the implemented Unity scripts.
- Maintaining clear and professional communication during the final preparation stage.

These challenges were managed through repeated testing, incremental improvement, and continuous refinement of the AI behaviour, game mechanics, and documentation.

22. Limitations

The current system has the following limitations:

- Full A* obstacle-aware pathfinding is not implemented.
- Learning is memory-based rather than machine learning or reinforcement learning.
- Animation depth and visual polish could be improved further.
- Predator cooperation and multi-agent communication are limited.
- Automated test coverage is limited because most tests were manual playthroughs.
- Behaviour logging and post-run analytics could be expanded.

These limitations do not prevent the artefact from meeting the core assignment requirements. Instead, they identify realistic areas for future technical improvement and further development beyond the coursework scope.

23. Conclusion

The project delivers a practical Unity 2D intelligent-agent artefact that demonstrates core artificial intelligence concepts in a clear and assessable form. The scout agent operates under partial observability, manages hunger, water, and energy, responds to predators and environmental events, uses remembered information, searches for resources, and changes observable behaviour according to runtime conditions.

The implementation demonstrates several intelligence traits, including perception, decision making, searching, memory, and status-based behaviour. The main decision process is handled through priority-based behaviour-tree style logic, while readable state labels are used to support debugging, screenshots, diagrams, testing, and viva explanation. This allows the artefact to show state-based behaviour without relying on a fixed scripted sequence.

The game also includes real-world physics indicators such as sensing ranges, distance checks, stat decay over time, movement modifiers, and collider-based interactions. Alternative outcomes, including exit success, helicopter rescue, starvation, dehydration, exhaustion, and predator capture, make the simulation more replayable and suitable for intelligent-agent evaluation.

The final artefact provides a strong game-based AI submission that visually demonstrates behaviour which would be difficult to show in a text-only chatbot project. The technical report, diagrams, testing evidence, user manual, and individual contribution supporting document together provide a complete submission package aligned with the coursework expectations.

24. Future Improvements

Future improvements could include:

- Integrating A* pathfinding for stronger route planning around obstacles.
- Extending adaptive behaviour through richer memory scoring or reinforcement learning.
- Improving character animation and movement presentation.
- Adding more advanced predator tactics and cooperation.

- Expanding weather interactions and environmental mechanics.
- Creating additional levels or maps for broader testing.
- Adding behaviour logs or post-run summaries for stronger evaluation evidence.
- Improving UI analytics and accessibility across more resolutions.

These improvements would help extend the project from a coursework artefact into a more advanced, polished, and replayable intelligent-agent game prototype.

25. References

Bourg, D.M. and Seemann, G. (2004) AI for Game Developers. Sebastopol, CA: O'Reilly Media.

Buckland, M. (2004) Programming Game AI by Example. Plano, TX: Wordware.

Ceylon Game Dev (2023) Unity Game Development Full Course in Sinhala. YouTube. Available at: https://www.youtube.com/watch?v=6CPKmwOimjg (Accessed: 4 May 2026).

Code Monkey (2022) The Unity Tutorial For Complete Beginners. YouTube. Available at: https://www.youtube.com/watch?v=XtQMytORBmM (Accessed: 4 May 2026).

Hart, P.E., Nilsson, N.J. and Raphael, B. (1968) 'A formal basis for the heuristic determination of minimum cost paths', IEEE Transactions on Systems Science and Cybernetics, 4(2), pp. 100-107. doi: 10.1109/TSSC.1968.300136.

Laird, J.E. and van Lent, M. (2001) 'Human-level AI's killer application: Interactive computer games', AI Magazine, 22(2), pp. 15-26. doi: 10.1609/aimag.v22i2.1558.

Microsoft (2026) C# documentation. Available at: https://learn.microsoft.com/dotnet/csharp/ (Accessed: 1 May 2026).

Millington, I. and Funge, J. (2016) Artificial Intelligence for Games. 3rd edn. Boca Raton: CRC Press.

Mixkit (2026) Free sound effects. Available at: https://mixkit.co/free-sound-effects/ (Accessed: 1 May 2026).

Russell, S. and Norvig, P. (2021) Artificial Intelligence: A Modern Approach. 4th edn. Harlow: Pearson.

Unity Technologies (2026) Unity Manual. Available at: https://docs.unity3d.com/Manual/ (Accessed: 1 May 2026).

26. Appendices

Appendix A: Annotated Source Code Listing

This appendix contains selected code evidence from the Unity source files. It is intentionally selective so the main report remains readable. The full Unity source code will be submitted with the coursework package.

A.1 Behaviour Tree Runtime

The behaviour-tree runtime provides selector, sequence, condition, and action nodes. This supports priority-based decision making rather than a hand-coded finite state machine.

```
public enum NodeStatus
{
    Success,
    Failure,
    Running
}

public abstract class BTNode
{
    public string DebugName;
    protected BTNode(string name) { DebugName = name; }
    public abstract NodeStatus Tick(BTContext ctx);
}

public class Selector : BTNode
{
    private readonly List<BTNode> children;

    public override NodeStatus Tick(BTContext ctx)
    {
        ctx.AppendPath(DebugName);
        for (int i = 0; i < children.Count; i++)
```

```
        {
            int markerLen = ctx.PathBuilder.Length;
            NodeStatus s = children[i].Tick(ctx);
            if (s == NodeStatus.Running) return NodeStatus.Running;
            if (s == NodeStatus.Success) return NodeStatus.Success;
            ctx.PathBuilder.Length = markerLen;
        }
        return NodeStatus.Failure;
    }
}
```

A.2 Scout Priority Decision Order

ScoutBrain.BuildRoot() organises behaviours by priority. Higher branches can pre-empt lower branches when conditions change.

```
return new Selector(
    "Root",
    criticalSeq,
    exitSeq,
    helicopterCommittedSeq,
    dangerShelterEscapeSeq,
    dangerSeq,
    hungerCrisisSeq,
    waterCrisisSeq,
    recoverEnergySeq,
    hungerSeekSeq,
    waterSeekSeq,
    shelterSeq,
    wanderAct);
```

A.3 Perception Check

ScoutPerception prevents the scout from having full map knowledge. Resources are detected through vision or short-range resource sense.

```
public bool CanDetectResource(Vector2 targetPos)
{
    return CanSee(targetPos) || CanSmellResource(targetPos);
}
```

A.4 Memory Support

ScoutMemory stores useful resource points and clears stale points when resources are consumed.

```
public void RememberFood(Vector2 pos)
{
    AddOrMerge(knownFood, pos);
}

public void RememberWater(Vector2 pos)
{
    AddOrMerge(knownWater, pos);
}

public void ForgetFoodNear(Vector2 pos)
{
    ForgetNear(knownFood, pos, mergeRadius);
}
```

A.5 Win and Lose State Locking

GameManager prevents conflicting terminal outcomes by returning early if the game is already won or lost.

```
public void WinGame(string reason)
{
    if (win || lose) return;
    win = true;
    ClearPendingRescueHold();
    ShowWin(reason);
}

public void LoseGame(string reason)
{
    if (win || lose) return;
    lose = true;
    ClearPendingRescueHold();
    ShowLose(reason);
}
```

Appendix B: Screenshots

This screenshot shows the game project opened in the Unity Editor, including the scene setup and project assets used during development. It supports the evidence that the artefact was implemented and tested as a Unity 2D project.

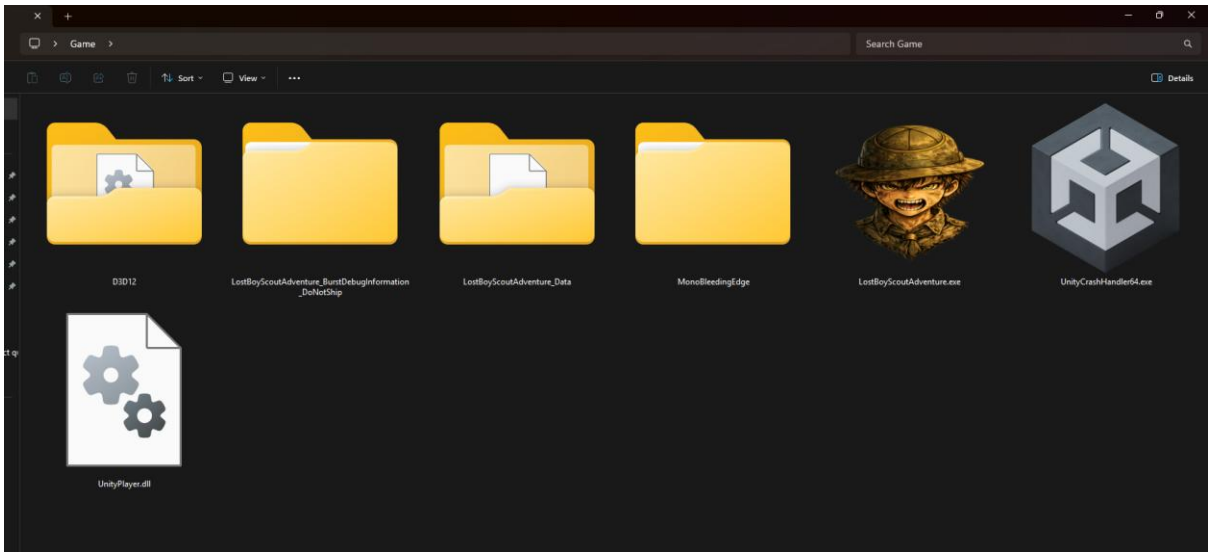

Figure 6: Unity Editor Project Evidence

This screenshot shows the scout using or approaching shelter as part of the survival and safe-zone behaviour.

Figure 7: Shelter Safety Evidence

This screenshot shows the main menu used to access play mode, AI mode, options, and help features.

Figure 8: Main Menu Screen

This screenshot shows the normal gameplay scene, including the scout, environment, HUD, resources, and survival indicators.

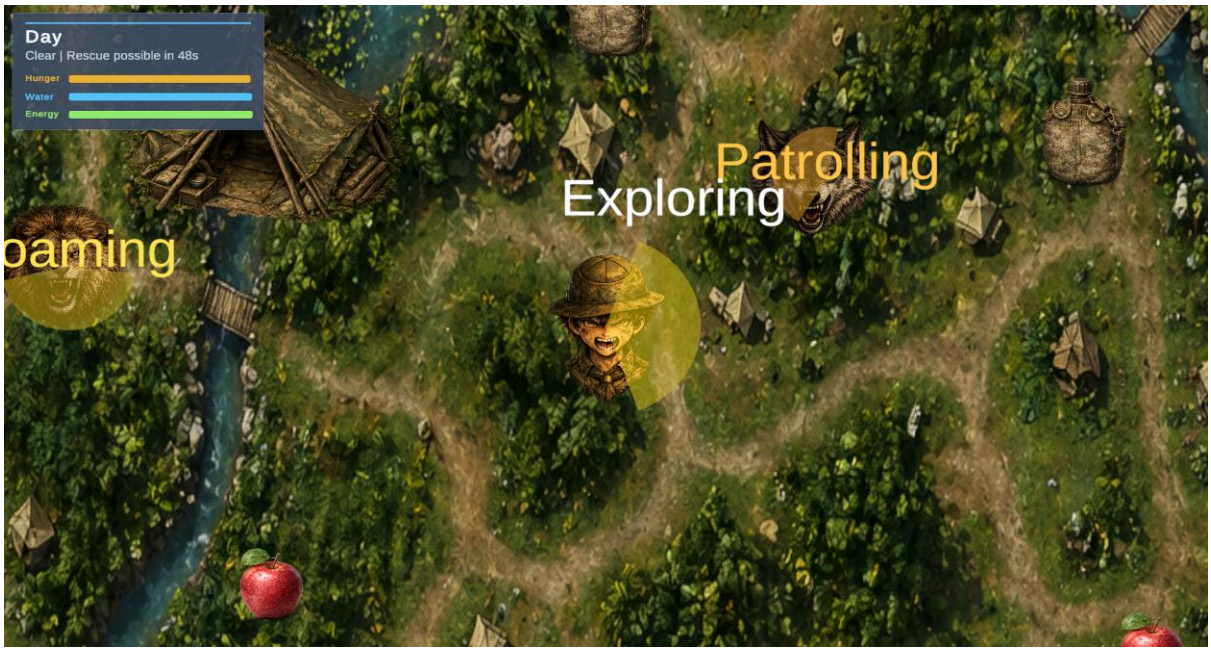

Figure 9: Gameplay Overview

This screenshot shows the scout searching for or moving towards a food resource during survival gameplay.

Figure 10: Scout Searching for Food

This screenshot shows the scout searching for or moving towards a water resource during survival gameplay.

Figure 11: Scout Searching for Water

This screenshot shows a wolf or lion danger situation where predator presence affects the scout's behaviour.

Figure 12: Predator Danger Behaviour

This screenshot shows the scout escaping from predator danger while the escape behaviour is active.

Figure 13: Escape Behaviour

This screenshot shows a successful game outcome, such as exit success or helicopter rescue success.

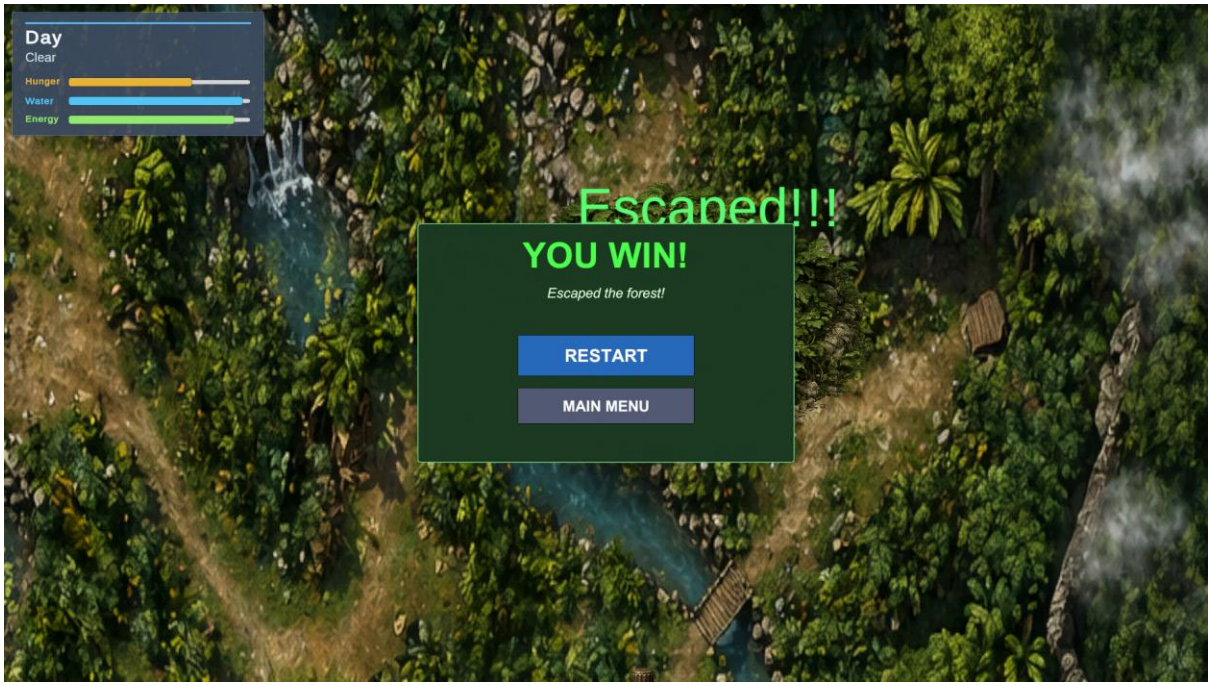

Figure 14: Win Screen

This screenshot shows a failure outcome, such as predator capture, starvation, dehydration, or exhaustion.

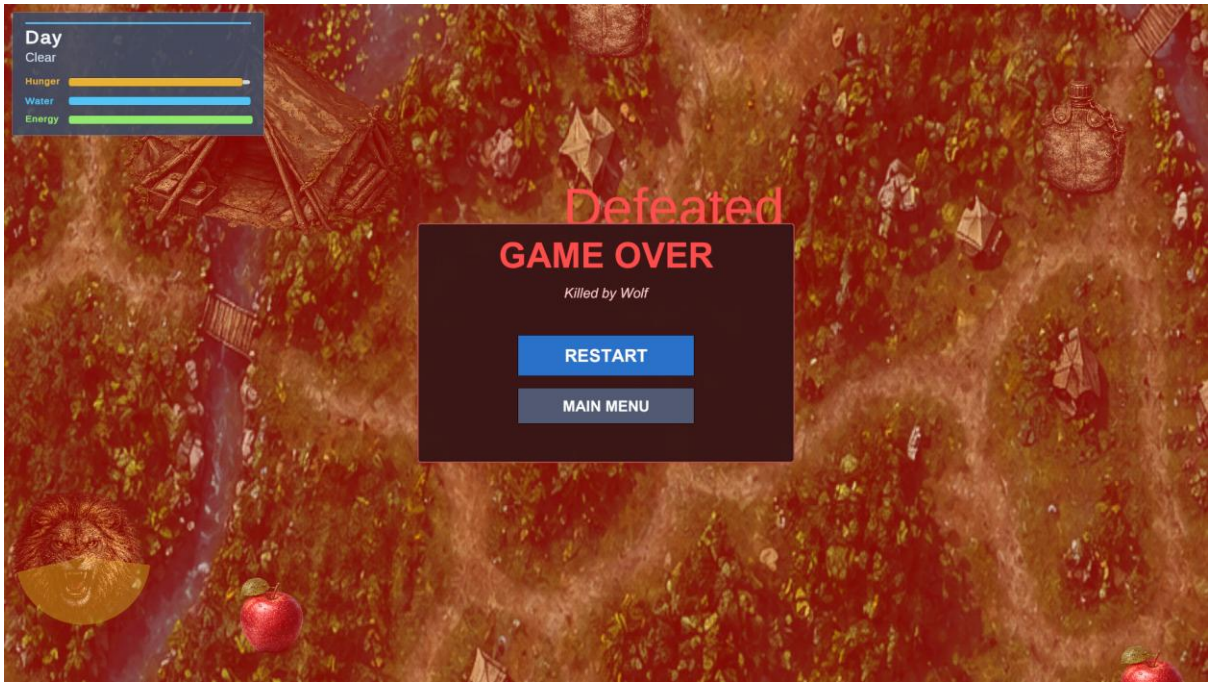

Figure 15: Lose Screen

This screenshot shows the AI state label, reason text, debug panel, or other explainability evidence used during testing.

Figure 16: Debug AI State Panel

This screenshot shows the night condition and how it changes the gameplay presentation or decision context.

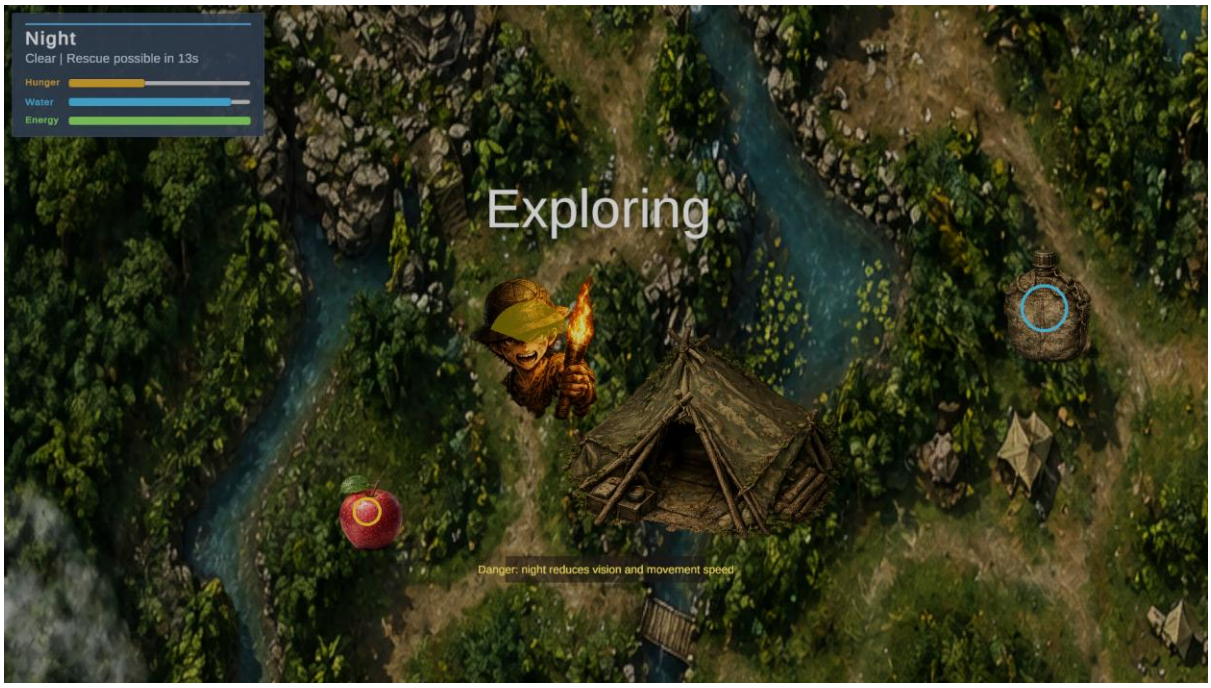

Figure 17: Night Gameplay Evidence

Appendix C: Diagram Sources

Editable draw.io files live under documentation/diagrams/final/. Submission mirrors are also available on Google Drive using the same filenames.

Report Figure	Repository Path	Google Drive
Figure 1	documentation/diagrams/final/scout_state_transition.drawio	https://drive.google.com/file/d/1tW5a2-bBY_4BPFv3yEJ9KEF_PGJ_xlfq/view?usp=sharing
Figure 2	documentation/diagrams/final/class_diagram.drawio	https://drive.google.com/file/d/1Ha8Dbp-wRtvvjm1BWeQiZ41EjGKZt7fy/view?usp=sharing
Figure 3	documentation/diagrams/final/scout_gameplay_loop_game_flowchart.drawio	https://drive.google.com/file/d/15TmlYgeITaejQ1kMq2wng69BbECL4Pbr/view?usp=sharing
Figure 4	documentation/diagrams/final/scout_peas_model.drawio	https://drive.google.com/file/d/1LmGUCKlwtq6zti6K2FUdJ1ImCSD1SCqV/view?usp=sharing
Figure 5	documentation/diagrams/final/predator_agents_peas_model.drawio	https://drive.google.com/file/d/1n-JIir-TUiyKf3wVgvKyNWWBi78pfefF/view?usp=sharing

Table 15: Repository Path Table

Appendix D: Final Preparation and Task Planning Evidence

D.1 Project Communication Evidence

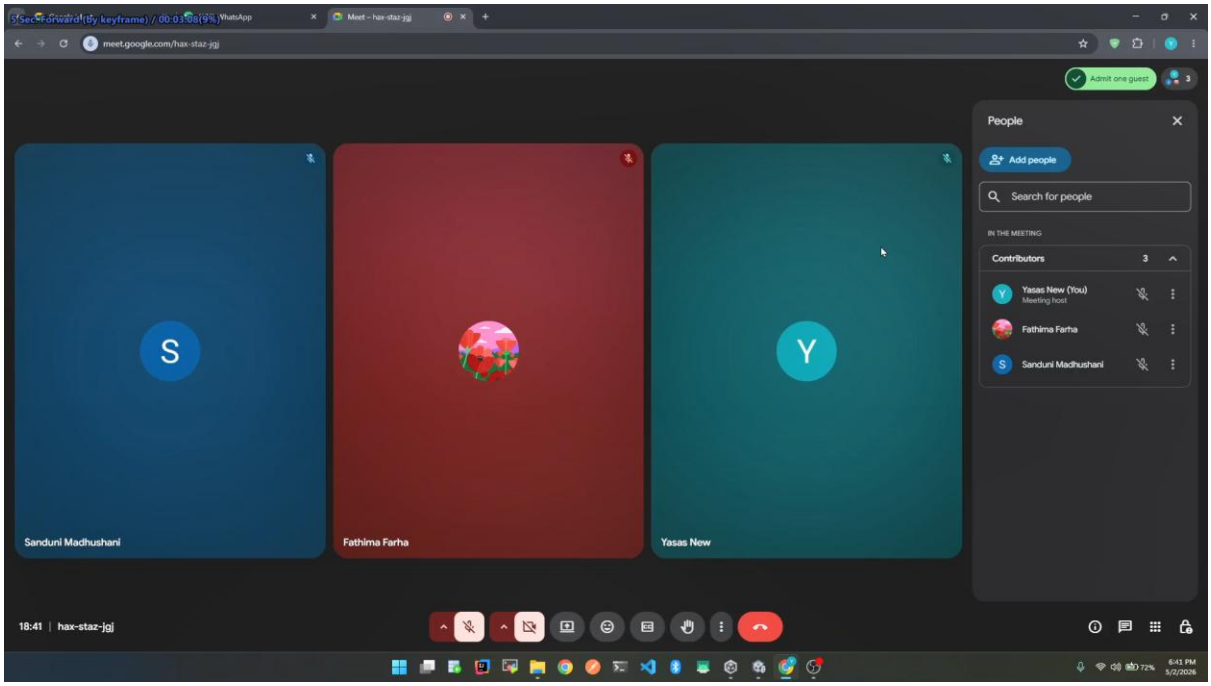

Figure 18: Project Communication Evidence

D.2 Task Planning and Evidence Record

AI Coursework 2 - Group Task Allocation and Contribution Evidence					
Scout vs Wolf: Intelligent Agent Survival Game					
Member	**Student ID**	**Planned Responsibility**	**Expected Output**	**Evidence / Status**	**Notes**
Yasas Pasindu Fernando	25026764	Main Unity development; scout AI; wolf and lion predator behaviour; environment setup; helicopter rescue logic; win/lose logic; testing, documentation, diagrams, user manual, asset preparation.	Playable Unity artefact, final technical report, diagrams, testing evidence, user manual, screenshots, packaged build.	Project files, screenshots, final report, user manual, diagrams, build evidence.	Main technical development and final preparation completed.
Farthima Farha	25026667	Planned project support, communication, review, and discussion input during group task planning.	Support for planned project tasks where available.	WhatsApp / Google Meet / task allocation evidence where available.	Included in task planning record. Do not state technical implementation unless supported by evidence.
Kosma Senuri Upadhya	25026728	Planned project support, communication, review, and discussion input during group task planning.	Support for planned project tasks where available.	WhatsApp / Google Meet / task allocation evidence where available.	Included in task planning record. Do not state technical implementation unless supported by evidence.
Note					
This sheet records planned coordination and contribution evidence for the group coursework submission.					

Figure 19: Task Planning and Evidence Record

D.3 Supporting Notes

This appendix provides supporting evidence related to final preparation, communication, and task organisation. It is included to show how the remaining work was planned and managed during the final submission stage.

The evidence in this appendix should be read together with the main report, Appendix E repository/build links, the Game User Manual, and the Supporting Document: Individual Development and Contribution Report.

Appendix E: Packaged Application Evidence

This appendix provides access information for the packaged game artefact and the Unity project source files submitted with this coursework.

Source Code Repository:
https://github.com/YasasPasinduFernando/LostBoyScoutAdventures

Packaged Windows Build:
https://drive.google.com/file/d/1ZMVtPFK3nxclhdXCmvtZDC6a9vMI5vYF/view?usp=sharing

The source code repository contains the Unity project files, scripts, scenes, assets, diagrams, and supporting development evidence where applicable. The packaged Windows build contains the exported playable version of the game.

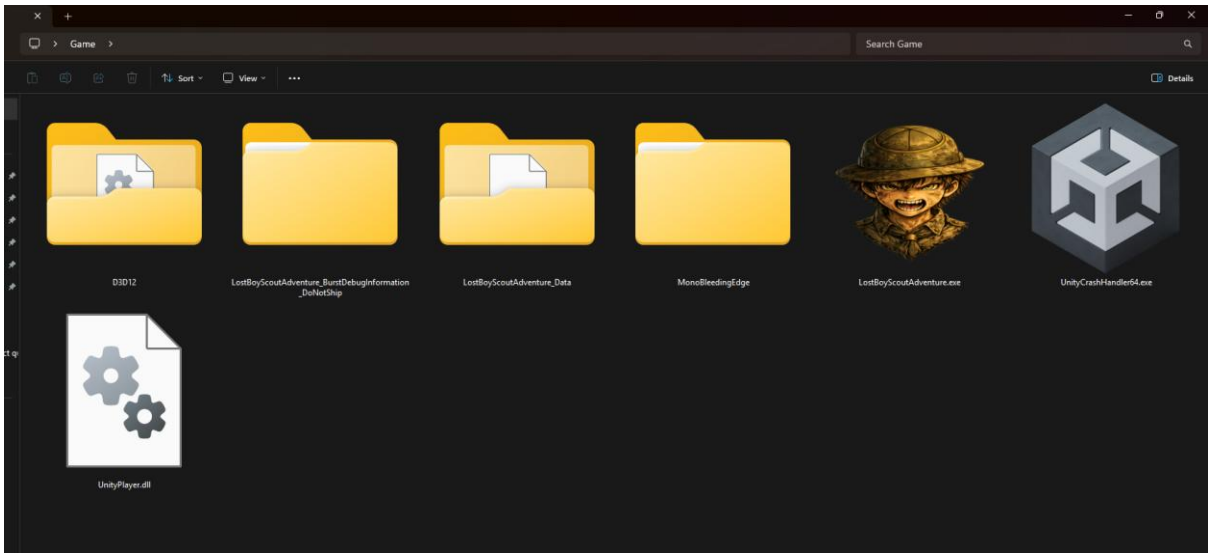

Figure 20: Packaged Windows Build Evidence

This screenshot shows the exported Windows build folder for the game. To run the game, open the build folder and double-click the game executable file, LostBoyScoutAdventure.exe. The UnityPlayer.dll file and the accompanying data folder must remain in the same directory as the executable. If these files are moved or deleted, the game may not launch correctly.

Appendix F: Game User Manual

This manual explains how to launch and use the Lost Boy Scout Adventure / Scout vs Wolf: Intelligent Agent Survival Game. It supports the final artefact submission by guiding assessors and users through the main menu, game modes, HUD indicators, manual controls, AI play mode, win/lose conditions, debug options, and cheat-code features used for testing and demonstration.

The user manual includes guidance on:

- System requirements for running the Unity build
- Main menu navigation
- Level selection
- Options and difficulty settings
- Manual play controls
- AI play mode
- HUD indicators such as hunger, water, energy, weather, and rescue timer
- Game mechanics, including food, water, shelter, predators, weather, and night effects
- Win conditions through helicopter rescue or exit discovery
- Debug mode and cheat-code support for testing
- Important notes for running and demonstrating the game

This appendix is included to show that the game artefact can be launched, understood, tested, and demonstrated by an assessor without needing additional explanation from the development team.

Purpose: User guidance, gameplay explanation, testing support, and demonstration evidence

Related evidence: Screenshots of the main menu, HUD, gameplay states, AI mode, rescue flow, and cheat-code help screen are included in the user manual.

USER MANUAL: LOST BOY SCOUT ADVENTURE

Supporting Document for Scout vs Wolf: Intelligent Agent Survival Game

1. Introduction

Lost Boy Scout Adventure is a 2D adventure survival game. You play as a young Boy Scout who has lost his way in a dense, dangerous forest. To survive, you must manage your basic needs, avoid deadly predators, and find a way to escape before it is too late.

Figure 21: Game Background

2.0 System Requirements

To ensure a smooth and optimal gameplay experience, your system should meet the following minimum requirements.

- Operating System: Windows 10 / Windows 11 (64-bit)
- Processor: Intel Core i3 or equivalent AMD processor
- Memory (RAM): 4 GB RAM minimum
- Graphics: Standard integrated graphics or dedicated GPU
- Storage: Minimum 500 MB of available space
- Controls: Standard Keyboard and Mouse

3.0 Getting Started: The Main Menu

When you launch the game, you will be greeted by the Main Menu. This menu acts as your primary navigation hub to configure your game session.

Figure 22: Main Menu Screen

3.1 Choosing a Play Mode

- PLAY MODE: Choose either PLAY (AI) for an automated simulation or MANUAL PLAY to control the Scout yourself.

3.2 Selecting a Level

The game features progressive difficulty through its level design.

- SELECT LEVEL: Choose between Level 1 (Hunted by a Wolf) or Level 2 (Hunted by a Wolf and a Lion).
    - Level 1 is highly recommended for beginners to understand basic mechanics.
    - Level 2 introduces a secondary apex predator, drastically increasing the difficulty and requiring advanced spatial awareness.

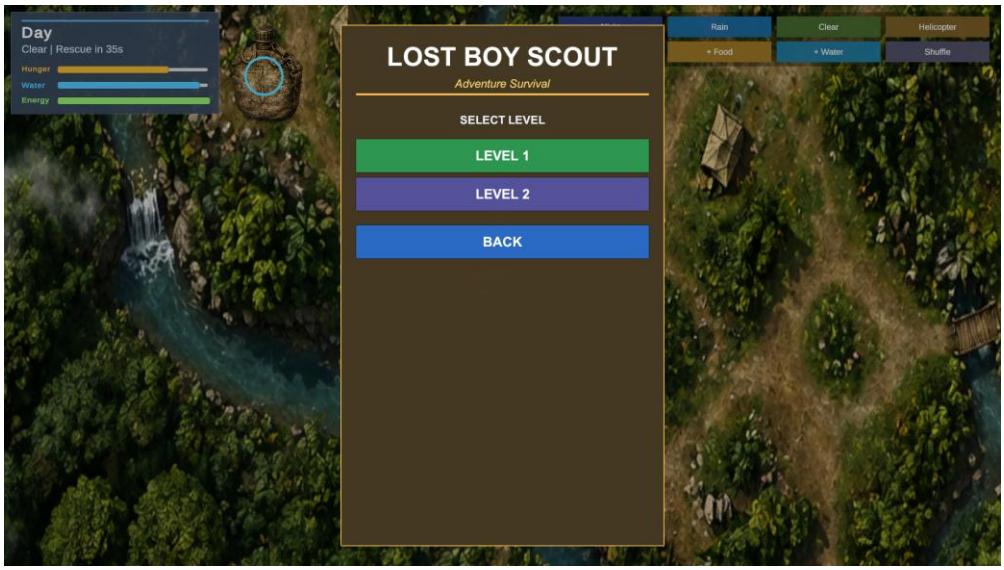

Figure 23: Level Screen

3.3 Accessing Options & Quitting

- OPTIONS: Adjust game difficulty and settings
- **HELP:** Opens the "How to Play" quick guide, which contains information on game modes, controls, survival rules, and cheat codes.
- QUIT: Exit the game.

Figure 24: Pause Screen

4. Options Menu

Navigate to the Options menu to customize your gameplay experience:

- **Difficulty:** Select between EASY, NORMAL, or HARD.
- **Auto Rescue / Auto Food+Water / Fix Energy:** Toggle these helper settings ON or OFF.
- **Debug Mode:** Turn ON for game testing features (See Section 10).
- **SOUND:** Toggle the game's audio ON or OFF.

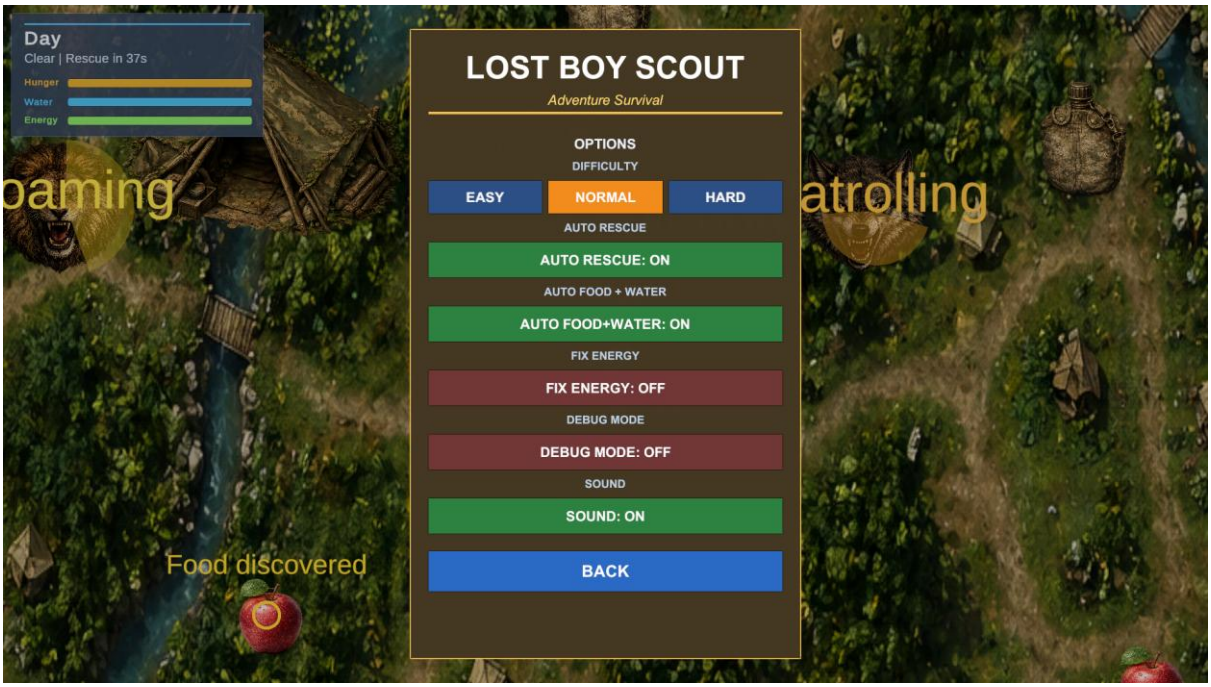

Figure 25: Options Menu

5. Game Interface (HUD)

Understanding the Heads-Up Display (HUD) is crucial for survival. The HUD is located at the top-left corner of the screen.

- **Day/Time & Weather:** Displays the current time of day (Day/Night), weather conditions (Clear/Rain), and the Rescue Countdown Timer.
- **Hunger Bar (Orange):** Depletes over time. Find and eat Food (Apples) to restore it. Warning: Find food immediately if it gets too low.
- **Water Bar (Blue):** Depletes over time. Find Water (Canteens/Flasks) to restore it.
- **Energy Bar (Green):** Depletes as you move. To recover energy, you must keep the Scout **idle** (stand still for a moment).

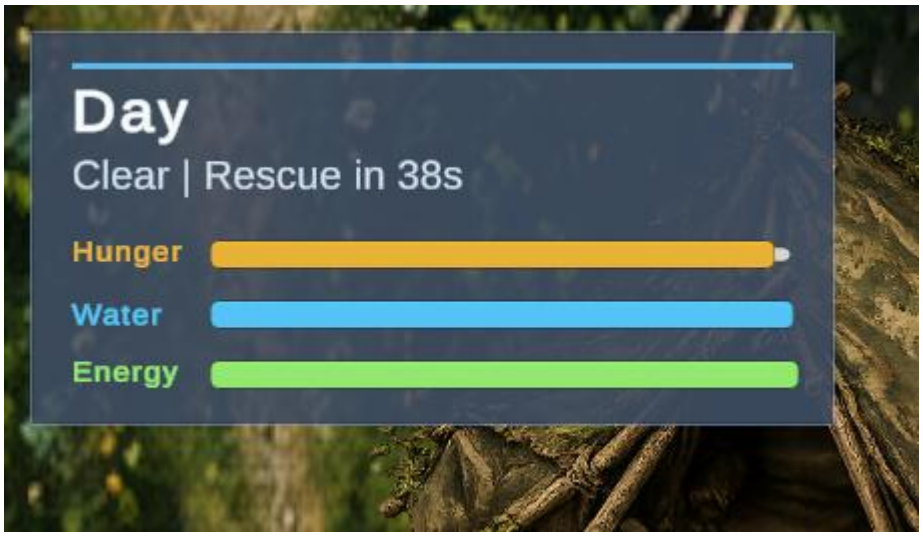

Figure 26: HUD Display

6. How to Play (Manual Mode)

In **MANUAL PLAY**, you have full control over the Scout's movements.

Controls

Action	Key/Input
Movement	Arrow Keys (Up, Down, Left, Right)
Interact / Collect	Automatic (Simply walk into Food, Water, or the Tent)
Pause Game	Esc Key

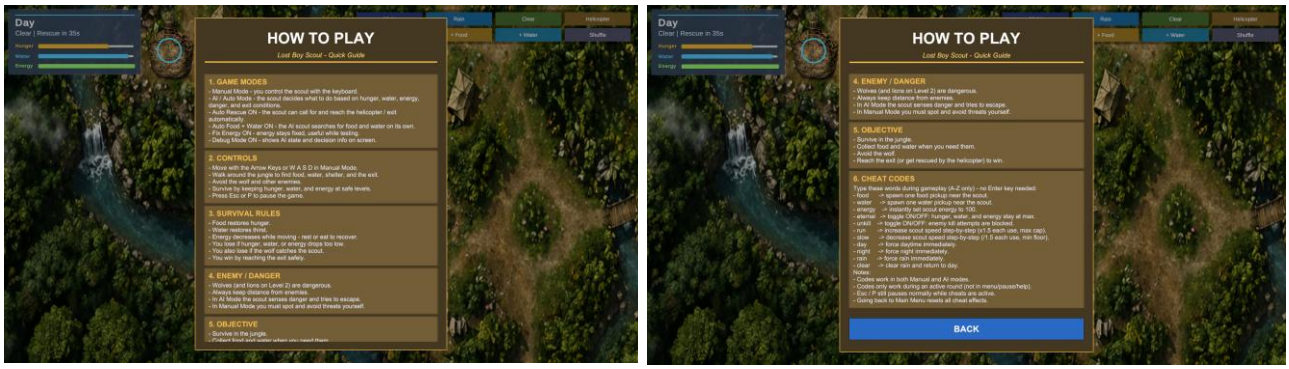

Figure 27: Guide Menu

7. Game Mechanics & Survival

To win, you must navigate the environment and utilize the items around you.

- **Finding Resources:** Walk towards Apples to reduce hunger and Canteens to reduce thirst.
- **Using Shelter:** Walk into the Tent to hide. The tent serves as a safe zone from predators and provides shelter during harsh weather.
- **Day/Night Cycle:** When night falls, your vision and movement speed are reduced. The Scout will automatically equip a torch to see in the dark.
- **Weather Effects:** Rain will also reduce your vision and movement speed. Seek shelter when it rains.

- **Predators:** Wild animals (Wolves and Lions) are patrolling the forest. If they spot you, they will hunt you. If caught, it is **GAME OVER**.

8. Play (AI) Mode

In **PLAY (AI)** mode, the game plays itself using an AI decision system designed to survive.

- **Behavioral Avoidance:** The AI recognizes animal movement patterns and actively flees from danger.
- **Advanced Memory System:** If the AI spots food or water while being chased, it stores that location in its memory. Once the threat is gone, the AI will use this memory to navigate back and collect the resources.

Figure 28: AI Mode

9. How to Win

There are two ways to escape the forest and win the game:

Method 1: Helicopter Rescue

Survive until the Rescue Timer in the top-left corner reaches zero.

1. The message "Helicopter rescue is now possible" will appear.
2. A green rescue circle will spawn on the map.

3. Move into the circle and hold your position until the helicopter arrives to pick you up.

Figure 29: Helicopter Rescue

Method 2: Find the Exit

Explore the map to locate the hidden "EXIT" path/cave. Reaching this location will instantly secure your escape.

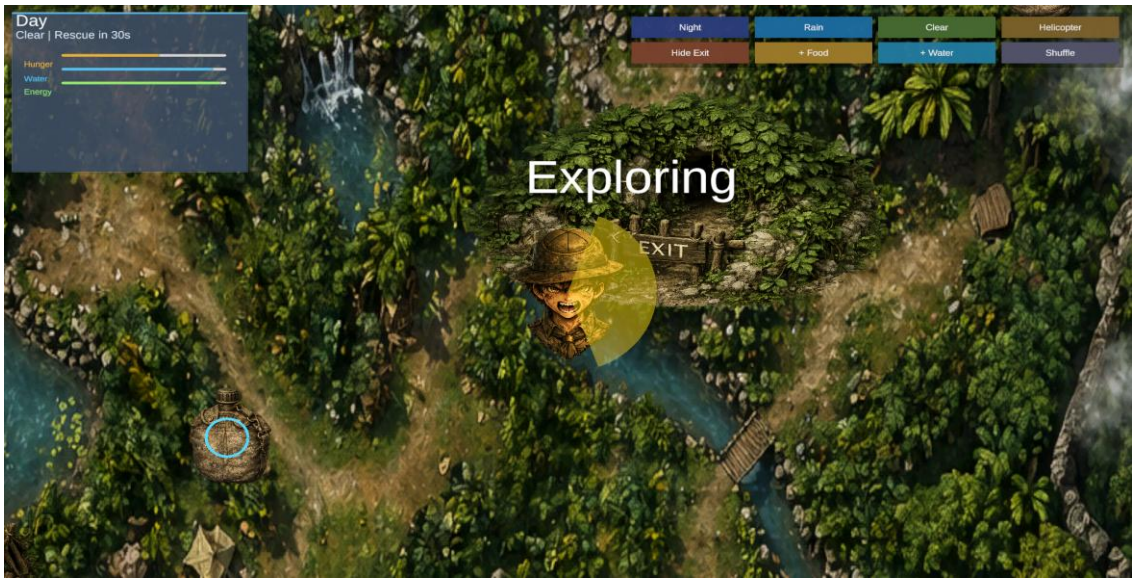

Figure 30: Exit

10. Debug Mode (For Testing)

If **Debug Mode** is turned ON from the Options menu, a special control panel will appear at the top of the screen during gameplay (available in both AI and Manual modes). This is used for testing game mechanics.

- **Controls include:** Toggle Day/Night, Start/Stop Rain, Show/Hide Exit, Add Food/Water, and instantly call the Helicopter.

Figure 31: Toggle Screen

11. In-Game Cheat Codes

Cheat codes are provided as testing and demonstration support features. They allow the game mechanics to be checked quickly during development, debugging, and assessment demonstrations. These codes work during an active game round only and are available in both Manual and AI modes. No Enter key is required after typing the code.

Cheat Code	Effect
food	Spawns one food pickup near the scout.
water	Spawns one water pickup near the scout.
energy	Instantly sets the scout's energy to 100.
eternal	Toggles maximum hunger, water, and energy ON/OFF.
unkill	Toggles blocking of enemy kill attempts ON/OFF.
run	Increases scout speed step-by-step, up to the maximum cap.
slow	Decreases scout speed step-by-step, down to the minimum floor.
day	Forces daytime immediately.
night	Forces night immediately.
rain	Forces rain immediately.
clear	Clears rain and returns the environment to daytime.
heli	Calls the helicopter and starts the rescue flow.
cocacola	Sets the water bar to 100.
kfc	Sets the hunger bar to 100.

Table 16: In-Game Cheat Codes

Important Notes:

- Cheat codes work in both Manual and AI modes.
- Cheat codes work only during an active round.
- Cheat codes do not work in the menu, pause screen, or help screen.
- Esc or P still pauses the game normally while cheats are active.
- Returning to the Main Menu resets all active cheat effects.
- These codes are included for testing and demonstration purposes and should not be considered part of the normal survival challenge.

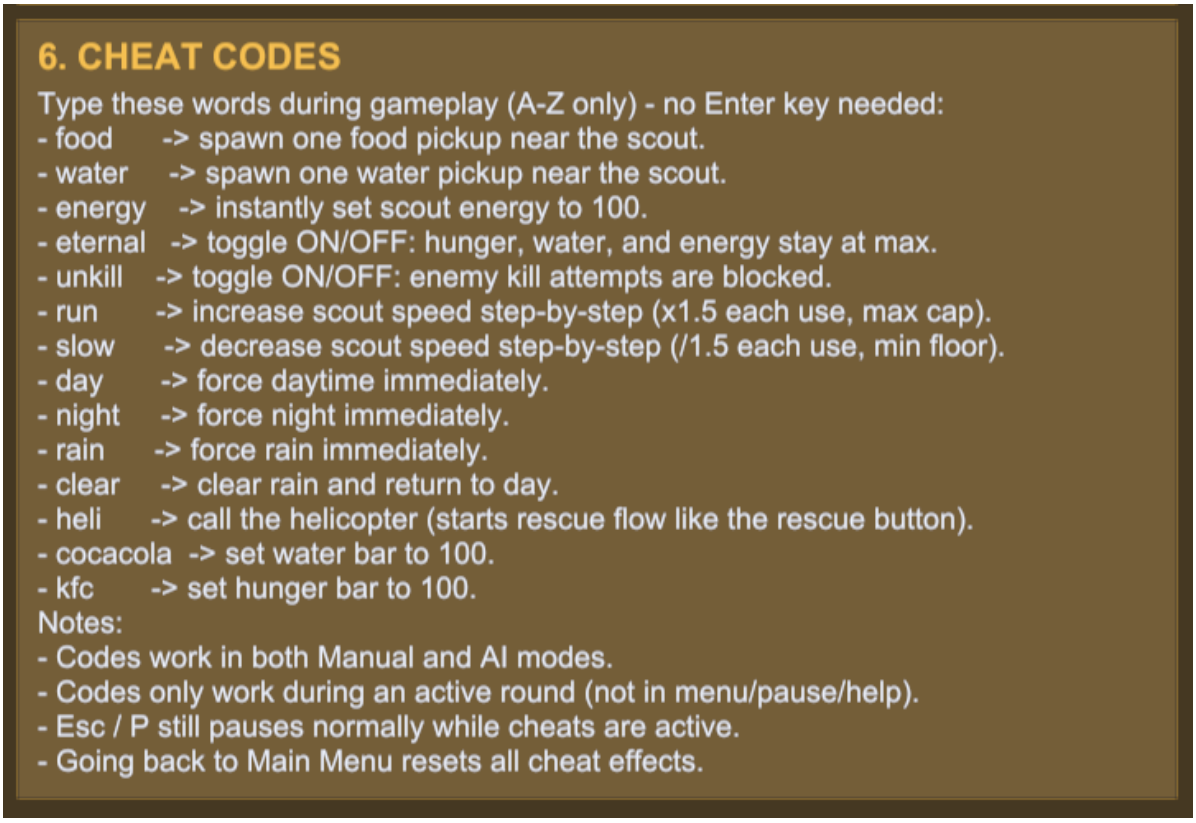

Figure 32: References

Supporting Document: Individual Development and Contribution Report

1. Introduction

This supporting document is attached to the main technical report for Lost Boy Scout Adventure / Scout vs Wolf: Intelligent Agent Survival Game. Its purpose is to record the author's individual development responsibility, contribution, and final preparation work for the Artificial Intelligence coursework artefact in a clear, professional, and evidence-based way.

Although the coursework was initially arranged as a group artefact, the author was later permitted by the module lecturer to proceed individually due to the limited remaining time and practical difficulties with the original group arrangement. Therefore, this supporting document records the author’s individual development responsibility and final preparation work.

The author made the main technical contribution to the development and final preparation of the coursework artefact. This included the core Unity game development, scout intelligent-agent behaviour, predator-related gameplay logic, environment setup, helicopter rescue logic, win and lose conditions, survival systems, testing, documentation, and final report preparation.

The contribution described in this document also includes diagrams, user manual preparation, supporting asset preparation, source-note checking, final preparation communication, and remaining work organisation. Visual assets and supporting UI elements were prepared using standard digital editing tools and legally sourced open-license materials. External audio and effect sources such as Mixkit and EventLabs were also used or referenced where relevant.

Final preparation communication, lecturer guidance, and task planning evidence is included where available, such as lecturer approval communication, project communication records, and task planning material.

This report should be read together with the main technical report, the game user manual, diagrams, screenshots, test evidence, source-code references, and packaged application evidence. Together, these items show how the final coursework artefact was planned, implemented, tested, documented, and prepared for submission.

2. Project Context

The project is a Unity 2D intelligent-agent survival game titled Lost Boy Scout Adventure / Scout vs Wolf: Intelligent Agent Survival Game. The game places a scout in a jungle environment where the scout must survive by managing hunger, water, and energy while responding to predators, rain, night conditions, shelter, an exit route, and helicopter rescue.

The artefact was designed as a game-based AI project rather than a chatbot-style project. This was important because game AI can be observed directly through movement, decisions, state labels, reactions, and win or lose outcomes. In the final artefact, the assessor can see the scout searching for resources, escaping danger, using shelter, moving towards an exit, or reaching a rescue outcome.

The game includes different levels of predator pressure. Level 1 includes a wolf predator, which creates a clear survival threat for the scout. Level 2 increases the challenge by including both wolf and lion predators. This level structure supports stronger demonstration value because the scout has to respond to a more demanding environment while still managing hunger, water, energy, weather, and possible outcomes.

The AI mode uses priority-based behaviour-tree style logic. The scout does not follow a fixed video-like route. Instead, decisions are selected at runtime using conditions such as predator danger, hunger, water level, energy, shelter need, exit availability, and rescue status. Manual mode also allows player control, which supports testing and demonstration of the same gameplay systems from a different control perspective.

Figure 33: Game Overview and Survival Environment

3. Personal Motivation and Project Selection

Many coursework teams selected chatbot-style artefacts because chatbots are familiar examples of AI systems and can be explained quickly. The author selected and supported a game-based intelligent agent because a game can show perception, decision making, search, danger response, and alternative outcomes in a more visible way.

The author had a long-term interest in game development from a young age. The coursework created a practical opportunity to connect that interest with Artificial Intelligence concepts and Unity development. This motivation was professional rather than emotional, because the project was still shaped around the coursework brief, marking criteria, and technical evidence required for assessment.

The game-based direction also helped the author explore a relevant area of the software industry. Games and interactive systems require real-time decisions, user feedback, event handling, asset integration, testing, and documentation. These areas connected well with the module topics because the scout could be designed as an autonomous agent that senses the environment and selects actions under changing conditions.

This project selection supported the assessment because the AI behaviour could be demonstrated visually. The scout's actions are not hidden inside text responses. Instead, decisions become visible through movement, HUD feedback, state labels, predator avoidance, search behaviour, and different outcomes

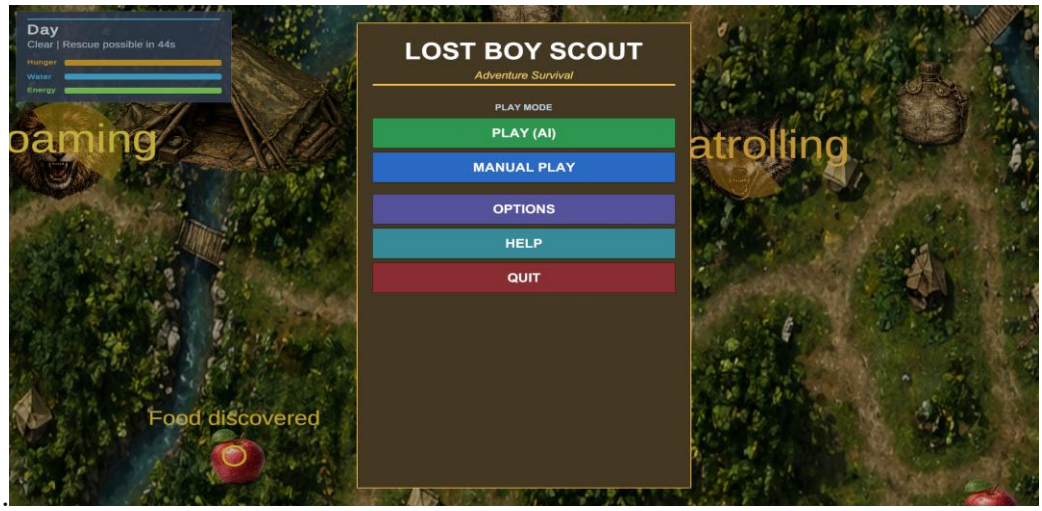

Figure 34: Main Menu and Game Mode Selection

4. Summary of Main Contribution

Table IC1 summarises the main contribution areas and the evidence that can support each area.

Area	Contribution Summary	Evidence / Figure Reference
Concept and motivation	Selected and supported a game-based intelligent-agent artefact that could demonstrate AI visually.	Main report motivation section, project concept notes
Game scenario design	Designed the jungle survival scenario around hunger, water, energy, resources, predators, shelter, exit, and rescue.	Main report Sections 3 to 5, Figure IC1
Level 1 and Level 2 predator design	Prepared the final predator-related gameplay behaviour for the Level 1 wolf scenario and the Level 2 wolf and lion variation.	Gameplay screenshots, Figure IC3, Figure IC4
Core Unity development	Completed the main Unity implementation and final integration of gameplay systems for the artefact.	Unity project files, playable build evidence
Unity scene and environment setup	Prepared and refined the 2D scene structure, object placement, environment layout, resources, shelter areas, exit context, and gameplay flow.	Unity project files, scene screenshots
Scout intelligent-agent logic	Took responsibility for the scout's survival stats, action selection, state labels, perception links, and runtime behaviour.	ScoutAI, ScoutBrain, debug screenshots

Behaviour-tree / priority-based decision logic	Refined the final AI approach around priority-based behaviour-tree style decision making.	BehaviorTree, ScoutBrain.BuildRoot(), pseudo-code in Section 6
Perception system	Supported limited sensing through vision, hearing, resource detection, and distance checks.	ScoutPerception, perception tests
Search and memory logic	Supported resource search, remembered targets, danger memory, stale target handling, and fallback exploration.	ScoutMemory, ScoutBrain, ItemPickup
Wolf and lion predator behaviour	Completed and integrated predator-related gameplay behaviour including detection, chase, catch, investigate, and roam-style pressure where used in the final artefact.	WolfAI, LionAI, WolfBrain, gameplay tests
Helicopter rescue logic	Supported rescue timing and rescue outcome behaviour as part of the game outcome system.	GameManager, user manual, test evidence
Win and lose logic	Helped connect exit win, helicopter rescue, starvation, dehydration, exhaustion, wolf capture, and lion capture outcomes.	GameManager, test evidence
UI and HUD feedback	Supported hunger, water, energy, weather, rescue, state labels, reason text, and debug feedback.	HUD screenshots, Figure IC5
Manual mode and AI mode support	Supported both manual play and autonomous AI mode for testing and demonstration.	User manual, main menu screenshots

Debug/cheat testing support	Supported cheat-code and debug features used for repeatable testing and demonstration.	CheatCodeManager, help screen screenshots
Assets and image generation	Supported image and asset preparation by creating manual 2D sprites and integrating royalty-free assets appropriate for the coursework.	Asset folders, source notes
Audio/effects sourcing	Supported external audio/effects sourcing and integration where relevant, including Mixkit/EventLabs source notes.	Mixkit/EventLabs references, audio assets
Diagrams and report documentation	Prepared and organised diagrams, report sections, tables, appendices, and references.	Draw.io files, main report, appendices
User manual preparation	Prepared the separate game user manual to explain launching, controls, AI mode, HUD, outcomes, debug features, and testing support.	User_Manual_AI_Coursework 2_Formatted.docx, Appendix F
Testing and debugging	Performed repeated scenario testing across resources, predators, weather, outcomes, UI, audio, and build behaviour.	Test plan, screenshots, build evidence
Final preparation communication and task planning	Maintained final preparation communication, followed lecturer guidance, organised remaining work, and supported task tracking.	Project communication, task planning record

Table 17: Summary of Individual Contribution

5. Concept Design and Scenario Development

I contributed to shaping the jungle survival concept so that it matched the Artificial Intelligence coursework requirements. The scout was selected as the main agent because the character naturally has survival goals. Hunger, water, and energy create internal pressure, while predators, rain, night, and limited resources create external pressure.

The game scenario includes food and water pickups, shelter areas, weather and night effects, predator danger, exit discovery, and helicopter rescue. These elements were selected because they create meaningful AI decisions. For example, the scout may need food, but a predator may be close enough to make escape more important. The same system can therefore produce different behaviour depending on runtime events.

Level 1 focuses on the wolf predator. This creates a clear starting challenge where the scout must learn to avoid one main predator threat while managing survival needs. Level 2 increases the difficulty by adding both wolf and lion predators. This gives stronger evidence of dynamic danger because the scout must respond to more than one threat source.

The author also contributed to the alternative outcome structure. Win outcomes include exit success and helicopter rescue. Loss outcomes include starvation, dehydration, exhaustion, wolf capture, and lion capture. This design makes the artefact non-scripted because the final result depends on the scout's decisions, internal status, predator positions, weather, and available routes.

The survival scenario supports AI coursework because it includes perception, decision making, searching, memory, state-based behaviour, and real-world physics indicators such as distance checks, sensing ranges, stat decay, movement changes, and collider or trigger interactions.

Figure 35: Level 1 Predator Scenario

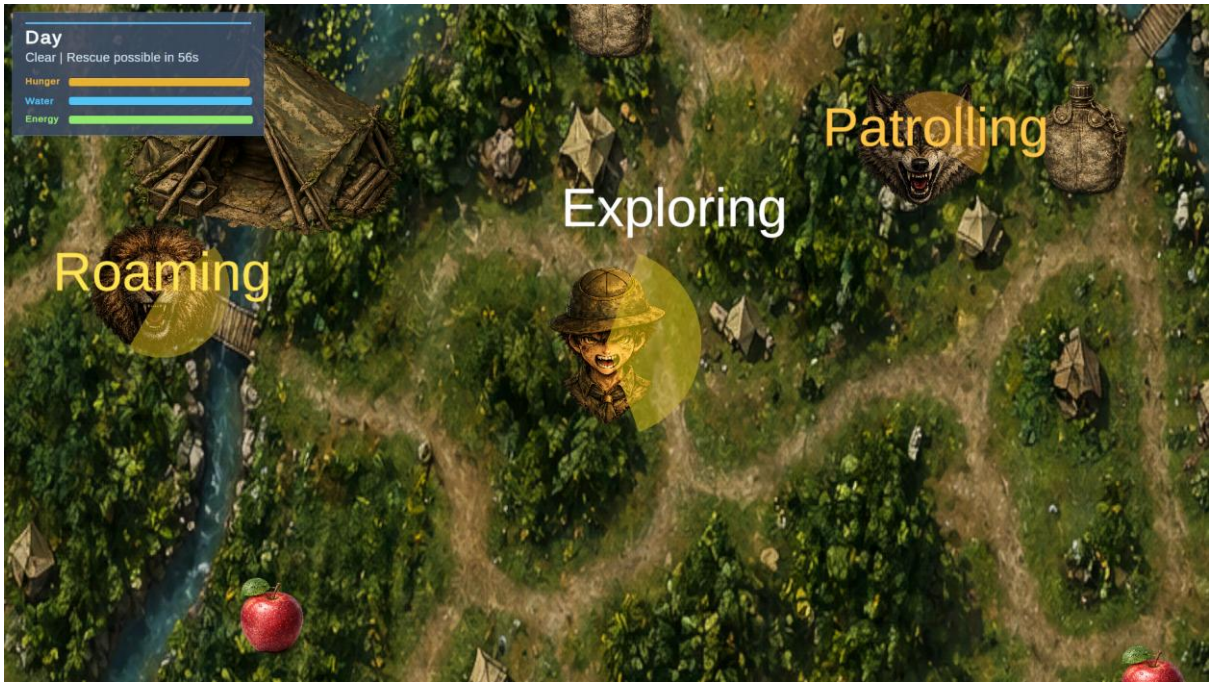

Figure 36: Level 2 Predator Scenario

6. AI Design and Behaviour Logic Contribution

The author initially explored simple state-style behaviour to understand the scout's possible modes, but the final implementation was refined around a priority-based behaviour-tree style decision structure. State labels such as Explore, SearchFood, SearchWater, EscapeDanger, GoToExit, Win, and Lose were kept for debugging, reporting, screenshots, diagrams, and viva explanation.

This wording is important because the project should not be described as a strict finite state machine controller. The final decision process is better explained as priority-based behaviour-tree driven action selection. The state labels help make behaviour understandable, but they are not the core engine that controls every transition.

The behaviour-tree style structure improved the AI because it allowed urgent conditions to interrupt lower-priority behaviours. Predator danger could override resource searching, hunger or water crisis could override normal exploration, and exit or rescue decisions could become active when survival conditions allowed. This made the scout more reactive and reduced the risk of a fixed video-like sequence.

I contributed to organising the decision priorities around survival and danger. The main priority groups can be explained as danger priority, exit or rescue priority, hunger and water priority, energy recovery, shelter response, and default exploration. This made the behaviour easier to test and explain because each visible action had a clear reason.

```
// Simplified priority order for explanation only
if (criticalPredatorDanger)
    EscapeDanger();
else if (exitIsSafeAndAvailable)
    MoveToExit();
else if (hungerIsCritical)
    SearchFood();
else if (waterIsCritical)
    SearchWater();
```

```
else if (energyIsLowAndSafe)
    RecoverEnergy();
else if (rainOrNightRequiresShelter)
    GoToShelter();
else
    Explore();
```

The real implementation contains more checks than this simplified example, but the pseudo-code shows the central idea. The scout selects the highest-priority valid behaviour from current conditions rather than following a fixed route.

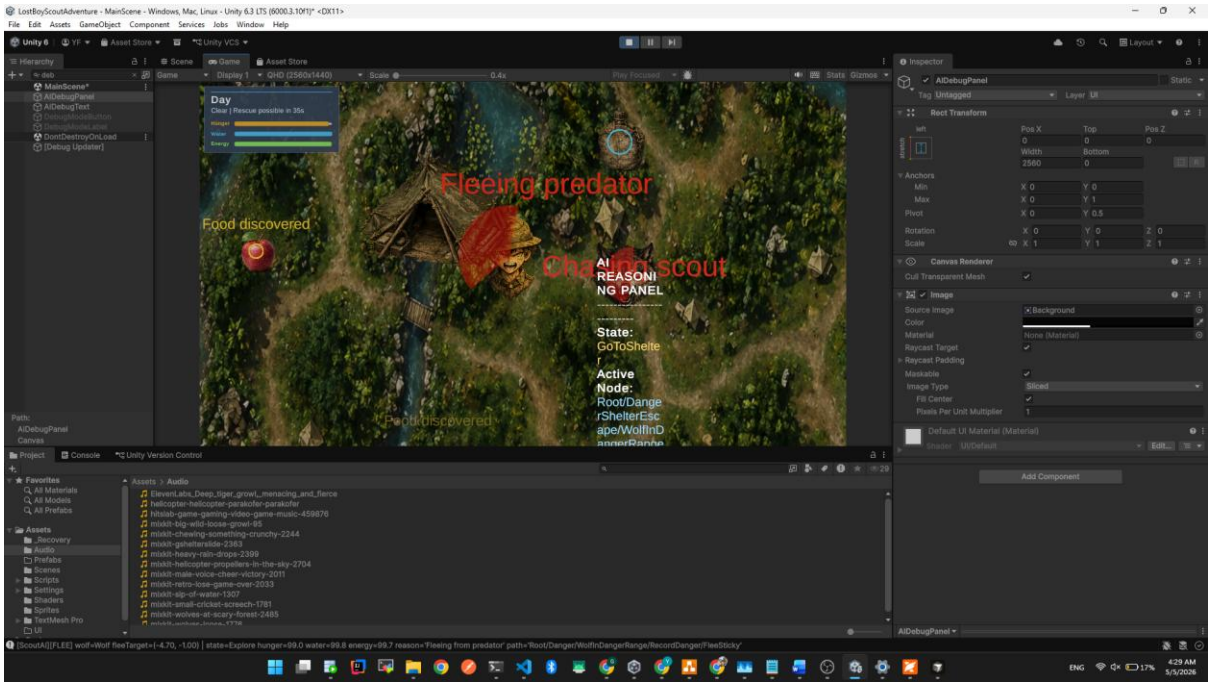

Figure 37: AI State and Debug Feedback

7. Contribution to Scout AI Implementation

I contributed to the core scout AI through scripts such as ScoutAI, ScoutBrain, BehaviorTree, and SteeringAgent. These scripts work together to make the scout an autonomous agent that observes, decides, and acts during gameplay.

ScoutAI acts as the Unity-facing component for the scout. It handles survival stat values, stat decay, trigger interactions, manual and AI mode links, and readable state information for the Inspector or debug UI. This script is important because the decision system needs current gameplay data before selecting a useful action.

ScoutBrain provides the main decision-making logic. It evaluates conditions such as critical danger, predator distance, hunger, water, energy, shelter need, exit availability, and rescue state. I contributed to refining this behaviour so that the scout could react to current events rather than follow one fixed path.

BehaviorTree supports the decision structure through selector, sequence, condition, and action nodes. This structure is suitable for the project because conditions can be re-evaluated during gameplay. As a result, new danger or a sudden survival need can change the scout's selected action.

SteeringAgent helps convert decisions into visible movement. Once the AI selects a target or escape direction, the steering logic supports movement, arrival behaviour, boundary clamping, and stuck recovery. This makes the AI behaviour visible during demonstration because decision making becomes movement on screen.

The author's contribution in this area is central to the coursework because it connects AI theory to a working Unity artefact. The scout's behaviour demonstrates perception, decision making, search, memory, status-based behaviour, and observable state changes.

8. Contribution to Perception, Search, and Memory

I contributed to perception by supporting limited sensing through ScoutPerception. The scout does not have perfect knowledge of the whole map. Instead, it uses vision, hearing, resource detection, distance checks, and trigger interactions to decide what information is currently available.

This supports partial observability, which is important for intelligent-agent coursework. If a resource is not visible or remembered, the scout should not act as if every object is already known. If a predator becomes close enough, the decision logic can respond and change the active behaviour.

Search behaviour was also an important contribution area. The scout can move towards visible resources, use remembered targets, perform emergency searching when survival is critical, and return to exploration when no immediate target is useful. This is heuristic search and steering rather than full A* pathfinding, which is correctly acknowledged in the main report.

Memory was supported through ScoutMemory. The scout can remember resource positions, shelter points, exit-related information, and danger zones. The memory system helps the scout use previous observations, while decay and stale target handling prevent the scout from relying on old information too strongly.

This memory contribution should be described as learning by experience. The project does not train a model using datasets or reinforcement learning. Instead, the scout adapts through remembered observations from gameplay.

```
// Simplified memory idea for explanation only
if (CanDetectResource(targetPosition))
    RememberResource(targetPosition);

if (rememberedTargetIsStillUseful)
    MoveToRememberedTarget();
else
    ExploreForNewTarget();
```

9. Contribution to Predator, Environment, and Outcome Logic

The author completed and integrated the predator-related gameplay behaviour that creates pressure around the scout in the final artefact. Level 1 uses a wolf predator, while Level 2 includes both wolf and lion predators. Predator behaviour contributes to the AI challenge because the scout must respond to danger instead of only searching for resources.

The predator logic is supported by scripts such as WolfAI, LionAI, and WolfBrain. These scripts provide detection, chase, catch, investigate, and roam-style behaviour. Predator mode labels may be used for display or explanation, but the behaviour should be understood as priority-based and behaviour-tree style where applicable, not as proof of a classic FSM system.

During planning discussions, different predator behaviours and catching logic were considered as part of final preparation task planning. However, the final core implementation and integration work presented in this submission was completed by the author. This wording is used to keep the contribution record accurate without assigning unsupported technical implementation to other students.

The author also contributed to environmental behaviour through rain and night effects, shelter areas, resource placement, and general scene preparation. These conditions affect sensing, movement, presentation, and shelter importance. The environment therefore becomes part of the decision context rather than acting only as background decoration.

Outcome logic was another contribution area. The game includes exit win, helicopter rescue win, starvation, dehydration, exhaustion, wolf capture, and lion capture. The author helped align these outcomes with the testing plan, screenshots, user manual, and main technical report.

```
// Simplified terminal outcome guard for explanation only
if (winAlreadyReached || loseAlreadyReached)
    return;

if (exitReached || rescueCompleted)
    WinGame(reason);
else if (predatorCaughtScout || survivalStatReachedZero)
    LoseGame(reason);
```

10. Contribution to Manual Mode, AI Mode, Debugging, and Cheat Support

The author supported both AI mode and manual mode. AI mode demonstrates autonomous scout behaviour through priority-based decision logic. Manual mode allows player control, which helps testers compare normal gameplay systems such as movement, pickups, HUD feedback, shelter use, win conditions, and lose conditions.

Debug and cheat features were included to support repeatable testing and demonstration. These features are not the main AI evidence, but they help verify behaviour more efficiently. For example, stat refill or testing shortcuts can be used to reach particular scenarios without replaying the full game from the beginning each time.

I contributed to documenting these features in the report and user manual so that assessors can understand which features are part of normal AI behaviour and which features are used for testing support. This distinction is important because the coursework evaluation should focus mainly on autonomous AI behaviour while still recognising testing tools as useful evidence.

Figure 38: Debug and Cheat-Code Support

11. Contribution to UI, HUD, Assets, Audio, and Diagrams

The author contributed to UI and HUD feedback so that the scout's behaviour is easier to understand. Hunger, water, energy, weather, rescue timer, win and lose screens, state labels, and reason text help make the AI readable. This is important for assessment because the marker needs to see both what the scout does and why the action was selected.

The author also supported asset preparation and integration. Images and supporting visual material were prepared or refined using AI-assisted image generation and editing tools where appropriate, including DALL·E 3 and Nano Banana. These tools were used as supporting asset-preparation resources rather than as evidence of AI logic inside the game.

External audio and effects sources, such as Mixkit and EventLabs where relevant, were also used or considered for presentation support. The author does not claim unsupported ownership of external media. The contribution in this area is practical sourcing, adaptation, integration, and acknowledgement.

Diagram preparation was another important part of the author's contribution. Draw.io diagrams were prepared and organised for the state transition view, class structure, gameplay flowchart, scout P.E.A.S model, and predator P.E.A.S model. These diagrams help connect the Unity implementation to academic AI concepts.

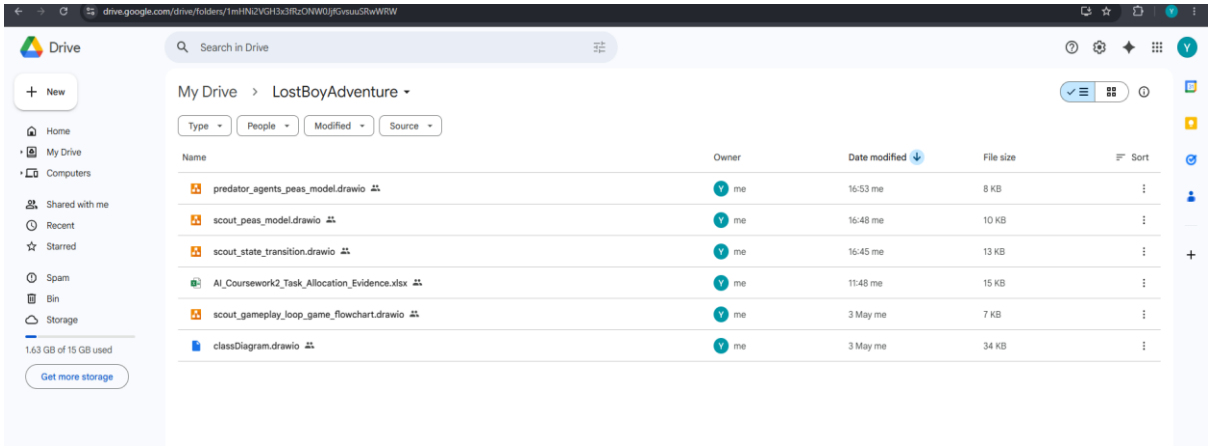

Figure 39: Diagram and Evidence Preparation

12. Contribution to Testing, Debugging, and Final Refinement

The author contributed to repeated testing across multiple gameplay scenarios.. Testing covered food search, water search, predator catch, escape danger, shelter behaviour, exit win, helicopter rescue, starvation, dehydration, exhaustion, night and rain effects, memory search, UI readability, audio behaviour, cheat-code support, and standalone executable behaviour.

Repeated playtesting was needed because the game is not a fixed sequence. A single run could not prove that the AI works across different conditions. The author therefore tested focused scenarios as well as longer play sessions to check whether alternative outcomes could occur naturally.

Debugging focused on both technical errors and behaviour quality. Issues such as stale target movement, unclear UI feedback, unsafe predator decisions, or unreliable outcome triggers could make the AI seem less coherent. I contributed to identifying and refining these issues so that the final artefact behaved more consistently.

Testing evidence also supported the final documentation. The main report, user manual, diagrams, screenshots, appendices, and contribution document were all prepared to help the assessor understand how the artefact works and how it was tested.

Evidence Type	**Purpose**	**Related Section**
Gameplay screenshots	Show AI behaviour, HUD, outcomes, and testing scenarios.	Figures IC1 to IC7, main report Appendix B
Diagrams	Explain AI design, class relationships, game flow, and P.E.A.S models.	Main report Appendix C
Source-code references	Link AI concepts to actual Unity scripts.	Main report Appendix A

Test table	Show checked behaviours and outcomes.	Main report Section 16
User manual	Explain launch process, controls, AI mode, manual mode, HUD, outcomes, and debug features.	Main report Appendix F
Individual submission approval evidence	Supports lecturer guidance, final preparation communication, and task planning record.	Main report Appendix D

Table 18: Evidence Prepared for Assessment

13. Contribution to Documentation, User Manual, and Research

The author took responsibility for preparing and refining the main technical documentation. This included the report structure, academic wording, requirement analysis, P.E.A.S explanation, intelligence trait mapping, state-based behaviour explanation, implementation descriptions, algorithms, testing, evaluation, limitations, references, appendices, and supporting evidence.

Documentation work required careful alignment between the written report and the actual code. The author avoided claiming unsupported features. For example, the report states that search and steering are heuristic rather than full A* pathfinding, and that memory-based learning is not machine learning or reinforcement learning.

The author also prepared the separate game user manual. The manual supports the artefact because it explains how to launch the Unity build, navigate the main menu, choose AI mode or manual mode, understand the HUD, use controls, interpret win and lose outcomes, and use debug or cheat-code features for testing. This helps the assessor run and evaluate the game without needing additional verbal explanation.

Research support included Unity documentation, Microsoft C# documentation, academic or game AI references, and practical learning resources such as Unity tutorials. The author also explained steps, shared technical guidance, and suggested relevant tutorials during development where this supported understanding of Unity, C#, gameplay implementation, and documentation preparation.

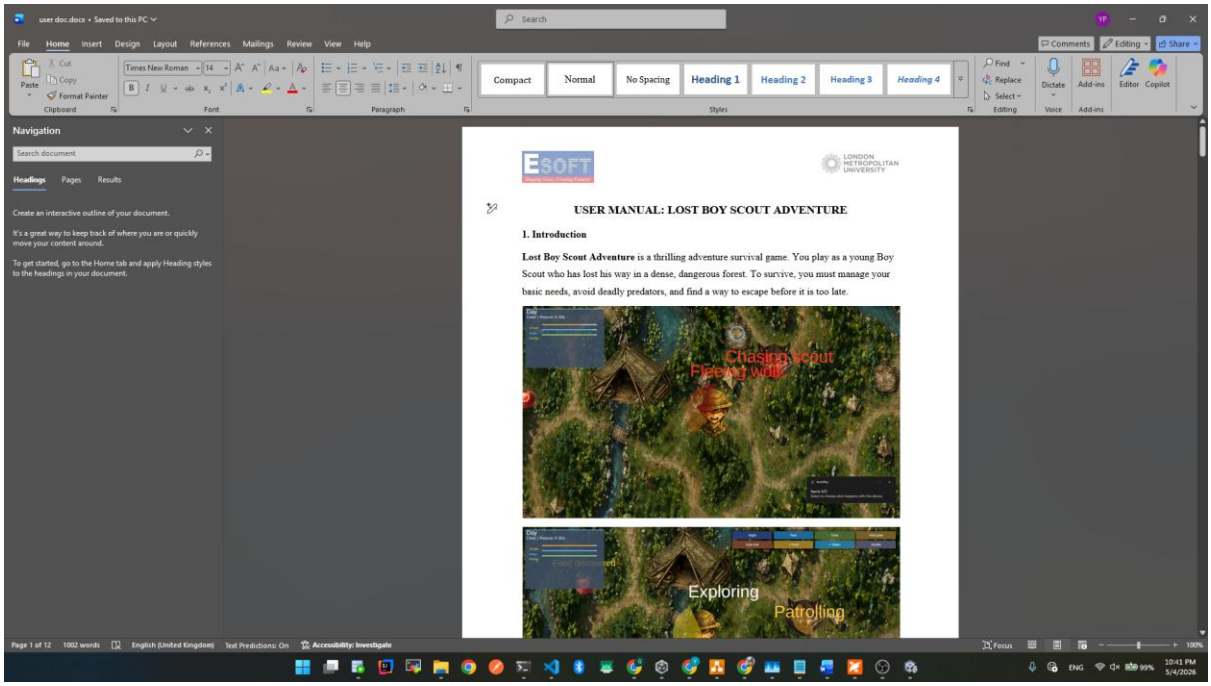

Figure 40: User Manual Supporting Evidence

14. Contribution to Final Preparation Communication and Task Planning

I contributed to final preparation communication, lecturer guidance follow-up, and task tracking support. WhatsApp and Google Meet were used for communication and clarification, and an Excel task planning record was used to organise remaining work areas, expected outputs, evidence, and notes.

The task planning record identifies the author as the main technical developer and final preparation lead. It is used as an evidence record for remaining work organisation and should not be treated as proof of completed technical implementation unless it is supported by project evidence.

During planning discussions, different predator behaviours and catching logic were considered as part of final preparation task planning. However, the final core implementation and integration work presented in this submission was completed by the author.

Coordination support included explaining the project direction, sharing development progress, preparing technical evidence, explaining implementation steps, suggesting tutorials, and connecting project work to report sections. This was useful because the artefact included several linked areas: Unity implementation, AI logic, diagrams, screenshots, testing, the user manual, and final documentation.

Final preparation communication and task planning evidence is included where available and should be interpreted as supporting material for the development process.

15. Challenges, Learning, and Final Preparation

The author faced several technical and preparation challenges during the project. One major challenge was managing several competing decision factors at the same time. Hunger, water, energy, predator danger, rain, night, shelter need, exit availability, and rescue timing can all influence behaviour. This made priority ordering important.

Another challenge was avoiding scripted behaviour. The scout needed to behave like an intelligent agent rather than a pre-recorded sequence. This required runtime sensing, priority evaluation, memory use, search fallback, and alternative outcome handling.

Balancing predator danger also required repeated testing. If predators were too weak, the scout had little reason to escape. If predators were too strong, resource searching and survival planning became less meaningful. I contributed to testing and documenting this balance so that the final game remained demonstrable.

The final preparation period also required careful document work. The author continued the core development work following lecturer guidance during the final preparation period. This included refining the report, correcting the AI architecture description, preparing appendices, linking the user manual, checking references, and ensuring that the individual contribution document was professional and evidence-based.

The project strengthened practical understanding of Unity, C#, game AI, behaviour-tree style decision logic, perception limits, memory systems, testing practice, and academic technical documentation.

16. Supporting Evidence

This contribution report is supported by project evidence included in the main submission package. The evidence helps demonstrate the author's involvement in technical development, testing, documentation, research, user manual preparation, final preparation communication, and remaining work organisation.

The following supporting evidence is included or referenced where available:

- Final main technical report
- Separate game user manual
- Unity project source files
- Build or executable evidence
- Gameplay screenshots for Level 1 and Level 2
- Screenshots of food search, water search, predator danger, escape, shelter, win, and lose outcomes
- Debug panel and state label screenshots
- Draw.io diagram source files and exported diagrams
- Test table and playtesting evidence
- Asset source notes, including AI-assisted image tools where used
- Audio/effects source notes, including Mixkit/EventLabs where relevant
- Final preparation communication evidence
- Project communication evidence
- Task planning / evidence record

17. Summary

The author made a substantial contribution to Lost Boy Scout Adventure / Scout vs Wolf: Intelligent Agent Survival Game. The work covered concept development, core Unity game development, scout intelligent-agent logic, predator-related gameplay behaviour, environment preparation, helicopter rescue logic, win and lose logic, testing, documentation, diagrams, supporting asset preparation, user manual development, final preparation communication, and remaining work organisation. It also included the Level 1 predator scenario and the Level 2 gameplay variation, where the wolf and lion increase the survival challenge.

The strongest contribution areas were the main technical development and final preparation of the artefact. This included the scout intelligent-agent logic, priority-based behaviour-tree style decision structure, survival gameplay systems, predator-related gameplay logic, testing evidence, and academic documentation. The final AI approach is best described as priority-based behaviour-tree style decision logic, with readable state labels used for debugging, reporting, screenshots, diagrams, and viva explanation.

The author also supported asset and source transparency. Visual assets and supporting images were prepared using AI-assisted image generation or editing tools such as DALL·E 3 and Nano Banana where appropriate. External audio and effect sources such as Mixkit and EventLabs were acknowledged where relevant. During development, the author also explained steps, shared technical guidance, and suggested tutorials where useful for project understanding and final preparation.

This supporting document records the author's contribution in an honest, professional, and evidence-based manner. It avoids unsupported claims and direct blame. It also links contribution areas to project evidence that can be reviewed during assessment.