Assignment 3

UNIVERSITY OF WOLLONGONG IN DUBAI

Assessment Cover Sheet

Subject Code:	CSIT356
Subject Name:	Game Engine Essentials
Submission Type:	Assessment 1
Assignment Title:	Frogert - 3D Platformer Game
Student Name:	Inna Grishina, Alim Izteleu , Vadim Ushak
Student Number:	8490697, 8193058 , 8770025
Student Phone/Mobile No.	0556670265
Student E-mail:	ig559@uowmail.edi.au
Lecturer Name:	Dr. HC Lim
Due Date:	25.01.2026
Date Submitted:	25.01.2026

Signature of Student: [illegible]

Optional **Marks:**

Comments:

Lecturer Assignment Receipt (To be filled in by student and retained by Lecturer upon return of assignment)

Subject:	**Assessment Title:**
Student Name:	**Student Number:**
Due Date:	**Date Submitted:**
Signature of Student:	

Student Assignment Receipt (To be filled in and retained by Student upon submission of assignment)

Subject:	**Assessment Title:**
Student Name:	**Student Number:**
Due Date:	**Date Submitted:**
Signature of Lecturer	

Table of content

Frogert - 3D Platformer Game

Game Title: Frogert
Game Genre: 3D Platformer
Platform: Unity 6
Team Members:
Alim Izteleu – 8193058
Inna Grishina – 8490697
Vadim Ushak – 8770025

Game Overview

The game is a three dimensional platformer where the player controls a frog navigating through a dynamic swamp environment. The core gameplay is based on physics driven movement, where the frog jumping ability and overall mobility are directly affected by its current weight and environmental weather conditions. The game world is designed as one large interconnected level divided into multiple locations. Progression through these locations requires the player to manage weight, movement, and weather effects. The player collects fireflies to increase weight and moves continuously to overcome environmental challenges and reach new areas.

The game features unique mechanics including a weight management system, dynamic weather conditions that affect gameplay, and AI-driven firefly collectibles. This combination creates a unique platformer experience focused on environmental problem-solving and strategic movement rather than combat-oriented gameplay.

Concept:

Frogert is a physics-based 3D platformer that challenges players to navigate a swamp environment by managing their character's weight in response to changing weather conditions. The game emphasizes strategic thinking, environmental awareness, and precise movement control.

Unique Selling Points:

Frogert distinguishes itself as a physics-based 3D platformer in which player progression is driven by weight management rather than combat mechanics. The core gameplay revolves around a dynamic weight system that directly influences movement, jumping behaviour, and resistance to environmental forces. In addition, the game features a dynamic weather system that continuously alters gameplay conditions, requiring players to adapt their strategy in response to changing environments. The interaction between physics, weather, and player decision-making creates an experience focused on strategic movement and environmental problem-solving, offering an alternative to traditional combat-oriented platformers.

Core Gameplay Loop:

1. **Explore** current location

2. **Observe** weather conditions

3. **Move and jump** across platforms

4. **Collect fireflies** to increase weight OR move continuously to reduce weight

5. **Experiment** with jumps and movement

6. **Reach campfire** and interpret firefly visual feedback

7. **Adjust weight** strategy based on feedback

8. **Progress** to next location

9. **Repeat** with increased environmental challenge

Controls:

- W/S – Forward and backward movement.
- A/D or Left/Right Arrow Keys – Horizontal movement.
- Space Bar – Charge-based jump.
- Hold the key to accumulate force.
- Release to execute the jump.
- Jump effectiveness depends on charge duration, weight, and environmental factors.
- CTRL – Slide.

Game Objectives

Primary objective

The main objective of the game is to progress through multiple connected locations by managing the frog weight in response to changing weather conditions. The player must correctly balance movement and firefly collection to perform required jumps and resist environmental forces.

Win Condition

The player wins the game by successfully reaching the final location of the level.

Failure Conditions

The player fails if:
- The frog is unable to progress due to incorrect weight management
- The frog falls into environmental hazards such as deep water
- The frog becomes trapped in unreachable gaps
- The frog cannot overcome environmental forces due to improper weight

Target Audience

Frogert is designed for casual players who are interested in physics-based platformers and experimental gameplay mechanics. The game appeals particularly to players who enjoy environmental problem-solving, movement-focused challenges, and strategic decision-making. By removing traditional combat elements, the project targets players who prefer non-violent, system-driven gameplay experiences built around physics interactions and player experimentation.

Gameplay Mechanics

Core Movement

The player controls a frog that moves and jumps using physics-based mechanics. The frog's movement and jumping behavior are influenced by:

- Dynamic weight value
- Current weather conditions
- Environmental forces
- Momentum accumulation

Charge-Based Jump System

The primary movement mechanic is a charge-based jump:

- Player holds the jump input to accumulate force
- Releasing the input executes the jump
- Jump effectiveness will depend on frog weight, accumulated momentum, external forces (wind, gravity), charge duration.

Momentum System

The Momentum System is a core mechanic that governs all movement actions through momentum, friction, and speed accumulation. During running, the player gradually accelerates and builds momentum over time, with acceleration rates affected by weight, surface friction, and weather conditions. Higher momentum results in faster movement and more powerful jumps, creating a direct relationship between movement investment and jump effectiveness.

Momentum carried into jumps affects jump distance and trajectory, while consecutive successful jumps slightly improve jump efficiency, encouraging rhythmic movement and flow-based gameplay. When stopping or changing direction, the frog decelerates based on friction and current momentum value - higher momentum requires more time and distance to stop, creating realistic physics-based movement. Surface friction directly affects both acceleration and deceleration rates, with weather conditions modifying friction to affect how quickly momentum is gained or lost.

All movement-related actions - including turning, stopping, and direction changes - respect the current momentum state, ensuring that surface properties and weight directly influence momentum behavior. This creates a physics-driven movement experience where players must understand momentum management to achieve optimal movement efficiency. The system encourages strategic momentum conservation during direction changes, careful timing of jumps to maximize momentum transfer, and flow-based gameplay that rewards maintaining continuous movement rather than stop-and-go patterns.

Belly Slide Mechanic

Under conditions of reduced surface friction, particularly during rainy weather, the frog can perform a belly slide. This mechanic allows for rapid movement across flat surfaces and is primarily used for traversal and repositioning. The belly slide functions as a situational tool rather than a core mechanic, adding variety to movement options while remaining dependent on environmental conditions.

Strategic Progression

Progression in Frogert requires players to continuously interpret environmental cues, manage weight strategically, and adapt movement behaviour in response to weather conditions. Successful advancement depends on balancing firefly collection with continuous movement, ensuring that the frog maintains an optimal weight for upcoming challenges.

Game Systems

Weight System

The weight system is a core gameplay mechanic that connects movement, physics, and environmental interaction. Weight changes dynamically throughout gameplay: collecting fireflies increases the frog's weight, while continuous movement gradually reduces it. These changes directly affect jump height, gravity influence, air control, movement responsiveness, and resistance to environmental forces such as wind.

Heavier frogs are more stable and resistant to external forces but have reduced jump height, while lighter frogs are capable of higher jumps and more agile movement but are more vulnerable to adverse weather conditions. Players must decide when to collect additional fireflies and when to prioritise movement in order to maintain an effective balance. Incorrect weight management can block progression by making certain jumps impossible or by exposing the frog to overpowering environmental forces.

Heavier Frog:

- More stable
- More resistant to wind
- Lower jump height
- Better environmental force resistance

Lighter Frog:

- Higher jump ability
- More vulnerable to strong weather effects
- More agile movement
- Less stable in adverse conditions

This system creates a balance-focused gameplay loop where the player must decide:

- When to collect additional fireflies
- When to move to maintain optimal weight
- How to balance weight for specific challenges

Incorrect weight management can prevent progression by:

- Making certain jumps impossible
- Exposing the frog to overpowering environmental forces
- Limiting movement options

Weather System

The weather system dynamically alters gameplay conditions across different locations, ensuring that each area presents unique challenges. Sunny weather provides baseline conditions with standard gravity and full movement control. Rainy weather gradually increases the frog's weight while reducing surface friction, enabling sliding but reducing movement precision. Windy weather applies strong horizontal forces, significantly affecting lighter frogs and often requiring players to increase weight before progressing. Foggy weather reduces visibility without altering physical behaviour, placing greater emphasis on spatial awareness and memory.

These effects are coordinated by the WeatherManager system, which controls weather transitions, modifies physics parameters in real time, adjusts surface properties, and influences AI behaviour. As a result, players must continuously adapt their strategy rather than relying on a single approach throughout the game.

Weather Types:

1. Sunny Weather
   - Normal gravity
   - Standard jump behavior
   - Full movement control
   - Baseline gameplay conditions

2. Rainy Weather
   - Frog gradually becomes heavier
   - Jump height decreases
   - Surface friction is reduced
   - Enables sliding behavior
   - Less precise movement control
   - Belly slide mechanic activated

3. Windy Weather

- Strong horizontal force applied to the frog
- Low weight = significant movement disruption
- Can prevent successful jumping if weight is too low
- Forces player to increase weight before progressing
- Creates strategic weight management requirement

4. Foggy Weather
   - Visibility is reduced
   - Platform navigation becomes more challenging
   - Physical behavior remains unchanged
   - Emphasizes spatial awareness and memory

The WeatherManager system:

- Controls weather transitions between locations
- Applies environmental effects to physics
- Modifies surface properties
- Affects AI behavior (fireflies)
- Creates dynamic gameplay challenges

Firefly Ai System

Fireflies are dynamic collectibles that use simple artificial intelligence behavior. They are not static objects and respond to both player proximity and environmental conditions.

AI Behavior

Basic Movement

- Fireflies move between predefined waypoints
- Maintain autonomous movement patterns
- Follow designated paths

Player Interactions:

- When player approaches within a certain distance, fireflies attempt to move away
- Makes collection more challenging
- Requires strategic approach

Weather Effects on AI

- Wind: Causes increased drift in firefly movement
- Fog: Reduces awareness range

- Demonstrates reactive movement influenced by environmental conditions

Design Purpose

- Creates dynamic collection challenges
- Demonstrates environmental interaction
- Shows reactive AI systems
- Adds depth to collectible mechanics

Campfire Feedback System

Campfires act as safe interaction points and progression indicators within the game world. At each campfire, glowing fireflies gather around the flame to visually represent the approximate weight required to reach the next location. This information is conveyed through environmental cues rather than numerical values, encouraging players to observe, interpret, and plan accordingly. The system reinforces the core theme of balance between weight, movement, and weather, while enhancing immersion and strategic decision-making.

The level is structured as:

- One large continuous environment
- Divided into multiple connected locations
- Each location requires specific weight range for traversal
- Campfires mark location transitions

Physics Design

The game uses Unity's physics system with custom modifications for:

- Weight-based mass calculations
- Dynamic gravity adjustments
- Environmental force applications
- Momentum accumulation
- Surface friction modifications

Physics Interactions:

Weight and Mass:

- Weight directly affects Rigidbody mass
- Mass influences all physics calculations
- Jump force calculations account for mass

Gravity System:

- Base gravity can be modified by weather
- Weight affects gravity perception
- Heavier objects fall faster (realistic physics)

Force Applications:

- Wind applies horizontal forces
- Force magnitude affected by weight
- Lighter objects more susceptible to forces

Friction System:

- Surface friction modified by weather
- Rain reduces friction (enables sliding)
- Affects movement control and precision

Level Design

Design Philosophy

Levels are designed around weight-based traversal challenges. The environment supports the core mechanics and encourages strategic weight management. The whole world is a single big level, divided into diverse segments, that are all connected.

Design Elements

Platforms are placed to require specific weight conditions. Gaps are sized for weight-dependent jump distances. Environmental obstacles are positioned strategically.

Weather Zones are positioned all around the world to force adaptation. Main idea is to acquire strategic weight management, as well as different movement strategies and varying gameplay challenges.

Players will progress to different locations. Each location will introduce new challenges and will become progressively harder. Progression is built on previous mechanics, such as weight and campfires.

Different locations have different hazards, such as instant failure trigger boxes, weather conditions, environmental forces and unreachable gaps.

Animations

Animation System

The frog character uses an Animator Controller to manage multiple animation states aligned with the player's movement and physics status. The animation set is designed to communicate player intention and gameplay state clearly, while also providing visual consistency with the physics-based control scheme. The core animation states include idle behaviour when the frog is stationary, movement animations during walking or running, jump animations covering take-off and landing, and a sliding animation that supports the belly slide mechanic.

Animation Control

Animation transitions are driven by gameplay parameters rather than scripted sequences. In particular movement speed is used to blend between idle and locomotion, while grounded detection and jump state determine transitions into aerial animations. Weather and weight states are also considered where appropriate to maintain consistency between physical behaviour and visual presentation. This ensures that the animation system supports readability during gameplay, especially in scenarios where momentum and surface friction noticeably affect movement.

Technical Specifications

Platform

Engine: Unity 6
Platform: PC (Windows)
Input: Keyboard
Rendering: Unity Universal Render Pipeline (URP)
Physics Engine: Unity Physics (PhysX)

Component Architecture

Core Components

The architecture of Frogert is based on a component-oriented approach, where each major gameplay system is implemented as a separate Unity component. This structure allows individual systems to be developed independently while remaining tightly integrated through shared data and events.

The FrogController is responsible for handling player input and controlling the frog's movement. It uses a Rigidbody component for physics simulation and collider components to interact with the environment. An Animator component is used to control animation states, while a reference to the WeightSystem ensures that movement and jump behaviour correctly reflect the frog's current weight.

The WeightSystem is implemented as a MonoBehaviour attached to the frog character. Its main role is to manage the dynamic weight value during gameplay. The system updates the Rigidbody mass in real time, processes weight increases caused by firefly collection, and calculates gradual weight reduction when the player maintains continuous movement. The current weight value is also made available to other systems that depend on it.

Weather-related behaviour is controlled by the WeatherManager, which acts as a central system for managing environmental conditions. It handles transitions between different weather states and applies their effects by adjusting physics parameters such as gravity, surface friction, wind forces, and visibility. The WeatherManager also coordinates with location-based logic to ensure that weather changes are applied consistently across the game world.

Firefly behaviour is handled by the FireflyAI component, which is attached to each firefly GameObject. This system controls waypoint-based movement and uses proximity detection to react to the player's presence. A simple state-based structure governs behaviour such as patrolling and fleeing. Firefly movement is influenced by active weather conditions, and a Rigidbody is used to maintain consistency with the physics-based environment.

Campfire functionality is managed through the CampfireSystem, which combines progression logic with visual feedback. This system controls the behaviour and placement of fireflies around campfires, represents weight requirements visually, and functions as a progression checkpoint when the player moves between locations.

Environmental elements are implemented using static colliders for platforms and level geometry, along with trigger colliders for hazards and collectibles. Physics materials are applied to define surface properties such as friction and bounciness. Additional trigger volumes are used to define weather zones and location boundaries, allowing the environment to interact dynamically with gameplay systems.

Collision Detection

Collision detection in Frogert is organised using Unity’s layer-based collision system. The frog character, environment geometry, hazards, collectibles, and trigger-based systems such as weather zones and campfires are separated into distinct layers to ensure predictable interaction behaviour.

Collision Resolution

Collision resolution relies primarily on Unity’s built-in physics system. Solid object interactions use impulse-based responses, while friction and elasticity are controlled through physics materials. Ground detection is implemented using raycasts or sphere casts to validate jump conditions and support stable movement on slopes. Platform interaction logic supports features such as one-way platforms, moving platform synchronisation, and precise edge detection.

Hazard interaction is handled through trigger-based detection, enabling immediate failure responses, respawn handling, and checkpoint restoration while preserving relevant game state. Collectible interaction also uses trigger validation to update the WeightSystem, remove collected fireflies from the scene, and activate feedback effects. Weather zone transitions are applied smoothly, with gradual changes and state preservation when zones overlap.

System Integration

From a physics perspective, system integration is achieved through direct coupling between the WeightSystem and the frog’s Rigidbody mass, ensuring that changes in weight immediately affect physical behaviour. The WeatherManager modifies global or local physics behaviour through gravity adjustments, surface friction manipulation, and direct force application such as wind. Jump forces are calculated with weight consideration to maintain consistent traversal behaviour across different weight states.

AI integration is handled through shared access to the current weather state, allowing fireflies to modify movement behaviour depending on wind or fog. Animation integration is achieved by feeding movement and state information from the FrogController into the Animator Controller, with weight and weather states used when needed to maintain coherence between visuals and physics. Optional UI integration may include displaying current weight and weather indicators, along with progression feedback. Audio integration can be implemented through event-driven triggers for movement, weather ambience, and collectible feedback. Communication between systems is supported through Unity Events or C# events, enabling weight changes, weather transitions, collection events, and progression updates to propagate cleanly across components.

Technical Requirements

The technical implementation is anchored in Unity's PhysX-based physics system with custom weight calculations, real-time mass updates, dynamic force application, and weather-driven modifications to gravity and surface friction. AI requirements include waypoint navigation, trigger-based proximity detection, reactive movement behaviour, state machine logic, and responsiveness to environmental conditions. Core systems are implemented through dedicated managers and components, including a WeatherManager, WeightSystem, FireflyAI, CampfireSystem, and optional managers for locations and input. Rendering requirements are supported through URP and can include particle effects for weather, post-processing for fog, dynamic lighting for campfires, and shader-based visual enhancement where appropriate.

Group Task Allocation

Member 1:

- Frog movement systems

- Physics systems implementation

- Weight mechanics

- Jumping behavior

- Core player controller

Member 2:

- Weather system implementation

- Level layout design

- Environmental effects

- WeatherManager system

- Environmental hazard placement

Member 3:

- Firefly AI implementation

- Animation setup and integration

- User interface elements

- AI waypoint system

- UI/UX design

Collaboration Areas:

- System integration

- Testing and balancing

- Level design review

- Gameplay refinement

- Documentation

Conclusion

Frogert demonstrates core aspects of game development using Unity with a strong emphasis on physics-based gameplay, artificial intelligence, and environmental interaction. The combination of weight management, weather effects, and diegetic feedback systems creates an innovative platformer experience while remaining technically achievable within the project scope.