CO1301 Games Concepts – Portfolio Assessment
Game 2: Future Space Frogs

You must follow the specification to implement a simple 3D game about saving the world from "radiation emitting future space" frogs by hitting them with a car controlled by the keyboard. You are expected to complete this in your own time outside of scheduled labs. But you should ask your lab tutor for advice if you are stuck.

Do not diverge from this specification. If you do, then you will lose marks. Ask for clarification if you are unsure!

Your submission should include the .cpp file of your project, the media folder containing all your models, and a word document containing you state transition diagram if you are aiming for marks higher than 60%.

In addition to the above, your formative feedback point submission should include a video (not more than 3 minutes) showing you game in execution.

The associated set of model files for this game can be found on Blackboard inside "Game2_Models.zip". The models include a frog (frog.x), an island (island.x), a cube portal (portal.x), a car (rover.x), a sky box (skybox.x) and water surface (surface.x).

Game Specification

You should aim to implement the features described below in order. To be eligible for a mark within any classification, you must have attempted all the features for all the previous classifications. Treat the grade boundaries as milestones in a real game development project.

Basic Scene Layout

To achieve any classification, you must first implement the basic scene layout and then meet the requirements of the classification. The basic scene layout is specified below:

- Set up a skybox using the skybox.x model provided at position (0, -960, 0).
- Using the surface.x model provided, create a "floor" of water at position (0, -5, 0).
- Using the island.x model, create an island as position (0, -6, 0).
- Create a manual camera at position(0, 120, -100) and rotate it about the X axis by 50 degrees, such that it is suspended in the air and looking downwards towards the island.
- Using the rover.x mesh, create a car model in position (0, 0, -50), this represents the player.
- Using the frog.x mesh, create two stationary enemies frog1 and frog2:
    - frog1 should be rotated 90 units along its Y axis and positioned at (`5, 0, 0`)
    - frog2 should be rotated -90 units along its Y axis and positioned at (`-5, 0, 0`)
- The scene should appear exactly like the picture below:

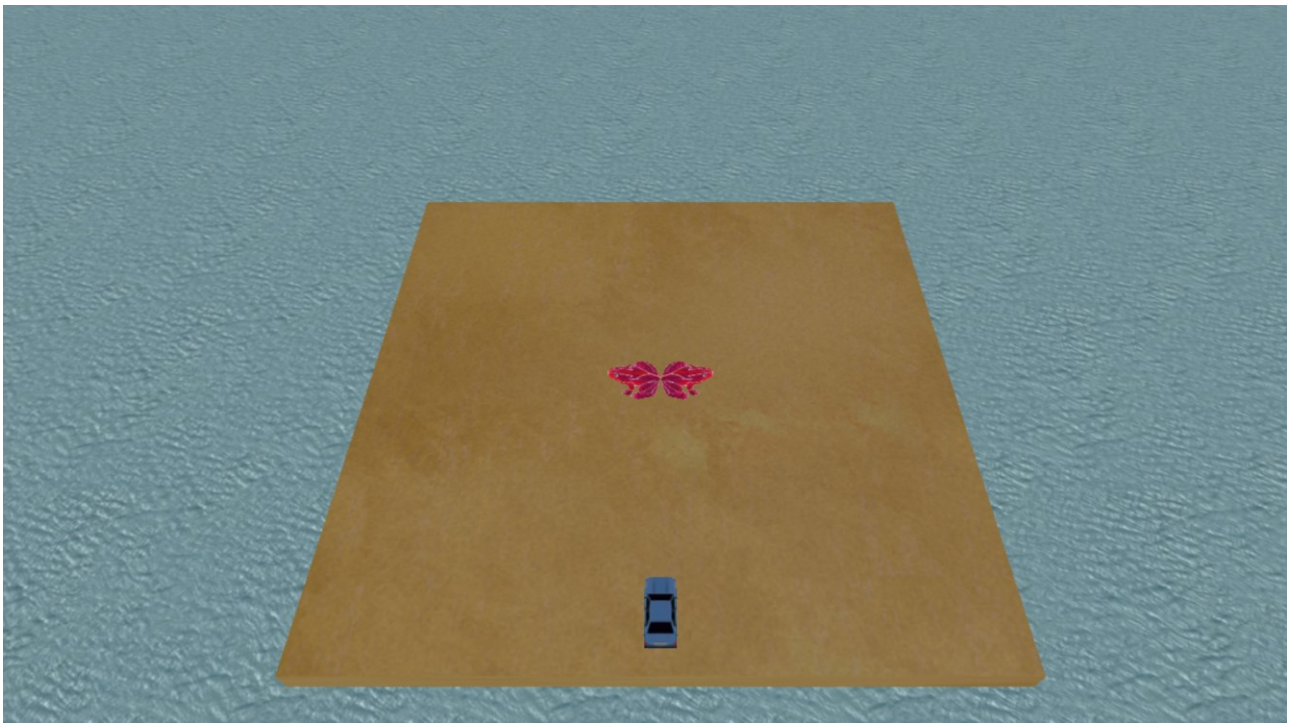

Figure 1 Screenshot of the basic scene layout

Pass Mark/Third Classification (40% +) Milestone 1

- The code you submit must compile without errors.
- Set up the scene exactly as described above.
- The game should have a state, which is one of **playing**, **paused** or **over**.
- The game should start in the **playing** state. In the **playing** state the player can control the car as follows:
    - 'W' moves the car forward
    - 'S' moves the car backwards
    - 'D' rotates the car clockwise
    - 'A' rotates the car anti-clockwise
- The speed at which the car moves must be defined in your code by a constant float named **kGameSpeed**.
- The P key causes the game to enter the paused state. In the Paused state:
    - Nothing should move
    - The P key causes the game to transition to the playing state
- In the Over state:
    - Nothing should move – the game is over.
- In all states, the player should be able to press the Escape key to quit the game.
- You should display the current score in the corner of the screen.
- When the game starts the score is 0.
- Implement sphere-box collision detection between the car and the frogs.
    - An efficient program will only use the X and Z dimensions, to effectively implement circle-to-rectangle collision detection.
    - If the car collides with a frog, the frog and should flip over (lie on its back) and the player's score should increase by 10.
    - Subsequent collisions should not result in any action.
    - If the car hits all two frogs, the player wins and an appropriate message should be displayed.
- If a player drives off the island, the player has lost and the game is over.

Lower second classification (50% +) Milestone 2

- Using the portal.x model provided, create a cube portal at position (0, 0, 0).
- You may want to scale it up a bit to properly house all frogs using **portal->Scale(1.2);**
- Add two frog models, frog3 and frog 4 so that they are initially positioned as follows
  - frog1: rotated 90 units along its Y axis and positioned at (5, 0, 0)
  - frog2: rotated -90 units along its Y axis and positioned at (-5, 0, 0)
  - frog3: rotated 180 units along its Y axis and positioned at (0, 0, -5)
  - frog4: positioned at (0, 0, 5)
- Each frog should now be in one of four states: **waiting**, **moving**, **dead** or **escaped**.
- When a frog is waiting, it should be fixed and not moving.
- When a frog is moving, it should be moving along either the world X or Z axis (depending on the direction it is facing) at half the speed of the car. Therefore, it should move forward in a straight line towards the water.
- When the game starts frog1 should be in moving state while the other three frogs should be in **waiting** state
- If the car collides with a frog, the frog transitions to its dead state, and should flip over, and the player's score should increase by 10.
- If frog1 enters its dead state, frog2 should enter moving state, if frog two enters its dead state, frog3 enters moving state, if frog3 enters dead state frog4 enters moving state and if all frogs are dead the player has won.
- If a frog gets into the water, it enters its escape state.
- Once a frog escapes, the player has lost and the game is over.
- Appropriate win or loss messages should be shown on the screen.
- Implement collision detection between the car and the cube to prevent the player from driving through it.
- Nothing should be able to move if the game is in any state other than 'playing'

Upper second classification (60% +) Milestone 3 – break-even point achieved

- Implement a chase camera view in which the camera is positioned above and behind the car.
- Use the "2" key to switch to this camera angle.
- Implement a first-person camera view. Use the "3" key to switch to this camera angle.
- Use the "1" key to switch to the default fixed camera.
- You must implement the collision detection for the enemies using arrays and functions.
- The water should have a state variable to store whether it is safe or poisoned.
- If a frog escapes into the water, it becomes poisoned and its skin should change to “poisoned.png”.
- Create state transition diagrams to show the behaviour of your game.

First Classification (70% +) Milestone 4 – bonus payments received

- A frog in moving state should slide 10 units in the appropriate direction (at the same speed as before), wait for 1 second and slide another 10 units.
- Add 6 plants at different positions around the island.
- Detect collisions (sphere-sphere) between the plants and the car, resolving them such that the car cannot pass through the branches of the plants, but most drive round them.

High First Classification Milestone 5 – extra bonus payments received

- Add a timer that counts down from 60 seconds.
- Rather than ending the game when all four frogs are dead, each dead frog should be moved back to the cube into waiting after being dead for one second.
- Therefore, dead frogs will be changed to waiting to be moving until the 60-second timer is expired. After that, dead frogs stay dead, but the player has to hit any remaining moving frogs before the game is won.
- When the game is **over**, pressing the space-bar should re-set everything and re-start the game.
- Use scaling in the Y axis to squash the frogs flat when they have been run over.
- You may have to lift them slightly off the ground, otherwise they might disappear into the road surface.
- When a frog moves forward one “space”, make it jump slightly.
- Update the state transition diagrams showing the behaviour of your game.