CO1301 Games Concepts – Portfolio Assessment
Game 3 – Desert Racer

You must follow the specification to implement elements of a racing game, including car movement, race stages and collision detection / resolution. In the "basic" version of the game the player controls a hover-car which must be driven along a short desert racetrack from a starting point, over three stages to a finishing gate. A student (Alan Raby) completed a similar assignment to this a few of years ago, and placed a video of it on YouTube: https://youtu.be/af6KhD06FcA. The stages must be completed in sequence and the player cannot finish the race unless they have completed all the stages. 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 documentation for your program as specified at the end of this document.

Game Specification

You should aim to implement the features described in this document 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.

Resources

The associated set of model files for this game can be found on Blackboard inside "Game3_Models.zip". The models include:

race2.x	A hover car
Checkpoint.x	A checkpoint and the finishing line
Skybox 07.x	A skybox
ground.x	The ground
TankSmall1.x	A water tank
TankSmall2.x	Another water tank
IsleStraight.x	A wall end
Wall.x	A wall section
Tribune1.x	A trap
Interstellar.x	An enemy spaceship
Cross.x	A large red cross
Flare.x	A bomb

A pack of "extra" models to enhance your scene will also be made available. You may use models from other sources as well if you wish (e.g. from earlier lab exercises), however, you must upload any extra models that you do use along with your source code when you submit your assignment. **You should only use extra media if you complete all aspects of this GDD.**

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:

- Use the models provided to create a racetrack. The basic layout consists of three checkpoints and two walls as illustrated below.

Figure 1 Screenshot of basic scene layout

- The walls are each made of three Isle models and a Wall model.
- The location of the models are:
    - checkpoint 1 (0, 0,0)
    - checkpoint 2 (10, 0, 120) – **rotated 90 degrees**
    - checkpoint 3 (25, 0, 56)
    - isle (-10, 0, 40)
    - isle (10, 0, 40)
    - isle (-10, 0, 56)
    - isle (10, 0, 56)
    - isle (-10, 0, 72)
    - isle (10, 0, 72)
    - wall (-10, 0, 48)
    - wall (10, 0, 48)
    - wall (-10, 0, 64)
    - wall (10, 0, 64)

- The checkpoints, aisles and walls MUST be stored either in arrays or STL vectors.
- For very high grades, you should read the positions of scenery items in from a file.
- You are not required to use any other models unless you are attempting the higher grades, but you may use other models if you wish.
    - If you are attempting a lower second grade, then you need to place some water tanks around the track
    - If you are attempting a higher second grade then you need to place more wall sections on the track.
- You may want to place some other buildings/models for effect.
- The hover-car should start some distance away from the first stage.
- The skybox should be created at location (0, -960, 0)

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

- The code you submit must compile without errors.
- Set up the scene as described above. (Note the orientation of checkpoint 2)
- Implement your own manual chase cam in which the camera is positioned above and behind the car.
- The player can only control the hover car
- Set up keys so that
    - 'Up' arrow moves the camera forward
    - 'Down' arrow moves the camera backward
    - 'Right' arrow moves the camera right.
    - 'Left' arrow moves the camera left.
    - Mouse movement rotates the camera. (You should not be able to “wriggle” the camera upside-down with the mouse!)
    - ‘1’ resets the camera position
    - 'W' applies forward thrust to the hover-car o 'S' applies backwards thrust to the hover-car
    - The maximum backward thrust should be half that of the maximum forward thrust.
    - 'D' turns the car clockwise
    - 'A' turns the car anti-clockwise

- Use "ui_backdrop.jpg" to create a backdrop for dialogue and other game information. It would probably look best if this is at the bottom of the screen. Use Draw to output dialogue and any other game information.
    - You could make your game look nicer by replacing ui_backdrop.jpg with something more exciting.

- Display the current state of the game over the backdrop.
- When the game loads the dialog should read "Hit Space to Start".
- When the player hits start, the dialog should flash “3” for a second, then “2”, then “1”, then “Go!”.
    - The player should not be able to move the hover-car until the dialog says “Go!”

- When the car crosses the first checkpoint the dialogue changes to "Stage 1 complete" and so on.
- When the car crosses the final checkpoint the dialogue changes to "Race complete"
- The car needs to cross the checkpoints in the correct order. You cannot complete a stage out of order.
    - You are being asked to implement states.
    - The best way is to think of the race stages as sub-states within the overall game’s Racing state.
        - Advanced: Think about how you might structure things if you had different racetracks, which had different numbers of stages. For a high mark, the race stages will not be hard coded within your game loop.

- Dialogue and state changes are triggered by moving through the checkpoints.
    - You should treat the gap between the struts of each checkpoint as an axis-aligned bounding box and implement a point-to-box collision function to detect the centre of the car passing through the checkpoint.
- Implement collision detection and simple resolution with the walls and with the legs of the checkpoints.
    - You may treat the hover-car as a sphere for collision detection purposes.
    - You should treat the two isle models and the wall section between them as a single axis-aligned bounding box. Implement collision detection with these as a sphere-to-box collision.
    - Implement a sphere-to-sphere calculation for the struts of the checkpoints.

In order to PASS, within your code:

- You must use frame timing to control the speed of EVERYTHING.
- This includes hover car movement/steering AND camera movement
- You must use an enumerated type to implement the game states.
- You must use functions for collision detection.
- You must use arrays to store elements of the scene.
- You must call your collision detection functions from within a loop.
- You must declare a structure for a 2D vector and implement functions to manipulate such vectors. At the very least you will need functions for scalar multiplication and vector addition.

Lower Second Classification (50% +) Milestone 2

- Add a speed readout to the dialogue – it should output the speed in whole numbers, as kilometres per hour according to the scale you have defined for your models (see the Report section).
  - The scale you define in your report should be implemented as a constant in your program.
  - Multiply your momentum vector's length by your scale constant to find out the speed of
  - your car in metres/second.
- Introduce at least one more checkpoint. You will need to extend the number of states to account for the increased number of checkpoints. You also need to add on the appropriate dialogue, e.g. "Stage 3 complete".
  - If you include any wall/isle sections, keep the walls axis-aligned (to keep collision detection simpler)
- Use water tanks placed sparingly alongside the track to suggest "corners".
- Also place a tank half-buried in the sand and leaning at an angle in the middle of the track in one of the sections as an obstacle to be avoided.
- Implement collision detection between the car and all of the stationary objects:
  - Implement collision detection between the car and the tanks as sphere-to-sphere.
  - Implement a basic damage model for the player hover-car. The car starts with 100 health points. Every time the car collides with an object it should lose 1 health point. Display the current health level in the user interface.
  - When the car's health reaches 0, it cannot be controlled at all, and the race is over.
  - When the car's health falls below 30%, the boost should not operate.
- Implement a first-person camera view. Use the "2" key to switch to this camera angle.
- Use the "1" key to switch back to the default chase camera.

Upper Second Classification (60% +) Milestone 3 – breakeven point achieved

- Resolve collisions with walls by manipulating the components of the momentum vector so the car appears to bounce off the walls.
- Introduce at least one further checkpoint. You will need to extend the number of states to account for the increased number of checkpoints. You also need to add appropriate dialogue.
  - Include some narrower walled sections that the car must travel through.
- Introduce a "Boost" facility to make the hover-car accelerate more quickly. This should be active while the space-bar is held down.
  - The thrust applied to the car should be 50% greater when the Boost is activated.
  - If the boost is active for too long (> 3 seconds), it will overheat. If this happens, the hover car should decelerate quickly (drag should be doubled), and the boost should not be usable for a period of time (5 seconds).
  - When the Boost is active, this should be shown on the dialog, with an additional warning
  - second before the booster overheats.
- Implement a non-player racing hover-car.
  - Use an array of dummy models to act as "waypoints" around the track which the non-player car should look at as it moves around the track.
  - As soon as it passes a waypoint (hint: use a distance calculation), the non-player car should
  - look at and move towards the next waypoint.
  - The number and positioning of these waypoints will affect how realistic the movement looks
  - Use the same model for the non-player car, but edit a copy of the skin graphic so that it is a different colour.
- Implement collision detection and simple resolution between the player and non-player hover-car(s).

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

- The hover-car should hover in the air as if it were on some sort of gravity cushion.
- The car gently bobbles up and down as it moves.
  - Make the car "lean" into the bends as it turns.
  - The car should lift up slightly at the front or rear (you choose!) as it accelerates.
- The car should bounce when it collides with any object. The bounce needs to be smooth.
- Identify a successful move through a checkpoint. Place a cross centred under the checkpoint when the player has successfully negotiated that check-point. Provide it with a life timer so that the cross disappears after a short while.
- Make the race track into a complete circuit so that a race of more than one lap can take place. Show the current lap and total laps on the dialogue (e.g. Lap 2/5)
- Add a "current race position" to the dialogue, and at the end of the race, display the race winner and their race time.
- Give players the option to restart the race instead of quitting at the end of the game.
- The positions of all objects in the game should be read in from a simple text file, formatted as follows:
  - Each line of the file represents one scenic object and has four fields separated by white space. The fields represent:
    - The type of object (text corresponding to the name of the .x file)
    - The x coordinate of the object
    - The z coordinate of the object
    - The rotation of the object around the y axis
  - The basic scene set-up (for 40%) would therefore look as follows:

Isle	-10	40	0
Isle	10	40	0
Isle	-10	56	0
Isle	10	56	0
Isle	-10	72	0
Isle	10	72	0
Wall	-10	48	0
Wall	10	48	0
Wall	-10	64	0
Wall	10	64	0
Checkpoint	0	0	0
Checkpoint	10	120	90
Checkpoint	25	56	0

High First Classification Milestone 4 – extra bonus payments received

Only attempt the following section if you've already achieved the first classification:
These are suggested extensions – feel free to add anything extra that you want, as long as it doesn't break the requirements of the previous sections.
You do not need to attempt all of these to score 100%!

- Place some barrels round the track which contain burning fires
  - Implement the fire by using a particle system for the flames.
- Scatter some unexploded bombs (using the Flare model) round the track.
  - When either car passes too close to a bomb, it should explode
  - If a bomb explodes close to the player car, the camera should shake, and the explosion should cause damage to the car.
- Implement a particle system to simulate the exhaust flames of the Booster coming from the rear of the hover-car, when the boost is active.
- Use an array to act as a "speed table" for the non-player car, so that it behaves in a more realistic way.
  - The car would thus move at a customisable speed between each waypoint
  - You could make this smoother by having the car accelerate/decelerate between each new speed.
- Add more non-player hover-cars.
- Implement a damage model for the hover cars
  - Bumping into the hover cars (and causing them to bump into obstacles) will cause them to be damaged and slow down.
- Damaged cars could emit smoke, and/or list to one side
  - The player car, when damaged, could become more difficult to steer.
- Give the player car a limited range "photon torpedo" style weapon with which to attack and damage the non-player cars.
- Add a menu screen with the option to load different racetracks.

Documentation

You need to produce a document to be submitted with your game. The document will state:

- The grade you expect to get.
- The length of your hover-car in TL-Engine units.
- The length of your hover-car in metres.
- The scale of objects your game (units / metre)
- The coefficient of drag
- (for a 2:2 or higher) The maximum speed of your hover car (without boost) in
  - metres / second,
  - kilometres / hour
  - miles / hour
- (for a 2:1 or higher) The maximum speed of your hover car (with boost) in
  - metres / second,
  - kilometres / hour
  - miles / hour
- The maximum thrust force you apply (length of thrust vector) in TL-Engine units and scaled to represent acceleration in metres/second/second.
- A brief explanation of how you resolve collisions between your hover car and other objects.
- This should include details of the mathematics used for any calculations.

You must also include a scale map of your race course, showing the course boundary and the positions of obstacles, checkpoints, etc. (all labelled).
The positions of objects may be given in TL-Engine units, but the map should have a scale in metres.
The map should also state the length (in metres) of the race track (or circuit).