CO1301 Games Concepts: 2017-2018
Assignment 3: Hover Racing

Release Date: ??/02/18 Hand in Date: Saturday 14/04/18 (11.59pm)

Introduction
For this assignment I want you to implement the 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 race-track from a starting point, over two stages to a finishing gate. The stages must be completed in sequence and the player cannot finish the race unless they have completed all of the stages.

I expect you to complete this project in your own time, NOT in the labs.

This is an **individual** project and no group work is permitted.

Do not diverge from the assignment specification. If you do not conform to the assignment specification then you will **lose marks**. If you do want to make some addition to the game (at the advanced level) and you are unsure whether the change will break the specification then check with Nick or Dean or Kris first. Some of the requirements at the advanced level have been left deliberately vague – use your creativity, but make sure you explain your approach carefully in the report.

Avoiding Plagiarism
- You will be held responsible if someone copies your work - unless you can demonstrate that have taken reasonable precautions against copying.
- Remember that the machines in the Games Lab have shared hard drives. You must therefore either password protect your work (e.g. by using WinZip and its password protection facility) or else delete your work from the hard drive each time you finish working on it (keep a copy on your pen-drive).

Learning Outcomes Assessed (see module descriptor for full list)
- Describe key games concepts, e.g. genres, terminology.
- Use a game engine to create simple computer game prototypes.
- Apply mathematical techniques for analysis and reasoning about problems

Deliverables, Submission and Assessment
- As before there are two deliverables – your **source code** and a **report**.
- Submission is electronic, via Blackboard.
    - For the source code, multiple files **are** permitted for this assignment.
        - If you create multiple files, you MUST also upload the Visual Studio project files to enable me to compile and run your project.
        - DO NOT upload compiled object files or executable files.
    - Make sure you include your name as a comment on the first line of EVERY source code file. Also include your name at the top of your report.
    - See the last page for details of what to include in your report.

- **Assessment will be by demonstration in your usual lab session during the week commencing 16/04/18**. You must demonstrate the code you uploaded – no tweaking!
    - **Important Note**: Your demonstration counts as part of your submission. If you are late demonstrating your assignment and do not have an extension, the usual lateness penalties will apply.

      PLEASE ALSO NOTE: The last possible date for demonstrations (with or without extensions) is Friday 27th April

Resources

This assignment has an associated set of files. The files can be found inside "Assignment 3 models.zip" on Blackboard. The files include:

- ***Models***
    - 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 that you do use along with your source code when you submit your assignment.

- Please note that all of the models and textures I supply are under license and must not be passed on to any else or used for any purpose other than your University work.

Game Specification
You should implement the features described below **in order**. To be eligible for a mark within any classification, you must have completed **all** the features for **all** the previous classifications. Only when you have completed the First classification may you add ideas of your own.

For a very high mark, you need to pay attention to the **playability** of the game as well as achieving the technical requirements.

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. This is the sort of thing you should be aiming for if you want a very high mark.

IMPORTANT NOTE: The bare pass mark is 40%, however if your course is **Games Development** or **Software Engineering**, I consider a mark of **60%** to be satisfactory.

Layout of the Level

- Use the models to create a straight race track. The basic layout consists of two stages (stage 1 and the finishing line), and two walls.
- The walls are each made of two Isle models and a Wall model.
- The location of the models are:
    - checkpoint 1 (0, 0, 0);
    - isle (-10, 0, 40);
    - isle (10, 0, 40);
    - wall (-10.5, 0, 46);
    - wall (9.5, 0, 46);
    - isle (10, 0, 53);
    - isle (-10, 0, 53);
    - checkpoint 2 (0, 0, 100);
- 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 some wall sections on the track.
- You may want to place some other buildings/models for effect.
- The player only controls the hover-car. The hover-car should start some distance away from the first stage.
- The skybox should be created at location (0.0, -960.0, 0.0).
- You may treat the hover-car as a sphere for collision detection purposes.

Bare Pass Mark = third classification (40% +) Milestone 1

- The code that you submit must compile without errors.
- Set up the scene as described above.
- Implement a chase cam in which the camera is positioned above and behind the car.
- Set up the keys so that
    - 'W' accelerates the hover-car forward (up to a limit)
    - 'S' decelerates the hover-car and allows it to move backwards (there should be a relatively low top-speed going backwards)
    - 'D' steers the car clockwise
    - 'A' steers the car anti-clockwise
- The player can only control the car.
- 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 dialogue reads "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 stage the dialogue changes to "Stage 1 complete".
- When the car crosses the finishing line the dialogue changes to "Race complete".
- The car needs to cross the checkpoint and the finishing line in the correct order. You cannot complete a stage out of order.
    - You are being asked to implement **states**. The game begins in state 1 (start) and then enters state 2 when the car goes through the first checkpoint. It will then enter state 3 when the car goes over the finishing line. You should define an **enumerated type** to record (and check) the current state of the game.
- Dialogue and state changes are triggered by crossing 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 with the **walls**, and with the **legs of the checkpoints**.
    - You should treat the isle models and wall 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.
- Implement your own chase camera. Set up the keys so that
    - 'Up' moves the camera forward
    - 'Down' moves the camera backward
    - 'Right' moves the camera right.
    - 'Left' rotates the camera left.
    - ‘1’ resets the camera position.
        - Mouse movement rotates the camera. (You should not be able to “wriggle” the camera upside-down with the mouse!)

- You **must** use **frame timing** to control the game speed.
- You **must** use an **enumerated type** to implement the game states

NOTE: I must be able to TEST that your collision-driven state changes work as they should. If you make the level such that this can't be tested, you won't get the marks!

Lower second classification (50% +) Milestone 2

- Add a speed readout to the dialogue – it should output the speed in **whole** numbers
- The car should have some forward momentum, and should slow down naturally when neither the accelerate or decelerate keys are held down.
- 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 2 complete".
    - Your race course should no longer be in a straight line. However, if you include any walled 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 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.

- You must implement the collision detection for the tanks using **array of tanks** and place your sphere-sphere collision detection algorithm in a **function**.

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

- 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.
    - If the boost is active for too long (> 3 seconds), it will overheat. If this happens, the hover car should decelerate quickly, 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 dialogue, with an additional warning 1 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, the non-player car should look at 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 between the player and non-player hover-cars.
- Implement a basic damage model for the player hover-car. The car starts with 100 hit points. Every time the car collides with an object it should lose 1 hit point. Display the current number of hit points in the user interface.

First 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 as it accelerates.
- The car should bounce when it collides with any object. The bounce needs to be smooth.
- Restart the game instead of quitting at the end of the game.
- Identify a successful move through the checkpoint. Place a cross centred on the circle when the player has successfully negotiated the 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.

- Use vectors to control the movement of the car.
    - Implement forces for **acceleration** (in the direction the car is **pointing**), and deceleration or **drag** (in the opposite direction to which the car is **moving**).
    - Calculate the car's **momentum** by combining the acceleration and drag vectors.
    - Move the car each according to its momentum vector.
        - You should be able to get it to slide round corners ☺
    - Resolve collisions by manipulating the components of the momentum vector.

High First classification (up to 100% and beyond!)

- Only implement 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.
- 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.

Not for marks, but if you really want to impress me…
This game is crying out for some development "tools" – primarily a level editor. If you want to get sophisticated, the track plan and way-points for the non-player cars should be loaded from a file (a simple text file would suffice) when the game plays, rather than having all the positions hard-coded in your program. The level editor would help you create this file. Develop it further, and you could release it as part of your game, to allow players to create and share their own race tracks.

Code Style and Layout

- Your code **MUST** be properly indented and laid out so that it is readable.
  - Brackets must line up (and should normally be on a line of their own).
  - Indentation must be consistent.
  - Appropriate use of white space should be made.
  - Over-long lines of code or comments should be split up
- You should have no "magic numbers" but instead make proper use of **constants**.
- **Variable names** must be meaningful.
- Your code should be **commented** appropriately.
- You should make proper use of **arrays**, with **loops** to process them:
  - **All** scenic items **MUST** be stored in arrays.
- You should make use of **enumerated types** to control the states within the game where appropriate.
- Your collision detection algorithms **MUST** be implemented as **functions** declared outside the main program.
- You should make use of **structures** where appropriate.

Report
You need to write a report about the program. The report must be succinct and to the point. The report is **not** a story. The report is **not** a chronological record of the process of development. The *maximum* size of the written report is 600 words (two sides of A4).

The report will state:

- A summary of **what** was achieved in the final implementation of the program.
- **How** the various technical aspects of the game work.
  - This should include details the mathematics used for all of your calculations.
- what **grade** you expect to get

You must also include, as a single-page appendix to your report, a **scale map** of your race course, showing the course boundary and the positions of obstacles, checkpoints, etc. (all labelled). The map *may* be hand-drawn and scanned, but this will not look very professional!