University of Central Lancashire UCLan Cyprus School of Sciences	**UCLan Coursework Assessment Brief**	2022-2023
	Module Title: Games Concepts Module Code: CO1301	Level 4
	Assignment 2 – Hover Racing	This assessment is worth 45% of the overall module mark

THE BRIEF/INSTRUCTIONS

Introduction

For this assignment I want you to implement the elements of a racing game, including car movement, race stages and collision detection and 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.

You have to submit the game and a report – read below.

This is an individual project and **no group work is permitted**. You are expected to complete the assignment in your own time, rather than in-class time (though you may ask for advice in the lab).

Do not diverge from the assignment specification. If you do not conform to the assignment specification, you will lose marks. If you do want to make some addition to the game and you are unsure whether the change will break the specification, check with me first.

Learning Outcomes Assessed

- Use a game engine to create simple computer game prototypes.
- Apply mathematical techniques for analysis and reasoning about problems

Assessment

- This assignment carries 100 marks.
- Your mark in Assignment 2 corresponds to **45%** of your final mark for this module.

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.

Alan Raby, a student from Preston, completed a similar assignment to this a few of years ago, and placed a video of it on YouTube here: http://www.youtube.com/watch?v=af6KhD06FcA

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

Milestone 1: Basic Scene Set-Up
Reward: 0% to 10%

- Use the models provided to create a straight race track. The basic layout consists of the skybox, two checkpoints, and two walls.
- Place the skybox at location (0, -960, 0).
- 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, 150)

- Further to the above, around the track you need to place some further models, for better visual effect:
  Add water tanks, additional wall sections, 15 or more buildings or anything else you find useful and interesting inside our Media folder.
- Position the hover-car some distance before the first checkpoint, as shown in the picture above.

Milestone 2: Basic Hover Movement and Collisions
Reward: 11% to 20%

- Set up the keys to allow some basic movement to the hover:
  'W' pushes the hover forward, with a constant speed.
  'S' makes the hover move backwards with a constant speed.
  Note: The hover's backward speed must be much smaller than its frontward speed.
  'D' steers the car clockwise
  'A' steers the car anti-clockwise

Note: You must choose how fast hover should move. Remember, we want to ensure our game is playable and fun.

Hint: You will find it very useful (and rewarding!) to avoid the using magic numbers!

Reminder: Anything that moves in the game **must use variable timing**, i.e. it should make use of dt, the time interval between the previous frame and the current frame.

- Implement collision detection between the hover and the walls:

For collision detection purposes, you must treat the hover as a sphere. Furthermore, each isle/wall model must be treated as a single axis-aligned bounding box. Therefore, you must implement a **sphere-to-box** collision detection.

- Implement collision resolution between the hover and the walls:

When a collision between the hover and any wall/isle is indeed detected, make the hover stop, so that it cannot travel through the walls.

Milestone 3: Backdrop and Game States
Reward: 21% to 35%

- Create a backdrop for dialogue and other game information:

  As a start, use the image "ui_backdrop.jpg" for the backdrop. It would probably look best if this is at the bottom of the screen.

  Hint: To achieve this, you will have to declare an object of type **ISprite*** and initialise it using the **CreateSprite()** member function of the **I3DEngine** class. You should then be able to figure out how to correctly position the backdrop using member functions of the **ISprite** class.

  You could make your game look nicer by replacing "ui_backdrop.jpg" with something more exciting.

- Implement the following states for the game:

  Demo:

  This is the default game state, i.e. it is the state that the game has when the game starts.

  When in the Demo state, a message on the screen should read "Press Space to Start".

  Hint: To display text, you will have to use declare an object of type **IFont***. Initialise this object using **I3DEngine::LoadFont()**, and draw text using **IFont::Draw()**.

  When in the Demo state, none of the keys should achieve anything, with the sole exception being the 'Space' key. When the player hits the 'Space' key, the game transits to the Count-Down state.

  - **Count-Down:**

    When in the Count-Down state, the message on the screen should read in red colour "3" for a second, then "2", then "1", then "Go, go, go!".

    No key presses are allowed during the Count-Down state, apart from Pausing the game Once enough time has passed, and once the "Go!" message has been displayed, the game transits to the "Stage 0 Complete" state.

  - **Stage 0 Complete**, and **State 1 Complete**:

    When in these states, the game is playable, meaning that the player can control the hover, according to the functionality described in Milestone 2.

    When passing through Checkpoint 1, the game transits from "Stage 0 Complete" to "Stage 1 Complete".

    When passing through Checkpoint 2, the game transits from "Stage 1 Complete" to "Race Complete".

    Hint: Use **sphere-to-sphere** collision detection to check whether a checkpoint has been passed.

    Note: Checkpoints must be passed in the correct order. If the player skipped Checkpoint 1, passing through Checkpoint 2 must not allow transition to the "Race Complete" state and the message "Checkpoint 1 missed!" should be issued on the screen.

  - **Race Complete:**

    In this case the game finishes, and nothing else happens.

  - **In all states:**

    Display the current game state over the backdrop.

    Pressing the 'Escape' key causes the application to terminate.

Milestone 4: Camera
Reward: 36% to 50%

Implement code which allows the camera to exhibit the following different behaviours:

- **Free-Moving Camera:**

  This is the default mode of operation of the camera, i.e. it is the behaviour that the camera exhibits when the game starts.
  At any time during the game, the Free-Moving camera is enabled by pressing '2'.

  When in this mode, the camera is simply positioned somewhere in the scene, at a viewpoint which allows the entire scene to be visible. You should apply limits to ensure that the camera does not roll more than 45 degrees

- **Third-Person Camera:**

  The third-person camera is enabled by pressing '3'.

  When in this mode, the camera is attached on a position behind and slightly above the hover-car, such that when the hover moves, the camera moves with it with the same velocity. The camera must be positioned in such a way that makes the game playable.
  Note: This type of camera is also known as a "Chase Camera".

- **First-Person Camera:**

  The first-person camera is enabled by pressing '1'.

  When in this mode, the camera takes the position of the hover-car's pilot. Take good care to position the camera correctly, such that it visually gives the experience of driving the hover-car.

- **Surveillance Camera:**

  This camera is enabled by pressing '4'.

  Like in the Free-Camera case, the Surveillance Camera is positioned somewhere in the scene, at a viewpoint which allows the entire scene to be visible. However, the surveillance camera automatically tracks (i.e. orients itself so it can look directly towards) the hover-car, as this moves around in the scene.

- **In all the camera behaviours described above:**
  The player must be able to control the camera, as follows:

  'Up' moves the camera forwards
  'Down' moves the camera backwards
  'Right' moves the camera rightwards.
  'Left' moves the camera leftwards.
  Mouse movement rotates the camera.
  'C' resets the camera's position and orientation.
  Hint: The camera's initial position and orientation may not be the same in all camera states!

  Note: Camera speed should be appropriate such that the game maintains its entertaining character.

Milestone 5: Better Piste
Reward: 51% to 60%

Make the piste more interesting:

- Introduce at least four more checkpoints.

  Hint: You will need to extend the number of game states to account for the increased number of checkpoints. You also need to add appropriate dialogue.

- In one of your checkpoints, include some narrower walled sections (i.e. a corridor) through which the car must travel before it passes the checkpoint.

- The piste must no longer be a straight line. However, if you include any walled sections, make sure the walls remain axis-aligned (in order to keep collision detection simpler).

- Use water tanks placed sparingly alongside the track to suggest its boundaries and corners.

- Also place 2 tanks half-buried in the sand and leaning at an angle in the middle of the track in one of the sections as obstacles to be avoided.

- Implement collision detection and resolution between hover and all stationary objects: Use sphere-to-box collision detection for walls, and sphere-to-sphere with everything else, including water-tanks etc.

  Note: Make sure the checkpoint's two legs are also collideable!

  Note: Place all tanks into an array, and call your sphere-to-sphere collision detection function in a for-loop. Do the same with all the walls, and call your sphere-to-box collision detection function again in a loop.

Milestone 6: Realistic Dynamics and Boost
Reward: 61% to 75%

Improve the way with which the hover moves:

- Introduce the idea of variable speed for your hover:

  When the game starts, the hover must have zero speed. From this point onwards, pressing the 'W' and 'S' keys simply increases or decreases the hover's speed.

  Make sure the speed is bounded within reasonable limits: The hover must be able to move fast frontwards, and slowly backwards.

  When the player doesn't press the 'W' key, the hover should gradually lose speed due to friction.

- Add a speed readout to the backdrop dialogue: The speed should be displayed in whole numbers.

- Adapt your collision resolution, such that upon collision, the hover bounces backwards. The bounce needs to be smooth.

  Hint: Upon collision, the hover's backward speed is somehow related to the hover's frontward speed right before the collision took place. Can you figure out how these two speeds are related?

- The hover should now float in the air, as if it were on some sort of gravity cushion: To achieve this, make the hover to gently wobble up and down, regardless whether it moves or not.

- Introduce 3 more additional obstacles to be avoided of you preference.

- Introduce a "Boost" facility to make the hover accelerate more quickly:

  The boost is activated while the space-bar is held down.

  If the boost is active for too long (e.g. for more than 4 seconds), it will overheat. If this happens, the hover car should decelerate quickly, and the boost should not be available for 6 seconds.

  When the Boost is active, this should be shown on the dialogue, with an additional warning, 2 seconds before the booster overheats.

  Introduce 2 speed packages which when the hover runs over it, it will double its speed for 2 seconds. The packages should then be destroyed.

Milestone 7: Opponent and Damages
Reward: 76% to 90%

- Implement an opponent 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 enemy hovers.

Implement a basic damage model for the player hover:

The car starts with 100 hit points. Every time the car collides with an object it should lose 5 hit points. Display the current number of hit points in the user interface.

Introduce obstacles that reduce health by 5, 10, and 15 depending on their size and impact.
Slow down the hover upon collision by 75% for -15 health, 50% for -10 health and 25% for -5 health

Milestone 8: Further Improvements
Reward: 91% to 100%

- 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)

- Give the player the opportunity to restart the game instead of quitting at the end of the game.

- Make the game acknowledge that the player achieved a successful move through a checkpoint:

  Place a cross centred on the checkpoint’s arc when the player has successfully negotiated the check-point. Provide the cross with a life timer so that the cross disappears after a short while.

- Indicate the player’s current race position to the dialogue, and at the end of the race, display the race winner and their race time.

- Improve the car’s dynamics when it accelerates, and when it turns:

  Make the car lean into the bends as it turns, e.g. like a motorcycle, or an airplane does when it turns.

  The car should lift up slightly at the front as it accelerates.

Milestone 9: A Truly Professional Game
Reward: 101% and beyond!

If you made it here, you are doing great – so don’t stop!!

What follows is a list of ideas which you could implement in order to make your game much more playable and fun. Also, feel free to add anything extra that you want, as long as it doesn’t break the requirements of the previous sections. I have lots of bonus points wanting to be distributed to good attempts, in this section.

- Add more opponent hovers.

  Hint: Do this by writing the COpponent class (if you haven’t already done so). Once this class is properly debugged, you can make an array of COpponent objects, and call each element’s **COpponent::Update()**.

- Scatter some unexploded bombs (e.g. using the Flare model) round the track. When a car passes too close to a bomb, it should explode. If a bomb explodes close to the player car, the First-Person and Third-Person cameras must shake (but the Free-Moving and Surveillance cameras should not 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.

- Place some barrels round the track which contain burning fires. Implement each fire by using a particle system for the flames.

- Use an array to act as a "speed table" for the opponent cars, so that they behave in a more realistic way. The enemy cars would thus move at a customisable speed between each waypoint. You could make this smoother by having the opponent(s) accelerate and decelerate between each new speed.

- Implement a damage model for the opponent hovers: Bumping into them (and causing them to bump into obstacles) will cause them to be damaged and slow down.

- Damaged hovers (both, the player's and the opponents') could emit smoke, and/or tilt 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.

- Finally, here's a big one: 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. In this case, the level editor would help you create this file. Develop it further, and you could release it as part of your game, to support **modding**, i.e. to allow players to actually create and share their own race tracks.

Code Style and Layout

The functionality described in the milestones above is not everything. **Around 40%** is dedicated to following good programming practices. Here's a reminder of what we are looking for:

- Your code must be commented appropriately.
- Your code must be properly indented and laid out.
- You must give meaningful names to your variables.
- You must make use of enumerated types or int's and macros to control the states within the game where appropriate.
- You must have no magic numbers, but instead make proper use of constants.
- You must make good and extensive use of modularisation, i.e. create functions.
- You must use variable timing on anything that moves in your game.
- You must make proper use of arrays, with loops to process them.
- Where appropriate, you must also use object orientation, i.e. create classes.

Report

You **must** write a one-page technical report of 600 words, which is around two pages A4.
The report must be about your game, describing:

- The functionality of your game, i.e. how it works.
- The implementation of your game, i.e. how you achieved the above functionality. Particularly, make sure you explain how your collision detection functions work (including maths).
- What grade you expect to get.

  Hint: Some students lose marks when their expectations are not realistic!

You must also include, as an extra single-page appendix to your report, a scale map (which may be hand-drawn and scanned, or electronically produced) of your race course, showing the course boundary and the positions of obstacles, checkpoints, etc. (all labelled).

Final Tips

This assignment is thus far your biggest opportunity to make it into the game- and software-development world, by creating something which truly feels like a game.

Remember: ***Getting stuck during your first software development assignments is a natural part of becoming a good programmer*** – so don't panic!

When you *do* get stuck, here are a few things which could help you:

- Try to remember our discussions in class
- Ask me for help – i may give you a hint, or even a full answer
- Talk about your problem with a classmate.
- Search for help online
- Read a book!

Finally, here are a few of things that you should definitely ***NOT*** do when you get stuck:

- **Do not** ask a friend to give you code
- **Do not** copy anyone's code
- **Do not** give your code to anyone
- **Do not** give up!

Follow the advice above, and you will become an ace developer before you even know it!

So, happy coding!

PREPARATION FOR THE ASSESSMENT

Resources

- This assignment has an associated set of media files. These can be found inside the file "Assignment 3 – Media.zip" on Blackboard.
- The zip file has been password protected with the usual password.
- The files include the following models:
    - race2.x A hover car
    - heckpoint.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 is also available in “Assignment 3 – Extra Models.zip”.
- 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.
- All of the models and textures are under license and must not be passed on to anyone else or used for any purpose other than your university work.

RELEASE DATES AND HAND IN DEADLINE

Assessment Release date: [20/01/2023]

Assessment Deadline:21/04/23 at 22:00 CY Time
Demonstration: TBA

Please note that this is the final time you can submit – not the time to submit!
Your feedback/feed forward and mark for this assessment will be provided on 13/05/2022

SUBMISSION DETAILS

Deliverables

There are two deliverables: Your code, and your report.

Submission

- Submission is electronic, via Blackboard.
- Submit your report as a Microsoft Word file, or as a .pdf file.
- For the code, multiple files are permitted. All files related to code must be either header files (.h) or source files (.cpp). Do not submit your entire Visual Studio solution!
- You must include your name as a comment on the first line of each source code file.
- Submit all your files (code and report) together in a single .zip file. The filename must be your surname and first name, e.g. if your name is Andreas Andreou, then your submission must be AndreouAndreas.zip.
- Assessment is by demonstration during our lab session on the date specified above.

Frequently Lost Marks

You are strongly encouraged to follow the guidelines in this assignment brief closely.
Deviating from the guidelines in any way causes you to lose marks.

Here are a few other ways of how you may lose marks:

- Plagiarising in any way and by any amount means that you receive 0.
- Submitting code which does not compile means you receive 0.
- Not submitting a report means you receive 0.
- Late submission by 1 minute to 5 working days means that your mark is capped to 40.
- Late submission by more than 5 working days means you receive 0.
- Not demonstrating your work at the specified date means you receive 0.
- Providing insufficient answers during your demonstration makes you lose marks.
- Achieving less than 40 in our Mathematics Test means that your mark is capped to 40.

Not completing all tasks of a given milestone causes you to lose all marks related to all following milestones

HELP AND SUPPORT

Please edit the below to describe how any questions arising from this assessment brief should be handled – e.g. tutorials in seminars, online forum, etc.

- Enter here details for how academic support for this assessment will be provided
- For support with using library resources, please contact CyprusLibrary@uclan.ac.uk. You will find links to lots of useful resources in the My Library tab on Blackboard.
- If you have not yet made the university aware of any disability, specific learning difficulty, long-term health or mental health condition, please contact the Student Support CyprusStudentSupport@uclancyprus.ac.cy.
- To access mental health and wellbeing support, please contact Student Support at CyprusStudentSupport@uclancyprus.ac.cy.
- If you have any valid mitigating circumstances that mean you cannot meet an assessment submission deadline and you wish to request an extension, you will need to apply online prior to the deadline. For this, contact the School Office Admin: SchoolsAdmin@uclancyprus.ac.cy.

Disclaimer: The information provided in this assessment brief is correct at time of publication. In the unlikely event that any changes are deemed necessary, they will be communicated clearly via e-mail and a new version of this assessment brief will be circulated.	Version: 1