# CSCU9N6 Assignment

*Due by 4pm, Friday the 26th March 2021*

For this assignment, you are required to create a 2D platform game of your choice. The type of game you choose to implement is up to you but it should be suitable for implementation as a 'platform' or maze type game where the underlying structure can be controlled using a tile map. The complete assignment is worth 70% of the overall mark for the CSCU9N6 module.

**Specification**

Your submission should be based around a single player, 2D game using the programming components and principles discussed in the lectures and practicals. Your game should include the following elements:

- Animations
- One or more moving player controlled sprites
- Multiple moving computer controlled sprites that interact with the player and environment
- Correct collision detection and handling for all sprites and the tile map
- Sounds, including playing a sound using your own novel sound filter
- Multiple keyboard and mouse event handlers
- 2D tile maps with at least 2 levels

Each of these elements is worth between 10% and 15% of the total assignment marks so please do not concentrate on achieving a small subset of them to a high standard while excluding the rest.

In order to achieve a good mark for each component, you must extend and expand upon the concepts discussed in the lectures. For example, when playing a sound filter, you should define one of your own and use this, rather than copy the sound filter code provided in lectures and practicals. When implementing collision detection, you should use multiple levels of collision analysis and aim to be efficient when detecting and handling collisions.

To achieve an excellent mark, your game world will need to be greater in size than the physical windowed game size. In this type of game, the world should appear to move around the player rather than the player move around the world (the player will appear stationary). You should be able to use the offset values discussed in lectures to achieve this. The best approach to implement this is to keep the complete game world coordinate system consistent and then adjust the perspective only at the point you *draw* it to the screen.

The game should have at least two playable levels and should demonstrate that you have learnt how to integrate the various elements discussed in the lectures into a cohesive game format. 20% of the marks will be awarded regarding the challenges involved in writing the

code that controls your game sprites. For example, it is more challenging to implement collision detection if the sprites are constantly trying to move (e.g. due to gravity or momentum).

Please note that the code should compile successfully - a penalty will be applied to any code that does not compile and only a minimal effort will be made to try and make it compile if there are problems. If you have code that attempts to implement a particular feature but it is causing problems, please just comment it out and make a note of what it was trying to do in your code and also in your report. This may still gain you some marks based on what you were trying to implement.

**Assignment Code**

The **initial source code** for the 2D assignment is based on the GameCore code that we have developed in lectures and practicals. This contains an improved version of the Sprite.java class that was seen in the lectures. It also contains two extra attributes 'scale' and 'rotation', which are used in a new method 'drawTransformed' to draw a transformed version of a Sprite when a suitable graphics parameter is supplied. This provides an alternative to the basic 'draw' method. 'scale' and 'rotation' can be set and retrieved using the methods 'setScale, getScale, setRotation' and 'getRotation' respectively. Examine the source code to see how these methods work.

The Velocity.java class provides the ability to manipulate and specify a Velocity as a speed and direction. Using this class, you can create a velocity object with a particular speed and direction and then query the object to find out what the corresponding change in vertical and horizontal pixels per millisecond should be. These queries are achieved using the Velocity methods 'getdx' and 'getdy' respectively.

With the exception of the Velocity.java class, you must use all of the above classes in your code (your main game class should extend the GameCore class and can be based off Game.java). ***You must not use a different code library or game engine to build your game*** or use any library code that you did not write yourself. *Any attempt to do this will result in a mark of 0 for your assignment*. You can add further functionality to the supplied classes but you should not remove code.

In addition to the game, you should submit a short report of up to 2 sides of A4 that should honestly evaluate your implementation of the game and point out where you think it was successful and where it needs improvement. You should also use this as an opportunity to highlight any features of your game that may not be readily obvious but that you think worked well. At the end of the report, please briefly note how your game could be extended based on the code that you have already written. This report is worth 10% of the total assignment marks.

**Submission Process**

Your assignment should be submitted in two parts. For the assignment code, please *compress the complete IDE project folder* sufficient to compile and run your game into a ZIP file and upload this as a single file onto the Canvas assignment submission page. For the

report, please submit this as a second file on the Canvas submission page. Feedback for the complete assignment including the code submission will be provided in response to the report submission on Canvas.

*Please ensure that you put your student ID number but not your name on the first page of the report and all code that you submit.*

**Late Submission**

If you cannot meet the assignment hand in deadline and have good cause, please see the module coordinator to explain your situation and discuss an extension as soon as possible after the problem becomes known. Coursework will be accepted up to seven days after the hand-in deadline (or expiry of any agreed extension) but the mark will be lowered by three marks per day or part thereof. After seven days, the work will be deemed a non-submission and will receive an X for this assignment.

**Plagiarism**

Work which is submitted for assessment must be your own work. Plagiarism means presenting the work of others as though it were your own. The University takes a very serious view of plagiarism, and the penalties can be severe (ranging from a reduced mark in the assessment, through a fail mark for the module, to expulsion from the University for more serious or repeated offences). We check submissions carefully for evidence of plagiarism, and pursue those cases we find.