CSCU9N6 2D Platform Game Assignment

Due by 4pm, Friday 22nd March 2019

For this assignment you are required to create a 2D 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 *60% 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 a similar amount of the total assignment marks so please do not concentrate on achieving a small subset of them while excluding the rest. Note however that a minimal attempt at each element will get a *passing mark* at best. 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.

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. Significant 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).

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.

Report

You need also to submit a short report of around 2 sides of A4 with your assignment. This report is worth 10% of the total assignment marks. This report should:

1. Give a brief summary of the game play.
2. Discuss how your prototype game was implemented and how it could be extended from a software implementation point of view.
3. Discuss how you would add further functionality to your game given extra time and honestly evaluate your application and point out where you think it was successful and where it needs improvement.

Outline Marking Scheme
Although the scope of this assignment is quite broad, the marking is based upon the inclusion and effective implementation of the following components:

Animations & Sprites (25%)
The animation and sprites should exhibit a combination of the following properties (exceeding in one area can compensate for weaker performance in another area):

- Player and multiple computer sprites should be present
- The player and computer-controlled sprites should move
- Modification of the default animation and sprite classes to improve visual appearance/performance, for example stopping animations if the player stops moving (or similar)
- The sprites should be well controlled (i.e. move when and where you would expect them to)
- The player sprite(s) should have keyboard and mouse events correctly associated with them (UI below).

Complexity & Collision Detection (20%)
Both player- and computer-controlled sprites should collide properly with other game elements. Collision detection should use bounding box and circle based collision detection in a two-step approach. Collisions should be checked against tiles and sprites. Bonus marks for correct handling of sprites affected by constant force (e.g. gravity).

2D Tile Map (20%)
Does the solution include a 2D tile map and is interaction with this tile map correctly handled? 2 map levels?

Sounds (10%)
Has a sound filter been included and used appropriately? Has an attempt been made at implementing and using multiple sounds? Are the sounds triggered at the right time? Are the sounds preloaded or just loaded when required to be played?

UI Events (15%)
Have several different types of keyboard and mouse event handlers been implemented? At a minimum, the system should respond to different keystrokes in different ways and incorporate some mouse actions. Use of the Mouse Wheel or similar advanced events would be considered a bonus. Are any special UI features used, e.g. use of custom buttons with mouse cursor mapping to button area?

The sum of the above is capped at 60% of the available mark if the game world is limited to the windowed world (i.e. the tile map does not scroll).

Report (10%)
Did the report give an honest appraisal of the implementation and provide suitable ideas on how it could be extended and improved? Did it show an understanding of the general topics discussed in lectures?

TOTAL:100%

Assignment Starter Code
The initial source code for the 2D assignment is the GameCore code you have been using in your practicals. A link to a zip file containing the relevant files can be found on Canvas in the *Assignment* section.

Within this zip file, you should find a *game2D* package with the following elements:

Animation.java, GameCore.java, ScreenManager.java, Tile.java, TileMap.java, *Sprite.java, Velocity.java*

There is an improved version of the Sprite.java class that was seen in the lectures. It now 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.

Please note that you must use all of the above classes in your code (your main game class should extend the GameCore class). ***You must not use a different code library to build your game or use any library code that you did not write yourself***. You can add further functionality to the supplied classes but you should not remove code. As a starting point, you may wish to use the code in Game.java.

Submission Process
Your assignment should be submitted in two parts. For the **assignment code**, please put a copy of *all* the relevant source and resource files required to compile and run your game in the folder:

\\wsv.cs.stir.ac.uk\CSCU9N6\<YourIDNumber>

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 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.

For the **report**, please use the Assignment page on Canvas to submit it.

Feedback for the complete assignment including the code submission will be provided in response to this report submission on Canvas.

Please ensure that you put your student ID number but not your name on the first page of any material 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.