6CS003 Graphics and Artificial Intelligence 2017\_2018

Assessment 1

**Tasks 1 to 6- Graphics**

**Please remember you only need to complete 5 of these for this part of the assessment.**

Contents

[Assignment Overview 3](#_Toc454798167)

[Marking Criteria 4](#_Toc454798168)

[Getting started 4](#_Toc454798169)

[Task 1 of 6: Drawing an Ellipse 5](#_Toc454798170)

[Purpose of Task: 5](#_Toc454798171)

[Getting Started 6](#_Toc454798172)

[Task 2 of 6: Transformation Matrices in 2D 8](#_Toc454798173)

[Purpose of Task: 8](#_Toc454798174)

[Getting Started 9](#_Toc454798175)

[Task 3 of 6: Drawing a Bezier Curve 11](#_Toc454798176)

[Purpose of Task: 11](#_Toc454798177)

[Getting Started 11](#_Toc454798178)

[Task 4 of 6: Pan & Zoom 14](#_Toc454798179)

[Purpose of Task: 14](#_Toc454798180)

[Task 5 of 6: Line Intersection 17](#_Toc454798181)

[Purpose of Task: 17](#_Toc454798182)

[Getting Started 18](#_Toc454798183)

[Task 6 of 6: Race Track 19](#_Toc454798184)

[Purpose of Task: 19](#_Toc454798185)

# Assignment Overview

**Deadlines: These tasks form the Graphics half of the portfolio and the programs are to be demonstrated to a tutor during the workshop classes. Deadlines for each part are given with the exercise's description.**

**The purpose of this assessment is for you to implement several of the standard algorithms in computer graphics in HTML 5, with JavaScript and Canvas.**

This assessment tests your practical understanding of some of the fundamental graphic algorithms.

The assessment is divided up into 5 parts, which are outlined on the following pages. For each part you are given specifications for one or more JavaScript Functions.

To help you to test the routines, we provide you with a test-bed program, and data where needed, for each part of the assessment. Provided that your routines are coded in accordance with the specifications given, the test-bed programs should allow you to test your routines through manipulation of simple graphics.

You must get each part signed-off, on the slip (below), by the staff during the workshop sessions. This page should be handed in at the end of the assessment (Week 6), together with your programs zipped together and submitted to WOLF. No further documentation is required.

======================================================================

6CS003 Assignment 1:

Tasks 1 to 6 sign-off slip.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Student Name:** |  | | **Student No:** |  | |
| **Exercise** | **Suggested Deadline** | **Comments** | | **Tutor**  **Signature** | **Marks** |
| **Task 1:**  ***Circle*** | **Week 3** |  | |  |  |
| **Task 2:**  ***Matrices*** | **Week 4** |  | |  |  |
| **Task 3:**  ***Bezier Curve*** | **Week 5** |  | |  |  |
| **Task 4:**  ***Pan & Zoom*** | **Week 10** |  | |  |  |
| **Task 5:**  ***Line Intersection*** | **Week 11** |  | |  |  |
| **Task 6:**  ***Race Track*** | **Week 12** |  | |  |  |

**You only need to complete 5 of these ☺**

## Marking Criteria

|  |  |
| --- | --- |
| **Criteria** | **Marks** |
| **For Each of the 5 Tasks** |  |
| Each exercise completed and signed off by the suggested deadline | 7 |
| *or* Each exercise completed and signed off after the suggested deadline | or 4 |
| *or* Each exercise signed off, but with some errors etc. still to be resolved | or 2 |
|  |  |
| **Coding (submitted)** |  |
| Code clarity, efficiency, layout, comments, | Up to 2 marks |
| Passes JSLint | 1 mark |
| **TOTAL** | 10 |
| **5 Tasks = 5 x 10** | 50 |

## Getting started

This assessment is to be coded using JavaScript sitting on HTML 5, which is available on Faculty PC’s.

All of the programs, data and other information that you require may be downloaded from Wolf. We assume you will use notepad++ or PS Pad.

All 5 programs can be run by clicking index.html

**You should not need to change any file except “jsToBeUpdated.js”**

# Task 1 of 6: Drawing an Ellipse

## Purpose of Task:

To draw an "Circle" to the given parameters.

**Deadline:** Workshop in Week 3

**Definition of Function to be edited:**

drawEllipse(x, y, r, nSteps)

The circle should be drawn with its centre at the given co-ordinates (**x** and **y**) and with radius of **r**.

Since it is impossible to draw a perfect circle on the screen, in practice, a circle is normally approximated by a set of short, straight lines; the more lines, the better the "circle". In this case the final parameter (nSteps) defines how many line segments to divide the circle into.

**Note:** The co-ordinates of any point on the edge of a circle **centred at the origin** may be obtained from the following equations.

Y-axis

X = Radius\*cosine() Y = Radius\*sine() {where is an angle.}

X-axis

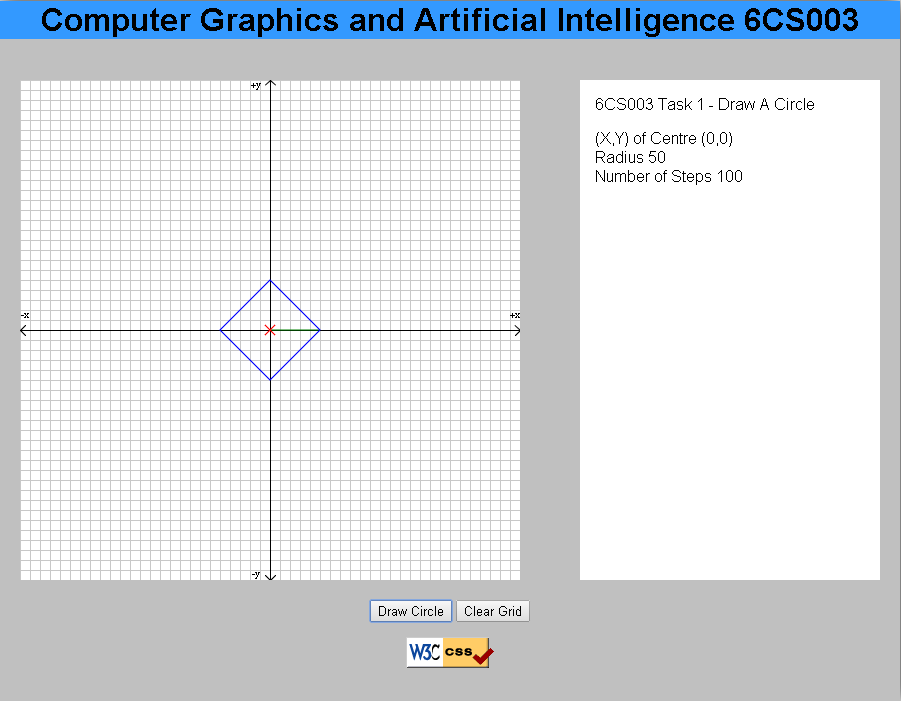
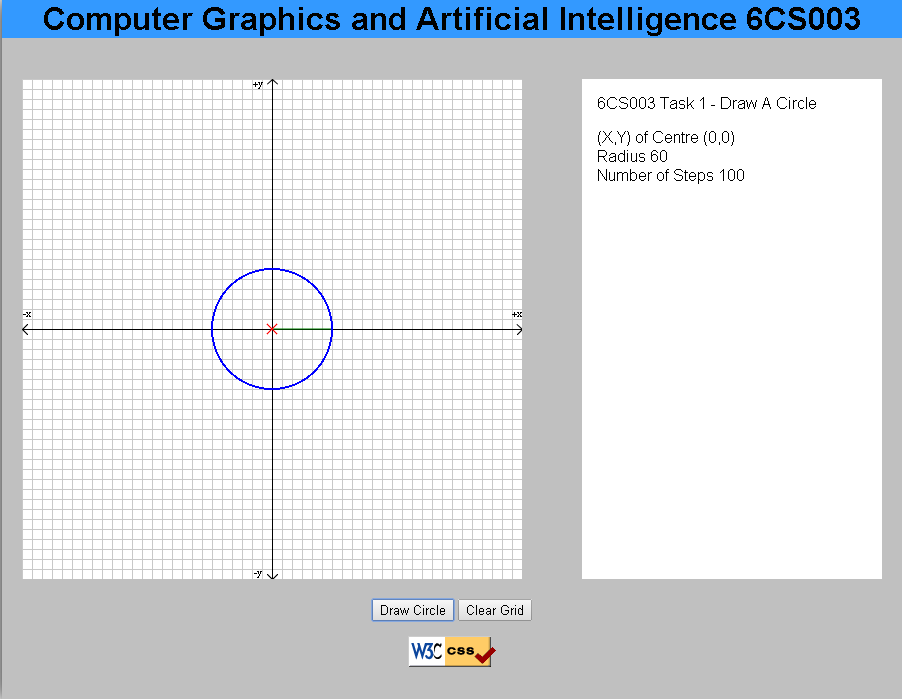
## 

## Getting Started

The Task 1 folder contains the files you need.

When you click on the “Draw Circle” button it reads in the parameter values from the user interface and calls the function drawCircle (…) to draw the circle.

A red cross is drawn at the centre of the circle and a green “radius-line” so that you can check that your circle is drawn in the correct position and at the correct size.

Screen shot from Task 1 without changes made Screen shot from Task 1 with changes made

There is no data file for this program; when you run the application, you will be asked to enter:

* X and Y of circle
* Radius
* Number of Steps

*N.B. The X and Y axes have their origin in the centre of the drawing area which is of size 800 by 600. (each square is 10x10)*

# Task 2 of 6: Transformation Matrices in 2D

## Purpose of Task:

To perform basic matrix transformations on 2d shapes.

**Deadline:** Workshop in Week 4

**Definition of Functions to be edited:**

* matrixMultiply ()
* transformCoordsX (position)
* transformCoordsY (position)
* matrix Multiply() Performs standard Matrix multiplication of two matrices,

i.e. M3 = M1 x M2

* transformCoordsX (position) and transformCoordsX (position) use the composite matrix M( ) to transform the original shapeCoords. Returning each coordinate as called.

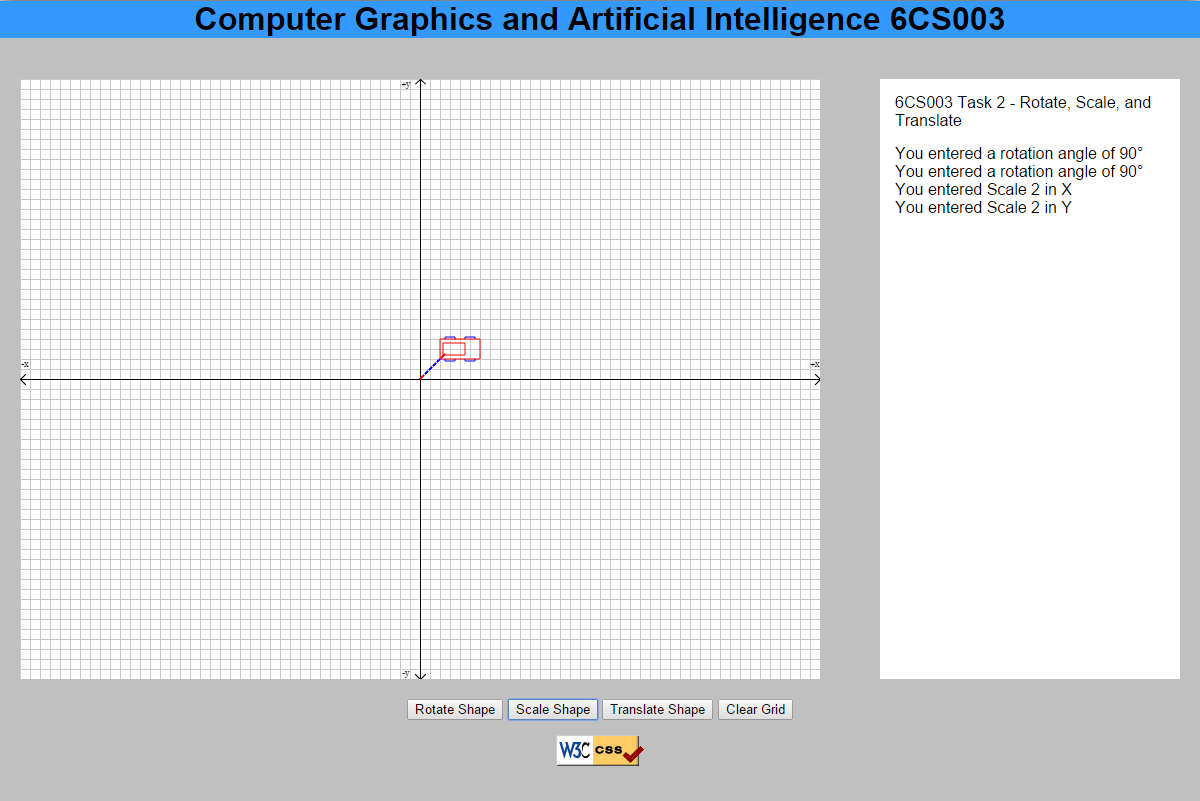
The arrays are all using 0-base indexing (i.e M1(0,0) is the top left element, M1(2,2) is the bottom right element). The **first index is the row number and the second index is the column number**. (i.e M1(0,2) is the top-right element.)

## Getting Started

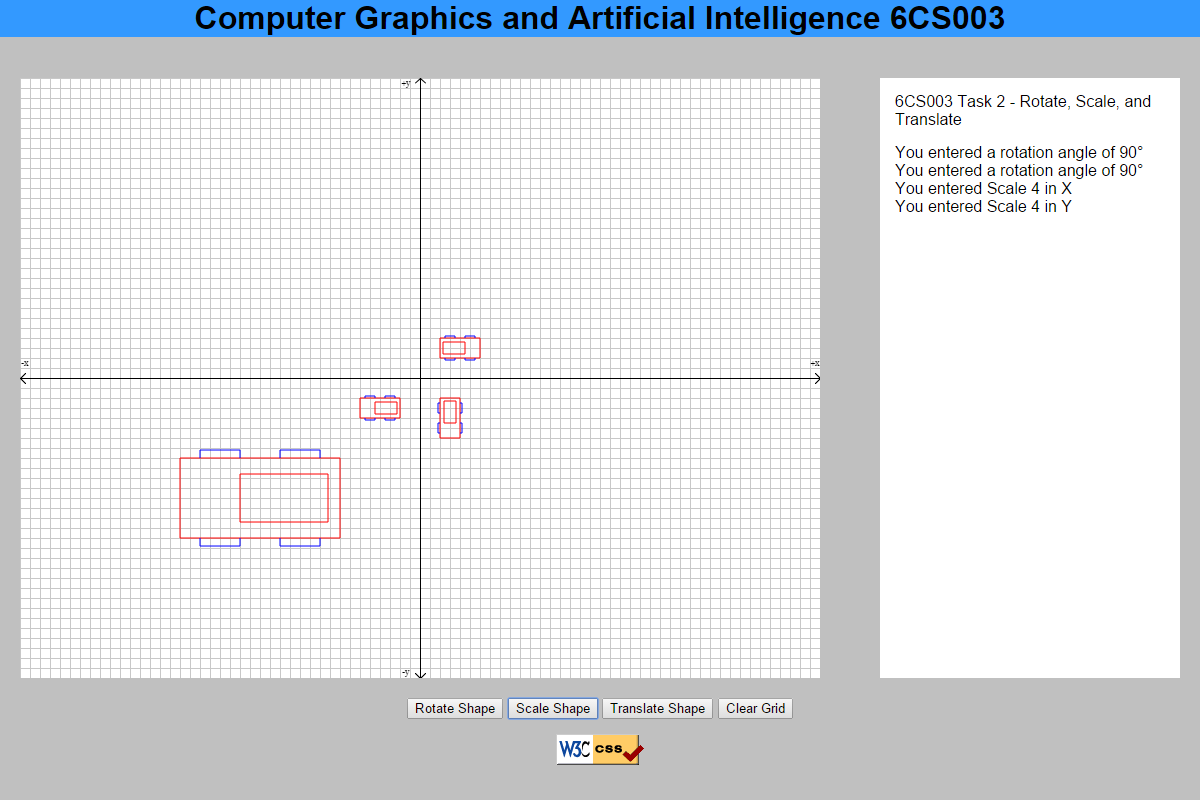
The Task 2 folder contains the files you need.

Once the application starts it displays a small car.

The user then selects a transformation option (Rotate, Translate, or Scale) and then enters the relevant value(s) to see the car redrawn.



Screen shot from Task 2 without changes made



Screen shot from Task 2 with changes made

The program:

* calls **matrixMultiply** to update the composite transformation matrix, by appending the latest transformation matrix to the previous ones.
* The program then calls **transformCoordsX and transformCoordsY** to transform each corner of the original polygon
* finally the transformed shape is drawn.
* The next transform can then be entered.
* Each is cumulative upon those preceding it.

There is no data file associated with this program.

Again, you will find “dummy” functions for **transformCoordsX and transformCoordsY** and **matrixMultiply** in “**jsToBeUpdated.js”**. The code that they contain is simply there to ensure that the program will run; it is not correct. Your task is to replace the dummy code with correct code.

# Task 3 of 6: Drawing a Bezier Curve

## Purpose of Task:

To draw a Bezier curve to fit 4 control points (cubic Curve).

**Deadline:** Workshop in Week 5.

**Definition of Function to be edited:**

drawBezierCurve()

You need only to alter the code that draws the blue lines, this needs altering to draw a Bezier curve.

N.B. The control points themselves and the straight lines joining them are already drawn in red by the main program in the initialiseExample () function.

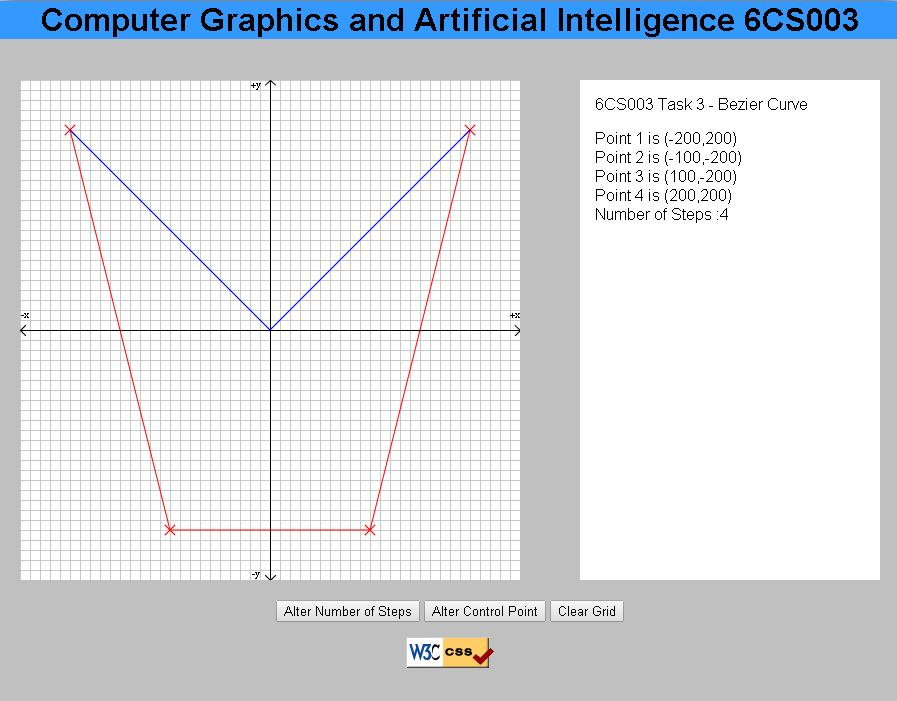
## Getting Started

The Task 3 folder contains the files you need.

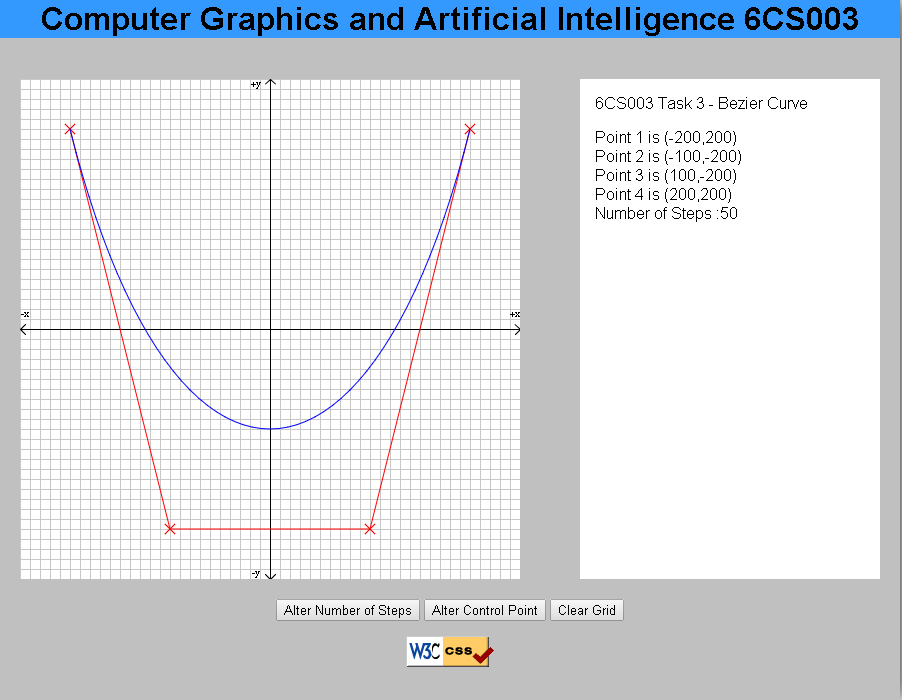
When the application first starts, you will see the initial locations for the four control points shown as red crosses. The red crosses are linked via straight lines to show the straight line path between points. Once these are drawn, the application calls the drawBezierCurve() function.

Until you correct it, the drawBezierCurve() function just draws two blue lines to represent the approximate position of the Bezier curve.

The initial number of steps for the Bezier curve is four, but as you can see from the drawBezierCurve() function, these currently have no effect on the lines.



Screen shot from Task 3 without changes made



Screen shot from Task 3 with changes made

The user is able to change the co-ordinates of any of the control points one at a time by clicking on the “Change Control Point” button. The control points and curve are then redrawn. The number of steps for the Bezier curve can be changed by clicking on the button “Alter Number of Steps” button. When the new number of steps has been entered, the control points and curve are redrawn.

**Further Guidance**

The arrayX and arrayY contain the coordinates of the X, Y points

# Task 4 of 6: Pan & Zoom

## Purpose of Task:

To calculate new Window co-ordinates to allow the user to "zoom" in or out on the image, or to look at the image which is "off-the-side" of the current window.

**Deadline:** Workshop in Week 10.

**Definition of Function to be edited:**

function zoomAndPan(direction) {

}

The parameters xMin, xMax etc. define, on input, the current window boundary. The direction value indicates the zoom or pan operation to be performed.

Values for direction are:

7 8 9

\ | / Pan directions

4 -- + -- 6 from the current

/ | \ position (marked by "+")

1 2 3

OR 5 to Zoom in,

0 to Zoom out.

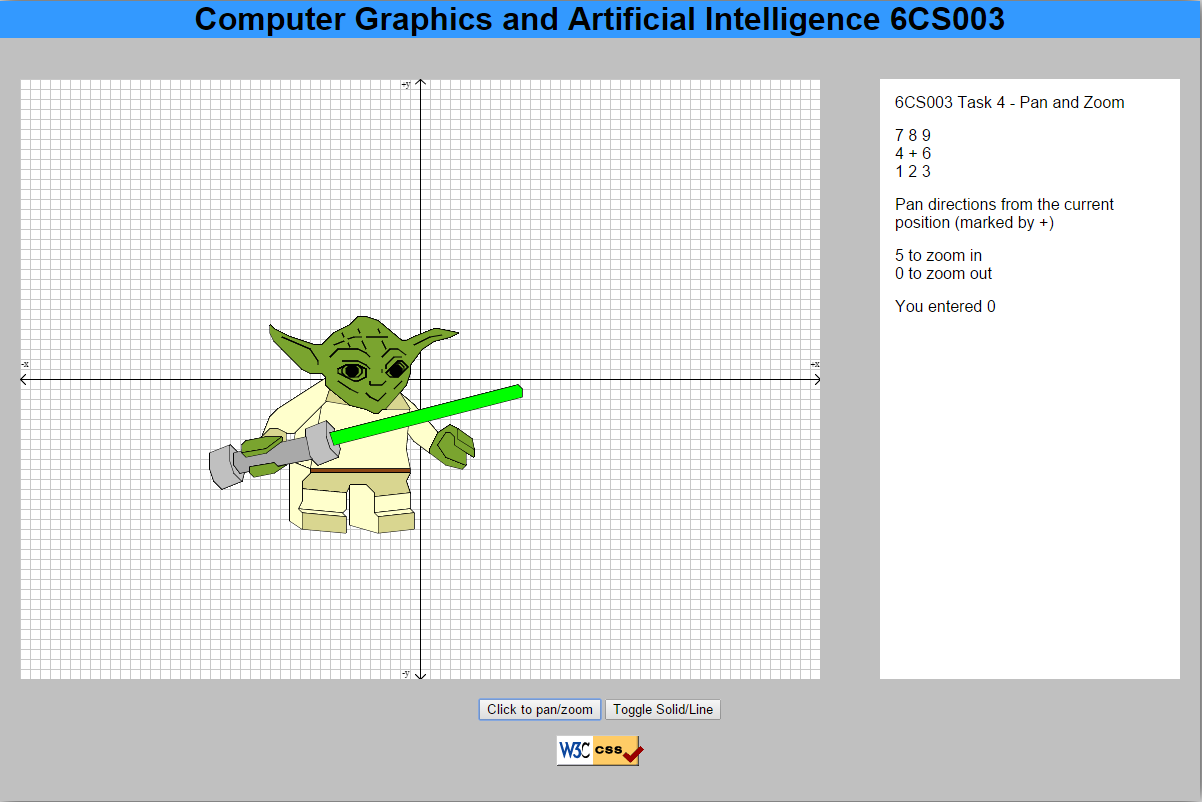
For Pan Operations there should be a 25% overlap in each direction (up, down, left, right) between the current window and the new one.

For Zoom in, the new window will be half the width and height of the current, but with the same centre co-ordinate.

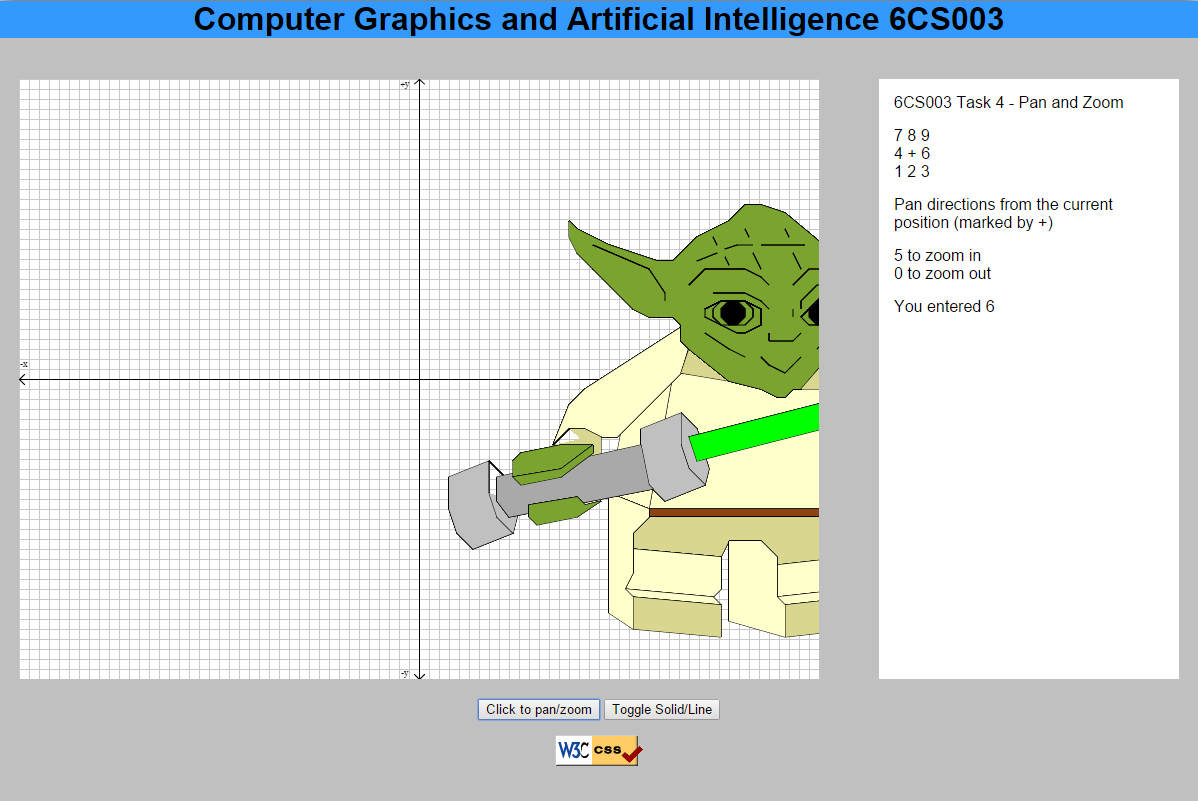
For Zoom out, the new window will be twice the width and height of the current, again with the same centre co-ordinate.

**Test-Bed Program: Task 5**

Viewport is fixed at the whole of the drawing area. Initially the program displays the whole model on the screen. The user then selects the zoom or pan value and the direction is passed to the function zoomAndPan(). The new Window co-ordinates are processed, and the Window/Viewport recalculated and the image redrawn.



Screen shot from Task 5 without changes made



Screen shot from Task 5 with changes made (5 then 6)

Appendix

This is included for information as the solution is incorporated into the framework you already have.

Transformation Matrices in 2D: Window to Viewport Transformations

Purpose:

To convert model co-ordinates into screen co-ordinates.

Definition of Function

setWindowViewport()

Given the:

* Window boundary co-ordinates (wxMin, wxMax, wyMin, wyMax)
* Viewport boundary co-ordinates (vxMin, vxMax, vyMin, vyMax)

This function creates a transformation matrix (composite), which is used when converting coordinates from model to screen space.

The format of the transformation is given by the equations:

Vx := AWx + B; {where Wx,Wy are the model co-ordinates..}

Vy := CWy + D; {... Vx,Vy are the screen co-ordinates..}

{... and A,B,C,D are constants for the..

{... Window to Viewport transformation}.

i.e. the composite matrix will have the format: **( A 0 B)**

**( 0 C D)**

**( 0 0 1)**

**Tips**

Remember how this translates into a transformation matrix i.e.

**( sx 0 tx)**

**( 0 sy ty)**

**( 0 0 1 )**

# Task 5 of 6: Line Intersection

## Purpose of Task:

To calculate and return the co-ordinates, if any, of the intersection point of two lines. In addition it will categorise the type of intersection (see below).

**Deadline:** Workshop in week 11

**Definition of student-written routines:**

lineIntersection(x1Coord1, y1Coord1, x1Coord2, y1Coord2, x2Coord1,

x2Coord1, x2Coord2, y2Coord2)

* Where
* (x1Coord1,y1Coord1)(x1Coord2,y1Coord2) Represent start and end points Line 1
* (x2Coord1,y2Coord1)(x2Coord2,y2Coord2) Represent start and end points Line 2

The values in result [] are the x and y co-ordinates of the intersection point, as calculated by the function, and an integer value which indicates what type of intersection we have.

|  |  |
| --- | --- |
| **IntersectType** | **Intersection type (between line segments 1 & 2)** |
| 3 | Intersection point lies on both line segments |
| 2 | Intersection point lies on line segment 2 but not on line segment 1 |
| 1 | Intersection point lies on line segment 1 but not on line segment 2 |
| 0 | Intersection point does not lie on either line segment |
| -1 | Line segments are parallel - no intersection point (values of x and y not set) |
| -2 | Line segments are co-linear - no intersection point (values of x and y not set) |

Your code should determine the (x, y) co-ordinates of the intersection (if they do cross) and the type of intersection (if there is one). Return value zero if there is no intersection.

## Getting Started

The Task 5 folder contains the files you need.

When you first run this code (without your amendments to correct this code) it will draw a pair of lines. The program needs the intersection point and type of intersection to function correctly.

There is test data built into the code which should check for all 6 types of intersection. Take care when testing vertical or horizontal lines. Do you need to write extra code to allow for these?

# Task 6 of 6: Race Track

## Purpose of Task:

To alter task 3 to a quadratic Bezier curve (3 point), and add functionality to the program to create a race track like layout.

**Deadline:** Workshop in Week 12.

**Definition of Requirements**

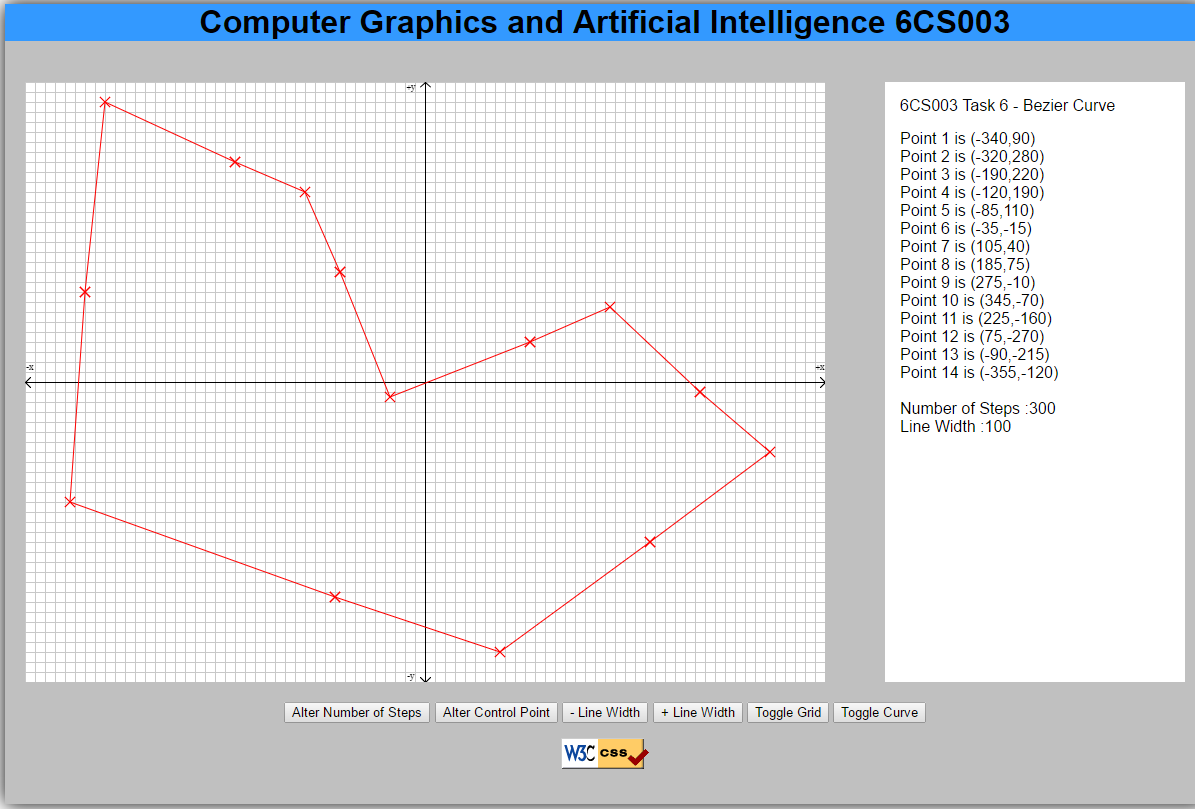
Step 1: Add your drawBezierCurve() from Task 3 to the “**jsToBeUpdated.js”** file.

Step 2: Alter your code to calculate 3 points (Quadratic curve) instead of 4 points (Cubic curve).

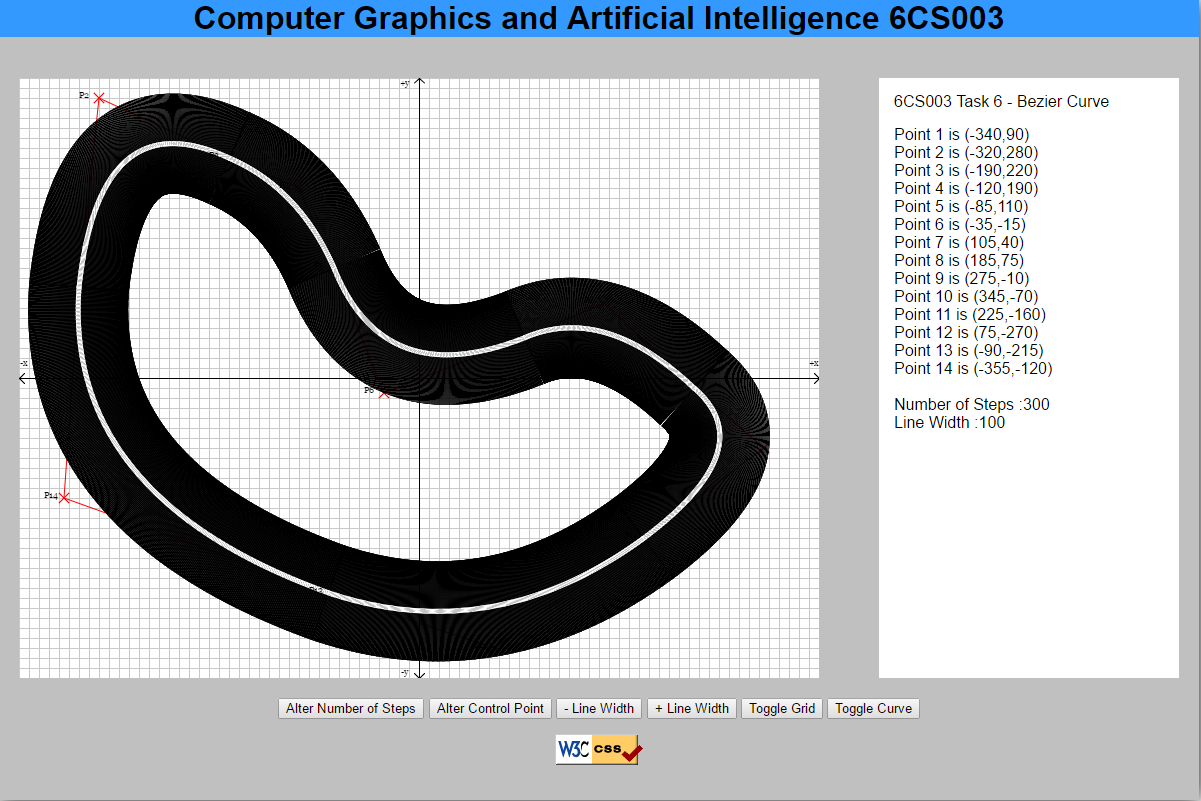
Step 3: Alter the line width to 100, and colour black, to create a wide Bezier curve.

Step 4: Add a second drawBezierCurve() function and make the line width in white to width 5, or alter the code to simply alter the width for 2 tracks.

Step 5: Add 2 car models from Task 2 to the track, but filled in as in task 4, i.e. 2 solid models rather than line drawings, facing the correct direction down the track.



Screen shot from Task 3 without changes made



Screen shot from Task 3 with changes made (no cars shown)