WORKSHEET A

Version 1.0
Computing
COMP270

Dr Ed Powley

Introduction

In this worksheet you will use cubic Bézier curves to implement a top-down 2D race track.

Begin by **forking** the following git repository:

`https://github.com/Falmouth-Games-Academy/comp270-worksheet-A`

Complete the tasks described below, remembering to **commit** your work regularly. To submit your work, open a **pull request** from your forked repository to the original repository.

The template program in this repository has a number of empty functions which the tasks below ask you to fill in. Feel free to add or modify any other code that makes your solutions easier to write or maintain — for example you are encouraged to add your own helper methods to the provided classes in order to enable code reuse.

The following video demonstrates what successfully completed solutions to the tasks should look like:

`https://youtu.be/KugZcz6YZTQ`

Background

Given four points in 2D space

$$\mathbf{p}_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}, \mathbf{p}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \mathbf{p}_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \mathbf{p}_3 = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix},$$

a *cubic Bézier curve* is defined by

$$b(t) = (1-t)^3\mathbf{p}_0 + 3(1-t)^2 t\mathbf{p}_1 + 3(1-t)t^2\mathbf{p}_2 + t^3\mathbf{p}_3$$

where $0 \leq t \leq 1$. See Figure 1.

The provided template program defines a `Bezier` class which represents a cubic Bézier curve. The program also defines a race track as a chain of Bézier curves, stored as `std::vector<Bezier> m_track`. The curves are set up such that the end point of one equals the start point of the next, and the start point of the first equals the end point of the last, giving the appearance of a continuous loop.

Task 1: drawing Bézier curves

For the purposes of rendering, a Bézier curve can be approximated by a series of straight lines. If $b(t)$ denotes the point on the curve with parameter t, then

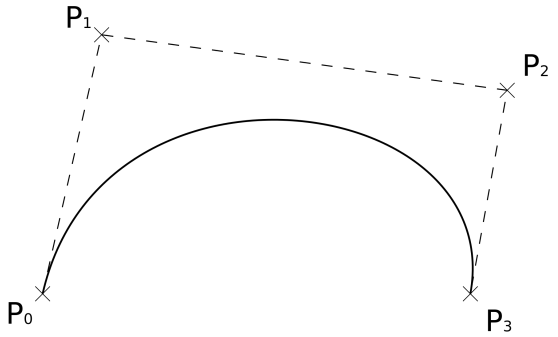

Figure 1: A cubic Bézier curve. Image source: `https://commons.wikimedia.org/wiki/File:Bezier_curve.svg`.

the curve can be drawn as a sequence of n line segments by drawing lines between points

$$b\left(\frac{i}{n}\right) \text{ and } b\left(\frac{i+1}{n}\right)$$

for $i = 0, \ldots, n-1$. For example if $n = 2$, the curve is approximated by two lines: one from $b(0.0)$ to $b(0.5)$, and one from $b(0.5)$ to $b(1.0)$. The larger n is, the smoother the curve appears.

In the code, the `Bezier` class has a `draw` method which currently does nothing. **Implement** the method so that it draws the curve, using `SDL_RenderDrawLine` or `SDL_RenderDrawLineF` to draw the line segments. The number of segments (n in the above discussion) should be defined as a constant so that it can be tuned; a value of 20 is a good default.

Task 2: following the curve

If the track is made up of n curves, denoted $b_0, \ldots, b_{n-1}$, then we can represent a position along the track as a number s with $0 \leq s \leq n$. The integer part of s gives the curve number, and the fractional part gives the t parameter along the curve. For example, $s = 1.2$ represents curve 1 and parameter value 0.2, so $b_1(0.2)$.

In C++, the integer part of a floating point number can be found[1] by using `static_cast<int>(s)`. The fractional part can be found using `fmodf(s, 1.0f)`.

The `Application` class has a `drawCarOnTrack` method which currently does nothing. It takes a single argument, `position`, which acts as s in the discussion above. **Implement** the method so that it draws a car sprite at the given position on the track; when done correctly, the sprite will be animated performing laps of the track.

Your implementation should call `drawCar` to actually draw the sprite. This function takes two arguments: a position (as a `Vector2`), and an angle. Pass an angle of 0 for now; the next task is to calculate a more appropriate angle.

Task 3: facing the tangent

At a point with parameter t, the tangent to the cubic Bézier curve is given by[2]

$$b'(t) = 3(1-t)^2(\mathbf{p}_1 - \mathbf{p}_0) + 6(1-t)t(\mathbf{p}_2 - \mathbf{p}_1) + 3t^2(\mathbf{p}_3 - \mathbf{p}_2)\,.$$

[1] Some caution is required using these with negative numbers; however we always have $s \geq 0$ so this is not an issue here.

[2] Those of you who know calculus can verify that this is the derivative of $b(t)$ with respect to t.

Modify your implementation of `Application::drawCarOnTrack` such that the car is drawn facing along the tangent to the curve — that is, in the direction of the track. Modify the angle passed to `drawCar` to achieve this. You will find the `atan2f` function useful in converting from the tangent vector to the required angle; however note that `atan2f` returns an angle in radians whereas `drawCar` expects an angle in degrees.

Task 4: movement speed

You will notice that the car appears to speed up and slow down on certain parts of the track. This is because we are incrementing t by a constant amount every frame (in `Application::updateCarPosition`, but this does not necessarily correspond to a constant distance along the curve.

Modify the implementation of `Application::updateCarPosition` to achieve a more consistent movement speed. Figuring out exactly how to achieve this is part of the exercise.

Marking Rubric

Criterion	Weight	Refer for Resubmission	Adequate	Competent	Very Good	Excellent	Outstanding
Basic competency threshold	30%	A reasonable attempt at the worksheet was not submitted by the formative deadline.	A reasonable attempt at the worksheet was submitted by the formative deadline. There is no evidence of academic misconduct.				
Functional coherence	40%	None of the tasks have been attempted.	Task 1 has been attempted and partially completed.	Task 1 has been successfully completed.	Tasks 1 and 2 have been successfully completed.	Tasks 1–3 have been successfully completed.	Tasks 1–4 have been successfully completed.
Maintainability	30%	The code is only sporadically commented, if at all, or comments are unclear. Few identifier names are clear or inappropriate. Code formatting hinders readability.	The code is well commented. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to a consistent formatting style. There is little obvious duplication of code or of literal values.	The code is reasonably well commented. Most identifier names are descriptive and appropriate. Most code adheres to a sensible formatting style. There is almost no obvious duplication of code or of literal values.	The code is reasonably well commented, with appropriate high-level documentation. Almost all identifier names are descriptive and appropriate. Almost all code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values. Some literal values can be easily "tinkered".	The code is very well commented, with comprehensive appropriate high-level documentation. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no obvious duplication of code or of literal values. Most literal values are, where appropriate, easily "tinkered".	The code is commented extremely well, with comprehensive appropriate high-level documentation. All identifier names are descriptive and appropriate. All code adheres to a sensible formatting style. There is no duplication of code or of literal values. Nearly all literal values are, where appropriate, easily "tinkered".