Physics Assignment 1 – Week 3.3 Vector basics

Introduction

With this programming assignment, *5 bonus points* can be gained which count towards the final course grade (see the course manual for grading details). For each assignment part, the points are indicated.

These bonus point can be gained by *signing off your solutions during the labs, at the latest during the third week* (week 3.5).

In addition, these assignments prepare for the final assignment, which should include most of these aspects.

All assignment parts are optional, all assignment parts for this week are strongly recommended, and serve as a basis for future assignments.

Assignment 1.1 Vec2 implementation – 1 point

Take the example 003 given on Blackboard as a starting point and complete the Vec2 struct with the following methods:

- - operator - returns the result of subtracting two vectors (without modifying either of them)
- * operator - returns the result of scaling a vector (by a float value, that can appear on the left or the right)
- Length - returns the length of the current vector
- Normalize - normalizes the current vector
- Normalized - returns a normalized version of the current vector without modifying it
- SetXY - sets the x & y of the current vector to the given two floats

Unit testing: Include a small test case for **every** operation that you implement. Here is an example for the scalar multiplication operator:

```
Vec2 myVec = new Vec2 (2,3);

Vec2 result = myVec * 3;
Console.WriteLine("Scalar multiplication right ok?: " +
   (result.x == 6 && result.y==9 && myVec.x==2 && myVec.y==3));

Vec2 result = 4 * myVec;
Console.WriteLine("Scalar multiplication left ok?: " +
   (result.x == 8 && result.y==12 && myVec.x==2 && myVec.y==3));
```

Use *nontrivial values* for your test cases (e.g. coordinates must be nonzero), and values for which you know the correct outcome. Chances are if you understand how to write a correct test case, you also know how to implement the operation. As you can see, test cases are very small and non-graphical, and immediately show whether the method is correctly implemented.

ASSIGNMENT 1.2 FIXING THE GIVEN EXAMPLE – 1 POINT

Start with the given example 003. Use your completed Vec2 struct and the information from the lecture on normalizing and scaling Vec2 to ensure that there is no speed difference between diagonal and non-diagonal movement.

ASSIGNMENT 1.3 AVOID THE BALL GAME – 2 POINTS

Create a simple game using Assignment 1.3 as a starting point:

- The ball follows the mouse, starting slowly.
- The ball speed increases with time.
- If the ball reaches (=overlaps) the mouse position the game is over. This hit test must be exact, so you cannot use the Sprite HitTest or HitTestPoint methods for this check; use the methods implemented in Vec2 instead.
- A timer is printed on screen; this is your score when the game ends. Display this score.

Everything must be done using operations defined in your Vec2 struct.

Steps involved:

- Calculate the delta vector between mouse and ball
- Normalize this delta
- Multiply with current speed
- Increase speed each frame
- Check distance between ball & mouse: either move, or go to game over

ASSIGNMENT 1.4 INERTIA – 1 POINT

Extend your game from 1.3 by adding a type of *inertia:* every frame, let the ball velocity be a combination of 99% of the previous frame's velocity, and 1% of the desired velocity (as computed in 1.3).

Answer the following questions:

- What happens if you use different percentages?
- What happens if you use percentages that don't add up to 100%?
- What happens if one of the percentages is negative?

Note: if you implemented it correctly, and the percentages add up to 100%, you implemented the *linear interpolation formula* for vectors – an essential tool to know!