gg1126

Guy Goudeau

December 1, 2015

ADPG-110 Assessment

**I.1. Description of the Problem**

a.) Name: Retro Redux

Problem Statement: Develop a static math library to that adheres to OOP design principals and promotes reuse and portability for better use and development of C++ code.

Problem Specification: Math Library must include the following aspects: 2 Dimensional & 3 Dimensional Vectors, Dot Product, Normalization, Cross Product, Magnitude, Operator overloading, Linear Interpolation, Vector 4 Normalization & Construct from Hexadecimal Color value, and conversion of degrees to radians.

**I.3.2. Output Items**

a.) Description: rPosition1 determines rectangle 1’s position

Type: Point2D<int>

Range of Acceptable Values: { 0-1600, 0-800 }

b.) Description: rPosition 2 determines rectangle 2’s position

Type: Point2D<int>

Range of Acceptable Values: { 0-1600, 0-800 }

c.) Description: rColor determines rectangle 1’s color

Type: Color<int>

Range of Acceptable Values: { 0-255, 0-255, 0-255, 0-255 }

d.) Description: rColor2 determines rectangle 2’s color

Type: Color<int>

Range of Acceptable Values: { 0-255, 0-255, 0-255, 0-255 }

e.) Description: cPosition determines circle’s position

Type: Point2D<int>

Range of Acceptable Values: { 0-1600, 0-800 }

f.) Description: TwoD1 defines 2D vector

Type: Vectors<int>

Range of Acceptable Values: (any integer, any integer)

g.) Description: TwoD2 defines 2D vector

Type: Vectors<int>

Range of Acceptable Values: (any integer, any integer)

h.) Description: ThreeD1 defines 3D vector

Type: Vectors<int>

Range of Acceptable Values: (any integer, any integer, any integer)

i.) Description: ThreeD2 defines 3D vector

Type: Vectors<int>

Range of Acceptable Values: (any integer, any integer, any integer)

j.) Description: Vec4 defines 4D vector

Type: Vectors<int>

Range of Acceptable Values: (any integer, any integer, any integer, any integer)

k.) Description: degrees integer determines degrees to be converted to radians

Type: integer

Range of Acceptable Values: any positive integer

**1.4.1 Description**

*Description*: In this program, I have refactored Ben Odom’s graphic-oriented engine to depict use of delta time, 3D vectors, and 4D vectors through creating two boxes with flags attached that scroll from the left to the right of the screen. The “ships” change their RGB color value as they move, and when they reach the right edge of the screen they are shifted back to the left edge and continue moving right indefinitely. There is also a graph shown. When the ‘p’ key is pressed, the console outputs all of the vector math computations from the linked static library. When the ‘o’ key is pressed, you are shown the delta time.

*ReadMe*: To test the math library, press the 'p' key.

To check delta time, press the 'm' key.

My website is located at - http://guygoudeau.github.io/Poppit-and-Locket/

**II.2 Information About the Objects**

a.) Name: Vectors

Description: Class holding functions for vector math library

Class Attributes

b.) Name: V x

Description: Used as placeholders for vector functions

Type: templated character

Range of Acceptable Values: n/a

c.) Name: V y

Description: Used as placeholders for vector functions

Type: templated character

Range of Acceptable Values: n/a

d.) Name: V z

Description: Used as placeholders for vector functions

Type: templated character

Range of Acceptable Values: n/a

e.) Name: V w

Description: Used as placeholders for vector functions

Type: templated character

Range of Acceptable Values: n/a

Class Operations

a.) Prototype: Vectors twoAdd(Vectors &other)

Description: adds two 2D vectors

Precondition: there are two existing 2D vectors with correct number of arguments

Postcondition: an added vector will be shown

Visibility: Public

b.) Prototype: Vectors threeAdd(Vectors &other)

Description: adds two 3D vectors

Precondition: there are two existing 3D vectors with correct number of arguments

Postcondition: an added vector will be shown

Visibility: Public

c.) Prototype: Vectors threeSub(Vectors &other)

Description: subtracts one 3D vector with another

Precondition: there are two existing 3D vectors with correct number of arguments

Postcondition: a subtracted vector will be shown

Visibility: Public

d.) Prototype: int twoMagnitude()

Description: find magnitude of 2D vector

Precondition: there is an existing 2D vector with correct number of arguments

Postcondition: magnitude will be outputted as an integer

Visibility: Public

e.) Prototype: int threeMagnitude()

Description: find magnitude of 3D vector

Precondition: there is an existing 3D vector with correct number of arguments

Postcondition: magnitude will be outputted as an integer

Visibility: Public

f.) Prototype: Vectors twoNormalize()

Description: will normalize a 2D vector

Precondition: there is an existing 2D vector with correct number of arguments

Postcondition: normalized vector will be outputted

Visibility: Public

g.) Prototype: Vectors threeNormalize()

Description: will normalize a 3D vector

Precondition: there is an existing 3D vector with correct number of arguments

Postcondition: normalized vector will be outputted

Visibility: Public

h.) Prototype: Vectors v4Normalize()

Description: will normalize a 4D vector

Precondition: there is an existing 4D vector with correct number of arguments

Postcondition: normalized vector will be outputted

Visibility: Public

i.) Prototype: int twoDotProd(Vectors &other)

Description: finds dot product of two 2D vectors

Precondition: there are two existing 2D vectors with correct number of arguments

Postcondition: dot product will be outputted as an integer

Visibility: Public

j.) Prototype: int threeDotProd(Vectors &other)

Description: finds dot product of two 3D vectors

Precondition: there are two existing 3D vectors with correct number of arguments

Postcondition: dot product will be outputted as an integer

Visibility: Public

k.) Prototype: Vectors twoCrossProd(Vectors &other)

Description: finds cross product of two 2D vectors

Precondition: there are two existing 2D vectors with correct number of arguments

Postcondition: cross product will be outputted as an integer

Visibility: Public

l.) Prototype: Vectors threeCrossProd(Vectors &other)

Description: finds cross product of two 3D vectors

Precondition: there are two existing 3D vectors with correct number of arguments

Postcondition: cross product will be outputted as an integer

Visibility: Public

m.) Prototype: void twoPrint()

Description: prints out 2D vector

Precondition: there is a 2D vector that needs to be printed

Postcondition: prints correctly syntaxed vector

Visibility: Public

n.) Prototype: void threePrint()

Description: prints out 3D vector

Precondition: there is a 3D vector that needs to be printed

Postcondition: prints correctly syntaxed vector

Visibility: Public

o.) Prototype: void fourPrint()

Description: prints out 4D vector

Precondition: there is a 4D vector that needs to be printed

Postcondition: prints correctly syntaxed vector

Visibility: Public

p.) Prototype: Vectors twoSub(Vectors &other)

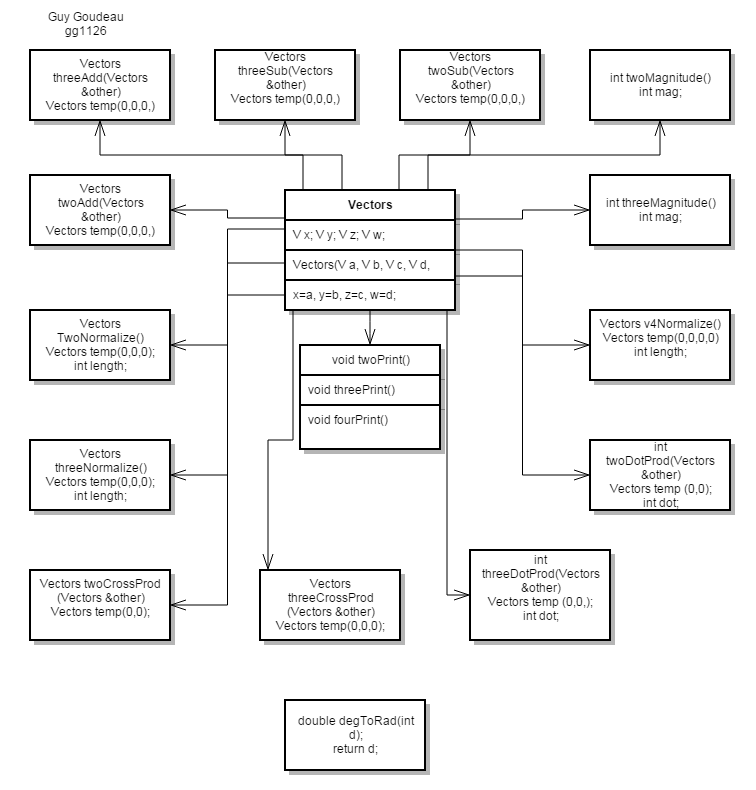
Description: subtracts one 2D vector from another

Precondition: there are two existing 2D vectors

Postcondition: will output subtracted vector

Visibility: Public

**II.4 Design Diagrams**



**III.1 Source Code**

*Vector Math Static Library:*

#pragma once

template <typename V>

class Vectors

{

public:

V x;

V y;

V z;

V w;

Vectors(V a, V b)

{

x = a;

y = b;

}

Vectors(V a, V b, V c)

{

x = a;

y = b;

z = c;

}

Vectors(V a, V b, V c, V d)

{

x = a;

y = b;

z = c;

w = d;

}

Vectors twoAdd(Vectors &other)

{

Vectors temp(0, 0);

temp.x = x + other.x;

temp.y = y + other.y;

return temp;

}

Vectors threeAdd(Vectors &other)

{

Vectors temp(0, 0, 0);

temp.x = x + other.x;

temp.y = y + other.y;

temp.z = z + other.z;

return temp;

}

Vectors twoSub(Vectors &other)

{

Vectors temp(0, 0, 0);

temp.x = x - other.x;

temp.y = y - other.y;

temp.z = z - other.z;

return temp;

}

Vectors threeSub(Vectors &other)

{

Vectors temp(0, 0, 0);

temp.x = x - other.x;

temp.y = y - other.y;

temp.z = z - other.z;

return temp;

}

int twoMagnitude()

{

int mag = sqrt((x ^ 2) + (y ^ 2));

return mag;

}

int threeMagnitude()

{

int mag = sqrt((x ^ 2) + (y ^ 2) + (z ^ 2));

return mag;

}

Vectors twoNormalize()

{

Vectors temp(0, 0);

int length = sqrt((x ^ 2) + (y ^ 2));

temp.x = x / length;

temp.y = y / length;

return temp;

}

Vectors threeNormalize()

{

Vectors temp(0, 0, 0);

int length = sqrt((x ^ 2) + (y ^ 2) + (z ^ 2));

temp.x = x / length;

temp.y = y / length;

temp.y = z / length;

return temp;

}

Vectors v4Normalize()

{

Vectors temp(0, 0, 0, 0);

int length = sqrt((x ^ 2) + (y ^ 2) + (z ^ 2) + (w ^ 2));

temp.x = x / length;

temp.y = y / length;

temp.z = z / length;

temp.w = w / length;

return temp;

}

int twoDotProd(Vectors &other)

{

int dot;

Vectors temp(0, 0);

temp.x = x \* other.x;

temp.y = y \* other.y;

dot = temp.x + temp.y;

return dot;

}

int threeDotProd(Vectors &other)

{

int dot;

Vectors temp(0, 0, 0);

temp.x = x \* other.x;

temp.y = y \* other.y;

temp.z = z \* other.z;

dot = temp.x + temp.y + temp.z;

return dot;

}

Vectors twoCrossProd(Vectors &other)

{

Vectors temp(0, 0);

temp.x = y \* other.x - x \* other.y;

temp.y = x \* other.y - y \* other.x;

return temp;

}

Vectors threeCrossProd(Vectors &other)

{

Vectors temp(0, 0, 0);

temp.x = y \* other.z - z \* other.y;

temp.y = z \* other.x - x \* other.z;

temp.z = x \* other.y - y \* other.x;

return temp;

}

void twoPrint()

{

cout << "( " << x << " , " << y << " )" << endl;

}

void threePrint()

{

cout << "( " << x << " , " << y << " , " << z << " )" << endl;

}

void fourPrint()

{

cout << "( " << x << " , " << y << " , " << z << " , " << w << " )" << endl;

}

};

double degToRad(int d)

{

return (d / 180)\*3.14;

}

*Main Source Code:*

#include "GameLoop.h" // include game loop header

#include "NewVectorLib.h" // include vector math library

#include <iostream> // include iostream library

#include <ctime> // include time library

using namespace std; // use std namespaces

SDL\_Keycode key; // makes key to be pressed for switch statements

System::Point2D<int> rPosition1 = { 0, 200 }; // rectangle 1's position

System::Point2D<int> rPosition2 = { 0, 700 }; // rectangle 2's position

System::Color<int> rColor = { 50, 50, 255, 255 }; // rectangle 1's color

System::Color<int> rColor2 = { 50, 50, 255, 255 }; // rectangle 2's color

System::Point2D<int> cPosition = { 800, 600 }; // circle's position

Vectors <int> TwoD1(6, 9); // define 2D vector

Vectors <int> TwoD2(9, 6); // define 2D vector

Vectors <int> ThreeD1(-6, 50, 6); // define 3D vector

Vectors <int> ThreeD2(4, 2, 14); // define 3D vector

Vectors <int> Vec4(25, 30, 35, 255); // define 4D vector

int degrees = 360;

Vectors <int> twoAdded = TwoD1.twoAdd(TwoD2);

Vectors <int> twoSubbed = TwoD1.threeSub(TwoD2);

Vectors <int> threeAdded = ThreeD1.threeAdd(ThreeD2);

Vectors <int> threeSubbed = ThreeD1.threeSub(ThreeD2);

int twoMag = TwoD1.twoMagnitude();

int threeMag = ThreeD1.threeMagnitude();

Vectors <int> twoNorm = TwoD1.twoNormalize();

Vectors <int> threeNorm = ThreeD1.threeNormalize();

Vectors <int> vec4Norm = Vec4.v4Normalize();

int TwoDot = TwoD1.twoDotProd(TwoD2);

int ThreeDot = ThreeD1.threeDotProd(ThreeD2);

Vectors <int> twoCross = TwoD1.twoCrossProd(TwoD2);

Vectors <int> threeCross = ThreeD1.threeCrossProd(ThreeD2);

float currentTime = 0; // define float currentTime equal to 0

float previousTime = 0; // define float previousTime equal to 0

float deltaTime = 0; // define float deltaTime equal to 0

void GameLoop::Loop() // start the game

{

while (m\_bRunning) // While the game is running, do the following

{

SDL\_Event sdlEvent; // Will hold the next event to be parsed

while (m\_bRunning) // While the game is running, do the following

{

// Events get called one at a time, so if multiple things happen in one frame, they get parsed individually through 'SDL\_PollEvent'

// The next event to parse gets stored into 'sdlEvent', and then passed to the 'EventHandler' class which will call it's appropriate function here

// 'SDL\_PollEvent' returns 0 when there are no more events to parse

while (SDL\_PollEvent(&sdlEvent))

{

// Calls the redefined event function for the EventHandler class

// Refer to its header file and cpp for more information on what each inherited function is capable of

// and its syntax

OnEvent(sdlEvent);

}

float div = 1000; // define float div equal to 1000

float ct = clock(); //define float ct equal to system clock time

currentTime = (float)ct / div; // currentTime equals ct \* div

deltaTime = currentTime - previousTime; // deltaTime equals currentTime - previousTime

Update(); // run update function

LateUpdate(); // run lateupdate function

Draw(); // run draw function

DrawGraph(); // run drawgraph function

previousTime = currentTime; // set previousTime equal to currentTime

Graphics::Flip(); // Required to update the window with all the newly drawn content

}

}

}

void GameLoop::DrawGraph() // draws graph function

{

Graphics::DrawLine({ 400, 600 }, { 900, 600 }, { 0, 0, 255, 255 }); // x line

Graphics::DrawLine({ 400, 600 }, { 400, 100 }, { 0, 0, 255, 255 }); // y line

float x = 400; // beginning x position of intervals

for (int i = 0; i < 10; i++) // i = number of intervals

{

x += 50; // space between each interval, plus means going right

Graphics::DrawLine({ x, 590 }, { x, 610 }, { 255, 0, 0, 255 }); // x intervals

}

float y = 600; // beginning y position of intervals

for (int i = 0; i < 10; i++) // i = number of intervals

{

y -= 50; // space between each interval, minus means going up

Graphics::DrawLine({ 390, y }, { 410, y }, { 255, 0, 0, 255 }); // y intervals

}

System::Point2D<int> cPosition2 = { 400, 600 }; // define circle 2's position

for (int i = 0; i < 11; i++) // run loop 11 times, and for each loop

{

Graphics::DrawCircle(cPosition2, 5, 50, { 0, 255, 0, 255 }); // draw a small circle

cPosition2.X += 50; // then add 50 to x

cPosition2.Y -= 50; // then subtract 50 from y

}

}

void GameLoop::Update() // Update function definition

{

rPosition1.X++\*deltaTime == currentTime; // change rectangle 1's position using deltatime

rPosition1.X++\*deltaTime == currentTime; // do the same thing to make it faster

rColor.Blue = rPosition1.X; // set rectangle 1's blue value to be equal to it's x value

if (rPosition1.X == 400) // if rectangle 1's x is a certain value

{

rColor = { 255, 50, 50, 255 }; // change the color values

}

if (rPosition1.X == 800) // if rectangle 1's x is a certain value

{

rColor = { 50, 255, 50, 255 }; // change the color values

}

if (rPosition1.X == 1200) // if rectangle 1's x is a certain value

{

rColor = { 50, 50, 255, 255 }; // change the color values

}

if (rPosition1.X == 1600) // if rectangle 1's x reaches the right edge of the screen

{

rPosition1.X = 0; // send it back to the left

}

rPosition2.X++\*deltaTime == currentTime; // change rectangle 2's position using deltatime

rColor2.Blue = rPosition2.X; // set rectangle 2's blue value to be equal to it's x value

if (rPosition2.X == 400) // if rectangle 2's x is a certain value

{

rColor2 = { 255, 50, 50, 255 }; // change the color values

}

if (rPosition2.X == 800) // if rectangle 2's x is a certain value

{

rColor2 = { 50, 255, 50, 255 }; // change the color values

}

if (rPosition2.X == 1200) // if rectangle 2's x is a certain value

{

rColor2 = { 50, 50, 255, 255 }; // change the color values

}

if (rPosition2.X == 1600) // if rectangle 1's x reaches the right edge of the screen

{

rPosition2.X = 0; // send it back to the left

}

switch (key) // depending on the key pressed, change the case

{

case SDLK\_p: // if p is pressed, display vector math from VectorMath.lib

cout << "Add 2D vectors: ";

twoAdded.twoPrint();

cout << "Subtract 2D vectors: ";

twoSubbed.twoPrint();

cout << "Add 3D vectors: ";

threeAdded.threePrint();

cout << "Subtract 3D vectors: ";

threeSubbed.threePrint();

cout << "Magnitude of 2D vector: " << twoMag << endl;

cout << "Magnitude of 3D vector: " << threeMag << endl;

cout << "Normalized 2D vector: ";

twoNorm.twoPrint();

cout << "Normalized 3D vector: ";

threeNorm.threePrint();

cout << "Normalized 4D vector: ";

vec4Norm.fourPrint();

cout << "Dot product of 2D vectors: " << TwoDot << endl;

cout << "Dot product of 3D vectors: " << ThreeDot << endl;

cout << "Cross product of 2D vectors: ";

twoCross.twoPrint();

cout << "Cross product of 3D vectors: ";

threeCross.threePrint();

cout << "Degrees to radians: " << degToRad(degrees) << endl;

break; // break case

case SDLK\_m: // if m is pressed

std::cout << "Previous time: " << previousTime << std::endl; // output previous time and end line

std::cout << "Current time: " << currentTime << std::endl; // output current time and end line

std::cout << "Delta time: " << deltaTime << std::endl; // output delta time and end line

break; // break case

}

}

void GameLoop::LateUpdate() // run LateUpdate function

{

key = KMOD\_NONE; // no key is being pressed

}

void GameLoop::Draw() // run Draw function

{

// Objects are drawn in a painter's layer fashion meaning the first object drawn is on the bottom, and the last one drawn is on the top

// just like a painter would paint onto a canvas

Graphics::DrawLine(rPosition1, { rPosition1.X, rPosition1.Y - 50 }, { 250, 250, 250, 255 }); //draw flagpole for rect1

Graphics::DrawCircle({rPosition1.X+14, rPosition1.Y-75}, 30, 3, { 255, 0, 0, 255 }); // draw flag for rect1

Graphics::DrawRect(rPosition1, {50, 50}, rColor); // draw rect1

Graphics::DrawLine(rPosition2, { rPosition2.X, rPosition2.Y - 50 }, { 250, 250, 250, 255 }); //draw flagpole for rect2

Graphics::DrawCircle({ rPosition2.X + 14, rPosition2.Y - 75 }, 30, 3, { 255, 0, 0, 255 }); // draw flag for rect2

Graphics::DrawRect(rPosition2, { 50, 50 }, rColor2); // draw rect2

}

void GameLoop::OnKeyDown(const SDL\_Keycode ac\_sdlSym, const Uint16 ac\_uiMod, const SDL\_Scancode ac\_sdlScancode) // function that happens when a key is pressed down

{

switch (ac\_sdlSym)

{

case SDLK\_ESCAPE: m\_bRunning = false; break; // End the loop

default: printf("%s\n",SDL\_GetKeyName(ac\_sdlSym)); // print they key that's pressed

key = ac\_sdlSym;

break; // break case

}

}

void GameLoop::OnKeyUp(const SDL\_Keycode ac\_sdlSym, const Uint16 ac\_uiMod, const SDL\_Scancode ac\_sdlScancode) // function that happens when a key is released

{

switch (ac\_sdlSym)

{

default: break; // break case

}

}

void GameLoop::OnExit() // on exit of the loop

{

m\_bRunning = false; // End loop

}

GameLoop::GameLoop() // actual loop

{

m\_bRunning = true; // game running bool is true

}

GameLoop::~GameLoop() // idk blame ben

{

}

**IV.3 Operating Directions**

1. Navigate to the ADGP-110 Assessment folder located on the desktop
2. Launch Project.exe
3. Press ‘p’ key to test vector library