**Donray Williams**

**12/14/16**

**Math for Games– Assessment 2 – Year 1**

# **1.0 Requirements Documentation**

# **1.1 Description of problem**

**A. Name:** Math for Games

**B. Problem Statement:** Create a redistributable math’s library for real-time applications.

**C. Problem Specification:** The library must implement the required classes and functions given to us. To ensure that the library functions correctly we were given a Unit Test Application that you must use to test the accuracy of your mathematical methods.

**1.2 Input Information**

A. Our Math Library was used in the creation of our own personal game in which it takes input from the keyboard.

**1.3 Output Information**

A. Using the aieBootStrap the game is drawn to the screen and playable by the user.

**1.4 User Interface**

A. User has all control of the playable characters in the game using the keyboard. You can also exit the game and restart the game using the keyboard.

# **2.0 System Architecture**

**2.1 Member Functions in the classes:**

**Class** *Vector2 ~*

**PROTOTYPE:** bool operator = = (const Vector2 &A)

**DESCRIPTION:** Returns a check if each point is equal to the set one

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns if they it is True or False

**VISIBILITY:** Public

**PROTOTYPE:** Vector2 Vector2::operator +(const Vector2 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and adds each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors added

**VISIBILITY:** Public

**PROTOTYPE:** Vector2 Vector2::operator -(const Vector2 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and subtracts each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors subtracted

**VISIBILITY:** Public

**PROTOTYPE:** Vector2 Vector2::operator \*(float Mult)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and multiplies each point to the number that is taken in and returns that Vector

**PRECONDITION:** Must have Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors multiplied

**VISIBILITY:** Public

**PROTOTYPE:** float Vector2::Magnitude()

**DESCRIPTION:** Multiplies each point times itself then adds each number up and square roots the whole thing

**PRECONDITION:** Must have a Vector

**POST CONDITION:** Returns the length of the Vector

**VISIBILITY:** Public

**PROTOTYPE:** Vector2 Vector2::Normalize()

**DESCRIPTION:** Creates a temporary vector that dives each point by the Magnitude function then returns that Vector

**PRECONDITION:** Must have a 2D Vector

**POST CONDITION:** Returns the Normalized Vector of the x and y divided by the magnitude.

**VISIBILITY:** Public

**PROTOTYPE:** float Vector2::DotProduct(const Vector2 &A)

**DESCRIPTION:** Returns each point times the taken in Vector and adds each of those numbers

**PRECONDITION:** Must have two Vectors

**POST CONDITION:** Returns the Dot Product of the two Vectors

**VISIBILITY:** Public

**Class** *Vector3 ~*

**PROTOTYPE:** bool operator = = (const Vector3 &A)

**DESCRIPTION:** Returns a check if each point is equal to the set one

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns if they it is True or False

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Vector3::operator +(const Vector3 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and adds each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors added

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Vector3::operator -(const Vector3 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and subtracts each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the three Vectors subtracted

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Vector3::operator \*(float Mult)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and multiplies each point to the number that is taken in and returns that Vector

**PRECONDITION:** Must have Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors multiplied

**VISIBILITY:** Public

**PROTOTYPE:** float Vector3::Magnitude()

**DESCRIPTION:** Multiplies each point times itself then adds each number up and square roots the whole thing

**PRECONDITION:** Must have a Vector

**POST CONDITION:** Returns the length of the Vector

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Vector3::Normalize()

**DESCRIPTION:** Creates a temporary vector that dives each point by the Magnitude function then returns that Vector

**PRECONDITION:** Must have a 3D Vector

**POST CONDITION:** Returns the Normalized Vector of the x, y, z divided by the magnitude.

**VISIBILITY:** Public

**PROTOTYPE:** float Vector3::DotProduct(const Vector3 &A)

**DESCRIPTION:** Returns each point times the taken in Vector and adds each of those numbers

**PRECONDITION:** Must have two Vectors

**POST CONDITION:** Returns the Dot Product of the two Vectors

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Vector3::CrossProduct(const Vector3 &A)

**DESCRIPTION:** Creates a temporary vector that values are given by the cross product

**PRECONDITION:** Must have two Vectors

**POST CONDITION:** Returns the Cross Product of the two Vectors

**VISIBILITY:** Public

**Class** *Vector4 ~*

**PROTOTYPE:** bool operator = = (const Vector4 &A)

**DESCRIPTION:** Returns a check if each point is equal to the set one

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns if they it is True or False

**VISIBILITY:** Public

**PROTOTYPE:** Vector4 Vector4::operator +(const Vector4 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and adds each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors added

**VISIBILITY:** Public

**PROTOTYPE:** Vector4 Vector4::operator -(const Vector4 &A)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and subtracts each point to the other and returns that Vector

**PRECONDITION:** Must have a Vector to check against

**POST CONDITION:** Returns the answer of the three Vectors subtracted

**VISIBILITY:** Public

**PROTOTYPE:** Vector4 Vector4::operator \*(float Mult)

**DESCRIPTION:** Creates a temporary Vector that takes the Vector in the arguments and multiplies each point to the number that is taken in and returns that Vector

**PRECONDITION:** Must have Vector to check against

**POST CONDITION:** Returns the answer of the two Vectors multiplied

**VISIBILITY:** Public

**PROTOTYPE:** float Vector4::Magnitude()

**DESCRIPTION:** Multiplies each point times itself then adds each number up and square roots the whole thing

**PRECONDITION:** Must have a Vector

**POST CONDITION:** Returns the length of the Vector

**VISIBILITY:** Public

**PROTOTYPE:** Vector4 Vector4::Normalize()

**DESCRIPTION:** Creates a temporary vector that dives each point by the Magnitude function then returns that Vector

**PRECONDITION:** Must have a 4D Vector

**POST CONDITION:** Returns the Normalized Vector of the x, y, z, and w divided by the magnitude.

**VISIBILITY:** Public

**PROTOTYPE:** float Vector3::DotProduct(const Vector3 &A)

**DESCRIPTION:** Returns each point times the taken in Vector and adds each of those numbers

**PRECONDITION:** Must have two Vectors

**POST CONDITION:** Returns the Dot Product of the two Vectors

**VISIBILITY:** Public

**Class** *Matrix2 ~*

**PROTOTYPE:** Matrix2 Matrix2::operator \* (const Matrix2 &a)

**DESCRIPTION:** Creates a temporary Matrix that holds each equation that will equal that number at that current point then returns that matrix

**PRECONDITION:** Must have a matrix to multiply

**POST CONDITION:** Uses the Matrix2 formula to return the outcome

**VISIBILITY:** Public

**PROTOTYPE:** Vector2 Matrix2::operator \* (const Vector2 &a)

**DESCRIPTION:** Creates a temporary Vector that holds each equation that will equal that number at that current point then returns that vector

**PRECONDITION:** Must have a Vector2 and a Matrix2 to multiply by

**POST CONDITION:**

**VISIBILITY:** Public

**Class** *Matrix3 ~*

**PROTOTYPE:** Matrix3 Matrix3::operator \* (const Matrix3 &a)

**DESCRIPTION:** Creates floats for each individual point in a 3x3 matrix equal to each equation that will get the number at the current point then sets a temporary matrix that has each float and returns it.

**PRECONDITION:** Must have a matrix to multiply

**POST CONDITION:** Uses the Matrix3 formula to return the outcome

**VISIBILITY:** Public

**PROTOTYPE:** Vector3 Matrix3::operator \* (const Vector3 &a)

**DESCRIPTION:** Creates a temporary Vector that holds each equation that will equal that number at that current point then returns that vector

**PRECONDITION:** Must have a Vector2 and a Matrix2 to multiply by

**POST CONDITION:** Returns a 3D Vector

**VISIBILITY:** Public

**PROTOTYPE:** Matrix3 Matrix3::setRotateX(float a)

**DESCRIPTION:** Function for rotating around the X axis for 3D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 3D Matrix

**POST CONDITION:** Shows the 3D Matrix rotated around the X

**VISIBILITY:** Public

**PROTOTYPE:** Matrix3 Matrix3::setRotateY(float a)

**DESCRIPTION:** Function for rotating around the Y axis for 3D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 3D Matrix

**POST CONDITION:** Shows the 3D Matrix rotated around the Y

**VISIBILITY:** Public

**PROTOTYPE:** Matrix3 Matrix3::setRotateZ(float a)

**DESCRIPTION:** Function for rotating around the Z axis for 3D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 3D Matrix

**POST CONDITION:** Shows the 3D Matrix rotated around the Z

**VISIBILITY:** Public

**Class** *Matrix4 ~*

**PROTOTYPE:** Matrix4 Matrix4::setRotateX(float a)

**DESCRIPTION:** Function for rotating around the X axis for 4D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 4D Matrix

**POST CONDITION:** Shows the 4D Matrix rotated around the X

**VISIBILITY:** Public

**PROTOTYPE:** Matrix4 Matrix4::setRotateY(float a)

**DESCRIPTION:** Function for rotating around the Y axis for 4D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 4D Matrix

**POST CONDITION:** Shows the 4D Matrix rotated around the Y

**VISIBILITY:** Public

**PROTOTYPE:** Matrix4 Matrix4::setRotateZ(float a)

**DESCRIPTION:** Function for rotating around the Z axis for 4D using the

overloading operator returning the current instance

**PRECONDITION:** Must have a 4D Matrix

**POST CONDITION:** Shows the 4D Matrix rotated around the Z

**VISIBILITY:** Public

**PROTOTYPE:** Matrix4 Matrix4::operator \* (const Matrix4 &a)const

**DESCRIPTION:** Creates a temporary matrix that is all zeros then sets them all equal

to the two that are multiplied together

**PRECONDITION:** Must have a 4D Matrix

**POST CONDITION:** Returns the multiplied 4D Matrix

**VISIBILITY:** Public

**PROTOTYPE:** Vector4 Matrix4::operator \* (const Vector4 &a)const

**DESCRIPTION:** Creates a temporary vector that is all zeros loops through the vector

and matrix to then multiply each one

**PRECONDITION:** Must have a 4D Vector and a 4D Matrix

**POST CONDITION:** Returns the multiplied 4D Matrix

**VISIBILITY:** Public

**Member Variables:**

**Class** Vector2:

**float** *VA* **[2]-** 1D Array to create my 2D Vector

**float x, y –** The X and Y value of the Vector

**Class** Vector3:

**float** *VA* **[3]-** 1D Array to create my 3D Vector

**float x, y, z –** The X, Y, and Z value of the Vector

**Class** Vector4:

**float** *VA* **[4]-** 1D Array to create my 4D Vector

**float x, y, z –** The X, Y, Z, and W value of the Vector

**Class** Matrix2:

**float** *matrix* **[2] [2]-** 2 x 2 Array to create my 2D Matrix

**Class** Matrix3:

**float** *matrix* **[9] –** 1D Array to create my 3D Matrix

**Class** Matrix4:

**float** *matrix* **[4] [4]-** 4 x 4 Array to create my 4D Matrix

**Class** Player:

**Vector2** *m\_Position*- A Vector for my player

**bool** *m\_Shooting*- A check to see if the player is shooting

**float** *m\_size*- A set size for the round player to check if they are getting hit or not

**bool** *m\_userHit*- A check to see if the User is getting hit or not

**bool** *m\_enemyHit*- A check to see if the Enemy is getting hit or not

**int** *m\_userHP*- A integer to store the Users HP

**int** *m\_enemyHP*- A integer to store the Enemy HP

**Class** Shoot:

**Vector2** *m\_Position*- A Vector for the Bullet

**int** m\_bulletSpeed- A integer value to store the bullet speed

**Class** Application2D:

**bool** *m\_GameOver- A check if the Game is over or not*

**int** *m\_healthBarCompensator-* A integer used to move the health bar right

Source Code for MathforGames:

#include "Math.h"

#include <iostream>

Vector2::Vector2() {};

Vector2::Vector2(float xpos, float ypos) : x(xpos), y(ypos)

{

VA[0] = xpos;

VA[1] = ypos;

}

bool Vector2::operator == (const Vector2 &A)

{

return x == A.x && y == A.y;

}

Vector2 Vector2::operator +(const Vector2 &A)

{

Vector2 tmp = Vector2(x + A.x, y + A.y);

return tmp;

}

Vector2 Vector2::operator -(const Vector2 &A)

{

Vector2 tmp = Vector2(x - A.x, y - A.y);

return tmp;

}

Vector2 Vector2::operator \*(float Mult)

{

Vector2 tmp = Vector2(x \* Mult, y \* Mult);

return tmp;

}

float Vector2::Magnitude()

{

return sqrt((x\*x) + (y\*y));

}

Vector2 Vector2::Normalize()

{

Vector2 tmp = Vector2(x / Magnitude(), y / Magnitude());

return tmp;

}

float Vector2::DotProduct(const Vector2 &A)

{

return (x \* A.x) + (y \* A.y);

}

Vector3::Vector3() {};

Vector3::Vector3(float xpos, float ypos, float zpos) : x(xpos), y(ypos), z(zpos)

{

VA[0] = xpos;

VA[1] = ypos;

VA[2] = zpos;

}

bool Vector3::operator == (const Vector3 &A)

{

return x == A.x && y == A.y && z == A.z;

}

Vector3 Vector3::operator +(const Vector3 &A)

{

Vector3 tmp = Vector3(x + A.x, y + A.y, z + A.z);

return tmp;

}

Vector3 Vector3::operator -(const Vector3 &A)

{

Vector3 tmp = Vector3(x - A.x, y - A.y, z - A.z);

return tmp;

}

Vector3 Vector3::operator \*(float Mult)

{

Vector3 tmp = Vector3(x \* Mult, y \* Mult, z \*Mult);

return tmp;

}

float Vector3::Magnitude()

{

return sqrt((x\*x) + (y\*y) + (z\*z));

}

Vector3 Vector3::Normalize()

{

Vector3 tmp = Vector3(x / Magnitude(), y / Magnitude(), z / Magnitude());

return tmp;

}

float Vector3::DotProduct(const Vector3 &A)

{

return (x \* A.x) + (y \* A.y) + (z \* A.z);

}

Vector3 Vector3::CrossProduct(const Vector3 &A)

{

Vector3 tmp = Vector3(y\*A.z - z\*A.y, x\*A.z - z\*A.x, z\*A.y - z\*A.x);

return tmp;

}

Vector4::Vector4() {};

Vector4::Vector4(float xpos, float ypos, float zpos, float wpos) : x(xpos), y(ypos), z(zpos), w(wpos)

{

VA[0] = xpos;

VA[1] = ypos;

VA[2] = zpos;

VA[3] = wpos;

}

bool Vector4::operator == (const Vector4 &A)

{

return x == A.x && y == A.y && z == A.z && w == A.w;

}

Vector4 Vector4::operator +(const Vector4 &A)

{

Vector4 tmp = Vector4(x + A.x, y + A.y, z + A.z, w + A.w);

return tmp;

}

Vector4 Vector4::operator -(const Vector4 &A)

{

Vector4 tmp = Vector4(x - A.x, y - A.y, z - A.z, w - A.w);

return tmp;

}

Vector4 Vector4::operator \*(float Mult)

{

Vector4 tmp = Vector4(x \* Mult, y \* Mult, z \*Mult, w \*Mult);

return tmp;

}

float Vector4::Magnitude()

{

return sqrt((x\*x) + (y\*y) + (z\*z) + (w\*w));

}

Vector4 Vector4::Normalize()

{

Vector4 tmp = Vector4(x / Magnitude(), y / Magnitude(), z / Magnitude(), w / Magnitude());

return tmp;

}

float Vector4::DotProduct(const Vector4 &A)

{

return (x \* A.x) + (y \* A.y) + (z \* A.z) + (w \* A.w);

}

Matrix2::Matrix2() {};

Matrix2::Matrix2(float a[2][2])

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

matrix2[i][j] = a[i][j];

}

}

}

Matrix2::Matrix2(float a, float b, float c, float d) {

matrix2[0][0] = a;

matrix2[0][1] = b;

matrix2[1][0] = c;

matrix2[1][1] = d;

}

Matrix2 Matrix2::operator \* (const Matrix2 &a)

{

Matrix2 tmp = Matrix2

(

matrix2[0][0] \* a.matrix2[0][0] + matrix2[0][1] \* a.matrix2[1][0],

matrix2[0][0] \* a.matrix2[0][1] + matrix2[0][1] \* a.matrix2[1][1],

matrix2[1][0] \* a.matrix2[0][0] + matrix2[1][1] \* a.matrix2[1][0],

matrix2[1][0] \* a.matrix2[0][1] + matrix2[1][1] \* a.matrix2[1][1]

);

return tmp;

}

Vector2 Matrix2::operator \* (const Vector2 &a)

{

Vector2 tmp = Vector2

(

a.x\*matrix2[0][0] + a.y\*matrix2[1][0],

a.x\*matrix2[0][1] + a.y\*matrix2[1][1]

);

return tmp;

}

Matrix3::Matrix3() {};

Matrix3::Matrix3(float a[9])

{

for (int i = 0; i < 3; i++)

{

matrix3[i] = a[i];

}

}

Matrix3::Matrix3(float a, float b, float c, float d, float e, float f, float g, float h, float i)

{

matrix3[0] = a;

matrix3[1] = b;

matrix3[2] = c;

matrix3[3] = d;

matrix3[4] = e;

matrix3[5] = f;

matrix3[6] = g;

matrix3[7] = h;

matrix3[8] = i;

}

Matrix3 Matrix3::operator \* (const Matrix3 &a)

{

float p1 = (matrix3[0] \* a.matrix3[0]) + (matrix3[1] \* a.matrix3[3]) + (matrix3[2] \* a.matrix3[6]);

float p2 = (matrix3[0] \* a.matrix3[1]) + (matrix3[1] \* a.matrix3[4]) + (matrix3[2] \* a.matrix3[7]);

float p3 = (matrix3[0] \* a.matrix3[2]) + (matrix3[1] \* a.matrix3[5]) + (matrix3[2] \* a.matrix3[8]);

float p4 = (matrix3[3] \* a.matrix3[0]) + (matrix3[4] \* a.matrix3[3]) + (matrix3[5] \* a.matrix3[6]);

float p5 = (matrix3[3] \* a.matrix3[1]) + (matrix3[4] \* a.matrix3[4]) + (matrix3[5] \* a.matrix3[7]);

float p6 = (matrix3[3] \* a.matrix3[2]) + (matrix3[4] \* a.matrix3[5]) + (matrix3[5] \* a.matrix3[8]);

float p7 = (matrix3[6] \* a.matrix3[0]) + (matrix3[0] \* a.matrix3[3]) + (matrix3[8] \* a.matrix3[6]);

float p8 = (matrix3[6] \* a.matrix3[1]) + (matrix3[1] \* a.matrix3[4]) + (matrix3[8] \* a.matrix3[7]);

float p9 = (matrix3[6] \* a.matrix3[2]) + (matrix3[2] \* a.matrix3[5]) + (matrix3[8] \* a.matrix3[8]);

Matrix3 tmp = Matrix3(p1, p2, p3, p4, p5, p6, p7, p8, p9);

return tmp;

}

Vector3 Matrix3::operator \* (const Vector3 &a)

{

float x = (matrix3[0] \* a.x) + (matrix3[3] \* a.y) + (matrix3[6] \* a.z);

float y = (matrix3[1] \* a.x) + (matrix3[4] \* a.y) + (matrix3[7] \* a.z);

float z = (matrix3[2] \* a.x) + (matrix3[5] \* a.y) + (matrix3[8] \* a.z);

Vector3 tmp = Vector3(x, y, z);

return tmp;

}

Matrix3 Matrix3::setRotateX(float a)

{

Matrix3 RotateX = Matrix3

(

1, 0, 0,

0, cos(a), -sin(a),

0, sin(a), cos(a)

);

\*this = RotateX\*\*this;

return \*this;

}

Matrix3 Matrix3::setRotateY(float a)

{

Matrix3 RotateY = Matrix3

(

cos(a), 0, sin(a),

0, 1, 0,

-sin(a), 0, cos(a)

);

\*this = RotateY\*\*this;

return \*this;

}

Matrix3 Matrix3::setRotateZ(float a)

{

Matrix3 RotateZ = Matrix3

(

cos(a), -sin(a), 0,

sin(a), cos(a), 0,

0, 0, 1

);

\*this = RotateZ\*\*this;

return \*this;

}

Matrix4::Matrix4() {};

Matrix4::Matrix4(float a[4][4])

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

matrix4[i][j] = a[i][j];

}

}

}

Matrix4::Matrix4(float a, float b, float c, float d, float e, float f, float g, float h, float i, float j, float k, float l, float m, float n, float o, float p) {

matrix4[0][0] = a;

matrix4[0][1] = b;

matrix4[0][2] = c;

matrix4[0][3] = d;

matrix4[1][0] = e;

matrix4[1][1] = f;

matrix4[1][2] = g;

matrix4[1][3] = h;

matrix4[2][0] = i;

matrix4[2][1] = j;

matrix4[2][2] = k;

matrix4[2][3] = l;

matrix4[3][0] = m;

matrix4[3][1] = n;

matrix4[3][2] = o;

matrix4[3][3] = p;

}

Matrix4 Matrix4::setRotateX(float a)

{

Matrix4 RotateX = Matrix4

(

1, 0, 0, 0,

0, cos(a), -sin(a), 0,

0, sin(a), cos(a), 0,

0, 0, 0, 1

);

\*this = RotateX\*\*this;

return \*this;

}

Matrix4 Matrix4::setRotateY(float a)

{

Matrix4 RotateY = Matrix4

(

cos(a), 0, sin(a), 0,

0, 1, 0, 0,

-sin(a), 0, cos(a), 0,

0, 0, 0, 1

);

\*this = RotateY\*\*this;

return \*this;

}

Matrix4 Matrix4::setRotateZ(float a)

{

Matrix4 RotateZ = Matrix4

(

cos(a), -sin(a), 0, 0,

sin(a), cos(a), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

\*this = RotateZ\*\*this;

return \*this;

}

Matrix4 Matrix4::operator \* (const Matrix4 &a)const

{

Matrix4 tmp = Matrix4(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

for (int i = 0; i < 64; i++)

{

tmp.matrix4[(i / 16)][(i % 16) / 4] += matrix4[(i / 16)][i % 4] \* a.matrix4[i % 4][(i % 16) / 4];

}

return tmp;

}

Vector4 Matrix4::operator \* (const Vector4 &a)const

{

Vector4 tmp = Vector4(0, 0, 0, 0);

for (int i = 0; i < 16; i++)

{

tmp.VA[i / 4] += matrix4[i % 4][i / 4] \* a.VA[i % 4];

}

tmp.x = tmp.VA[0];

tmp.y = tmp.VA[1];

tmp.z = tmp.VA[2];

tmp.w = tmp.VA[3];

return tmp;

}

Source Code for aieBootStrap:

bool Application2D::startup() {

m\_2dRenderer = new aie::Renderer2D();

m\_texture = new aie::Texture("./textures/numbered\_grid.tga");

m\_shipTexture = new aie::Texture("./textures/ship.png");

m\_shipTexture1 = new aie::Texture("./textures/frown.png");

m\_grass = new aie::Texture("./textures/tallgrass.png");

m\_grassball = new aie::Texture("./textures/grassball.png");

m\_font = new aie::Font("./font/consolas.ttf", 32);

m\_shooting = new aie::Audio("./audio/powerup.wav");

m\_audio = new aie::Audio("./audio/music.wav");

m\_pain = new aie::Audio("./audio/pain.wav");

m\_cameraX = 0;

m\_cameraY = 0;

m\_timer = 1;

User.m\_Position = Vector2(50, 370);

Enemy.m\_Position = Vector2(1240, 370);

Bullet.m\_Position = Vector2(40, 370);

Bullet2.m\_Position = Vector2(1240, 370);

User.m\_Shooting = False;

Enemy.m\_Shooting = False;

User.m\_userHit = False;

Enemy.m\_enemyHit = False;

User.m\_userHP = 5;

Enemy.m\_enemyHP = 5;

m\_healthBarCompensator = 0;

m\_GameOver = False;

return True;

}

void Application2D::shutdown() {

delete m\_shooting;

delete m\_audio;

delete m\_pain;

delete m\_font;

delete m\_texture;

delete m\_shipTexture;

delete m\_shipTexture1;

delete m\_grass;

delete m\_grassball;

delete m\_2dRenderer;

}

void Application2D::update(float deltaTime) {

m\_timer += deltaTime;

aie::Input\* input = aie::Input::getInstance();

// play old school music whole time

m\_audio->play();

// use arrow keys to move player

if (input->isKeyDown(aie::INPUT\_KEY\_W) && User.m\_Position.y < 680) //To not go out of range of windows size

User.m\_Position = User.m\_Position + Vector2(0, 10);

if (input->isKeyDown(aie::INPUT\_KEY\_S) && User.m\_Position.y > 40)

User.m\_Position = User.m\_Position - Vector2(0, 10);

if (input->isKeyDown(aie::INPUT\_KEY\_UP) && Enemy.m\_Position.y < 680) //To not go out of range of windows size

Enemy.m\_Position = Enemy.m\_Position + Vector2(0, 10);

if (input->isKeyDown(aie::INPUT\_KEY\_DOWN) && Enemy.m\_Position.y > 40)

Enemy.m\_Position = Enemy.m\_Position - Vector2(0, 10);

// Player movement

(input->isKeyDown(aie::INPUT\_KEY\_SPACE) && User.m\_Shooting == False) ?

(Bullet.m\_Position.y = User.m\_Position.y,

Bullet.m\_bulletSpeed = 0,

m\_shooting->play(),

User.m\_Shooting = True) : 0;

// Second player movement

(input->isKeyDown(aie::INPUT\_KEY\_RIGHT\_CONTROL) && Enemy.m\_Shooting == False) ?

(Bullet2.m\_Position.y = Enemy.m\_Position.y,

Bullet2.m\_bulletSpeed = 0,

m\_shooting->play(),

Enemy.m\_Shooting = True) : 0;

//Bullet Speed in a direction

(User.m\_Shooting == True) ? Bullet.m\_bulletSpeed += 3 : 0;

(Enemy.m\_Shooting == True) ? Bullet2.m\_bulletSpeed -= 3 : 0;

// End of Bullet // Reset the position to shoot from middle of face

(Bullet.m\_Position.x > 1280) ? (Bullet.m\_Position.x = 40, User.m\_Shooting = False) : 0;

(Bullet2.m\_Position.x < 0) ? (Bullet2.m\_Position.x = 1240, Enemy.m\_Shooting = False) : 0;

// Update the position

(User.m\_Shooting == True) ? Bullet.m\_Position.x += Bullet.m\_bulletSpeed : 0;

(Enemy.m\_Shooting == True) ? Bullet2.m\_Position.x += Bullet2.m\_bulletSpeed : 0;

// Collision check for User bullet on enemy

if (Enemy.m\_Position.x - Enemy.m\_size / 2 < Bullet.m\_Position.x && Enemy.m\_Position.x + Enemy.m\_size / 2 > Bullet.m\_Position.x

&& Enemy.m\_Position.y - Enemy.m\_size / 2 < Bullet.m\_Position.y && Enemy.m\_Position.y + Enemy.m\_size / 2 > Bullet.m\_Position.y)

{

m\_pain->stop(); // Stops sound each time to so sound can play every time

m\_pain->play();

Enemy.m\_enemyHit = True; // Set this bool to true to be used later

Enemy.m\_enemyHP -= 1; // Reduce hp by one each time

m\_healthBarCompensator++; // To compensate for HP bar moving right instead of left

}

// Collision check for enemy bullet on user

if (User.m\_Position.x - User.m\_size / 2 < Bullet2.m\_Position.x && User.m\_Position.x + User.m\_size / 2 > Bullet2.m\_Position.x

&&User.m\_Position.y - User.m\_size / 2 < Bullet2.m\_Position.y && User.m\_Position.y + User.m\_size / 2 > Bullet2.m\_Position.y)

{

m\_pain->stop();

m\_pain->play();

User.m\_userHit = True;

User.m\_userHP -= 1;

}

// Game Reset in any state

(input->isKeyDown(aie::INPUT\_KEY\_R) && m\_GameOver == False) ? (shutdown(), startup()) : 0;

// If HP gets to 0 close game

(User.m\_userHP == 0 || Enemy.m\_enemyHP == 0) ? quit() : 0;

// Exit the application

(input->isKeyDown(aie::INPUT\_KEY\_ESCAPE)) ? quit() : 0;

}

void Application2D::draw() {

// wipe the screen to the background colour

clearScreen();

// set the camera position before we begin rendering

m\_2dRenderer->setCameraPos(m\_cameraX, m\_cameraY);

// begin drawing sprites

m\_2dRenderer->begin();

//if (m\_GameOver == False) // Started to add a end game screen did not finish it

// {

m\_2dRenderer->drawSprite(m\_grass, 640, 360, 0, 0, 0, 1);

//m\_2dRenderer->drawSprite(m\_grassball, 640, 360, 0, 0, m\_timer, 0);

// demonstrate spinning playable character

m\_2dRenderer->setUVRect(0, 0, 1, 1);

m\_2dRenderer->drawSprite(m\_shipTexture, User.m\_Position.x, User.m\_Position.y, User.m\_size, User.m\_size, m\_timer);

// draw a moving frowny face

m\_2dRenderer->setRenderColour(1, 0, 1, 1);

m\_2dRenderer->drawSprite(m\_shipTexture1, Enemy.m\_Position.x, Enemy.m\_Position.y, Enemy.m\_size, Enemy.m\_size, m\_timer);

// output some text, uses the last used colour

char fps[32];

m\_2dRenderer->setRenderColour(1, 1, 0, 1);

sprintf\_s(fps, 32, "FPS: %i", getFPS());

m\_2dRenderer->drawText(m\_font, fps, 0, 720 - 32);

m\_2dRenderer->drawText(m\_font, "Press Space and Right Ctrl to Shoot! / Press R to Restart!", 0, 720 - 64);

m\_2dRenderer->drawText(m\_font, "Arrow Keys to Move Teacher and W and S to Move Donray!", 0, 690 - 64);

m\_2dRenderer->drawText(m\_font, "Your HP", 0, 55);

m\_2dRenderer->drawText(m\_font, "His HP", 1170, 55);

// Draws intergers that get lowered on each hit

m\_2dRenderer->setRenderColour(1, 1, 1, 1);

char test[16];

sprintf\_s(test, 16, "%i", User.m\_userHP);

m\_2dRenderer->drawText(m\_font, test, 0, 0);

char test2[16];

sprintf\_s(test2, 16, "%i", Enemy.m\_enemyHP);

m\_2dRenderer->drawText(m\_font, test2, 1260, 0);

//// Game counter

//char counter[15];

//sprintf\_s(counter, 15, "%i", count);

//m\_2dRenderer->drawText(m\_font, counter,300,0);

// Draw HP bars that lower on each hit

m\_2dRenderer->setRenderColour(0, 1, 0, 1);

m\_2dRenderer->drawBox(0 + (User.m\_userHP \* 50) / 2, 10, User.m\_userHP \* 50, 20, 0, 0);

m\_2dRenderer->setRenderColour(1, 0, 0, 1);

m\_2dRenderer->drawBox(1160 + (m\_healthBarCompensator \* 50) / 2, 10, Enemy.m\_enemyHP \* 50, 20, 0, 0);

// Draw Bullet to screen following the X and Y of the faces

if (User.m\_Shooting == True)

{

m\_2dRenderer->setRenderColour(0, 1, 0, 1);

m\_2dRenderer->drawBox(Bullet.m\_Position.x, Bullet.m\_Position.y, 20, 5, 0, 0);

}

if (Enemy.m\_Shooting == True)

{

m\_2dRenderer->setRenderColour(0, 0, 1, 1);

m\_2dRenderer->drawBox(Bullet2.m\_Position.x, Bullet2.m\_Position.y, 20, 5, 0, 0);

}

// }

//else

//{

//}

// done drawing sprites

m\_2dRenderer->end();

}

**-Read Me –**

You can access the aieBootStrap file via this link: <https://github.com/wdonray/aieBoostrap/tree/master/project2D>

You can access the Math Library file via this link:

<https://github.com/wdonray/Math-Library>

Click the button that says “Clone or download” button listed on the top right in this page, then click “Download ZIP” this will allow you to access each file individually.

The Math-Library-master folder contains the Math Library Folder which contains my source code.

The aieBootstrap-master folder contains the project2D Folder which contains my source code.

An Assessment documentation is enclosed in the repository.

To access Assessment documentation which is on Word, click on the file named “Math Library.docx” which is inside the Math Library Repository you can click on the link above to access it and you can also download that file from there.

To access my test that I used to test my math go into the Math Library folder then open up the Math.cpp