Tutorial – Matrix Transformations

Introduction:

In this tutorial we will continue our study of Matrices by introducing transformation matrices. We will focus on trying to build an intuitive understanding of the structure of a 2D transformation matrix before moving on to code implementations of functions that build rotation, scale and translation matrices.

Exercises:

The following are transformation matrices.
For each one say whether it is translating, scaling and/or rotating, and by how much of each.

$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos(20°) & -\sin(20°) & 0 \\ \sin(20°) & \cos(20°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 * \cos(45°) & -2.5 * \sin(45°) & 6 \\ 3 * \sin(45°) & 2.5 * \cos(45°) & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

Practical:

Open up the matrix class you started in the last session. Add the following functions

- functions for creating
  - rotation
  - translation and
  - scale matrices.
- overloaded multiply operator for translating a vector

Here's the updated .h file. The new parts you need to add have been highlighted.

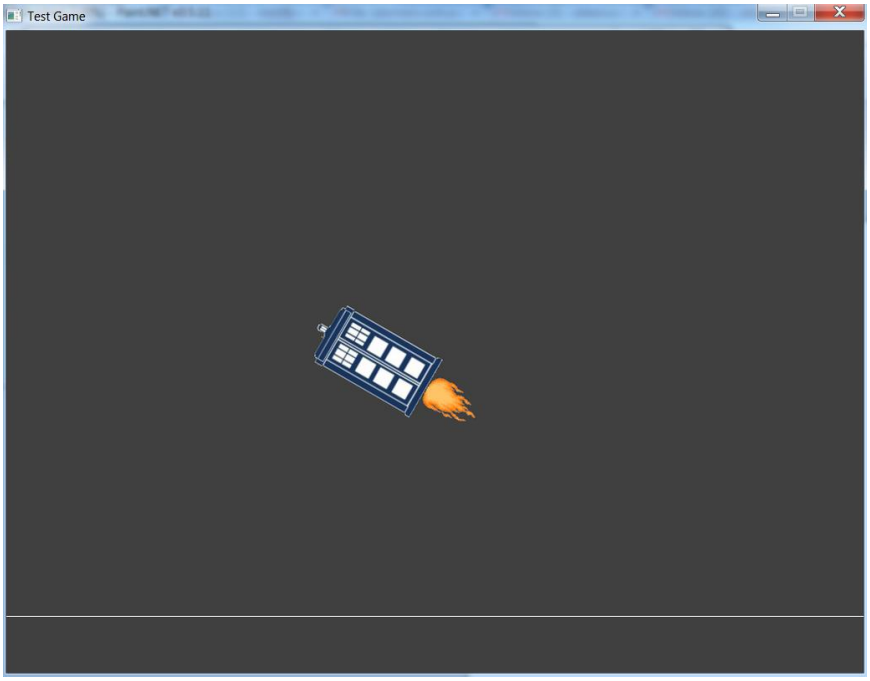
```
#ifndef _MATRIX3X3_H_
#define _MATRIX3X3_H_

class Matrix3x3
{
public:
	Matrix3x3();
	~Matrix3x3(){}

	//builds and returns a new identity matrix
	static Matrix3 CreateIdentity();
	//builds and returns a new rotation matrix
	static Matrix3x3 CreateRotation(float a_fRadians);
	//builds and returns a new scale matrix
	static Matrix3x3 CreateScale(const Vector3& a_Scale);
	//builds and returns a new translation matrix
	static Matrix3x3 CreateTranslation(const Vector3& a_Translation);


	//transposes matrix and returns *this
	Matrix3x3& Transpose();
	//builds and returns a new matrix that is the transpose of this matrix
	Matrix3x3 GetTranspose() const;

	Matrix3x3 operator +(const Matrix3x3& a_RHS) const;
	Matrix3x3 operator -(const Matrix3x3& a_RHS) const;
	Matrix3x3 operator *(const Matrix3x3& a_RHS) const;

	Vector3 operator *(const Vector3& a_RHS) const;

	Matrix3x3& operator +=(const Matrix3x3& a_RHS);
	Matrix3x3& operator -=(const Matrix3x3& a_RHS);
	Matrix3x3& operator *=(const Matrix3x3& a_RHS);

	float m_afM[3][3];
};

#endif //_MATRIX3X3_H_
```

Download and open the project provided on Portal for this tutorial.

This project contains a simple version of the classic Moon Lander arcade game. However, the code is missing the Vector3 and Matrix3 classes.

Include your Vector3 and Matrix3 classes in order to test them with the provided code. No changes to the provided code should be necessary, although you may need some minor modifications if you used a different interface for either your Matrix or Vector classes. These updates (if any) should be trivial.

The finished product should resemble the image below.



This task relates directly to your assignment! If your maths classes work here they will also work in your assignment. Use this place to get them right before you test them in your arcade game.

(Note, you cannot use the Moon Lander game presented in this tutorial for your assignment)