HONG KONG INSTITUTE OF VOCATIONAL EDUCATION
Laboratory 2: Simple 2D Game – Phase I

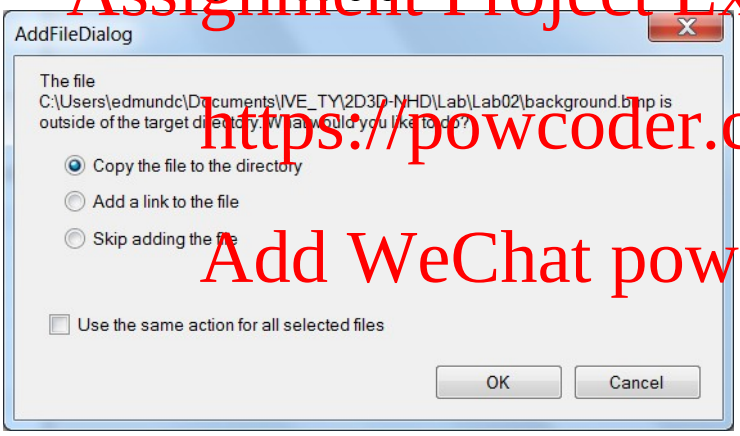
Module Intended Learning Outcome (#2):
On completion of the module, students are expected to be able to:
- develop 2D and 3D graphics programs for general gaming purposes;

Lesson Intended Learning Outcome:
On completion of this lab, students are expected to be able to:
- Create a simple program for Windows using MonoGame and Visual Studio.
- Compile, debug and execute a game program using MonoGame and Visual Studio.

In this lab, we will start creating a simple 2D game, in which the player character will try to escape from the falling asteroids. We will create the falling rocks first in this phase.

Start from the Scratch – Background Picture

1. Follow the instruction given in the last lab to create a new empty MonoGame Project.
2. Add the background image into your Content Pipeline Tool. You may want to add a sub-folder to make your contents more organized. When you add the file into the tool, you will see the following dialog popup:



   Choose "Copy the file to the directory" so that the files are copied into an appropriate location in your project directory.
3. Build and save your Content project before quitting the Content Pipeline Tool.
4. Add a class level ***Texture2D*** variable in "Game1.cs" in your MonoGame project to store your background image:

```
Texture2D   bgTexture;
```

5. Use content manager to load your image file in ***LoadContent()*** function.

```
bgTexture = Content.Load<Texture2D>("Images\\background");
```

6. Use sprite batch to draw your image on screen in ***Draw()*** function.

```
spriteBatch.Begin();
spriteBatch.Draw(bgTexture,
      GraphicsDevice.Viewport.Bounds, Color.White);
spriteBatch.End();
```

   `GraphicsDevice.Viewport.Bounds` is the rectangle representing the full rectangle of the viewable area of your application; with the top-left corner as (0, 0) of the rectangle.

Drawable Game Component

In general, game objects can be generalized into a class. In XNA (MonoGame) framework, we can create such a class by inheriting the class **GameComponent.** If your game object class contains graphics rendering and you want to make it render onto the screen itself, you may inherit your game object class from the class **DrawableGameComponent,** which is a sub-class of **GameComponent**.

The game object class we are going to use in this lab should contain the following attributes:

- A **Texture2D** object to store the sprite for the game object.
- **Vector2** objects to store the position, center and the velocity of the object
- A **float** variable to store the rotation angle of the game object.
- A **float** variable to store the rotation speed of the game object.
- A **static Random** object (Random number generator) for generating random numbers. It must be declared static so that different rock objects will use the same random number generator and not repeatedly appear in the same position.

The first game component class that we will add is a falling rock. The rock will fall from a random position at the top of the screen to the bottom at a random speed. When it hits the ground, a new rock will fall at another random position with a new speed. While the rock is falling, we add some rotation to it so that it looks like it is rolling.

TASK 1 – Create a Rock class

1. Add a new Item to your Project through the menu bar. Choose **Class** from **Visual C#** category. Name the class **Rock.cs**.
2. Add the using statements for XNA framework to access MonoGame capability:


```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

3. Change the class definition so that it inherits the base class **DrawableGameComponent** as the rocks will be drawn to the screen later.
4. A **DrawableGameComponent** works similar to the **Game** class. It will have a constructor (with a Game object as the parameter), ***Initialize()***, ***LoadContent()***, ***Update()*** and ***Draw()*** functions as in **Game** class. These functions in the **DrawableGameComponent** will be called respectively when the corresponding function in **Game** is called and the Component is added into the game. The detail steps will be discussed below.
5. First of all, we will add the implementation of the constructor to the Rock class. The constructor is just simply calling the constructor of the base class (super class) as follows:

```
public Rock(Game g) : base(g) { }
```

   The constructor will be publicly accessible. The parameter will be the **Game** object that the component belongs to; and can be accessible in this class using the variable **Game**.
6. Then, add the class-level instance variables required for your **Rock** class (see above). When adding your **Random** object, you should also initialize it right in the declaration:

```
static Random r = new Random();
```

   Apart, you also need to create a **SpriteBatch** variable to store the ***spriteBatch*** that draws 2D object in this class.

7. The next thing to add is the ***Initialize()*** function. When you type:

```
public override void Initialize()
```

Visual Studio should help you to complete the rest and your should have these things:

```
public override void Initialize() {
      base.Initialize();
}
```

Add statements before `base.Initialize()`, to set the position, velocity and rotation speed of the rock.

- The initial X-position of the rocks will be generated randomly, between 0 and width of the screen; and the Y-position of the rocks will be 0, i.e. at the top of the screen.
- As the rocks fall down straight, X-value for the velocity is 0 and Y-value is a random value between 0.5 and 2.5.
- The rotation speed will be 1/10 of the falling velocity. The value denotes the radian value of clockwise turn in each update.

All values that are not initialized in the process are by default set to 0.

To generate random values, you may use the following method of the **Random** class:

```
r.Next(); // Returns a non-negative random number
r.Next(int); // Returns a non-negative random number less than the parameter
r.Next(int, int); // Returns a random number within a specified range.
r.NextDouble(); // Returns a random number between 0.0 and 1.0.
```


8. Copy the ***LoadContent()*** function from your **Game1.cs**. ***LoadContent()*** in the **DrawableGameComponent** class works similarly as in the **Game1** class.

Inside your ***LoadContent()*** function, you should keep the initialization of the ***spriteBatch*** object. Apart, you should load the texture for the rock – remember to add the asteroid image to your **Content Pipeline Tool**. Then, load the image to the **Texture2D** variable you've declared. However, you need to use the **Content** manager from your **Game** object to load the images, as you cannot own any **Content** object in your **DrawableGameComponent**:


```
texture = Game.Content.Load<Texture2D>(...);
```

The rotation center of the game object should also be set in ***LoadContent()*** function after the texture is loaded, so that the width and height of the texture can be known. The center of your game object is at the half the texture's width and height respectively.

9. Add your ***Update()*** function by typing and choosing `public override void Update`

Inside your ***Update()*** function, you will update the position and rotation angle of your rock before the `base.Update(gameTime)` statement.

```
position.Y += velocity.Y;
rotateAngle = (rotateAngle + rotateSpeed) % MathHelper.TwoPi;
```

The rotation angle is kept in between 0 and 2π.

You should also check if the rock hits the ground also. The Y-coordinate of the ground of the background is approximately at 225. When the Y-coordinate reaches such value, re-initialize the rock so that it goes back to a random position at the top of the screen.

10. Add your ***Draw()*** function by typing and choosing `public override void Draw`

In your ***Draw()*** function, use the ***spriteBatch*** object to draw the rocks onto the screen.

```
spriteBatch.Begin();
spriteBatch.Draw(texture, position, // image and position of rock
      null, Color.White, // draw whole image, no tinting
      rotateAngle, center, // rotate angle along the center, center position
      1.0f, SpriteEffects.None, 0.5f);    // no scaling, no flipping
spriteBatch.End();
```

TASK 2 – Add an Array of Rocks to the Game

1. Declare an array of **Rock** in **Game1.cs**:

```
Rock[] rocks;
```

2. Initialize the array in the ***Initialize()*** function.

```
rocks = new Rock[3];
```

3. Create the rocks and add them to the game's component list (***Components***) in the ***Initialize()*** function.

```
for (int i = 0; i < 3; i++) {
      rocks[i] = new Rock(this);
      Components.Add(rocks[i]);
}
```

   The ***GameComponent*** must be added to the component list inside the ***Initialize()*** function so that the ***Initialize()*** and ***LoadContent()*** functions of the rock objects are called properly.

4. Once the rocks are added to the component list, no extra function call is needed. The ***Update()*** and ***Draw()*** functions of the rock objects will be called automatically when ***base.Update()*** and ***base.Draw()*** are called.