7.4H: Something Awesome

SplashKit Sprites Tutorial

In this task, I have created a game called Sky Dodger. The functionality of this game is that there are rocks which keep falling from above. There is a player which is represented by a plane. The player has 3 lives and the score keeps increasing as the user spends time without getting killed. In the background, there is a picture full of stars and it keeps moving which gives us a feel that the plane is actually moving whereas only the background is moving and the plane is as it is. The player can move the plane to its left or right depending upon the situation. There is background music which keeps playing. If there is a collision between the plane and the rock, then there is a background sound of collision and the visual represents a blast. Once there is a collision, life decreases by one. Once the lives are over, the screen shows a purple background in which the message displayed is “GAME OVER!!! YOUR SCORE IS: XX”. The user can close the window if they want else the window will automatically be closed after 9 seconds.

The unique part about this game is that I have created it using **Sprites**. A Sprite is a 2D image or animation used in games and graphical applications. Sprites are typically used to represent characters, objects and effects in a game.Instead of drawing graphics pixel by pixel, we use sprites as pre-made images that can be moved, scaled, rotated, and animated efficiently.

Below is the code regarding which I have provided the explanation as well:

Program.cs

```
using System;
using SplashKitSDK;
using static SplashKitSDK.SplashKit;

namespace _7_4HD_Something_Awesome
{
   public class Program
   {
       public static void Main()
       {
           Game game = new Game();
           game.Run();
```

```
        }
    }
}
```

The above code means that we have a namespace _7_4HD_Something_Awesome which helps organize the code and avoid conflicts with other parts of the program.

`Game game = new Game();` This statement indicates that we have created an object of the Game class. It is used to access the function of the “Game” class.

Player.cs

```
using System;
using System.Collections.Generic;
using SplashKitSDK;

public class Player
{
    public Sprite Sprite
    {
        get; private set;
    }

    public Player(string imagePath, int startX, int startY)
    {
        Bitmap playerBitmap = new Bitmap("Player", imagePath);
        Sprite = SplashKit.CreateSprite(playerBitmap);
        SplashKit.SpriteSetX(Sprite, startX);
        SplashKit.SpriteSetY(Sprite, startY);

    }

    public void Move()
    {
        if(SplashKit.KeyDown(KeyCode.LeftKey) && SplashKit.SpriteX(Sprite) > 0)
        {
            SplashKit.SpriteSetX(Sprite, SplashKit.SpriteX(Sprite) - 5);
        }

        if(SplashKit.KeyDown(KeyCode.RightKey) && SplashKit.SpriteX(Sprite) < 800 -
Sprite.Width)
        {
            SplashKit.SpriteSetX(Sprite, SplashKit.SpriteX(Sprite) + 5);
        }

    }

    public void Draw()
    {
        SplashKit.DrawSprite(Sprite);

    }
}
```

This class handles the player, its movement, its velocity, drawing, etc. in SplashKit.

Sprite: Sprite stores the player's image as a Sprite object. The *get; private set;* means that the Sprite can be read from anywhere, but can only be modified inside this class.

The constructor ensures that the player appears at a specific position when the game starts.

The imagepath is the file path of the player's image. *StartX* and *StartY* are the initial positions of the players. The system uses "*SplashKit.CreateSprite(playerBitmap);*" to create a Sprite from the loaded image.

The below statements are used to set the X and Y locations of the Sprite:

SplashKit.SpriteSetX(Sprite, startX);
SplashKit.SpriteSetY(Sprite, startY);

The *Move()* function is responsible for moving the Player within the screen when the key is pressed. Hence, when the left key is pressed, the player moves to the left by 5 pixels and when the right key is pressed, the player moves right by 5 pixels.

The Draw() function executes the statement "*SplashKit.DrawSprite(Sprite);*". It helps in rendering the player Sprite on the screen.

Obstacle.cs

```
using SplashKitSDK;

public class Obstacle
```

```
{
    public Sprite Sprite
    {
        get; private set;
    }

    public Obstacle(string imagePath, int X)
    {
        Bitmap obstacleBitmap = new Bitmap(Guid.NewGuid().ToString(), imagePath);
        Sprite = SplashKit.CreateSprite(obstacleBitmap);
        SplashKit.SpriteSetX(Sprite, X);
        SplashKit.SpriteSetY(Sprite, -Sprite.Height);
    }

    public void Move()
    {
        SplashKit.SpriteSetY(Sprite, SplashKit.SpriteY(Sprite) + 5);
    }

    public bool IsOffScreen()
    {
        return SplashKit.SpriteY(Sprite) > 600;
    }

    public void Draw()
    {
        SplashKit.DrawSprite(Sprite);
    }
}
```

This class defines the falling obstacles in the Sky Dodger game that is created. It manages the functions like Creating Obstacles, Moving them Downward, Checking if they go off-screen and Drawing them on the Screen.

In this code, the Sprite object stores the obstacles image. It can be read from anywhere but can only be modified inside the class.

The system loads the obstacle image using "*Bitmap obstacleBitmap = new Bitmap(Guid.NewGuid().ToString(), imagePath);*". The imagePath is the path to the obstacle image file. **Guid.NewGuid().ToString()** generates a **unique name** for each obstacle bitmap so that multiple obstacles can exist without conflicts because every obstacle needs a unique name.

The statement “*SplashKit.SpriteSetX(Sprite, X);*” means that the system places the obstacle at the given X coordinate.

The statement “*SplashKit.SpriteSetY(Sprite, -Sprite.Height);*” sets the obstacle’s Y position just above the screen, so it falls down naturally.

This ensures that the obstacles randomly appear across the screen and fall from the top.

The below code creates a falling effect as it moves the obstacles downward by 5 pixels per frame. It should be called every frame inside the game loop.

```
public void Move()
{
   SplashKit.SpriteSetY(Sprite, SplashKit.SpriteY(Sprite) + 5);
}
```

The below code checks if the obstacle has moved past the bottom of the screen (assuming screen height = 600 pixels). If true, then the obstacle should be removed from the game.

```
public bool IsOffScreen()
{
   return SplashKit.SpriteY(Sprite) > 600;
}
```

The below code renders the obstacle sprite on the screen. This should be called every frame inside the game loop.

```
public void Draw()
{
   SplashKit.DrawSprite(Sprite);
}
```

Background.cs

```
using SplashKitSDK;


public class Background
{
    private Sprite _sprite;
    private float _yPosition;

    public Background(string imagePath)
    {
        Bitmap background = new Bitmap("Background", imagePath);
        _sprite = SplashKit.CreateSprite(background);
        _yPosition = 0;
    }

    public void Update()
    {
        _yPosition += 2;
        if(_yPosition >= 600)
        {
            _yPosition = 0;
        }
    }

    public void Draw()
    {
        SplashKit.DrawSprite(_sprite, 0, _yPosition);
        SplashKit.DrawSprite(_sprite, 0, _yPosition - 600);
    }
}
```

This class is responsible for the scrolling background in our game. It creates a looping effect to simulate motion as if the player's plane is flying forward.

Here, we have declared 2 variables *_sprite* and *_yPosition*.
The variable *_sprite* holds the background image and the variable *_yPosition* tracks the vertical position of the background.

The explanation of the constructor is below :

The statement “*Bitmap background = new Bitmap("Background", imagePath);*” loads the background image from the imagePath.

The statement “*_sprite = SplashKit.CreateSprite(background)*;” creates a sprite from the loaded image.

The variable *_yPosition* is assigned 0 so that the background starts from the top.

The below code moves the background by 2 pixels per frame. The system resets the value of _yPosition to 0 once the value reaches 600 so as to create an infinite effect.

```
public void Update()
{
   _yPosition += 2;
   if(_yPosition >= 600)
   {
      _yPosition = 0;
   }
}
```

The *Draw()* function draws the first background image at *(0, _yPosition)* and a second copy of it just above the first one at *(0, _yPosition - 600)* to ensure smooth transition when scrolling. It basically creates a continuous scrolling effect where the top image seamlessly replaces the bottom one.

Sound.cs

```
using SplashKitSDK;

public class Sound
{
   private SoundEffect sound;
   private Music backgroundMusic;

   public Sound()
   {
       sound = new SoundEffect("Explosive Sound", "explosion.wav");
       backgroundMusic = new Music("Background Music", "background_music.mp3");
       PlayBackgroundMusic();
   }

   public void ExplosionNoise()
   {
       sound.Play();
   }

   public void PlayBackgroundMusic()
```

```
    {
        if(!SplashKit.MusicPlaying())
        {
            SplashKit.PlayMusic("background_music.mp3", -1);
        }
    }

    public void StopBackgroundMusic()
    {
        SplashKit.StopMusic();
    }
}
```

This class manages sound effects and background music in the game. It handles explosion sound effects, looping background music and starting and stopping background music.

The variable sound stores the explosion sound effect. The variable backgroundMusic stores the background music file. It keeps the audio data private and prevents accidental modifications outside the class.

Below is the explanation of the constructor:

In the statement : “*sound = new SoundEffect("Explosive Sound", "explosion.wav");*”, Explosive Sound is the internal name used to reference the sound and explosion.wav is the file name.

The statement “*backgroundMusic = new Music("Background Music", "background_music.mp3");*” represents the internal name of the music as Background Music and the filename background_music.mp3.

The function “*PlayBackgroundMusic();*” ensures the music starts playing as soon as the window game opens.

The below code plays the explosion sound effect when called.

```
public void ExplosionNoise()
{
   sound.Play();
}
```

The below code prevents restarting the music if it is already playing.

```
public void PlayBackgroundMusic()
{
    if (!SplashKit.MusicPlaying())
    {
        SplashKit.PlayMusic("background_music.mp3", -1);
    }
}
```

The below code stops the background music when called.

```
public void StopBackgroundMusic()
{
    SplashKit.StopMusic();
}
```

Explosion.cs

```
using SplashKitSDK;

public class Explosion
{
   private Sprite _sprite;
   public int _frameCount;
   public bool _active;

   public Explosion(string imagepath, float x, float y)
   {
       Bitmap explosionBitmap = new Bitmap(Guid.NewGuid().ToString(), imagepath);
       _sprite = SplashKit.CreateSprite(explosionBitmap);
       SplashKit.SpriteSetX(_sprite, x);
       SplashKit.SpriteSetY(_sprite, y);
       _frameCount = 30;
       _active = true;
   }

   public void Update()
   {
       if(_frameCount > 0)
       {
            _frameCount--;
        }
        else
        {
            _active = false;
        }
    }

    public bool isActive()
    {
        return _active;
    }

    public void Draw()
    {
        if(_active)
        {
            SplashKit.DrawSprite(_sprite);
        }
    }
}
```

This class creates an explosion effect in the game when the rock hits the plane.

The variable _*sprite* stores the explosion image.
The variable _*frameCount* tracks how long the explosion should be visible (30 frames).
The variable _*active* is true when the explosion is visible and false when the explosion disappears.

Below is the explanation of the constructor:

In the statement "*Bitmap explosionBitmap = new Bitmap(Guid.NewGuid().ToString(), imagepath);*", "**Guid.NewGuid().ToString()**" generates a unique name for each explosion image. It ensures that multiple explosions can exist at the same time.
The statement "*_sprite = SplashKit.CreateSprite(explosionBitmap);*" creates a sprite from the explosion image.
The statement "*SplashKit.SpriteSetX(_sprite, x);*" and "*SplashKit.SpriteSetY(_sprite, y);*" sets the position of the explosion to x and y.

The code statement "*_frameCount = 30;*" means that the explosion will last 30 Frames i.e. ~ 0.5 seconds, equivalent to 60 Frames Per Second.

The below code ensures that each frame decreases by 1. Once it reaches 0, _active is set to false. This removes the explosion effect after 30 frames.

```
public void Update()
{
   if(_frameCount > 0)
   {
      _frameCount--;
   }
   else
   {
      _active = false;
   }
}
```

The function *isActive()* returns true if the explosion is still visible and returns false if the explosion has disappeared.

The function *Draw()* only draws the explosion when the *_active* is true. Once it is false, the function stops rendering.

Game.cs

```
using SplashKitSDK;

public class Game
{
   private Window _GameWindow;
   private Player _Player;
   private List<Obstacle> obstacles;
   private int score;
   private int lives;
   private Random _random;
   private SplashKitSDK.Timer _timer;
   private List<Explosion> explosions;
   private Sound soundEffect;
   private Background background;

   public Game()
   {
        _GameWindow = new Window("Sky Dodger", 800, 600);
        _Player = new Player("Plane.png", 400, 500);
        obstacles = new List<Obstacle>();
        score = 0;
        lives = 3;
        _random = new Random();
        _timer = new SplashKitSDK.Timer("myTimer");
        _timer.Start();
        explosions = new List<Explosion>();
        soundEffect = new Sound();
        background = new Background("background.jpg");
    }

    public void Run()
    {
        while(!_GameWindow.CloseRequested)
        {
            SplashKit.ProcessEvents();
            SplashKit.ClearScreen(Color.Navy);

            soundEffect.PlayBackgroundMusic();

            background.Update();
            background.Draw();

            if(SplashKit.Rnd() < 0.02)
            {
                obstacles.Add(new Obstacle("Rock.png", _random.Next(0,
_GameWindow.Width - 50)));
            }

            _Player.Move();
            _Player.Draw();

            obstacles.RemoveAll(obstacle => {
                if(SplashKit.SpriteCollision(_Player.Sprite, obstacle.Sprite) &&
_timer.Ticks > 1000)
                {
                    explosions.Add(new Explosion("Explosion.png",
SplashKit.SpriteX(_Player.Sprite), SplashKit.SpriteY(_Player.Sprite)));
                    soundEffect.ExplosionNoise();
                    lives--;
                    _timer.Reset();
                    return true;
                }
                return obstacle.IsOffScreen();
            });

            explosions.RemoveAll(explosion => !explosion.isActive());
            foreach(var explosion in explosions)
            {
                explosion.Update();
                explosion.Draw();
            }

            foreach(var obstacle in obstacles)
            {
                obstacle.Move();
                obstacle.Draw();
            }

            if(lives <= 0)
            {
                GameOver();
                return;
            }

            score++;

            SplashKit.DrawText($"Score: {score}", Color.Orange, "Arial", 20, 10, 10);
            SplashKit.DrawText($"Lives: {lives}", Color.Orange, "Arial", 20, 10, 40);

            _GameWindow.Refresh(60);
        }

        soundEffect.StopBackgroundMusic();
    }

    private void GameOver()
    {
        _GameWindow.Clear(Color.BlueViolet);
        SplashKit.DrawText($"GAME OVER! FINAL SCORE: {score}", Color.White, "Arial",
60, 275, 275);
        _GameWindow.Refresh();
        SplashKit.Delay(9000);
        _GameWindow.Close();
    }
}
```

This class manages the main logic of the game. It handles:

- Player Movement
- Obstacle spawning & collision detection
- Explosion effects
- Score and lives tracking
- Game over sequence
- Background music and sound

The usage and description of the variables are below:

- _GameWindow: Creates the game window (800×600).
- _Player: The player-controlled airplane.
- obstacles: Stores falling obstacles (rocks).
- score: Keeps track of the player's score.
- lives: Tracks how many lives the player has (starts with 3).
- _random: Generates random numbers (used for spawning obstacles).
- _timer: Ensures a delay after a collision to prevent instant game over.
- explosions: Stores temporary explosion effects.
- soundEffect: Manages background music & sound effects.
- background: Manages scrolling background animation.

In the constructor, below are the actions performed in sequence as per the code:

- Creates a game window → "Sky Dodger" with size 800×600.
- Creates the player at (400,500) with "Plane.png".
- Initializes obstacle list (empty at start).
- Sets score to 0 and lives to 3.
- Creates a random number generator (used for spawning obstacles).
- Starts a game timer to add collision delay.
- Creates an empty list of explosions.
- Initializes background music & sound effects.
- Loads the scrolling background image.

The constructor prepares the game environment before it starts.

The below code represents that the system will keep the game running until the window is closed. It processes user input keys, mouse, etc.. It clears the screen to refresh for the next frame.

```
public void Run()
{
   while (!_GameWindow.CloseRequested)
   {
      SplashKit.ProcessEvents();
      SplashKit.ClearScreen(Color.Navy);
```

The statement “*soundEffect.PlayBackgroundMusic();*” ensures that the background music is playing. It keeps the game engaging with continuous music.

The statement “*background.Update();*” and “*background.Draw();*” moves and redraws the scrolling background. It gives an illusion of flying movement. It basically adds visuals to the game.

The function of the statements “*_Player.Move();*” and “*_Player.Draw();*” is to move the player and draw the player sprite on the screen.

The code below checks if the player collides with any obstacle. If a collision is detected and at least 1 second has passed, then it creates an explosion effect. It plays an explosive sound and reduces life by 1. It resets the timer and removes the collided obstacle. It also removes obstacles that move off the screen so as to prevent instant death from multiple hits.

```
obstacles.RemoveAll(obstacle => {
   if(SplashKit.SpriteCollision(_Player.Sprite, obstacle.Sprite) && _timer.Ticks > 1000)
   {
      explosions.Add(new Explosion("Explosion.png", SplashKit.SpriteX(_Player.Sprite),
SplashKit.SpriteY(_Player.Sprite)));
      soundEffect.ExplosionNoise();
      lives--;
      _timer.Reset();
      return true;
   }
   return obstacle.IsOffScreen();
});
```

The below code removes inactive explosions from the memory and ensures that the explosion disappears from the screen. It updates and draws active explosions.

```
explosions.RemoveAll(explosion => !explosion.isActive());
foreach(var explosion in explosions)
{
   explosion.Update();
   explosion.Draw();
}
```

The below code moves the obstacles downwards and draws them on the screen.

```
foreach(var obstacle in obstacles)
{
   obstacle.Move();
   obstacle.Draw();
}
```

The statement "*soundEffect.StopBackgroundMusic();*" stops the background music when the user exits the game.

The below code displays the "GAME OVER" message once the game is over. It waits for 9 seconds and then closes the window. It gives the player time to see their final score.

Please find below the URL of the game in action:

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=098b7c7b-45ed-4e2e-a621-b27700907efb