Game Framework Project

Submitted To:

Dr. Awais Hassan

Submitted By:

Muhammad Safee ullah, 2020-CS-13

Department of Computer Sicence

University of Engineering and Technonlogy Lahore

Framework for Making Platformer Games

Problem Statement:

We need to add falling functionality for different objects (picture box) such as enemies and other players.

Previous Solution:

Pseudo code:

This program will allow user to add enemies and other objects such as player to the game.

```
Make an enemy class that contains all the enemy logic and a function to update it's position.
Make a player class that has all the player logic and a function to update its position.
Make a list in Form1 that allows user to add enemies.
On every timer tick:
    Loop through the enemies and run their update methods.
    Run the update method for the player.
```

Demerits:

The business logic is not entirely separate from Form1.

There is no separate Game class.

If we want to add physics to our classes, we have to write it for every class.

Code:

Form1:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace dragonSlayer
{
    public partial class MainGame : Form
    {
        int dragonCounter = 0;
        public List<PathFinder> dragons;
        public List<ProgressBar> dragonHealths;
        public int EnemyHealth = 2;
        public MainGame()
        {
            InitializeComponent();
            dragons = new List<PathFinder>();
            dragons.Add(dragon1);
            dragons.Add(dragon2);
            dragons.Add(dragon3);
            dragons.Add(dragon4);
            dragonHealths = new List<ProgressBar>();
            dragonHealths.Add(dragon1Health);
            dragonHealths.Add(dragon2Health);
            dragonHealths.Add(dragon3Health);
            dragonHealths.Add(dragon4Health);
        }


        private void MainGame_KeyDown(object sender, KeyEventArgs e)
            player.setStatesTrue(e.KeyCode);
        private void MainGame_KeyUp(object sender, KeyEventArgs e)
            player.setStatesFalse(e.KeyCode);

        private void gameLoopTimer_Tick(object sender, EventArgs e)
        {
            this.score.Text = $"Score: {player.score}";
            playerHealth.Value = player.health;
            enemyHeathProgressbar.Value = EnemyHealth;
            player.update();
            foreach (PathFinder dragon in dragons)
            {
                dragonHealths[dragons.IndexOf(dragon)].Value = dragon.Health;
                if (dragon.Enable) dragon.followPlayer();
            }
            if (EnemyHealth <= 0)
            {

                end("You Win!", player.score);
            }
        }
        public void end(string message, int score) {
            Wizard.Hide();
            System.Threading.Thread.Sleep(10);
            Hide();
            End endWindow = new End(message, score);
            endWindow.Show();
            this.Close();
        }

        private void DragonSpawn_Tick(object sender, EventArgs e)
        {
            if(dragonCounter < dragons.Count)
            {
                dragons[dragonCounter++].Enable = true;
                Wizard.Image = dragonSlayer.Properties.Resources.WizardSummon;
            }
        }
    }

}
```

Current Solution:

Design Decision:

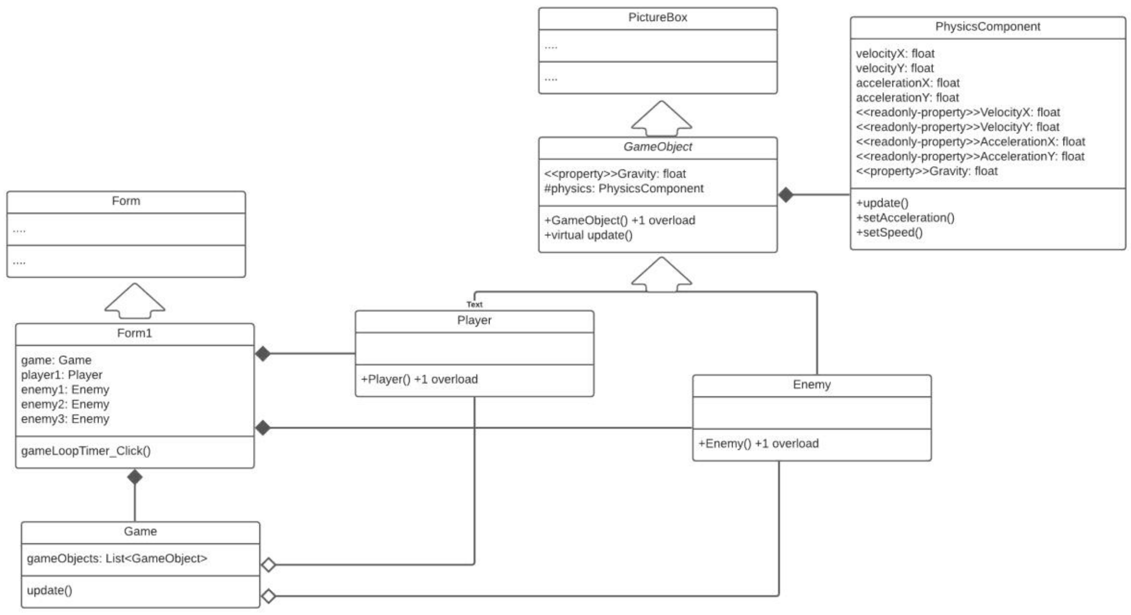
What we have to do:

1. Make a Game class separate from Form1
2. Make separate class for player and enemies
3. Adda method in our Game class to update positions of all objects.

Optionals:

1. Make an abstract class for enemy that defines the common behavior of enemies and can be inherited by the user of our framework.
2. Make a separate physics class instead of repeating the code for every class in our game.

UML Diagram:



Merits/Demerits:

1. This approach saves us from a lot of repetitive code
2. The physics behavior can be updated for every element
3. We can add complex physics to out objects without having to add all the code
4. The code is very readable.
5. The code in Player and Enemy class looks the same for now but it'll be different in future when the behavior of both classes diverges

Code:

Form1:

```
public partial class Form1 : Form
    {
        Game game;
        public Form1()
        {
            InitializeComponent();
            game = new Game();
            game.addGameObject(player1);
            game.addGameObject(enemy1);
            game.addGameObject(enemy2);
            game.addGameObject(enemy3);
        }

        private void gameLoopTimer_Tick(object sender, EventArgs e)
        {
            game.update();
        }
    }
```

Game:

```
class Game
    {
        List<GameObject> gameObjects = new List<GameObject>();
        public void addGameObject(GameObject gameObject) => gameObjects.Add(gameObject);
        public void update()
        {
            foreach (GameObject gameObject in gameObjects)
                gameObject.update();
        }
    }
```

PhysicsComponent:

```
class PhysicsComponent
    {
        float velocityX, velocityY;
        float accelerationX, accelerationY;
        float gravity;
        Control objectToAttach;

        public float VelocityX { get => velocityX;}
        public float VelocityY { get => velocityY;}
        public float AccelerationX { get => accelerationX;}
        public float AccelerationY { get => accelerationY;}
        public float Gravity { get => gravity; set => gravity = value; }

        public PhysicsComponent(Control objectToAttach, float gravity = 0)
        {
            this.objectToAttach = objectToAttach;
            this.gravity = gravity;
        }

        public void setSpeed(float velocityX, float velocityY)
        {
            this.velocityX = velocityX;
            this.velocityY = velocityY;
        }
        public void setAcceleration(float accelerationX, float accelerationY)
        {
            this.accelerationX = accelerationX;
            this.accelerationY = accelerationY;
        }
        public void update()
        {
            objectToAttach.Top += (int)velocityY;
            objectToAttach.Left += (int)velocityX;
            velocityX += accelerationX;
            velocityY += accelerationY + gravity;
        }
    }
```

GameObject:

```
abstract class GameObject : PictureBox
    {
        protected PhysicsComponent physics;
        //For adjusting gravity from the properties panel
        public float Gravity { get => physics.Gravity; set => physics.Gravity = value; }
        public GameObject()
        {
            //for creating object from toolbox
            physics = new PhysicsComponent(this);
        }
        public GameObject(Image objectImage, float objectGravity)
        {
            //for creating object programatically
            this.Image = objectImage;
            physics = new PhysicsComponent(this, objectGravity);
        }
        public virtual void update()
        {
            physics.update();
            //Refresh();
        }
    }
```

Enemy:

```
class Enemy :GameObject
    {
        public Enemy() : base() { }
        public Enemy(Image enemyImage, float enemyGravity) : base(enemyImage,
enemyGravity) { }
    }
```

Player:

```
class Player: GameObject
    {
        public Player() : base() { }
        public Player(Image playerImage, float playerGravity) : base(playerImage,
playerGravity) { }

    }
```