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

Table of Contents

Problem Statement:

As we are making our framework more easy to use for a developer, we need to use composition instead of inheritance, so we can vary the movement at runtime.We also need to add movement with keys for player.

Previous Solution:

This code defines the movement of the player and enemies in their own classes and we cannot change them outside of the object. This code uses inheritance and creates player and enemies as inherited from pictureBoxes.

Demerits:

The demerits of this approach is that you have to add a new enemy type if you ever want to change the behavior and the mevements are fixed an cannot be modified at runtime and are fixed for every object and we have to write behaviors separately for every type of enemy.

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;
        protected Movement movement = new Movement();
        //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 void setMovement(Movement movement)
        {
            this.movement = movement;
        }
        public virtual void update()
        {
            movement.update(physics);
            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) { }

    }
```

Movement:

```
class Movement
    {
        protected float movementSpeed = 10;
        public void setMovementSpeed(float movementSpeed)
        {
            this.movementSpeed = movementSpeed;
        }
        public virtual void update(PhysicsComponent physics)
        {
            //base behaviour to move rightwards
            physics.setSpeed(movementSpeed, 0);
        }
    }
```

MovementUpDown:

```
class MovementUpAndDown : Movement
    {
        public override void update(PhysicsComponent physics)
        {
            if (physics.VelocityY > 20) physics.Gravity = -1;
            else if (physics.VelocityY < -20) physics.Gravity = 1;

        }
    }
```

MovementAtAngle:

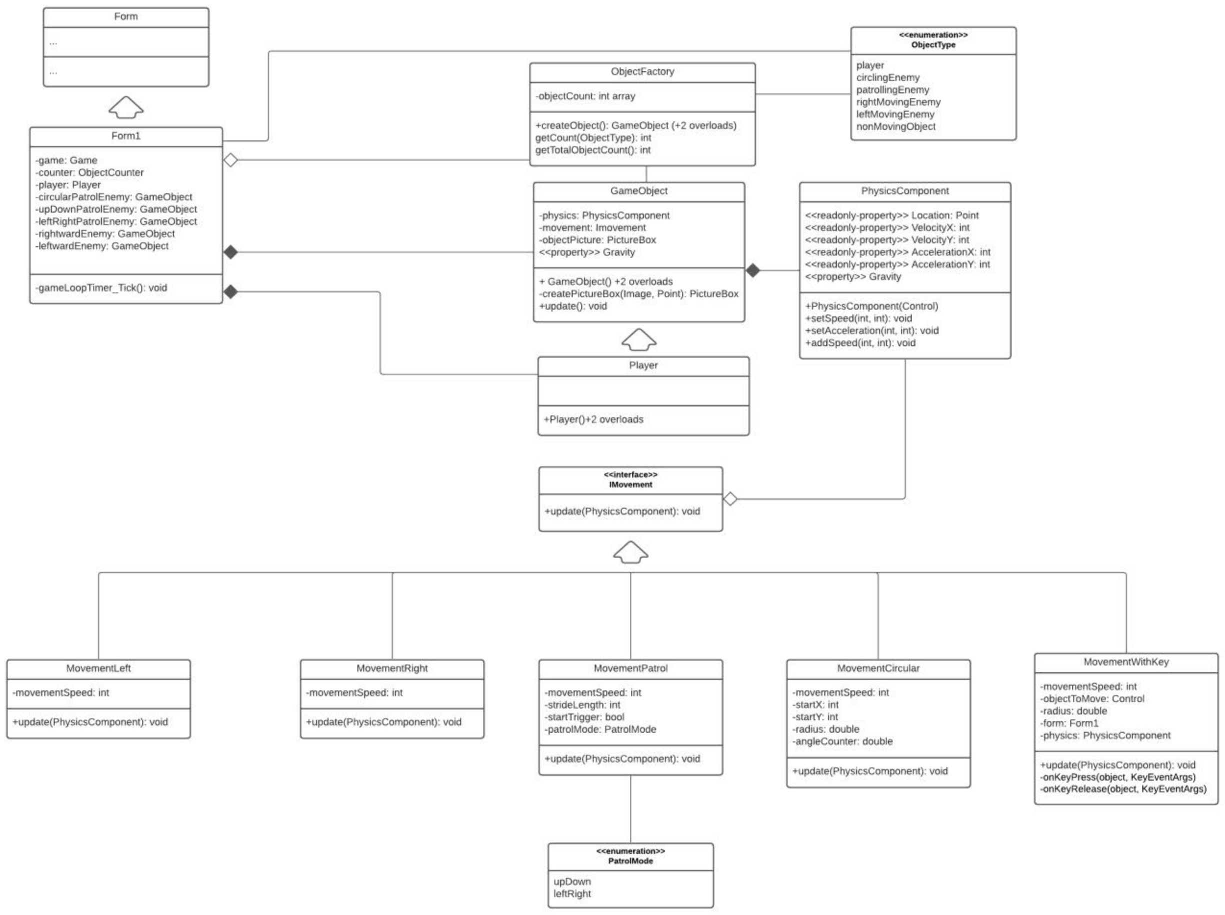
```
class MovementAtAngle :Movement
    {
        bool startFlag = true;
        public override void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            if (startFlag)
            {
                physics.setAcceleration(1, 1);
                startFlag = false;
            }
            if (physics.VelocityY > 20) physics.setAcceleration(-1, -1);
            else if (physics.VelocityY < -20) physics.setAcceleration(1, 1);

        }
    }
```

Design Decision:

Here we have some choices to make:

1. Make a virtual function for updating movement in GameObject and write our movement code in there.
   But the demerits of this approach is that we cannot reuse our movement code and have to write it separately for every class.
2. Second solution makes much more sense it is to make separate classes for movement and pass the object of these classes to our gameobjects for handling the behavior.
   In this scenario we can just make change to our movement classes without touching the code I the gameObject classes and it'll change everywhere automatically.
3. We can compose picturebox object inside of the gameobject class instead of inheriting from the picturebox.
4. We can use factory design pattern for creating objects. The benefit is the decoupling of object creation and object classes from our main code. The downside is the that the code becomes a bit lengthier.

UML Diagram:


Form
...
...
Form1
-game: Game
-counter: ObjectCounter
-player: Player
-circularPatrolEnemy: GameObject
-upDownPatrolEnemy: GameObject
-leftRightPatrolEnemy: GameObject
-rightwardEnemy: GameObject
-leftwardEnemy: GameObject
-gameLoopTimer_Tick(): void
ObjectFactory
-objectCount: int array
+createObject(): GameObject (+2 overloads)
getCount(ObjectType): int
getTotalObjectCount(): int
<<enumeration>>
ObjectType
player
circlingEnemy
patrollingEnemy
rightMovingEnemy
leftMovingEnemy
nonMovingObject
GameObject
-physics: PhysicsComponent
-movement: Imovement
-objectPicture: PictureBox
<<property>> Gravity
+ GameObject() +2 overloads
-createPictureBox(Image, Point): PictureBox
+update(): void
PhysicsComponent
<<readonly-property>> Location: Point
<<readonly-property>> VelocityX: int
<<readonly-property>> VelocityY: int
<<readonly-property>> AccelerationX: int
<<readonly-property>> AccelerationY: int
<<property>> Gravity
+PhysicsComponent(Control)
+setSpeed(int, int): void
+setAcceleration(int, int): void
+addSpeed(int, int): void
Player
+Player()+2 overloads
<<interface>>
IMovement
+update(PhysicsComponent): void
MovementLeft
-movementSpeed: int
+update(PhysicsComponent): void
MovementRight
-movementSpeed: int
+update(PhysicsComponent): void
MovementPatrol
-movementSpeed: int
-strideLength: int
-startTrigger: bool
-patrolMode: PatrolMode
+update(PhysicsComponent): void
MovementCircular
-movementSpeed: int
-startX: int
-startY: int
-radius: double
-angleCounter: double
+update(PhysicsComponent): void
MovementWithKey
-movementSpeed: int
-objectToMove: Control
-radius: double
-form: Form1
-physics: PhysicsComponent
+update(PhysicsComponent): void
-onKeyPress(object, KeyEventArgs)
-onKeyRelease(object, KeyEventArgs)
<<enumeration>>
PatrolMode
upDown
leftRight

Code:

Form1:

```
public partial class Form1 : Form
    {
        Game game;
        ObjectCounter counter = ObjectCounter.Instance();
        public Form1()
        {
            InitializeComponent();
            game = Game.Instance();
            GameObject player = new GameObject(playerPictureBox, new
MovementWithKey(playerPictureBox, 15));
            GameObject circularPatrolEnemy = new GameObject(CircularPictureBox, new
MovementCircular(CircularPictureBox, 100));
            GameObject upDownPatrolEnemy = new GameObject(UDPatrolPictureBox, new
MovementPatrol(500, 5, PatrolMode.upDown));
            GameObject leftRightPatrolEnemy = new GameObject(LRPatrolPictureBox, new
MovementPatrol(500, 5, PatrolMode.leftRight));
            GameObject rightwardEnemy = new GameObject(rightwardPictureBox, new
MovementRight(5));
            GameObject leftwardEnemy = new GameObject(leftwardPictureBox, new
MovementLeft(5));
            game.addGameObject(player);
            game.addGameObject(circularPatrolEnemy);
            game.addGameObject(upDownPatrolEnemy);
            game.addGameObject(upDownPatrolEnemy);
            game.addGameObject(leftRightPatrolEnemy);
            game.addGameObject(rightwardEnemy);
            game.addGameObject(leftwardEnemy);
        }

        private void gameLoopTimer_Tick(object sender, EventArgs e)
        {
            game.update();
            objectCountLabel.Text = $"Objects: {counter.ObjectCount}";
        }
    }
```

GameObject:

```
class GameObject
    {
        protected PhysicsComponent physics;
        protected IMovement objectMovement;
        public float Gravity { get => physics.Gravity; set => physics.Gravity = value; }
        public GameObject(Control objectPicture, IMovement objectMovement, float 
objectGravity = 1)
        {
            //for creating object from a component
            physics = new PhysicsComponent(objectPicture, objectGravity);
            this.objectMovement = objectMovement;
        }
        public GameObject(Image objectImage, Point objectPosition, IMovement 
objectMovement, float objectGravity = 1)
        {
            //for creating object from a an Image
            PictureBox objectPB = createPictureBox(objectImage, objectPosition);
            objectPB.SizeMode = PictureBoxSizeMode.AutoSize;
            physics = new PhysicsComponent(objectPB, objectGravity);
        }
        public GameObject(Image objectImage, Point objectPosition, Size objectSize, 
IMovement objectMovement, float objectGravity = 1)
        {
            //for creating object of custom size from a an Image
            PictureBox objectPB = createPictureBox(objectImage, objectPosition);
            objectPB.Size = objectSize;
            physics = new PhysicsComponent(objectPB, objectGravity);
        }
        PictureBox createPictureBox(Image objectImage, Point objectPosition)
        {
            //Utility function
            PictureBox objectPB = new PictureBox();
            objectPB.Image = objectImage;
            objectPB.Location = objectPosition;
            objectPB.BackColor = Color.Transparent;
            return objectPB;
        }
        public virtual void update()
        {
            objectMovement.update(physics);
            physics.update();
            //Refresh();
        }
    }
```

Player:

```
class Player: GameObject
    {
        public Player(Control objectPicture, IMovement objectMovement, float
objectGravity = 1):base(objectPicture, objectMovement, objectGravity) { }
        public Player(Image objectImage, Point objectPosition, IMovement objectMovement,
float objectGravity = 1) : base(objectImage, objectPosition, objectMovement,
objectGravity) { }
        public Player(Image objectImage, Point objectPosition, Size objectSize, float
objectGravity, IMovement objectMovement) : base(objectImage, objectPosition, objectSize,
objectMovement, objectGravity) { }
        public override void update()
        {
            //Custom player code here
            base.update();
        }
    }
```

IMovement:

```
interface IMovement
    {
        void update(PhysicsComponent physics);
    }
```

MovementPatrol:

```
class MovementPatrol : IMovement
    {
        int strideLength, movementSpeed;
        PatrolMode patrolMode;
        bool startFlag;
        Point startPoint;
        public MovementPatrol(int strideLength, int movementSpeed, PatrolMode patrolMode)
        {
            this.strideLength = strideLength;
            this.movementSpeed = movementSpeed;
            this.patrolMode = patrolMode;
        }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            if(patrolMode == PatrolMode.upDown)
            {
                if (!startFlag)
                {
                    physics.setSpeed(0, movementSpeed);
                    startPoint = new Point(physics.Location.X, physics.Location.Y);
                    startFlag = true;
                }
                if (physics.Location.Y < startPoint.Y + 20) physics.setSpeed(0,
movementSpeed);
                if (physics.Location.Y > strideLength) physics.setSpeed(0, -
movementSpeed);
            }
            else if (patrolMode == PatrolMode.leftRight)
            {
                if (!startFlag)
                {
                    physics.setSpeed(movementSpeed, 0);
                    startPoint = new Point(physics.Location.X, physics.Location.Y);
                    startFlag = true;
                }
                if (physics.Location.X < startPoint.X + 20)
physics.setSpeed(movementSpeed, 0);
                if (physics.Location.X > strideLength) physics.setSpeed(-movementSpeed,
0);
            }

        }
    }
```

MovementWithKey:

```
class MovementWithKey : IMovement
    {
        Control gameObject;
        int movementSpeed;
        Form form;
        PhysicsComponent physics;
        public MovementWithKey(Control gameObject, int movementSpeed)
        {
            this.gameObject = gameObject;
            this.movementSpeed = movementSpeed;
            form = gameObject.FindForm();
        }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            this.physics = physics;
            form.KeyDown += new KeyEventHandler(keyDownHandler);
            form.KeyUp += new KeyEventHandler(keyUpHandler);

        }
        private void keyDownHandler(object sender, KeyEventArgs e)
        {
            if(physics.VelocityX + physics.VelocityY < movementSpeed)
            {
                if (e.KeyCode == Keys.Up) physics.setSpeed(0, -movementSpeed);
                if (e.KeyCode == Keys.Down) physics.setSpeed(0, movementSpeed);
                if (e.KeyCode == Keys.Left) physics.setSpeed(-movementSpeed, 0);
                if (e.KeyCode == Keys.Right) physics.setSpeed(movementSpeed, 0);
            }
        }
        private void keyUpHandler(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Up) physics.setSpeed(physics.VelocityX, 0);
            if (e.KeyCode == Keys.Down) physics.setSpeed(physics.VelocityX, 0);
            if (e.KeyCode == Keys.Left) physics.setSpeed(0, physics.VelocityY);
            if (e.KeyCode == Keys.Right) physics.setSpeed(0, physics.VelocityY);
        }
    }
```

MovementRight:

```
class MovementRight: IMovement
    {
        int movementSpeed;
        public MovementRight(int movementSpeed)
        {
            this.movementSpeed = movementSpeed;
        }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            physics.setSpeed(movementSpeed, 0);
        }
    }
```

MovementLeft:

```
class MovementLeft : IMovement
    {
        int movementSpeed;
        public MovementLeft(int movementSpeed)
        {
            this.movementSpeed = movementSpeed;
        }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            physics.setSpeed(-movementSpeed, 0);
        }
    }
```

MovementCircular:

```
class MovementCircular :IMovement
    {
        Control objectToMove;
        int startX, startY;
        double angleCounter;
        double radius;
        public MovementCircular(Control objectToMove, double radius)
        {
            this.objectToMove = objectToMove;
            startY = objectToMove.Top;
            startX = objectToMove.Left;
            this.radius = radius;
        }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            angleCounter += 0.05;

            objectToMove.Top = (int)(startY + radius * Math.Sin(angleCounter));
                objectToMove.Left = (int)(startX + radius * Math.Cos(angleCounter));

        }
    }
```

ObjectFactory:

```
class ObjectFactory
    {
        int[] objectCount = new int[20];
        private static ObjectFactory counterInstance;
        private ObjectFactory() { }
        public static ObjectFactory Instance()
        {
            if (counterInstance == null)
                counterInstance = new ObjectFactory();
            return counterInstance;
        }
        public GameObject createObject(Control objectPicture, IMovement objectMovement, 
ObjectType objectType, float objectGravity = 1)
        {
            ++objectCount[(int)objectType];
            return new GameObject(objectPicture, objectMovement, objectGravity);
        }
        public GameObject createObject(Image objectImage, Point objectPosition, IMovement 
objectMovement, ObjectType objectType, float objectGravity = 1)
        {
            ++objectCount[(int)objectType];
            return new GameObject(objectImage, objectPosition, objectMovement, 
objectGravity);
        }
        public GameObject createObject(Image objectImage, Point objectPosition, Size 
objectSize, IMovement objectMovement, ObjectType objectType, float objectGravity = 1)
        {
            ++objectCount[(int)objectType];
            return new GameObject(objectImage, objectPosition, objectSize, 
objectMovement, objectGravity);
        }
        public int getCount(ObjectType objectType) => objectCount[(int)objectType];
        public int getTotalObjectsCount()
        {
            int count = 0;
            foreach (int objCount in objectCount) count += objCount;
            return count;
        }
    }
```

ObjectType:

```
    enum ObjectType
    {
        player,
        circlingEnemy,
        patrollingEnemy,
        rightMovingEnemy,
        leftMovingEnemy,
        nonMovingObject
    }
```

PatrolMode:

```
enum PatrolMode
    {
        upDown,
        leftRight
    }
```