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:

We want developers to use our framework but we don't want to give them our code. Also we don't want the developers to be able to instantiate the GameObject object independently but only through the ObjectFactory class.

Previous Solution:

No previous attempt to solve this problem.

Solution/Current Approach:

Design Decision:

1. For the first problem there is only one solution that is to make our project into a class library and ship it to our developers.
2. For the second problem we have 2 choices:
    a. Make the constructors of GameObject class as protected and inherit the factory class from it.
    b. Make the constructors of the GameObject class as **internal** so the developers that use our framework cannot instantiate it.

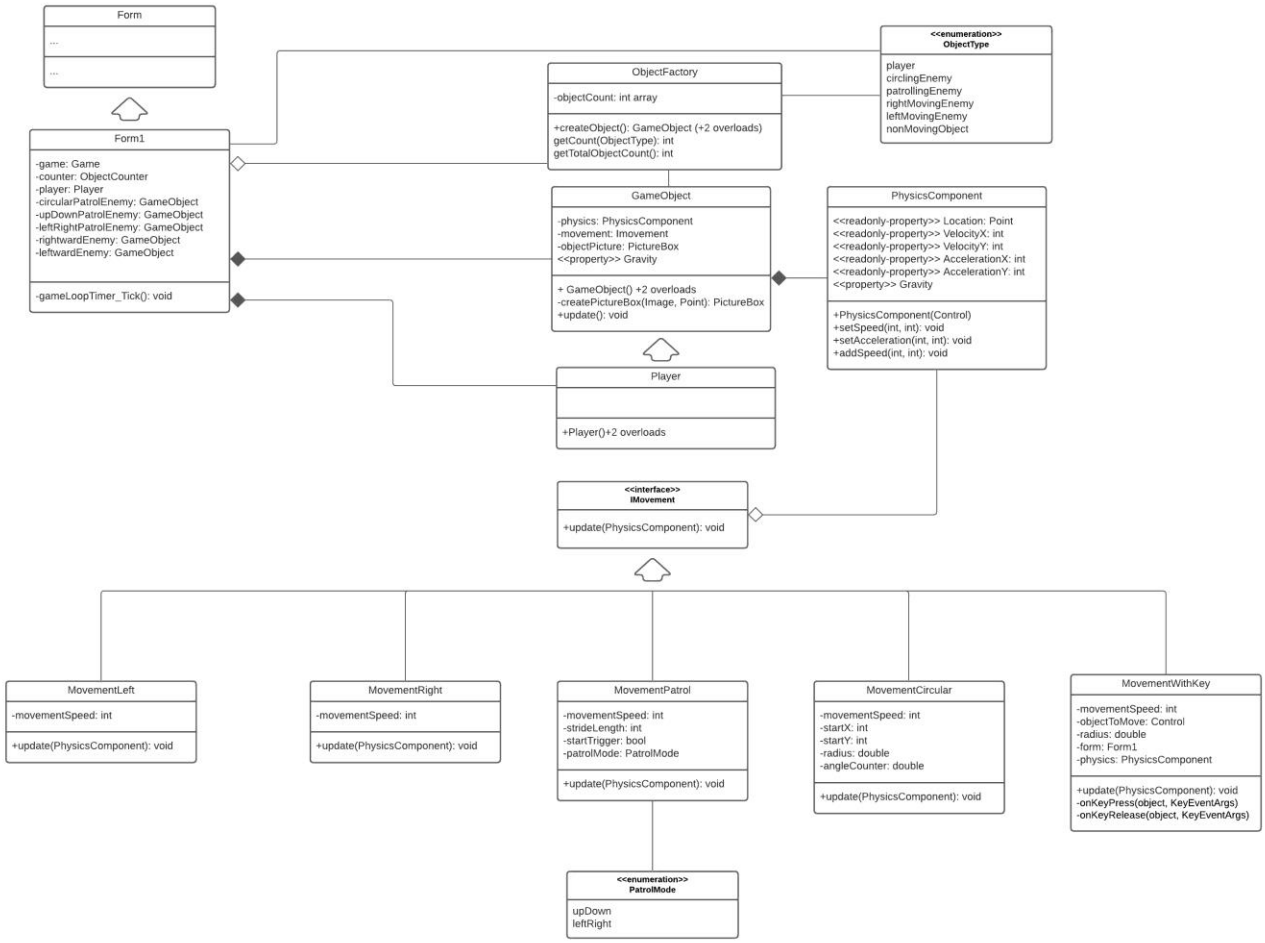
    Demerits:

    The demerits of the first approach are as follows:

    1. ObjectFactory ***is not*** a GameObject.
    2. We don't want the ObjectFactory to contain the attribute sand methods of GameObject.
    3. In future if we want to use multiple classes in out ObjectFactory, we it'll be impossible due to multiple inheritance problem.
    4. This approach is not an elegant approach.

    Decision: Due to above mentioned factors, we pick the second option of using ***internal*** as the access modifier for construtors of the GameObject class.

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();
            factory = ObjectFactory.Instance();
            GameObject player = factory.createObject(playerPictureBox, new
MovementWithKey(playerPictureBox, 15), ObjectType.player);
            GameObject circularPatrolEnemy = factory.createObject(CircularPictureBox, new
MovementCircular(CircularPictureBox, 100), ObjectType.circlingEnemy);
            GameObject upDownPatrolEnemy = factory.createObject(UDPatrolPictureBox, new
MovementPatrol(500, 5, PatrolMode.upDown), ObjectType.patrollingEnemy);
            GameObject leftRightPatrolEnemy = factory.createObject(LRPatrolPictureBox,
new MovementPatrol(500, 5, PatrolMode.leftRight), ObjectType.patrollingEnemy);
            GameObject rightwardEnemy = factory.createObject(rightwardPictureBox, new
MovementRight(5), ObjectType.rightMovingEnemy);
            GameObject leftwardEnemy = factory.createObject(leftwardPictureBox, new
MovementLeft(5), ObjectType.leftMovingEnemy);
            game.addGameObject(player);
            game.addGameObject(circularPatrolEnemy);
            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: {factory.getTotalObjectsCount()}";
        }
    }
```

Game:

```
public class Game
    {
        List<GameObject> gameObjects = new List<GameObject>();
        private static Game gameInstance;
        private Game() { }
        public static Game Instance()
        {
            if (gameInstance == null)
                gameInstance = new Game();
            return gameInstance;
        }
```

```
        public void addGameObject(GameObject gameObject)
        {
            gameObjects.Add(gameObject);
        }
        public void update()
        {
            foreach (GameObject gameObject in gameObjects)
                gameObject.update();
        }
    }
```

GameObject:

```
public class GameObject
    {
        protected PhysicsComponent physics;
        protected IMovement objectMovement;
        public float Gravity { get => physics.Gravity; set => physics.Gravity = value; }
        internal GameObject(Control objectPicture, IMovement objectMovement, float
objectGravity = 1)
        {
            //for creating object from a component
            physics = new PhysicsComponent(objectPicture, objectGravity);
            this.objectMovement = objectMovement;
        }
        internal 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);
        }
        internal 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();
        }
```

```
    }
```

IMovement:

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

MovementPatrol:

```
public 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:

```
public 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:

```
public 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:

```
public 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:

```
public 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:

```
public 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:

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

PatrolMode:

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