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 to reuse the same movement object for multiple game objects instead of making unnecessary clones of the same movement objects. This is to optimize our framework and increase performance.

Previous Solution:
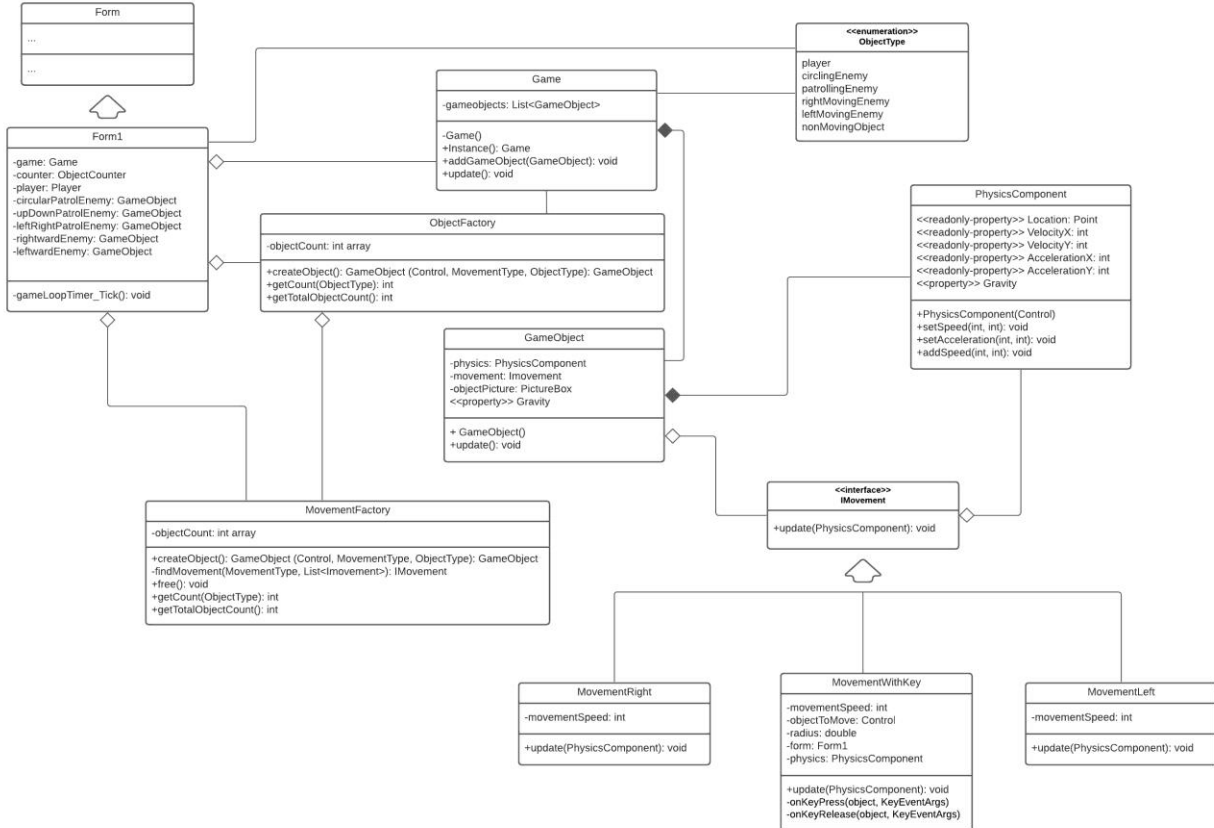
No previous attempt to solve this problem.

Solution/Current Approach:

Design Decision:

1. We want to reuse our existing movement objects. One common method for that can be object pool pattern.
2. Merits:
    a. By using oject pooling we can increase the performance of the framework.
    b. We can decrease the amount of space on the disk.
    c. We can get a higher frame rate.
3. Demerits:
    a. This approach can have a significant impact only in projects that deal with hundreds or thousands of object at a time.
    b. It feels like an unnecessary increase in complexity for smaller projects.
    c. Taking control of object disposing prevents us from using the features of modern programming language's garbage collector.
    d. Finding the object, removing and adding it to the available or occupied list etc might sometimes be even more expensive then creating the new object.

UML Diagram:

Form

...

...

Form1

-game: Game
-counter: ObjectCounter
-player: Player
-circularPatrolEnemy: GameObject
-upDownPatrolEnemy: GameObject
-leftRightPatrolEnemy: GameObject
-rightwardEnemy: GameObject
-leftwardEnemy: GameObject

-gameLoopTimer_Tick(): void

Game

-gameobjects: List<GameObject>

-Game()
+Instance(): Game
+addGameObject(GameObject): void
+update(): void

**<<enumeration>>
ObjectType**

player
circlingEnemy
patrollingEnemy
rightMovingEnemy
leftMovingEnemy
nonMovingObject

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

ObjectFactory

-objectCount: int array

+createObject(): GameObject (Control, MovementType, ObjectType): GameObject
+getCount(ObjectType): int
+getTotalObjectCount(): int

GameObject

-physics: PhysicsComponent
-movement: Imovement
-objectPicture: PictureBox
<<property>> Gravity

+ GameObject()
+update(): void

MovementFactory

-objectCount: int array

+createObject(): GameObject (Control, MovementType, ObjectType): GameObject
-findMovement(MovementType, List<Imovement>): IMovement
+free(): void
+getCount(ObjectType): int
+getTotalObjectCount(): int

**<<interface>>
IMovement**

+update(PhysicsComponent): void

MovementRight

-movementSpeed: int

+update(PhysicsComponent): void

MovementWithKey

-movementSpeed: int
-objectToMove: Control
-radius: double
-form: Form1
-physics: PhysicsComponent

+update(PhysicsComponent): void
-onKeyPress(object, KeyEventArgs)
-onKeyRelease(object, KeyEventArgs)

MovementLeft

-movementSpeed: int

+update(PhysicsComponent): void

Code:

Form1:

```
public partial class Form1 : Form
    {
        Game game;
        ObjectFactory factory;
        MovementFactory movementFactory;
        public Form1()
        {
            InitializeComponent();
            game = Game.Instance();
            factory = ObjectFactory.Instance();
            movementFactory = MovementFactory.Instance();
            game.addGameObject(factory.createObject(playerPictureBox,
MovementType.keyBoard, ObjectType.player, 0));
            game.addGameObject(factory.createObject(CircularPictureBox,
MovementType.left, ObjectType.circlingEnemy));
            game.addGameObject(factory.createObject(UDPatrolPictureBox,
MovementType.left, ObjectType.patrollingEnemy));
            game.addGameObject(factory.createObject(LRPatrolPictureBox,
MovementType.right, ObjectType.patrollingEnemy));
            game.addGameObject(factory.createObject(rightwardPictureBox,
MovementType.right, ObjectType.rightMovingEnemy));
            game.addGameObject(factory.createObject(leftwardPictureBox,
MovementType.left, ObjectType.leftMovingEnemy));
        }

        private void gameLoopTimer_Tick(object sender, EventArgs e)
        {
            game.update();
            objectCountLabel.Text = $"Objects: {factory.getTotalObjectsCount()},
Movements: Left:{movementFactory.getCount(MovementType.left)}
Right:{movementFactory.getCount(MovementType.right)}
KeyBoard:{movementFactory.getCount(MovementType.keyBoard)}";
        }
  }
```

Game:

```
public class Game
    {
        List<GameObject> gameObjects = new List<GameObject>();
        private static Game gameInstance;
        private static readonly object locker = new object();
        private Game() { }
        public static Game Instance()
        {
            lock (locker)
            {
                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;
        MovementFactory movementFactory;
        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;
            movementFactory = MovementFactory.Instance();
        }
        public virtual void update()
        {
            objectMovement.update(physics);
            physics.update();
            //Refresh();
        }
        ~GameObject() => movementFactory.free(objectMovement);
  }
```

IMovement:
```
// Interface for using in game objects
    public interface IMovement
    {
        public MovementType MovementType { get; }
        public bool IsExclusive { get; }
        void update(PhysicsComponent physics);
    }
```

MovementWithKey:
```
public class MovementWithKey : Movement, IMovement
    {
```

```csharp
        PhysicsComponent physics;
        int movementSpeed = 5;
        bool firstTimeCheck;
        public MovementWithKey():base(MovementType.keyBoard, true) { }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = -1;
            if (!firstTimeCheck)
            {
                this.physics = physics;
                physics.ObjectForm.KeyDown += new KeyEventHandler(keyDownHandler);
                physics.ObjectForm.KeyUp += new KeyEventHandler(keyUpHandler);
                firstTimeCheck = true;
            }

        }
        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:

```csharp
public class MovementRight : Movement, IMovement
    {
        int movementSpeed = 5;
        public MovementRight() : base(MovementType.right, false) { }
        public void update(PhysicsComponent physics)
        {
            physics.Gravity = 0;
            physics.setSpeed(movementSpeed, 0);
        }
    }
```

MovementLeft:

```csharp
public class MovementLeft : Movement, IMovement
    {
        int movementSpeed = 5;
        public MovementLeft():base(MovementType.left, false) { }
        public void update(PhysicsComponent physics)
        {
```

```
            physics.Gravity = 0;
            physics.setSpeed(-movementSpeed, 0);
        }
    }
```

ObjectFactory:
```
public class ObjectFactory
    {
        int[] objectCount = new int[10];
        MovementFactory movementFactory = MovementFactory.Instance();
        private static ObjectFactory factoryInstance;
        private static readonly object locker = new object();
        private ObjectFactory() { }
        public static ObjectFactory Instance()
        {
            lock (locker)
            {
                if (factoryInstance == null)
                    factoryInstance = new ObjectFactory();

                return factoryInstance;

            }
        }
        public GameObject createObject(Control objectPicture, MovementType movementType,
ObjectType objectType, float objectGravity = 1)
        {
            objectCount[(int)objectType]++;
            return new GameObject(objectPicture,
movementFactory.createMovement(movementType), objectGravity);
        }
        public int getCount(ObjectType objectType) => objectCount[(int)objectType];
        public int getTotalObjectsCount()
        {
            int count = 0;
            foreach (int objCount in objectCount) count += objCount;
            return count;
        }
    }
public class MovementFactory
    {
        static MovementFactory factoryInstance;
        private static readonly object locker = new object();
        List<IMovement> available = new List<IMovement>();
        List<IMovement> occupied = new List<IMovement>();
        int[] movementCount = new int[3];
        private MovementFactory() { }
        public static MovementFactory Instance()
        {
            lock (locker)
            {
                if (factoryInstance == null) factoryInstance = new MovementFactory();
                return factoryInstance;
            }
```

```csharp
        }
        public IMovement createMovement(MovementType movementType)
        {
            IMovement movement = findMovement(movementType, available);
            if (movement != null)
            {
                if (movement.IsExclusive)
                {
                    available.Remove(movement);
                    occupied.Add(movement);
                }
                return movement;
            }

            else
            {
                lock (locker)
                {
                    IMovement newMovement;
                    if (movementType == MovementType.right) newMovement = new
MovementRight();
                    else if (movementType == MovementType.left) newMovement = new
MovementLeft();
                    else newMovement = new MovementWithKey();

                    if (newMovement.IsExclusive) occupied.Add(newMovement);
                    else available.Add(newMovement);

                    movementCount[(int)newMovement.MovementType]++;
                    return newMovement;
                }
            }
        }
        public int getCount(MovementType movementType) =>
movementCount[(int)movementType];
        public int getTotalMovementCount()
        {
            int count = 0;
            foreach (int movCount in movementCount) count += movCount;
            return count;
        }
        IMovement findMovement(MovementType typeToLookFor, List<IMovement> listToSearch)
        {
            foreach(IMovement movement in listToSearch)
                if (movement.MovementType == typeToLookFor) return movement;
            return null;
        }
        public void release(IMovement movement)
        {
            if (movement.IsExclusive)
            {
                occupied.Remove(movement);
                available.Add(movement);
            }
        }
```

```
    }
```

ObjectType:

```csharp
public enum ObjectType
    {
        player,
        circlingEnemy,
        patrollingEnemy,
        rightMovingEnemy,
        leftMovingEnemy,
        nonMovingObject
    }
public enum MovementType
{
    keyBoard,
    right,
    left
}
```