



Game Framework Project

Submitted To:

Dr. Awais Hassan

Submitted By:

Muhammad Safee ullah, 2020-CS-13

Department of Computer Science

University of Engineering and Technology Lahore

Framework for Making Platformer Games

Table of Contents

Problem Statement:..... 3

Previous Solution: 3

Solution/Current Approach: 3

 Design Decision:..... 3

 UML Diagram: 3

Code: 4

 Form1:..... 4

 Game:..... 4

 GameObject: 5

 IMovement:..... 5

 MovementWithKey:..... 5

 MovementRight: 6

 MovementLeft: 6

 ObjectFactory:..... 7

 ObjectType: 10

Problem Statement:

We want to make a dynamic collision detection mechanism that user can use to add and define which objects are supposed to collide with each other and what they do after colliding (colliding behavior).

Previous Solution:

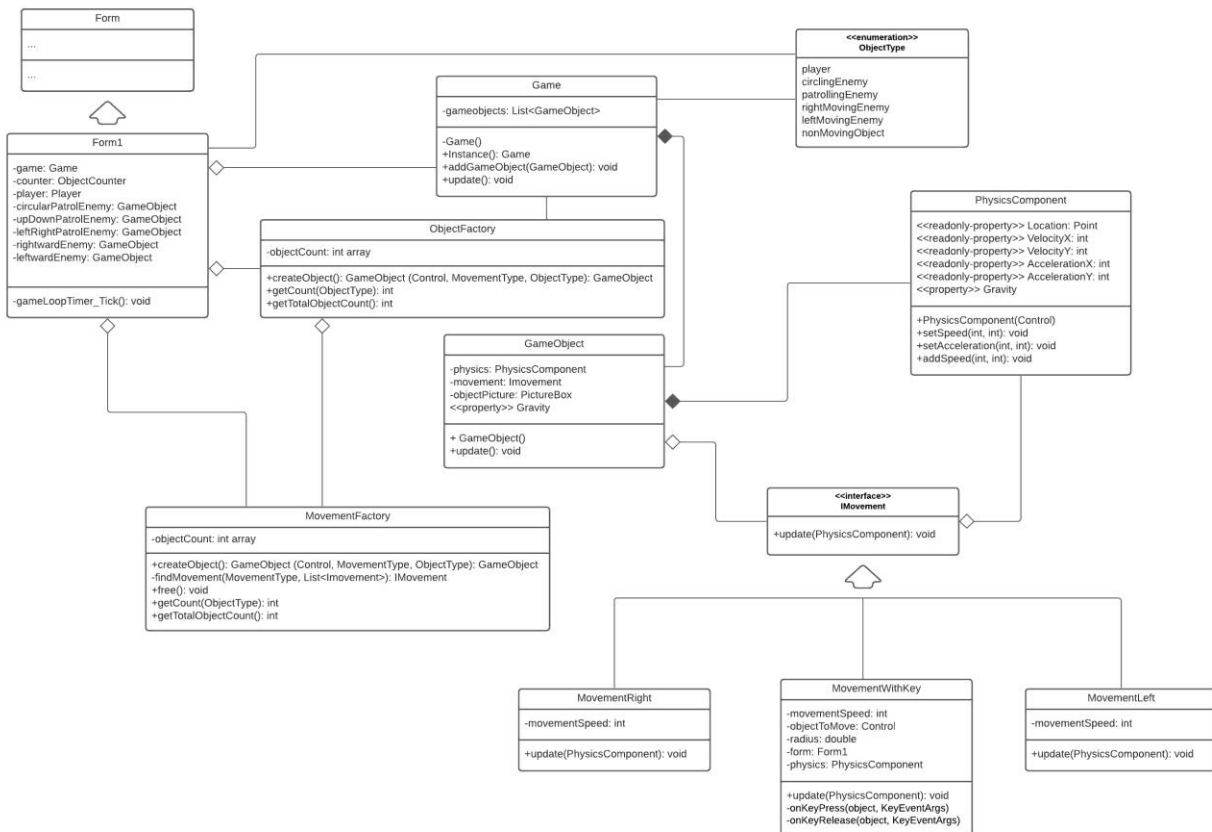
No previous attempt to solve this problem.

Solution/Current Approach:

Design Decision:

1. We can manually add collision behavior for each object
 2. We can use the object type enum to check for object type
- We go with the second choice because it is more scalable and general

UML Diagram:



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.addObject(factory.createObject(playerPictureBox,
MovementType.keyBoard, ObjectType.player, 0));
        game.addObject(factory.createObject(CircularPictureBox,
MovementType.left, ObjectType.circlingEnemy));
        game.addObject(factory.createObject(UDPatrolPictureBox,
MovementType.left, ObjectType.patrollingEnemy));
        game.addObject(factory.createObject(LRPatrolPictureBox,
MovementType.right, ObjectType.patrollingEnemy));
        game.addObject(factory.createObject(rightwardPictureBox,
MovementType.right, ObjectType.rightMovingEnemy));
        game.addObject(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();
        }
    }
}
```

University of Engineering and Technology, Lahore

```
        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
{

```

University of Engineering and Technology, Lahore

```
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:

```
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:

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

University of Engineering and Technology, Lahore

```
        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;
        }
    }
}
```

University of Engineering and Technology, Lahore

```
}
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);
    }
}
```



```
}
```

CollisionDetector:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SectionA2020CS13Framework
{
    public class CollisionDetector
    {
        ObjectType first, second;
        ICollisionBehavior behavior;
        public CollisionDetector(ObjectType first, ObjectType second, ICollisionBehavior
behavior)
        {
            this.first = first;
            this.second = second;
            this.behavior = behavior;
        }
        public void check(List<GameObject> objects)
        {
            for (int i = 0; i < objects.Count - 1; i++)
                for (int j = i + 1; j < objects.Count; j++)
                {
                    GameObject firstObject = objects[i];
                    GameObject secondObject = objects[j];

                    if (firstObject.Type == first && secondObject.Type == second
                        && firstObject.collidesWith(secondObject))
                    {
                        behavior.apply(firstObject, secondObject);
                    }
                }
        }
    }
}
```

ICollisionBehavior:

```
public interface ICollisionBehavior
{
    public void apply(GameObject first, GameObject second);
}
```

DeleteObject:

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

namespace SectionA2020CS13Framework
{
    public class DeleteObject : ICollisionBehavior
    {
        Game game;
```

University of Engineering and Technology, Lahore

```
ObjectType typeToDelete;
public DeleteObject(Game game, ObjectType typeToDelete)
{
    this.game = game;
    this.typeToDelete = typeToDelete;
}
public void apply(GameObject first, GameObject second)
{
    if(first.Type == typeToDelete)
    {
        first.removeSelf();
        game.removeGameObject(first);
        GC.Collect();
    }
    else if (second.Type == typeToDelete)
    {
        second.removeSelf();
        game.removeGameObject(second);
        GC.Collect();
    }
}
}
```

ObjectType:

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