Game FrameWork
(Task 03: Movement)

Task 03

Your framework should be developed in a way that if user of your framework wants to extend the behaviour of the motion then it could do that easily without opening the code of the framework.
For example, user of the framework want to move an object zig zag or diagonally then it only need to write the code for this movement and then inject into your framework so now framework allow gameObjects to move diagonally. (Note this change will only for the user)

Task 03: Solution

Whenever, we need to extend the functionality of a class, What pillar of OOP is used?

Task 03: Solution

Whenever, we need to extend the functionality of a class, What pillar of OOP is used?

Answer: Inheritance

Task 03: Solution

But our task is more like the function update should work differently under different conditions.

E.g., move differently in horizontal, vertical and keyboard movement.

Task 03: Solution

But our task is more like the function update should work differently under different conditions.

E.g., move differently in horizontal, vertical and keyboard movement.

This case sounds like dynamic Polymorphism.

Task 03: Solution

But our task is more like the function update should work differently under different conditions.

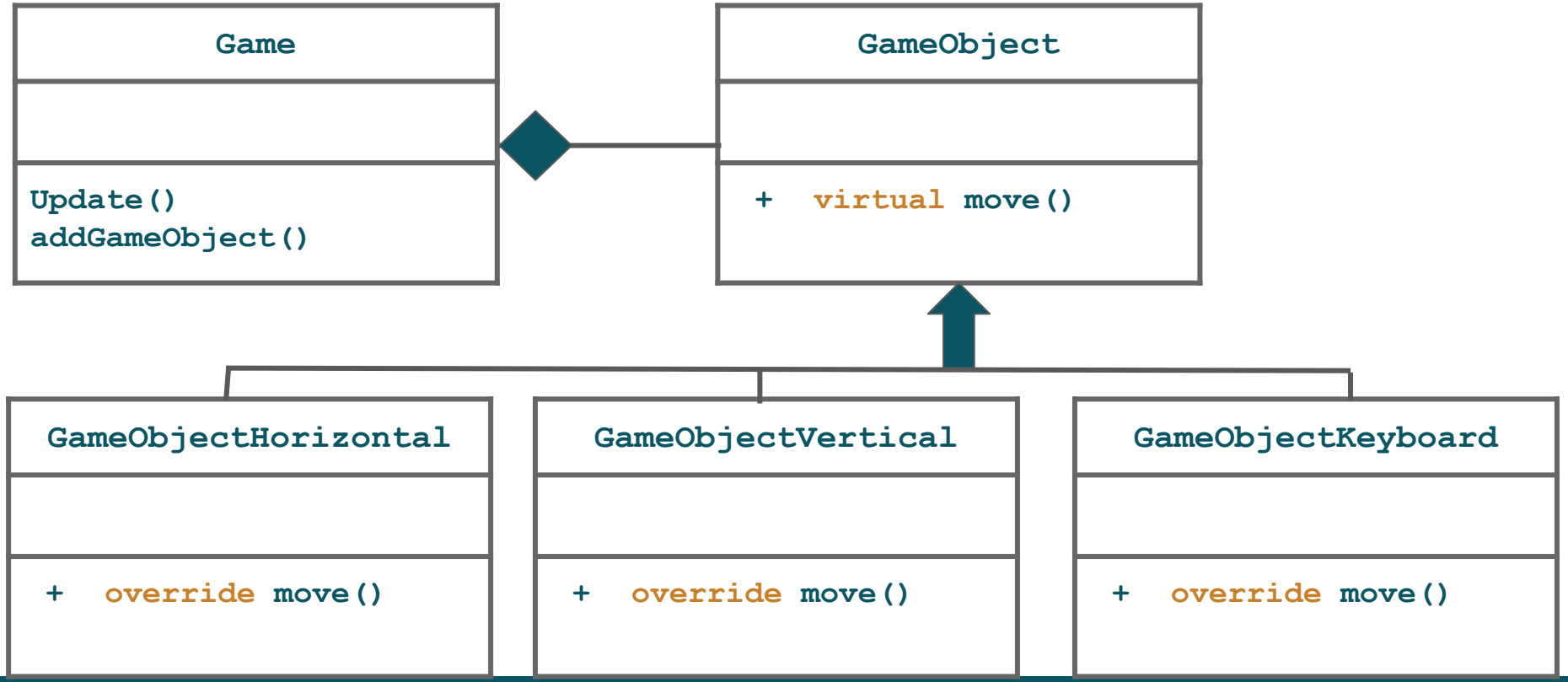
E.g., move differently in horizontal, vertical and keyboard movement.

So how to implement that?

Task 03: Solution

Let's make the GameObject base class and derive Horizontal, Vertical and Keyboard movement from it in the framework.
And make the update function virtual in base class and override in the derived classes.

Task 03: Solution



Task 03: Solution

```
public void addGameObject(Image img, int top, int left, string motion)
{
    if (motion == "vertical")
        GameObjectVertical ob = new GameObjectVertiacal(img, top, left);
    else if (motion == "horizontal")
        GameObjectHorizontal ob = new GameObjectHorizontal(img, top, left);
    else if (motion == "keyboard")
        GameObjectKeyboard ob = new GameObjectKeyboard(img, top, left);
    gameObjects.Add(ob);
    OnGameObjectAdded?.Invoke(ob.Pb, EventArgs.Empty);
}
```

Task 03: Solution

But how to pass the new functionality (ZigZag) movement from outside the framework?

Task 03: Solution

But how to pass the new functionality (ZigZag) movement from outside the framework?

In the current solution we are again using if statements in the framework and we have to change the condition to add the zigzag motion.

Task 03: Solution

But how to pass the new functionality (ZigZag) movement from outside the framework?

In the current solution we are again using if statements in the framework and we have to change the condition to add the zigzag motion.

So what is the correct solution?

Task 03: Solution

But how to pass the new functionality (ZigZag) movement from outside the framework?

In the current solution we are again using if statements in the framework and we have to change the condition to add the zigzag motion.

So what is the correct solution?

Task 03: Solution

We can pass the object from the form to the Game class in the Framework and then store in the list of GameObjects

```
public void addGameObject(Image img, int top, int left, GameObject ob)
{
    gameObjects.Add(ob);
    OnGameObjectAdded?.Invoke(ob.Pb, EventArgs.Empty);
}
```

```
g.addGameObject(Resources.enemyBlack1, 200, 200,
                new HorizontalMovement(10, boundary, "left"));
```

Task 03: Solution

In this way we can make a new class for Zigzag movement in the Consumer Project and just pass the object to the Game class in the framework.

The override move function will be called according to the object of (horizontal, vertical, keyboard or zigzag movement class) through dynamic polymorphism.

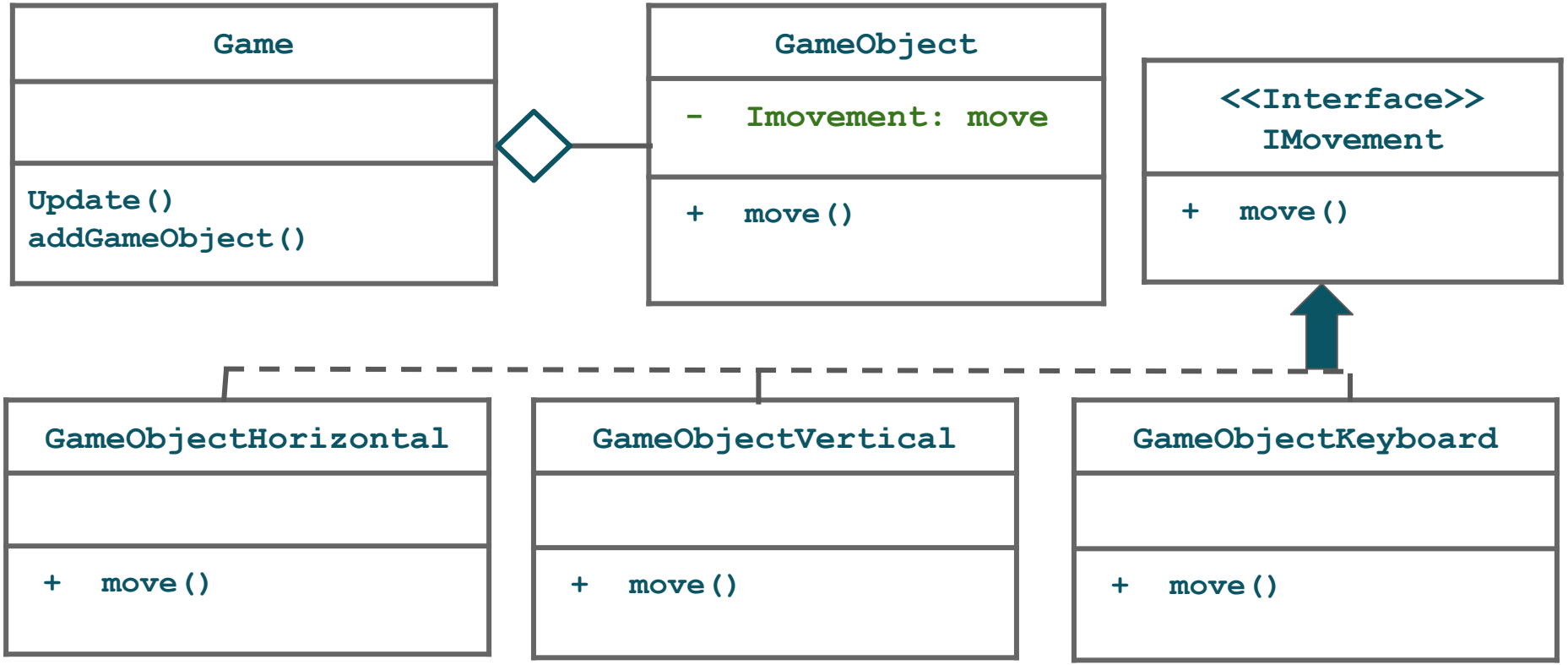
Task 03: Solution

Okay. Now we have achieved the desired functionality. But can we further improve the solution?

Task 03: Solution

Yes, we can.
It is through interfaces.

Task 03: Solution through Interface



Task 03: Solution

```
public void addGameObject(Image img, int top, int left, IMovement movement)
{
    GameObject ob = new GameObject(img, top, left, movement);
    gameObjects.Add(ob);
    OnGameObjectAdded?.Invoke(ob.Pb, EventArgs.Empty);
}
```

Task 03: Solution

Now, we can implement more than one interface in a class.

So, whenever we have a choice between inheritance vs Interface, we should go with interfaces.

Task 03: Solution

Make an Interface. And make a function move

```
public interface IMovement
{
    5 references
    Point move(Point location);
}
```

Task 03: Solution

Make a Class that will implement the interface.

```
public class HorizontalMovement: IMovement
{
    private int speed;
    private Point boundary;
    private string direction;
    private int offset = 90;
    1 reference
    public HorizontalMovement(int speed, Point boundary, string direction)
    {
        this.speed = speed;
        this.boundary = boundary;
        this.direction = direction;
    }
```

```
public Point move(Point location)
{
    if ((location.X + offset) >= boundary.X)
    {
        direction = "left";
    }
    else if (location.X + speed <= 0)
    {
        direction = "right";
    }

    if (direction == "left")
    {
        location.X -= speed;
    }
    else
    {
        location.X += speed;
    }
    return location;
}
```

Task 03: Game Class

```
public class Game
{
    private List<GameObject> gameObjects;
    public event EventHandler OnGameObjectAdded;
    1 reference
    public Game()
    {
        gameObjects = new List<GameObject>();
    }

    4 references
    public void addGameObject(Image img, int top, int left, IMovement movement)
    {
        GameObject ob = new GameObject(img, top, left, movement);
        gameObjects.Add(ob);
        OnGameObjectAdded?.Invoke(ob.Pb, EventArgs.Empty);
    }

    1 reference
    public void update()
    {
        foreach (GameObject go in gameObjects)
        {
            go.move();
        }
    }
```

Task 03: GameObject Class

```
public class GameObject
{
    private PictureBox pb;
    private IMovement movement;
    1 reference
    public GameObject(Image img, int top, int left, IMovement movement)
    {
        Pb = new PictureBox();
        Pb.Image = img;
        Pb.Width = img.Width;
        Pb.Height = img.Height;
        Pb.BackColor = Color.Transparent;
        Pb.Top = top;
        Pb.Left = left;
        this.Movement = movement;
    }
    2 references
    public IMovement Movement { get => movement; set => movement = value; }
    10 references
    internal PictureBox Pb { get => pb; set => pb = value; }

    1 reference
    public void move()
    {
        this.pb.Location = Movement.update(this.pb.Location);
    }
}
```

Task 03: Form

```
private void gameLoop_Tick(object sender, EventArgs e)
{
    g.update();
}
```

```
private void Space_Load(object sender, EventArgs e)
{
    g = new Game();
    // Event Handlers
    g.OnGameObjectAdded += new EventHandler(OnGameObjectAdded_Game);

    //making a point of the form's width and height.

    Point boundary = new Point(this.Width, this.Height);

    // adding GameObjects
    g.addGameObject(SpaceShooter.Properties.Resources.enemyBlack1, 200, 200, new HorizontalMovement(10, boundary, "left"));
    g.addGameObject(SpaceShooter.Properties.Resources.laserBlue02, 150, 150, new VerticalMovement(10, boundary, "up"));
    g.addGameObject(SpaceShooter.Properties.Resources.meteorGrey_small2, 190, 300, new ZigZagMovement(9, boundary, "left"));
    g.addGameObject(SpaceShooter.Properties.Resources.playerShip3_blue, 10, 250, new KeyboardMovement(8, boundary));

}
1 reference
private void OnGameObjectAdded_Game(object sender, EventArgs e)
{
    PictureBox pb = (PictureBox)sender;
    this.Controls.Add(pb);
}
```

Game FrameWork
(Task 04: Collision)

Task 04:

Now, we need to detect the collision between any two types of the objects and on collision we want to attach the behaviour.

For example, we want when a collision occur between object and ground the player shows stop behaviour means it stop falling.

Another example is when the player touches the enemy it should die.

Task 04:

Now, we need to detect the collision between any two types of the objects and on collision we want to attach the behaviour.

For example, we want when a collision occur between object and ground the player shows stop behaviour means it stop falling.

Another example is when the player touches the enemy it should die.

Task 04: Solution

Let's make a Collision Class

```
public class CollisionClass
{
    private ObjectTypes g1;
    private ObjectTypes g2;
    private ICollisionAction behaviour;

    1 reference
    public CollisionClass(ObjectTypes g1, ObjectTypes g2, ICollisionAction action)
    {
        this.G1 = g1;
        this.G2 = g2;
        this.Behaviour = action;
    }

    2 references
    public ObjectTypes G1 { get => g1; set => g1 = value; }
    2 references
    public ObjectTypes G2 { get => g2; set => g2 = value; }
    2 references
    internal ICollisionAction Behaviour { get => behaviour; set => behaviour = value; }

}
```

Task 04: Solution

Let's make an Interface for Game class to give the access for only subscribing the events

```
public interface IGame
{
    2 references
    void RaisePlayerDieEvent(PictureBox pb);
}
```

Task 04: Solution

Let's define the interface

```
public interface ICollisionAction
{
    2 references
    void performAction(IGame game, GameObject source1, GameObject source2);
}
```

Task 04: Player Action

```
public class PlayerCollision : ICollisionAction
{
    2 references
    public void performAction(Core.IGame game, GameObject source1, GameObject source2)
    {
        GameObject player;
        if (source1.Otype == ObjectTypes.player)
        {
            player = source1;
        }
        else
        {
            player = source2;
        }
        game.RaisePlayerDieEvent(player.Pb);
    }
}
```

Task 04: Game Object

```
public class GameObject
{
    private PictureBox pb;
    private IMovement movement;
    private ObjectTypes otype;
    1 reference
    public GameObject(Image img, ObjectTypes otype, int top, int left, IMovement movement)
    {
        Pb = new PictureBox();
        Pb.Image = img;
        Pb.Width = img.Width;
        Pb.Height = img.Height;
        Pb.BackColor = Color.Transparent;
        Pb.Top = top;
        Pb.Left = left;
        this.Movement = movement;
        this.Otype = otype;
    }
    4 references
    public IMovement Movement { get => movement; set => movement = value; }
    4 references
    public ObjectTypes Otype { get => otype; set => otype = value; }
    11 references
    internal PictureBox Pb { get => pb; set => pb = value; }
    1 reference
    public void move()
    {
        this.pb.Location = Movement.move(this.pb.Location);
    }
```

Task 04: Game Class

```
public class Game: IGame
{
    private List<GameObject> gameObjects;
    private List<CollisionClass> collisions;
    public event EventHandler OnGameObjectAdded;
    public event EventHandler OnPlayerDie;
    1 reference
    public Game()
    {
        gameObjects = new List<GameObject>();
        collisions = new List<CollisionClass>();
    }

    2 references
    public void RaisePlayerDieEvent(PictureBox playerGameObject)
    {
       //this is proxy of raising event handler
        OnPlayerDie?.Invoke(playerGameObject, EventArgs.Empty);

    }

    4 references
    public void addGameObject(Image img, ObjectTypes otype, int top, int left, IMovement movement)
    {
        GameObject ob = new GameObject(img, otype, top, left, movement);
        gameObjects.Add(ob);
        OnGameObjectAdded?.Invoke(ob.Pb, EventArgs.Empty);
    }
```

Task 04: Game Class

```
public void update()
{
    detectCollision();
    foreach (GameObject go in gameObjects)
    {
        go.move();
    }
}
1 reference
public void detectCollision()
{
    for(int x = 0; x < gameObjects.Count; x++)
    {
        for(int y = 0; y < gameObjects.Count; y++)
        {
            if (gameObjects[x].Pb.Bounds.IntersectsWith(gameObjects[y].Pb.Bounds))
            {

                foreach (CollisionClass c in collisions)
                {
                    if (gameObjects[x].Otype == c.G1 && gameObjects[y].Otype == c.G2)
                    {
                        c.Behaviour.performAction(this, gameObjects[x], gameObjects[y]);
                    }
                }
            }
        }
```

Task 04: Game Class

```
public void update()
{
    detectCollision();
    foreach (GameObject go in gameObjects)
    {
        go.move();
    }
}
1 reference
public void detectCollision()
{
    for(int x = 0; x < gameObjects.Count; x++)
    {
        for(int y = 0; y < gameObjects.Count; y++)
        {
            if (gameObjects[x].Pb.Bounds.IntersectsWith(gameObjects[y].Pb.Bounds))
            {

                foreach (CollisionClass c in collisions)
                {
                    if (gameObjects[x].Otype == c.G1 && gameObjects[y].Otype == c.G2)
                    {
                        c.Behaviour.performAction(this, gameObjects[x], gameObjects[y]);
                    }
                }
            }
        }
```

```
public void addCollision(CollisionClass c)
{
    collisions.Add(c);
}
```

Task 04: Form

```
private void Space_Load(object sender, EventArgs e)
{
    g = new Game();
    // Event Handlers
    g.OnGameObjectAdded += new EventHandler(OnGameObjectAdded_Game);
    g.OnPlayerDie += new EventHandler(removePlayer);

    //making a point of the form's width and height.

    Point boundary = new Point(this.Width, this.Height);

    // adding GameObjects
    g.addGameObject(SpaceShooter.Properties.Resources.enemyBlack1, ObjectTypes.otherObject, 200, 200,
                    new HorizontalMovement(10, boundary, "left"));
    g.addGameObject(SpaceShooter.Properties.Resources.laserBlue02, ObjectTypes.otherObject, 150, 150,
                    new VerticalMovement(10, boundary, "up"));
    g.addGameObject(SpaceShooter.Properties.Resources.meteorGrey_small2, ObjectTypes.otherObject, 190, 300,
                    new ZigZagMovement(9, boundary, "left"));
    g.addGameObject(SpaceShooter.Properties.Resources.playerShip3_blue, ObjectTypes.player, 10, 250,
                    new KeyboardMovement(8, boundary));

    // adding Collisions
    CollisionClass c = new CollisionClass(ObjectTypes.player, ObjectTypes.otherObject,new PlayerCollision());
    g.addCollision(c);
}
```

Task 04: Form

```
private void removePlayer(object sender, EventArgs e)
{
    this.Controls.Remove((PictureBox)sender);
}

1 reference
private void OnGameObjectAdded_Game(object sender, EventArgs e)
{
    PictureBox pb = (PictureBox)sender;
    this.Controls.Add(pb);
}

1 reference
private void gameLoop_Tick(object sender, EventArgs e)
{
    g.update();
}
```