

**Game Development Framework**

**Submitted By:**

**Asad Mehmood , 2020-CS-10**

**Submitted To:**

**Sir. Laeeq Khan Niazi**

**For fulfillment of**

**CS 162 Object Oriented Programming**

**Department of Computer Science**

**University of Engineering and Technology, Lahore**

**Phase 1:**

**Problem Statement:**

* Add falling functionality for different objects (picturebox) such as enemies and other players.

**Solution (In procedural programming):**

If we use procedural programming to solve this problem statement, we have used Gotoxy functionality to control the movement of different objects such as enemies and other players. In this we have used conditional statements in such a way that objects keep on coming down until it not strike the ground.

**Solution (In Object oriented programming):**

**1ST Approach**

In OOP, when we first design that we simply added all the functionalities in MainTickGame() function and also used a enum class .

**Demerits of the solution**

In this approach, we get the expected output but the way was not much dynamic. The objects were not protected. We have to add all the falling conditions for the object to fall under gravity. That was not that much dynamic and was also making code lengthy. From this approach, if we add pictureBox in form then we have to add all the added conditions again for that added pictureBox.

**2nd Approach**

**Code:**

**Form1 Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Gravity\_Framework

{

public partial class Gravity : Form

{

Game game = new Game(3);

public Gravity()

{

InitializeComponent();

}

private void timer1\_Tick(object sender, EventArgs e)

{

game.update(ground);

}

private void Gravity\_Load(object sender, EventArgs e)

{

GameObject playerObj = new GameObject(pbPlayer);

GameObject enemyObj1 = new GameObject(enemy1);

GameObject enemyObj2 = new GameObject(enemy2);

GameObject enemyObj3 = new GameObject(enemy3);

game.addGameobjects(playerObj);

game.addGameobjects(enemyObj1);

game.addGameobjects(enemyObj2);

game.addGameobjects(enemyObj3);

}

private void ground\_Click(object sender, EventArgs e)

{

}

}

}

**Game Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Collections;

namespace Gravity\_Framework

{

class Game

{

public int gravity;

List<GameObject> addobjects = new List<GameObject>();

public Game(int g)

{

this.gravity = g;

}

public void addGameobjects(GameObject gameobj)

{

addobjects.Add(gameobj);

}

public void update(PictureBox ground)

{

foreach (object get\_object in addobjects)

{

GameObject obj = (GameObject)get\_object;

obj.updateObjects(gravity,ground);

}

}

}

}

**GameObject Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Gravity\_Framework

{

class GameObject

{

private PictureBox real\_pictureBox;

public GameObject(PictureBox pB)

{

this.real\_pictureBox = pB;

}

public void updateObjects(int gravity,PictureBox ground)

{

if(this.real\_pictureBox.Bounds.IntersectsWith(ground.Bounds))

{

this.real\_pictureBox.Left += gravity;

}

else

{

this.real\_pictureBox.Top += gravity;

}

}

}

}

**Merits of solution:**

In this approach, we used different classes which are associated with each other and data is Encapsulated. The major benefit of this approach is that we not need to add all the conditions when a new pictureBox (object) is added. Simply, we have to create its object and pass it as parameter to Game class then it will fall under gravity without having additional conditions.

**Demerits of solution:**

The solution is still limited to limited number of objects that are allowed to be created. The only objects are allowed to fall under gravity. The objects are also created manually using the form and the movement of object is not generic.

**UML Diagram:**

Form1

-Gravity : Form

+Gravity()

-Gravity\_Load(object sender , EventArgs e)

-timer1\_Tick(object sender , EventArgs e)

+Game

+GameObject

real\_pictureBox : PictureBox

+gravity : int

-addobjects : List<>

+GameObject(PictureBox pB)

+updateObjects(int gravity , PictureBox ground)

+Game(int g)

+addGameObjects(GameObjects gameobj)

+update(PictureBox ground)

**Phase 2:**

**Problem Statement:**

* Add falling functionality and moving by keyboard keys for different objects (picturebox) such as enemies and other players.

**Solution (Procedural programming):**

This problem can be solve in procedural programming by using Gotoxy Functionality in such a way that the object created for left or right movement are assigned specific value and condition so that they can move in their respective direction. We will set their value and boundary so then they will move in that specific direction.

**Solution (In Object oriented programming):**

**Demerits of the previous solution**

In the previous approach, solution is still limited to limited number of objects that are allowed to be created. The only objects are allowed to fall under gravity. The objects are also created manually using the form and the movement of object is not generic.

**1st Approach:**

**Code:**

**Form1 Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Gravity\_Framework

{

public partial class Gravity : Form

{

Game game = new Game(3);

public Gravity()

{

InitializeComponent();

}

private void timer1\_Tick(object sender, EventArgs e)

{

game.update(ground);

}

private void Gravity\_Load(object sender, EventArgs e)

{

GameObject playerObj = new GameObject(pbPlayer);

GameObject enemyObj1 = new GameObject(enemy1);

GameObject enemyObj2 = new GameObject(enemy2);

GameObject enemyObj3 = new GameObject(enemy3);

game.addGameobjects(playerObj);

game.addGameobjects(enemyObj1);

game.addGameobjects(enemyObj2);

game.addGameobjects(enemyObj3);

}

private void ground\_Click(object sender, EventArgs e)

{

}

}

}

**Game Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Collections;

namespace Gravity\_Framework

{

class Game

{

public int gravity;

List<GameObject> addobjects = new List<GameObject>();

public Game(int g)

{

this.gravity = g;

}

public void addGameobjects(GameObject gameobj)

{

addobjects.Add(gameobj);

}

public void update(PictureBox ground)

{

foreach (object get\_object in addobjects)

{

GameObject obj = (GameObject)get\_object;

obj.updateObjects(gravity,ground);

}

}

}

}

**GameObject Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Gravity\_Framework

{

class GameObject

{

private PictureBox real\_pictureBox;

public GameObject(PictureBox pB)

{

this.real\_pictureBox = pB;

}

public void updateObjects(int gravity,PictureBox ground)

{

if(this.real\_pictureBox.Bounds.IntersectsWith(ground.Bounds))

{

this.real\_pictureBox.Left += gravity;

}

else

{

this.real\_pictureBox.Top += gravity;

}

}

}

**Solution of the previous problem:**

Now the new approach for this is that we have made separate classes for left and right movement. In this we simply have to give pictureBox , value of gravity and movement direction object and the object will then move according to movement direction object given to it. It makes the movement for generic and it can simply implement to any object .

**Code:**

**Form1 Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

public partial class Form1 : Form

{

List <GameObject> gameObjectList = new List <GameObject>();

public Form1()

{

InitializeComponent();

}

private void pictureBox3\_Click(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

GameObject player1Object = new GameObject(player1 , 5 , new LeftMovement());

GameObject player2Object = new GameObject(player2 , 6 , new RightMovement());

GameObject player3Object = new GameObject(player3, 6, new MoveWithKeyboard(this));

addObjects(player1Object);

addObjects(player2Object);

addObjects(player3Object);

}

private void addObjects(GameObject obj)

{

gameObjectList.Add(obj);

}

private void mainGameTimer\_Tick(object sender, EventArgs e)

{

foreach(GameObject obj in gameObjectList)

{

obj.update();

}

}

}

}

**GameObject Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class GameObject

{

PictureBox pictureBox;

int gravity;

Movement real\_movement;

public GameObject(PictureBox gameObj , int g , Movement movement)

{

pictureBox = gameObj;

gravity = g;

this.real\_movement = movement;

}

public void update()

{

real\_movement.moveObject(pictureBox, gravity);

}

}

}

**Movement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class Movement

{

public virtual void moveObject(PictureBox pictureBox, int gravity)

{

}

}

}

**LeftMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class LeftMovement : Movement

{

public override void moveObject(PictureBox pictureBox,int gravity)

{

pictureBox.Left -= gravity;

}

}

}

**RightMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class RightMovement : Movement

{

public override void moveObject(PictureBox pictureBox,int gravity)

{

pictureBox.Left += gravity;

}

}

}

**MoveWithKeyboard Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class MoveWithKeyboard : Movement

{

Form1 form;

PictureBox pictureBox;

int gravity;

public MoveWithKeyboard(Form1 form)

{

this.form = form;

form.KeyDown += new KeyEventHandler(keyholder);

}

public override void moveObject(PictureBox pictureBox, int gravity)

{

this.pictureBox = pictureBox;

this.gravity = gravity;

}

private void keyholder(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left)

{

this.pictureBox.Left -= gravity;

}

else if (e.KeyCode == Keys.Right)

{

this.pictureBox.Left += gravity;

}

}

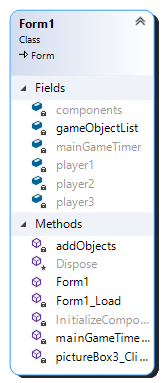
}

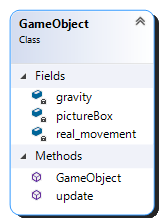
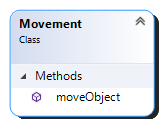
}

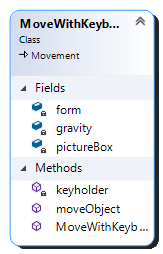
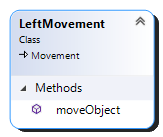
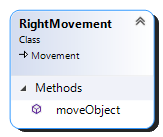
**Merit of the solution:**

Now the new approach for this is that we have made separate classes for left and right movement. In this we simply have to give pictureBox , value of gravity and movement direction object and the object will then move according to movement direction object given to it. It makes the movement for generic and it can simply implement to any object. In this , we have simply inherited leftmovement rightmovement and movewithkeyboard class with movement class and it had made the movement more generic.

**UML Diagram:**



**Phase 3:**

**Problem Statement:**

* Add falling functionality, moving by keyboard keys, petrol movement and jumping movement for different objects (picturebox) such as enemies and other players.

**Solution (In Object oriented programming):**

**Demerits of the previous solution**

In the previous approach, solution was not interfaced. Classes and their functions were not handled with interface there was no restriction for the class.

**1st Approach**

**Code:**

**Form1 Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

public partial class Form1 : Form

{

List <GameObject> gameObjectList = new List <GameObject>();

public Form1()

{

InitializeComponent();

}

private void pictureBox3\_Click(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

GameObject player1Object = new GameObject(player1 , 5 , new LeftMovement());

GameObject player2Object = new GameObject(player2 , 6 , new RightMovement());

GameObject player3Object = new GameObject(player3, 6, new MoveWithKeyboard(this));

addObjects(player1Object);

addObjects(player2Object);

addObjects(player3Object);

}

private void addObjects(GameObject obj)

{

gameObjectList.Add(obj);

}

private void mainGameTimer\_Tick(object sender, EventArgs e)

{

foreach(GameObject obj in gameObjectList)

{

obj.update();

}

}

}

}

**GameObject Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class GameObject

{

PictureBox pictureBox;

int gravity;

Movement real\_movement;

public GameObject(PictureBox gameObj , int g , Movement movement)

{

pictureBox = gameObj;

gravity = g;

this.real\_movement = movement;

}

public void update()

{

real\_movement.moveObject(pictureBox, gravity);

}

}

}

**Movement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class Movement

{

public virtual void moveObject(PictureBox pictureBox, int gravity)

{

}

}

}

**LeftMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class LeftMovement : Movement

{

public override void moveObject(PictureBox pictureBox,int gravity)

{

pictureBox.Left -= gravity;

}

}

}

**RightMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class RightMovement : Movement

{

public override void moveObject(PictureBox pictureBox,int gravity)

{

pictureBox.Left += gravity;

}

}

}

**MoveWithKeyboard Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

class MoveWithKeyboard : Movement

{

Form1 form;

PictureBox pictureBox;

int gravity;

public MoveWithKeyboard(Form1 form)

{

this.form = form;

form.KeyDown += new KeyEventHandler(keyholder);

}

public override void moveObject(PictureBox pictureBox, int gravity)

{

this.pictureBox = pictureBox;

this.gravity = gravity;

}

private void keyholder(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left)

{

this.pictureBox.Left -= gravity;

}

else if (e.KeyCode == Keys.Right)

{

this.pictureBox.Left += gravity;

}

}

}

}

**Solution of the previous problem:**

Now the new approach for this is that we have used interface. Now classes are restricted to inherit their interfaced parent class and define the function created in interfaced class. With this approach, we have also added two more features Petrol movement and Jumping Movement.

**Code:**

**Form1 Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

public partial class Form1 : Form

{

List<GameObject> gameObjectList = new List<GameObject>();

public Form1()

{

InitializeComponent();

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

GameObject player1Object = new GameObject(player1, 5, new LeftMovement());

GameObject player2Object = new GameObject(player2, 6, new RightMovement());

GameObject player3Object = new GameObject(player3, 6, new MoveWithKeyboard(this));

GameObject player4Object = new GameObject(player4, 6, new PetrolMovement());

GameObject player5Object = new GameObject(player5, 6, new Jumping(this));

addObjects(player1Object);

addObjects(player2Object);

addObjects(player3Object);

addObjects(player4Object);

addObjects(player5Object);

}

private void addObjects(GameObject obj)

{

gameObjectList.Add(obj);

}

private void mainGameTimer\_Tick(object sender, EventArgs e)

{

foreach (GameObject obj in gameObjectList)

{

obj.update();

}

}

}

}

**GameObject Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

interface IGameObject

{

void update();

}

class GameObject : IGameObject

{

PictureBox pictureBox;

int gravity;

Movement real\_movement;

public GameObject(PictureBox gameObj, int g, Movement movement)

{

pictureBox = gameObj;

gravity = g;

this.real\_movement = movement;

}

public void update()

{

real\_movement.moveObject(pictureBox, gravity);

}

}

}

**Movement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

interface Movement

{

void moveObject(PictureBox pictureBox, int gravity);

}

}

**LeftMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

class LeftMovement : Movement

{

public void moveObject(PictureBox pictureBox, int gravity)

{

pictureBox.Left -= gravity;

}

}

}

**RightMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

class RightMovement : Movement

{

public void moveObject(PictureBox pictureBox, int gravity)

{

pictureBox.Left += gravity;

}

}

}

**MoveWithKeyboard Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

class MoveWithKeyboard : Movement

{

Form1 form;

PictureBox pictureBox;

int gravity;

public MoveWithKeyboard(Form1 form)

{

this.form = form;

form.KeyDown += new KeyEventHandler(keyholder);

}

public void moveObject(PictureBox pictureBox, int gravity)

{

this.pictureBox = pictureBox;

this.gravity = gravity;

}

private void keyholder(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left)

{

this.pictureBox.Left -= gravity;

}

else if (e.KeyCode == Keys.Right)

{

this.pictureBox.Left += gravity;

}

else if (e.KeyCode == Keys.Up)

{

this.pictureBox.Top -= gravity;

}

else if (e.KeyCode == Keys.Down)

{

this.pictureBox.Top += gravity;

}

}

}

}

**PetrolMovement Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

class PetrolMovement : Movement

{

int verticalSpeed = 3;

public void moveObject(PictureBox pictureBox, int gravity)

{

pictureBox.Top += verticalSpeed;

if (pictureBox.Top < 81 || pictureBox.Top > 380)

{

verticalSpeed = -verticalSpeed;

}

}

}

}

**Jumping Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework\_4

{

class Jumping : Movement

{

Form1 form;

PictureBox pictureBox;

int gravity;

int getJumping;

public Jumping(Form1 form)

{

this.form = form;

form.KeyDown += new KeyEventHandler(keyholder);

}

public void moveObject(PictureBox pictureBox, int gravity)

{

this.pictureBox = pictureBox;

this.gravity = gravity;

}

private void keyholder(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Space)

{

getJumping = this.pictureBox.Top;

this.pictureBox.Top = this.pictureBox.Top - 40;

}

this.pictureBox.Top = getJumping;

}

}

}

**Merit of the solution:**

With this approach, we basically have made interface in classes. This has made the classes to define the

Function created in interface classes. With this we have made our framework more efficient and classes are more punctual to define the interface function. Also, with the addition of two more movement Petrol and Jumping Movement, objects now just get the Movement object and start moving in the direction they get

**UML Diagram:**

