**<https://www.youtube.com/channel/UC0ZpJKLuJL4U03c7m23rnGw>**

**GUSFRING 2D Game**



Session: 2022 – 2026

**Submitted by:**

Harmain Iftikhar 2022-CS-07

**Supervised by:**

Prof. Dr. Laeeq Khan Nazi

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table of Contents**

**Introduction……………………………………………………….….………...……2-6**

**Wire Frames…........……………………………………………………..……..........7-9**

**Class Responsibility Collaboration Card……………………..………..….……..…10**

**Object Oriented Programming…………………………………….….………….…10**

**Design Pattern Implementation..…………………………………………….……...11**

**Conclusion……........……………………………………………………..…….....…..12**

**Code……........…………………………………………………………..….......….13-43**

## Short Description and Story of the Car Game

Basically in my Game there is a one main player the goal of the game is that you have to collect keys and then you will open the gate of winning there are lot hurdles in game like there is a bird and also different knife thrown from above and if you touch anyone of them your health will decrease and you lost your life there are also an enemy in the game you to kill them. You can increase your health by drinking energy drink.

## Game Characters Description

### Player:

There is one human player in the Game.

### Enemies

There are many enemy of different types in the game:

1. There is an enemy in human form that moves horizontal.
2. There is a bird that shoot form above.
3. Many knife and life killing things thrown from upward.

## Game Objects Description

Following are the Objects in the Game

* Gate
* Bird
* Energy Drink
* Coins
* Rock
* Tree
* Ground Surface
* Wall
* Knife

## Shooting System:

* The Player-Car will be able to shoot.
* Bird Shoot from above in explosion form.

## Rules & Interactions

You have to protect yourself from bird, bullets, knife and enemy.

## Goal of the Game

The goal of the game is to collect key and then open the gate.

**Wire Frames:**

****

****

**Object Oriented Programming:**

In this Game I have completely followed the OOP concepts.

**Inheritance:**

I have implemented inheritance at multiple places like in GameObject and Health & Game class and free fall.

**Encapsulation:**

I have implemented encapsulation in my game I have made private attributes and getter setters.

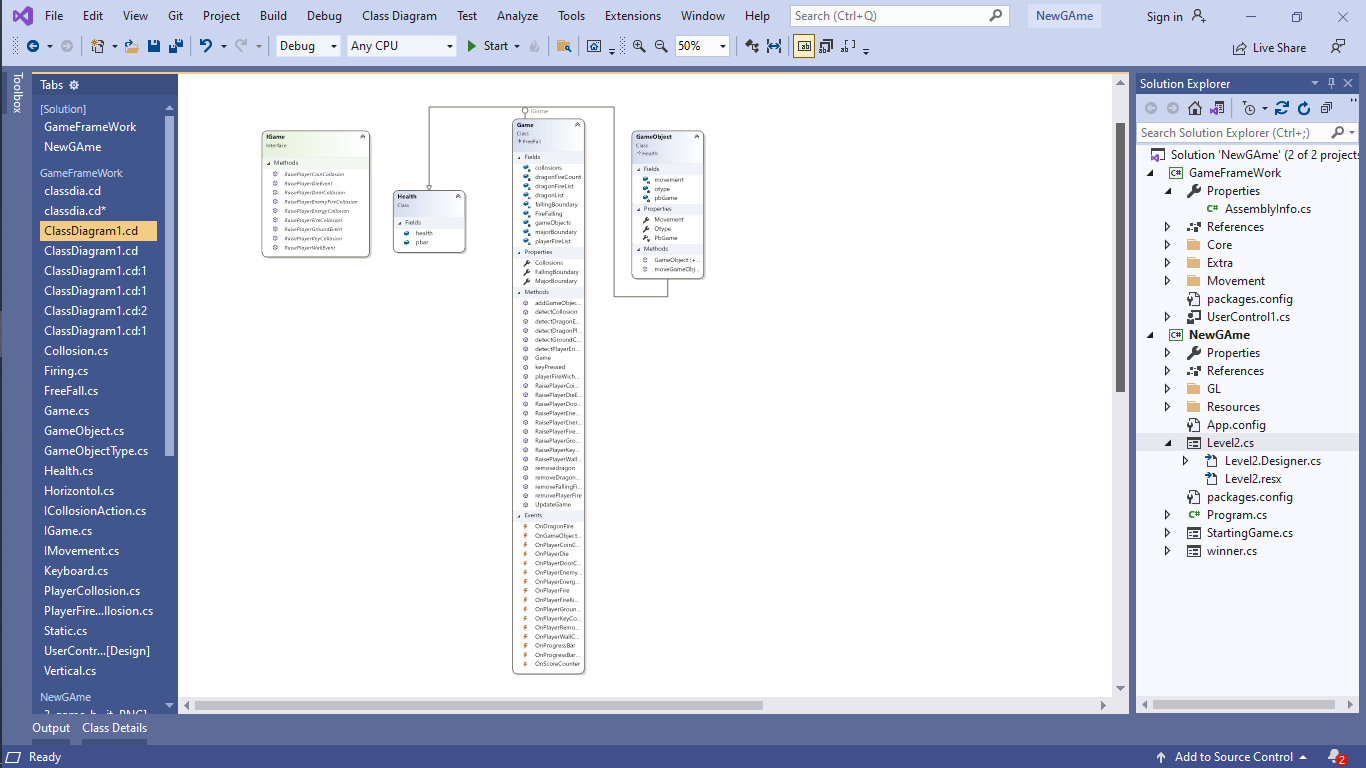
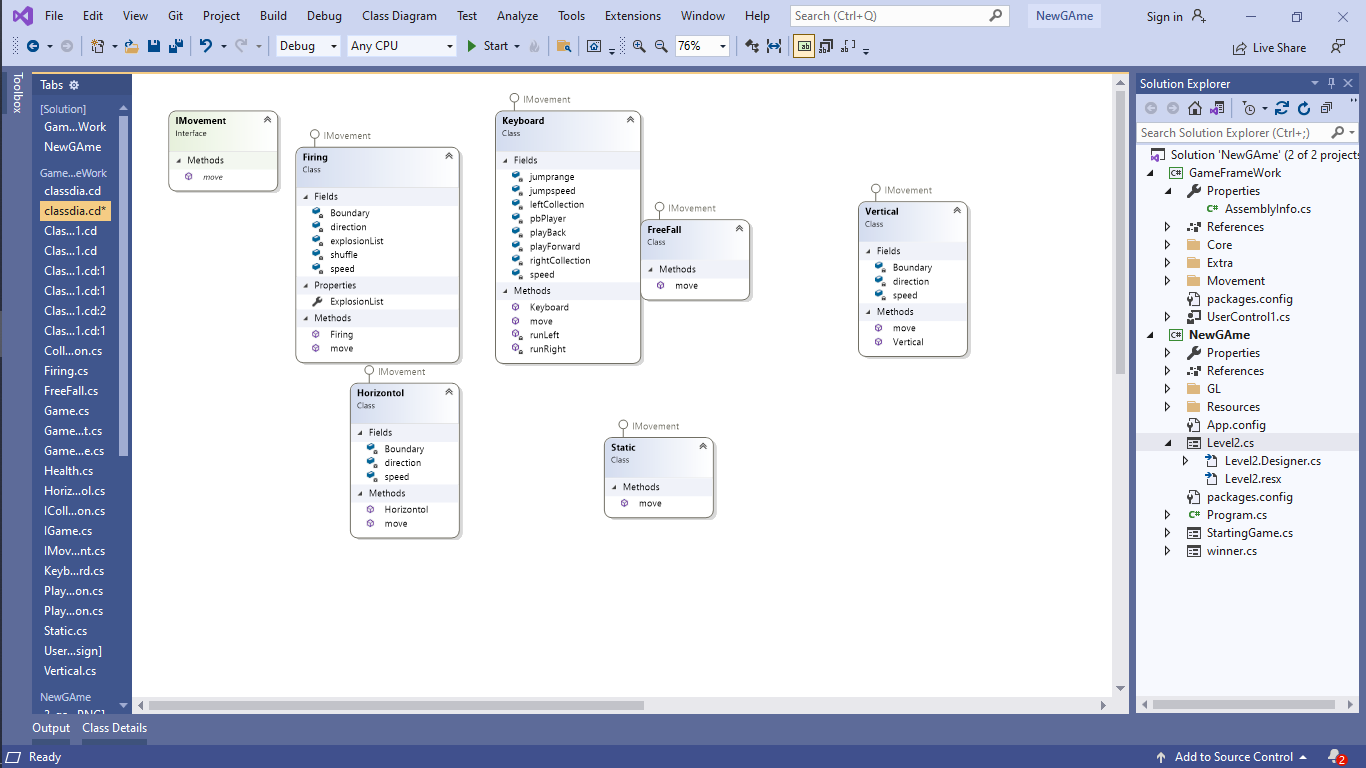
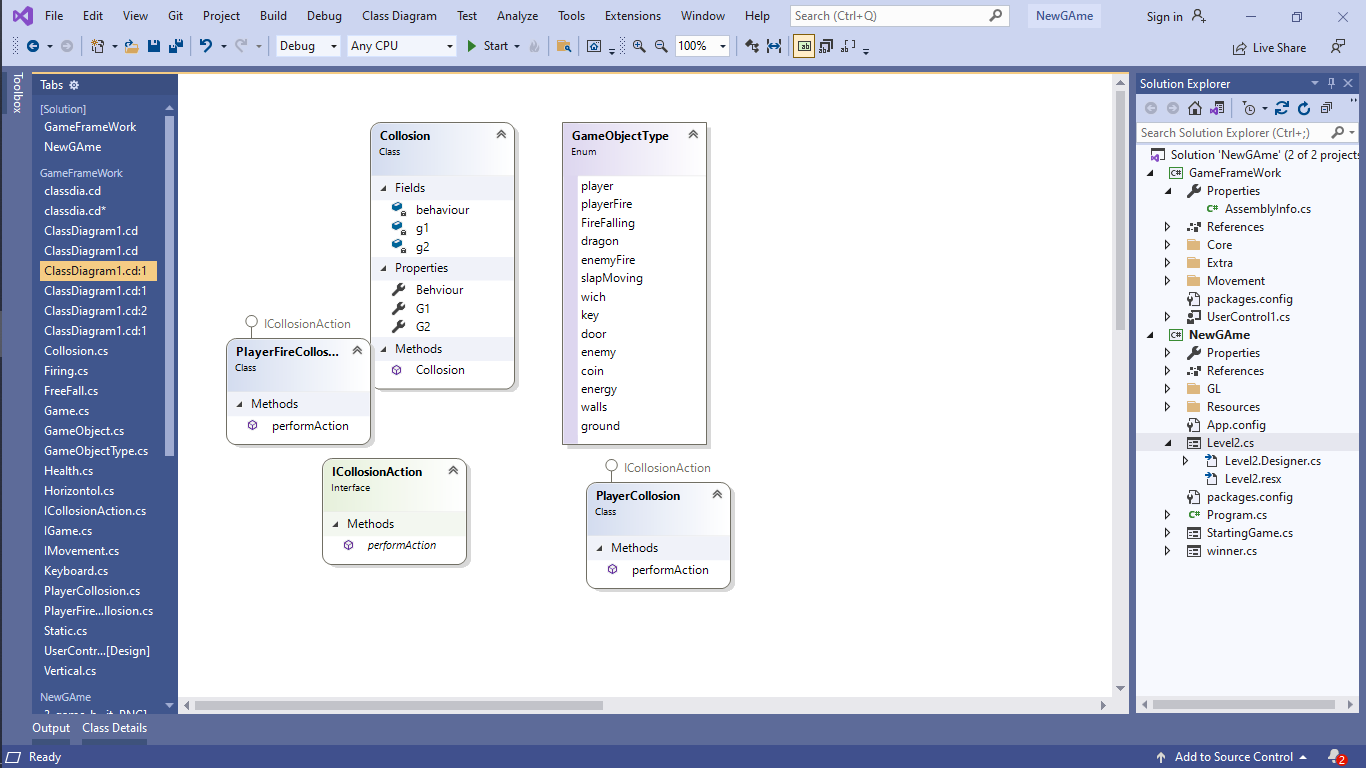
**Enumeration:**

I have also implemented enumeration classes in my Game that is a Game Object Type.

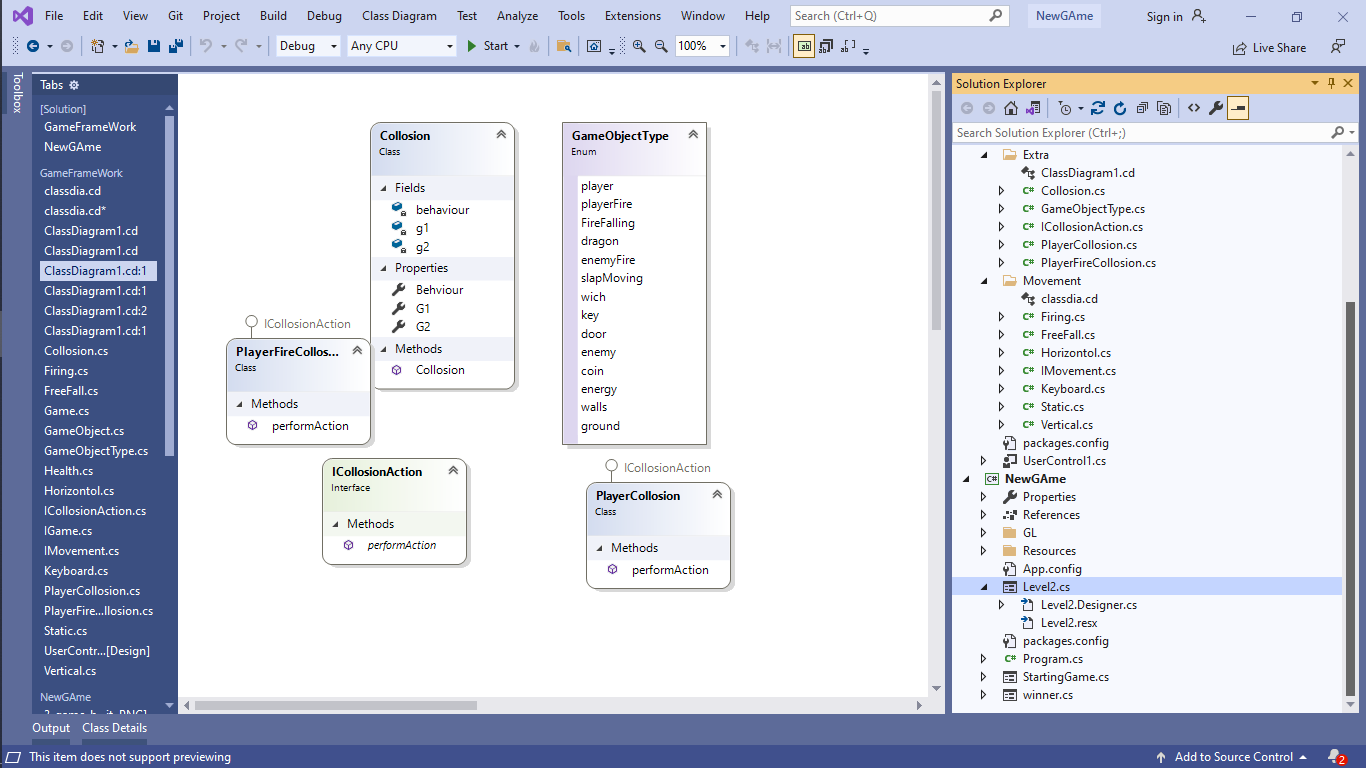
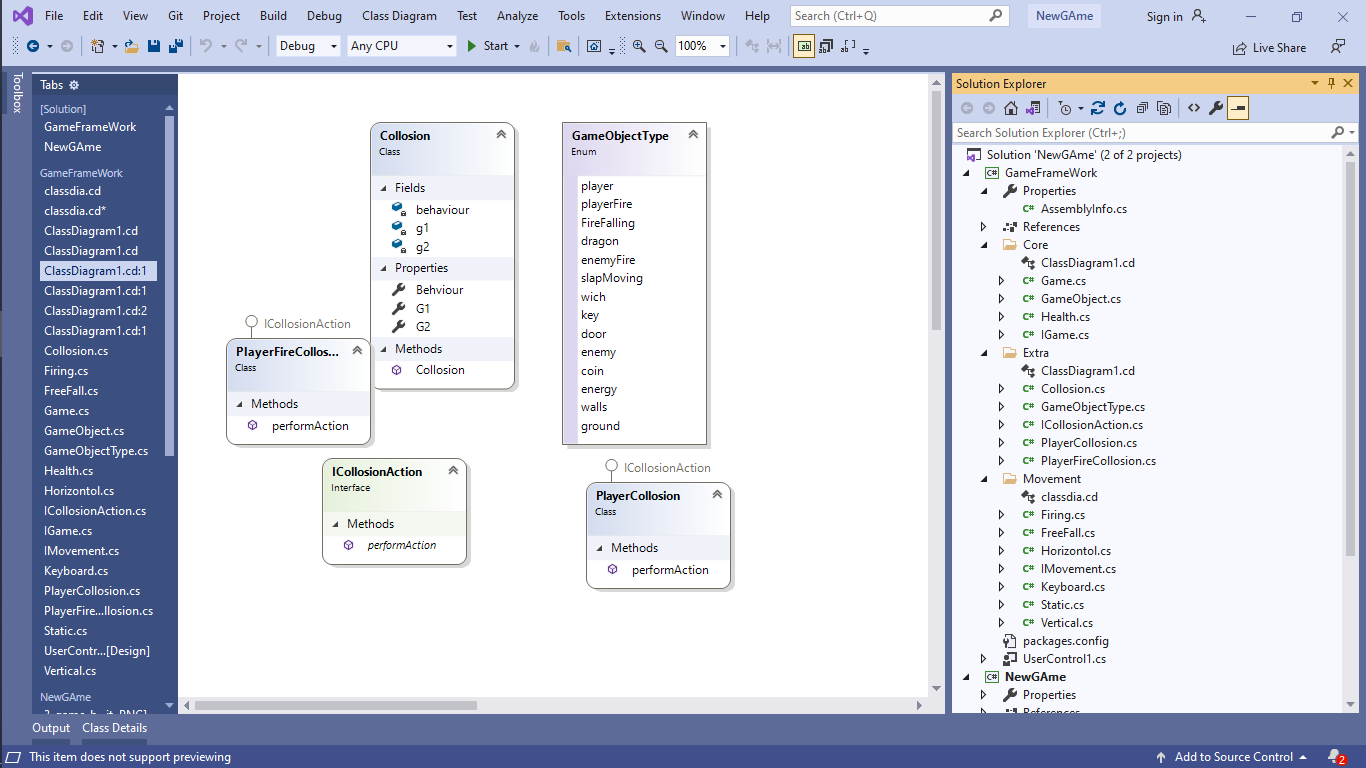
**Interface:**

I have also implemented interface classes in my Game like free fall and IMovement, keyboard IMovement, Firing and IMovement, Horizontal and IMovement, Static and IMovement.

**Class Diagram:**

******Design Pattern Implementation**

The directory structure for the project is given below:

****

**Conclusion:**

In conclusion, my game is built using the object-oriented programming approach. Its key functionalities includes the Firing, Moving and implementation of Framework. Important concepts object-oriented concepts such as association, inheritance and interface, encapsulation are used in this system. I faced several challenges during this phase. I faced difficulty in designing an effective class diagram collaboration model for the game and managing the key concepts of OOP paradigm. Throughout the period of designing, production and development of this project, I have learned how to create an effective game using object-oriented theory. The object-oriented approach can be really helpful in scaling of the project. It also helps the programmers in future to maintain and develop Games.

**Code:**

Collision Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameFrameWork.Extra

{

public class Collosion

{

private GameObjectType g1;

private GameObjectType g2;

private ICollosionAction behaviour;

public Collosion(GameObjectType g1,GameObjectType g2,ICollosionAction action)

{

this.g1 = g1;

this.g2 = g2;

this.behaviour = action;

}

public GameObjectType G1 { get => g1; set => g1 = value; }

public GameObjectType G2 { get => g2; set => g2 = value; }

public ICollosionAction Behviour { get => behaviour; set => behaviour = value; }

}

}

Firing Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public class Firing:IMovement

{

private int speed;

private Point Boundary;

private string direction;

private List<Image> explosionList;

int shuffle = 1;

public Firing(int speed, Point Boundary, string direction)

{

this.speed = speed;

this.Boundary = Boundary;

this.direction = direction;

}

public List<Image> ExplosionList { get => explosionList; set => explosionList = value; }

public PictureBox move(PictureBox pb)

{

if (direction == "right")

{

pb.Left += speed;

}

else if (direction == "left")

{

pb.Left -= speed;

}

else if (direction == "up")

{

pb.Top -= speed;

}

else if (direction == "down")

{

pb.Top += speed;

}

else if(direction== "diagonally")

{

shuffle++;

pb.Top += 7;

pb.Left -= 7;

for(int i = shuffle; i <= shuffle; i++)

{

pb.Image = ExplosionList[i];

if (i == 6)

{

shuffle = 0;

}

}

}

return pb;

}

}

}

Free Fall Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public class FreeFall:IMovement

{

public PictureBox move(PictureBox pb)

{

pb.Top += 1;

return pb;

}

}

}

Game Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

using GameFrameWork.Movement;

using GameFrameWork.Extra;

namespace GameFrameWork.Core

{

public class Game: FreeFall,IGame

{

List<GameObject> gameObjects;

private List<Collosion> collosions;

List<GameObject> playerFireList;

List<GameObject> FireFalling;

List<GameObject> dragonList;

List<GameObject> dragonFireList;

int dragonFireCount = 0;

private Point majorBoundary;

private Point fallingBoundary;

public event EventHandler OnGameObjectAdded;

public event EventHandler OnPlayerDie;

public event EventHandler OnPlayerWallCollosion;

public event EventHandler OnPlayerGroundCollosion;

public event EventHandler OnPlayerFire;

public event EventHandler OnPlayerRemoveFire;

public event EventHandler OnDragonFire;

public event EventHandler OnPlayerEnemyFireCollosion;

public event EventHandler OnPlayerCoinCollosion;

public event EventHandler OnScoreCounter;

public event EventHandler OnPlayerFireKiCollosion;

public event EventHandler OnProgressBar;

public event EventHandler OnProgressBarRemove;

public event EventHandler OnPlayerEnergyCollosion;

public event EventHandler OnPlayerKeyCollosion;

public event EventHandler OnPlayerDoorCollosion;

public List<Collosion> Collosions { get => collosions; set => collosions = value; }

public Point MajorBoundary { get => majorBoundary; set => majorBoundary = value; }

public Point FallingBoundary { get => fallingBoundary; set => fallingBoundary = value; }

public Game()

{

playerFireList = new List<GameObject>();

gameObjects = new List<GameObject>();

collosions = new List<Collosion>();

FireFalling = new List<GameObject>();

dragonList = new List<GameObject>();

dragonFireList = new List<GameObject>();

}

public void RaisePlayerDieEvent(PictureBox pb)

{

OnPlayerDie?.Invoke(pb, EventArgs.Empty);

}

public void RaisePlayerEnemyFireCollosion(PictureBox pb)

{

OnPlayerEnemyFireCollosion?.Invoke(pb, EventArgs.Empty);

}

public void RaisePlayerEnergyCollosion(GameObject ob)

{

gameObjects.Remove(ob);

OnPlayerEnergyCollosion?.Invoke(ob.PbGame, EventArgs.Empty);

}

public void RaisePlayerKeyCollosion(GameObject ob)

{

gameObjects.Remove(ob);

OnPlayerKeyCollosion?.Invoke(ob.PbGame, EventArgs.Empty);

}

public void RaisePlayerDoorCollosion(GameObject ob)

{

OnPlayerDoorCollosion?.Invoke(ob.PbGame, EventArgs.Empty);

}

public void RaisePlayerWallEvent(PictureBox pb)

{

OnPlayerWallCollosion?.Invoke(pb, EventArgs.Empty);

}

public void RaisePlayerFireCollosion(GameObject go)

{

go.pbar.Value = go.health-88;

if (go.pbar.Value<15)

{

dragonList.Remove(go);

OnPlayerFireKiCollosion?.Invoke(go.PbGame, EventArgs.Empty);

OnProgressBarRemove?.Invoke(go.pbar, EventArgs.Empty);

}

}public void RaisePlayerCoinCollosion(GameObject go)

{

gameObjects.Remove(go);

OnPlayerCoinCollosion?.Invoke(go.PbGame, EventArgs.Empty);

}

public void RaisePlayerGroundEvent(PictureBox pb)

{

OnPlayerGroundCollosion?.Invoke(pb, EventArgs.Empty);

}

public void keyPressed(Keys keyCode)

{

foreach (var go in gameObjects)

{

if (keyCode == Keys.F)

{

foreach (GameObject Go in gameObjects)

{

if (Go.Otype == GameObjectType.player)

{

OnPlayerFire?.Invoke(Go.PbGame, EventArgs.Empty);

}

}

}

}

}

public void addGameObject(Image img,GameObjectType Otype,int height,int width,int left,int top,IMovement movement)

{

GameObject ob = new GameObject(img, Otype,height, width, top, left, movement);

if (ob.Otype == GameObjectType.player)

{

gameObjects.Add(ob);

}

if (ob.Otype == GameObjectType.playerFire)

{

playerFireList.Add(ob);

}

if (ob.Otype == GameObjectType.FireFalling)

{

FireFalling.Add(ob);

}

if (ob.Otype == GameObjectType.dragon|| ob.Otype == GameObjectType.wich)

{

dragonList.Add(ob);

if (ob.Otype == GameObjectType.wich)

{

ob.pbar = new ProgressBar();

ob.health = 100;

ob.pbar.Value = 100;

ob.pbar.Height = 10;

ob.pbar.Width = 40;

ob. pbar.Top = ob.PbGame.Top - ob.pbar.Height;

ob.pbar.Left = ob.PbGame.Left+3;

OnProgressBar?.Invoke(ob.pbar, EventArgs.Empty);

}

}

if (ob.Otype == GameObjectType.enemyFire)

{

dragonFireList.Add(ob);

}

OnGameObjectAdded?.Invoke(ob.PbGame, EventArgs.Empty);

}

public void addGameObject(PictureBox pbGame, GameObjectType Otype, IMovement movement)

{

GameObject ob = new GameObject(pbGame, Otype,movement);

gameObjects.Add(ob);

}

public void UpdateGame()

{

detectCollosion();

detectPlayerEnemyFireCollosion();

detectDragonEnemyFireCollosion();

detectDragonPlayerCollosion();

playerFireWichCollosion();

foreach (var go in gameObjects)

{

go.moveGameObject();

if (go.Otype == GameObjectType.player)

{

for (int i = 0; i < 10; i++)

{

detectGroundCollosion();

go.PbGame = move(go.PbGame);

}

}

}

foreach (var far in playerFireList)

{

far.moveGameObject();

OnScoreCounter?.Invoke(null, EventArgs.Empty);

}

foreach (var fire in FireFalling)

{

detectCollosion();

fire.moveGameObject();

}

foreach(var dragon in dragonList)

{

dragon.moveGameObject();

if (dragonFireCount > 150)

{

OnDragonFire?.Invoke(dragon.PbGame, EventArgs.Empty);

dragonFireCount = 0;

}

if (dragon.Otype == GameObjectType.wich)

{

dragon.pbar.Top = dragon.PbGame.Top - dragon.pbar.Height;

dragon.pbar.Left = dragon.PbGame.Left + 3;

}

}

foreach(var dFire in dragonFireList)

{

dFire.moveGameObject();

}

dragonFireCount++;

removePlayerFire();

removeFallingFire();

removedragon();

removeDragonFire();

}

public void removePlayerFire()

{

for(int i=0;i< playerFireList.Count; i++)

{

if (playerFireList[i].PbGame.Location.X >= MajorBoundary.X)

{

OnPlayerRemoveFire?.Invoke(playerFireList[i].PbGame, EventArgs.Empty);

playerFireList.RemoveAt(i);

}

}

}

public void removeFallingFire()

{

for (int i = 0; i < FireFalling.Count; i++)

{

if (FireFalling[i].PbGame.Top+FireFalling[i].PbGame.Height >= fallingBoundary.Y)

{

OnPlayerRemoveFire?.Invoke(FireFalling[i].PbGame, EventArgs.Empty);

FireFalling.RemoveAt(i);

}

}

}

public void removedragon()

{

for (int i = 0; i < dragonList.Count; i++)

{

if (dragonList[i].Otype == GameObjectType.wich)

{

if (dragonList[i].PbGame.Left <= 1000)

{

OnPlayerRemoveFire?.Invoke(dragonList[i].PbGame, EventArgs.Empty);

OnProgressBarRemove?.Invoke(dragonList[i].pbar, EventArgs.Empty);

dragonList.RemoveAt(i);

}

}

else

{

if (dragonList[i].PbGame.Left <= 0)

{

OnPlayerRemoveFire?.Invoke(dragonList[i].PbGame, EventArgs.Empty);

dragonList.RemoveAt(i);

}

}

}

}

public void removeDragonFire()

{

for (int i = 0; i < dragonFireList.Count; i++)

{

if (dragonFireList[i].PbGame.Top >=492)

{

OnPlayerRemoveFire?.Invoke(dragonFireList[i].PbGame, EventArgs.Empty);

dragonFireList.RemoveAt(i);

}

}

}

public void detectCollosion()

{

for(int x = 0; x < gameObjects.Count; x++)

{

for(int y = 0; y < gameObjects.Count; y++)

{

if (gameObjects[x].PbGame.Bounds.IntersectsWith (gameObjects[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if(gameObjects[x].Otype==c.G1 && gameObjects[y].Otype == c.G2)

{

c.Behviour.performAction(this,c.G2,gameObjects[x], gameObjects[y]);

}

}

}

}

}

}

public void detectPlayerEnemyFireCollosion()

{

for (int x = 0; x < gameObjects.Count; x++)

{

for (int y = 0; y < FireFalling.Count; y++)

{

if (gameObjects[x].PbGame.Bounds.IntersectsWith(FireFalling[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if (gameObjects[x].Otype == c.G1 && FireFalling[y].Otype == c.G2)

{

c.Behviour.performAction(this, c.G2, gameObjects[x], FireFalling[y]);

}

}

}

}

}

}

public void detectDragonEnemyFireCollosion()

{

for (int x = 0; x < gameObjects.Count; x++)

{

for (int y = 0; y < dragonFireList.Count; y++)

{

if (gameObjects[x].PbGame.Bounds.IntersectsWith(dragonFireList[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if (gameObjects[x].Otype == c.G1 && dragonFireList[y].Otype == c.G2)

{

c.Behviour.performAction(this, c.G2, gameObjects[x], dragonFireList[y]);

}

}

}

}

}

}

public void detectDragonPlayerCollosion()

{

for (int x = 0; x < gameObjects.Count; x++)

{

for (int y = 0; y < dragonList.Count; y++)

{

if (gameObjects[x].PbGame.Bounds.IntersectsWith(dragonList[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if (gameObjects[x].Otype == c.G1 && dragonList[y].Otype == c.G2)

{

c.Behviour.performAction(this, c.G2, gameObjects[x], dragonList[y]);

}

}

}

}

}

}

public void detectGroundCollosion()

{

for (int x = 0; x < gameObjects.Count; x++)

{

if (gameObjects[x].Otype == GameObjectType.player || gameObjects[x].Otype == GameObjectType.ground)

{

for (int y = 0; y < gameObjects.Count; y++)

{

if (gameObjects[y].Otype == GameObjectType.player || gameObjects[y].Otype == GameObjectType.ground) {

if (gameObjects[x].PbGame.Bounds.IntersectsWith(gameObjects[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if (gameObjects[x].Otype == c.G1 && gameObjects[y].Otype == c.G2)

{

c.Behviour.performAction(this, c.G2, gameObjects[x], gameObjects[y]);

}

}

}

}

}

}

}

}

public void playerFireWichCollosion()

{

for (int x = 0; x < playerFireList.Count; x++)

{

for (int y = 0; y < dragonList.Count; y++)

{

if (playerFireList[x].PbGame.Bounds.IntersectsWith(dragonList[y].PbGame.Bounds))

{

foreach (Collosion c in Collosions)

{

if (playerFireList[x].Otype == c.G1 && dragonList[y].Otype == c.G2)

{

c.Behviour.performAction(this, c.G2, playerFireList[x], dragonList[y]);

}

}

}

}

}

}

}

}

GameObject Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

using GameFrameWork.Movement;

using GameFrameWork.Extra;

namespace GameFrameWork.Core

{

public class GameObject:Health

{

private PictureBox pbGame;

private IMovement movement;

private GameObjectType otype;

public GameObject(Image img,GameObjectType Otype,int height,int width,int top,int left,IMovement movement)

{

PbGame = new PictureBox();

PbGame.Image = img;

PbGame.Height = height;

PbGame.Width = width;

PbGame.Left = left;

PbGame.Top = top;

pbGame.SizeMode = PictureBoxSizeMode.StretchImage;

PbGame.BackColor = Color.Transparent;

this.Movement = movement;

pbGame.BringToFront();

this.Otype = Otype;

}

public GameObject(PictureBox pbGame,GameObjectType Otype,IMovement movement)

{

this.pbGame = pbGame;

this.Otype = Otype;

this.movement=movement;

}

public IMovement Movement { get => movement; set => movement = value; }

internal PictureBox PbGame { get => pbGame; set => pbGame = value; }

public GameObjectType Otype { get => otype; set => otype = value; }

public void moveGameObject()

{

PbGame = Movement.move(PbGame);

pbGame.BringToFront();

}

}

}

GameObjectType Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameFrameWork.Extra

{

public enum GameObjectType

{

player,

playerFire,

FireFalling,

dragon,

enemyFire,

slapMoving,

wich,

key,

door,

enemy,

coin,

energy,

walls,

ground,

}

}

Health Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace GameFrameWork.Core

{

public class Health

{

public ProgressBar pbar;

public int health = 100;

}

}

Horiontal Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public class Horizontol:IMovement

{

private int speed;

private Point Boundary;

private string direction;

public Horizontol (int speed,Point Boundary,string direction)

{

this.speed = speed;

this.Boundary = Boundary;

this.direction = direction;

}

public PictureBox move(PictureBox pb)

{

if (pb.Left <= Boundary.X)

{

direction = "right";

}

else if (pb.Left+pb.Width >= Boundary.Y)

{

direction = "left";

}

if (direction == "right")

{

pb.Left += speed;

}

else if (direction == "left")

{

pb.Left -= speed;

}

return pb;

}

}

}

ICollisonAction Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameFrameWork.Core;

namespace GameFrameWork.Extra

{

public interface ICollosionAction

{

void performAction(IGame game,GameObjectType type, GameObject g1, GameObject g2);

}

}

IGame Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace GameFrameWork.Core

{

public interface IGame

{

void RaisePlayerDieEvent(PictureBox pb);

void RaisePlayerWallEvent(PictureBox pb);

void RaisePlayerGroundEvent(PictureBox pb);

void RaisePlayerEnemyFireCollosion(PictureBox pb);

void RaisePlayerCoinCollosion(GameObject ob);

void RaisePlayerFireCollosion(GameObject ob);

void RaisePlayerEnergyCollosion(GameObject ob);

void RaisePlayerKeyCollosion(GameObject ob);

void RaisePlayerDoorCollosion(GameObject ob);

}

}

IMovements Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public interface IMovement

{

PictureBox move(PictureBox pb);

}

}

Keyboard Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

using EZInput;

namespace GameFrameWork.Movement

{

public class Keyboard:IMovement

{

private PictureBox pbPlayer;

private int playForward = 1;

private int playBack = 1;

private int jumpspeed =1;

private int jumprange = 60;

private int speed;

List<Image> leftCollection;

List<Image> rightCollection;

public Keyboard(int speed, List<Image> leftCollection,List<Image> rightCollection )

{

this.leftCollection = leftCollection;

this.rightCollection = rightCollection;

this.speed = speed;

this.leftCollection = leftCollection;

this.rightCollection = rightCollection;

}

public PictureBox move(PictureBox pb)

{

pbPlayer = pb;

if (true)

{

if (EZInput.Keyboard.IsKeyPressed(Key.LeftArrow))

{

if (pb.Left > 5)

{

pb.Left -= speed;

runLeft();

}

}

if (EZInput.Keyboard.IsKeyPressed(Key.RightArrow))

{

if (pb.Left < 1400)

{

pb.Left += speed;

runRight();

}

}

if (EZInput.Keyboard.IsKeyPressed(Key.Space))

{

if(pb.Top>200)

for(int i = jumpspeed; i < jumprange; i +=jumpspeed)

{

pb.Top -= jumpspeed;

}

}

}

return pb;

}

private void runLeft()

{

playForward = 1;

for(int i = playBack; i <= playBack; i++)

{

pbPlayer.Image = leftCollection[i];

if (playBack == 5)

{

pbPlayer.Image = leftCollection[i];

playBack = 1;

}

}

playBack++;

}

private void runRight()

{

playBack = 1;

for (int i = playForward; i <= playForward; i++)

{

pbPlayer.Image = rightCollection[i];

if (playForward == 5)

{

pbPlayer.Image = rightCollection[i];

playForward = 1;

}

}

playForward++;

}

}

}

PlayerCollison Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameFrameWork.Core;

namespace GameFrameWork.Extra

{

public class PlayerCollosion:ICollosionAction

{

public void performAction(IGame game,GameObjectType type,GameObject source1, GameObject source2)

{

GameObject player;

GameObject other;

if (source1.Otype == GameObjectType.player)

{

player = source1;

other = source2;

}

else

{

player = source2;

other = source1;

}

game.RaisePlayerDieEvent(player.PbGame);

if (GameObjectType.walls == type)

{

game.RaisePlayerWallEvent(player.PbGame);

}

if (GameObjectType.ground == type|| GameObjectType.slapMoving ==type)

{

game.RaisePlayerGroundEvent(player.PbGame);

}

if (GameObjectType.FireFalling == type|| GameObjectType.enemyFire == type|| GameObjectType.dragon == type|| GameObjectType.wich == type)

{

player.PbGame.Top = 490;

player.PbGame.Left = 21;

game.RaisePlayerEnemyFireCollosion(player.PbGame);

}

if (GameObjectType.coin == type)

{

game.RaisePlayerCoinCollosion(other);

}

if (GameObjectType.energy == type)

{

game.RaisePlayerEnergyCollosion(other);

}

if (GameObjectType.key == type)

{

game.RaisePlayerKeyCollosion(other);

}

if (GameObjectType.door == type)

{

game.RaisePlayerDoorCollosion(other);

}

}

}

}

PlayerFireCollison:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameFrameWork.Core;

namespace GameFrameWork.Extra

{

public class PlayerFireCollosion:ICollosionAction

{

public void performAction(IGame game, GameObjectType type, GameObject source1, GameObject source2)

{

GameObject playerFire;

GameObject other;

if (source1.Otype == GameObjectType.playerFire)

{

playerFire = source1;

other = source2;

}

else

{

playerFire = source2;

other = source1;

}

game.RaisePlayerFireCollosion(other);

}

}

}

Static Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public class Static:IMovement

{

public PictureBox move(PictureBox pb)

{

return pb;

}

}

}

Vertical Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace GameFrameWork.Movement

{

public class Vertical:IMovement

{

private int speed;

private Point Boundary;

private string direction;

public Vertical(int speed, Point Boundary, string direction)

{

this.speed = speed;

this.Boundary = Boundary;

this.direction = direction;

}

public PictureBox move(PictureBox pb)

{

if (pb.Top <= 200)

{

direction = "down";

}

else if (pb.Top + pb.Height >= Boundary.Y)

{

direction = "up";

}

if (direction == "down")

{

pb.Top += speed;

}

else if (direction == "up")

{

pb.Top -= speed;

}

return pb;

}

}

}