Escape Plan

Session 2023 - 2027

Submitted by:

Mian Saad Tahir 2023-CS-62

Supervised by:

Sir Laeeq Khan Niazi

Course:

CSC-102 Object Oriented Programming

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Contents

1. Short Description:

Escape Plan is an engaging Windows Form game built with C# utilizing advanced Object-Oriented Programming (OOP) concepts such as Interfaces, Singleton pattern, and Enumerations. The game presents a thrilling challenge where players control a dynamic character amidst a barrage of tanks and bullets. Your mission is to dodge incoming tanks, evade their relentless gunfire, and retaliate with precision shots to emerge victorious.

2. OOP Concepts:

- **Interfaces**:

Interfaces are employed to define a contract for classes to implement specific functionalities. The IMovement interface outlines methods for object movement, enabling developers to create custom movement behaviors by implementing this interface. This fosters

flexibility and interoperability, as different movement behaviors can be easily integrated into the framework.

- **Singleton Pattern**:

The Singleton pattern ensures that only one instance of a class is created, which is particularly useful for managing global game state or resources. In the context of the framework, the Singleton pattern is applied to the Game class to ensure that only a single instance of the game exists at any given time. This prevents unintended duplication of game instances and simplifies access to game-related functionalities.

- **Enumeration**:

Enumerations are employed to represent a fixed set of named constants, providing a more expressive and type-safe way to work with data. Within the framework, enumerations are used to define the direction and type of enemies, enhancing code readability and maintainability. By using enumerations, developers can easily identify and manage different enemy types and directions within the game.

3. Key Features:

- **Flexible Framework**:

  Escape Plan is not just a game; it's a framework that can be easily customized and applied to various front-end game designs with minimal modifications. Its modular architecture allows for seamless integration of new features and enhancements.

- **Player Control**:

Take command of your character with intuitive controls, navigating through a battlefield filled with adversarial tanks and obstacles. Fight back against the enemy onslaught by firing bullets from your own arsenal. Aim carefully and time your shots to destroy enemy tanks before they overwhelm you.

- **Enemy Tanks**:

Encounter waves of enemy tanks descending from the top of the screen. Each tank presents a unique challenge, requiring strategic maneuvers to avoid collisions and survive their relentless assaults. Stay alert as enemy tanks unleash a barrage of bullets in your direction. Use quick reflexes and nimble movements to evade incoming fire and stay alive.

- **Health and Score system**:

Monitor your player's health, lives, and score, displayed prominently in the left corner of the screen. Collect health pills scattered throughout the battlefield to replenish your health and extend your survival. Keep an eye out for score boosts to enhance your performance and climb the leaderboard.

- **Boss Battle**:

Prepare for the ultimate showdown against a formidable boss plane. Survive its onslaught and unleash your full firepower to emerge victorious and claim victory.

3. How to Play:

- Use the arrow keys to move your character.
- Press the spacebar to fire bullets at enemy tanks.
- Dodge incoming enemy bullets and tanks to avoid taking damage.
- Defeat all enemy tanks and conquer the final boss plane to win the game.

4. Wireframes:

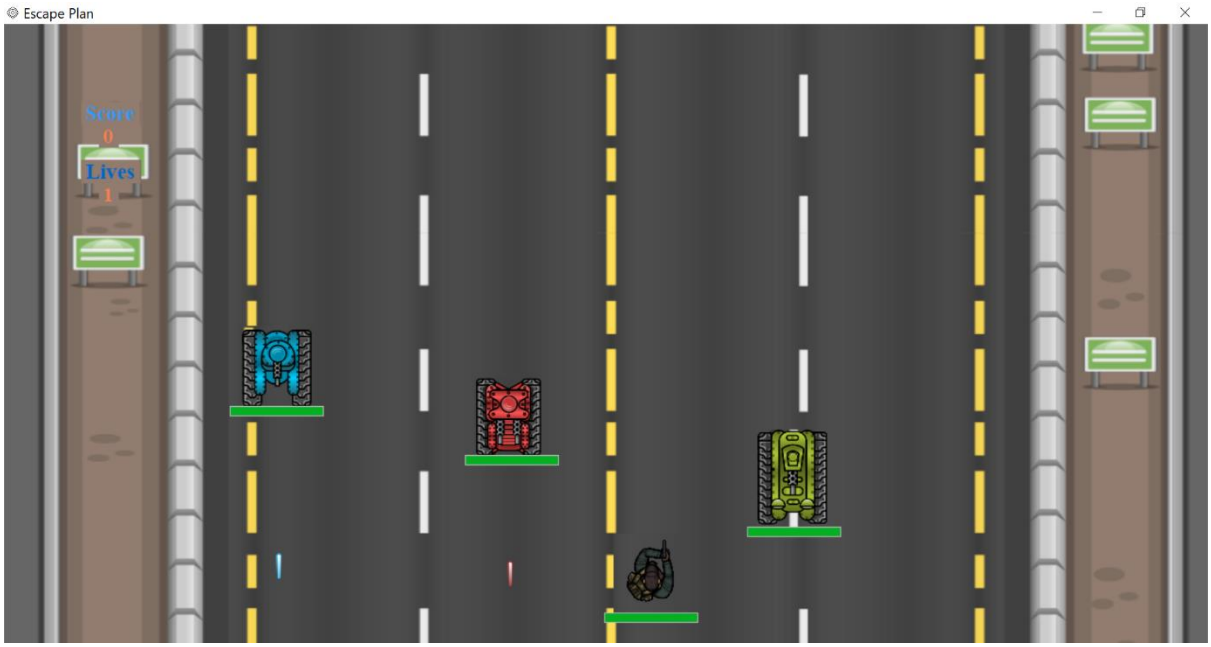


Fig.: Tank Battle

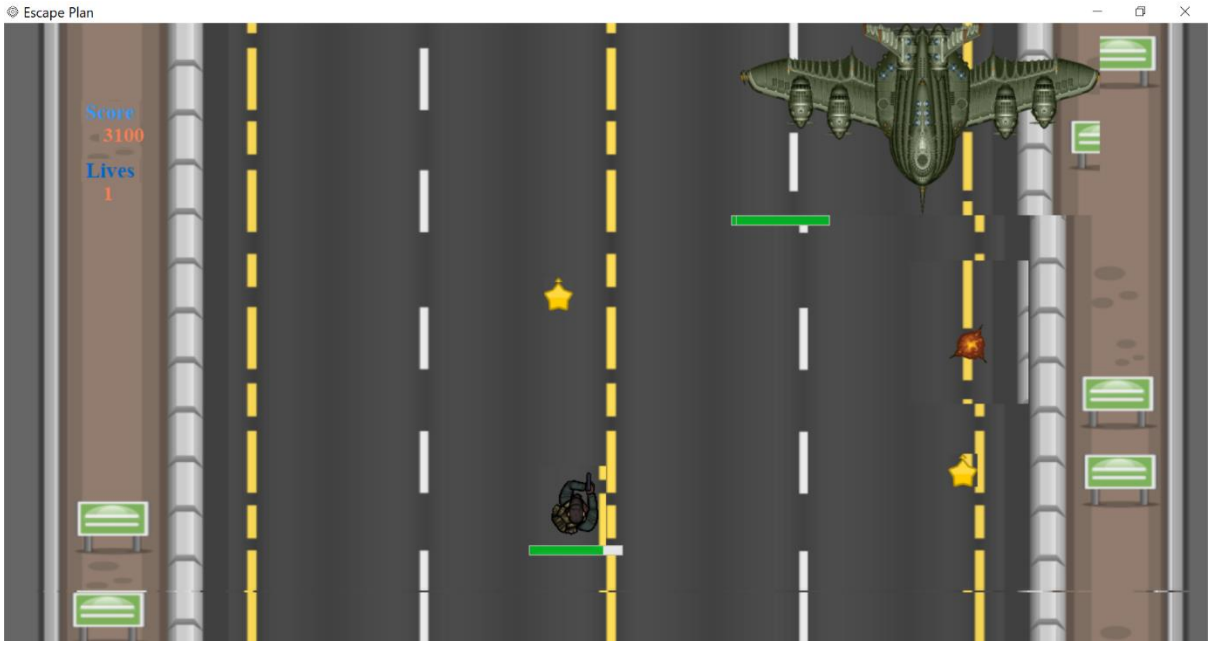


Fig.: Boss Fight

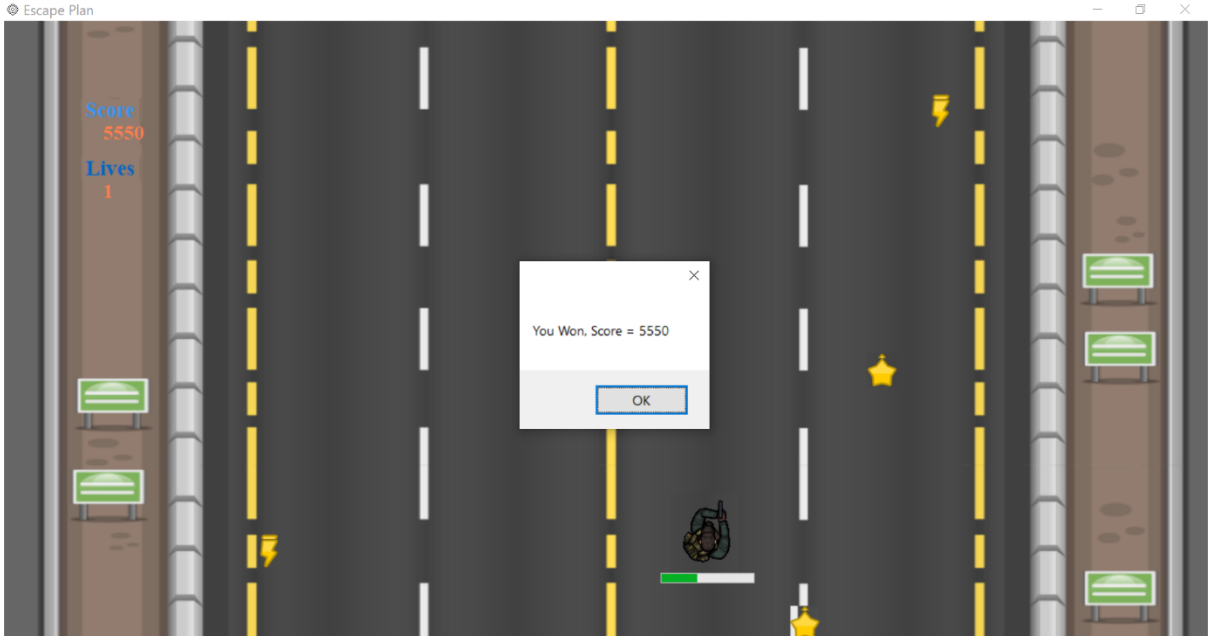


Fig.: Won the game

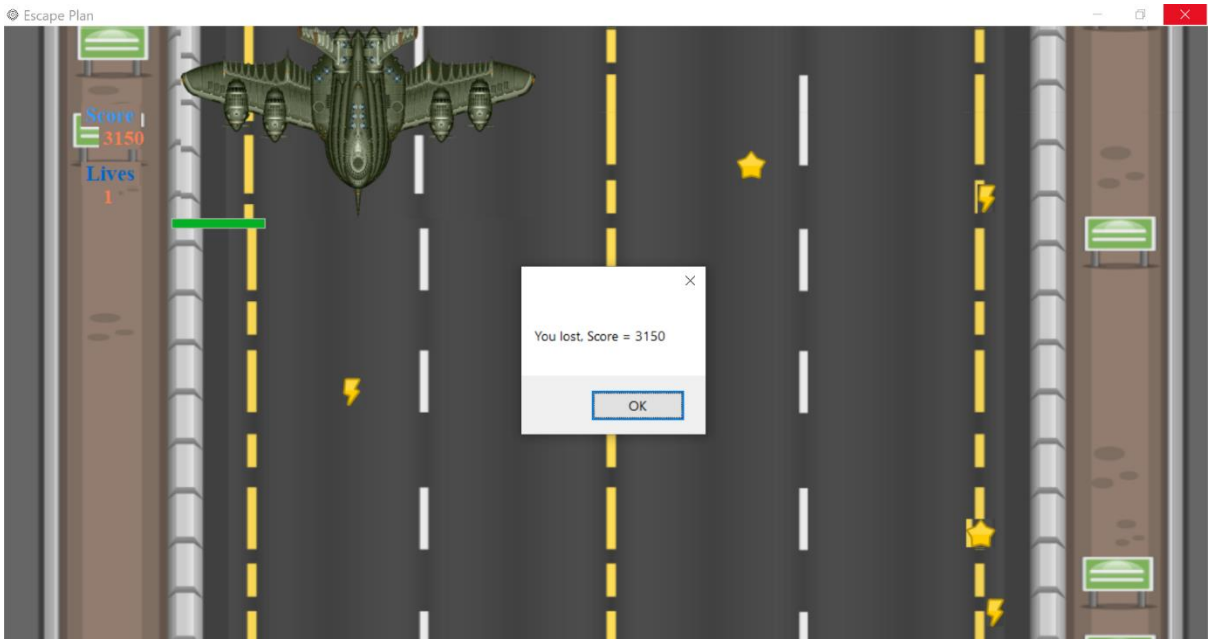
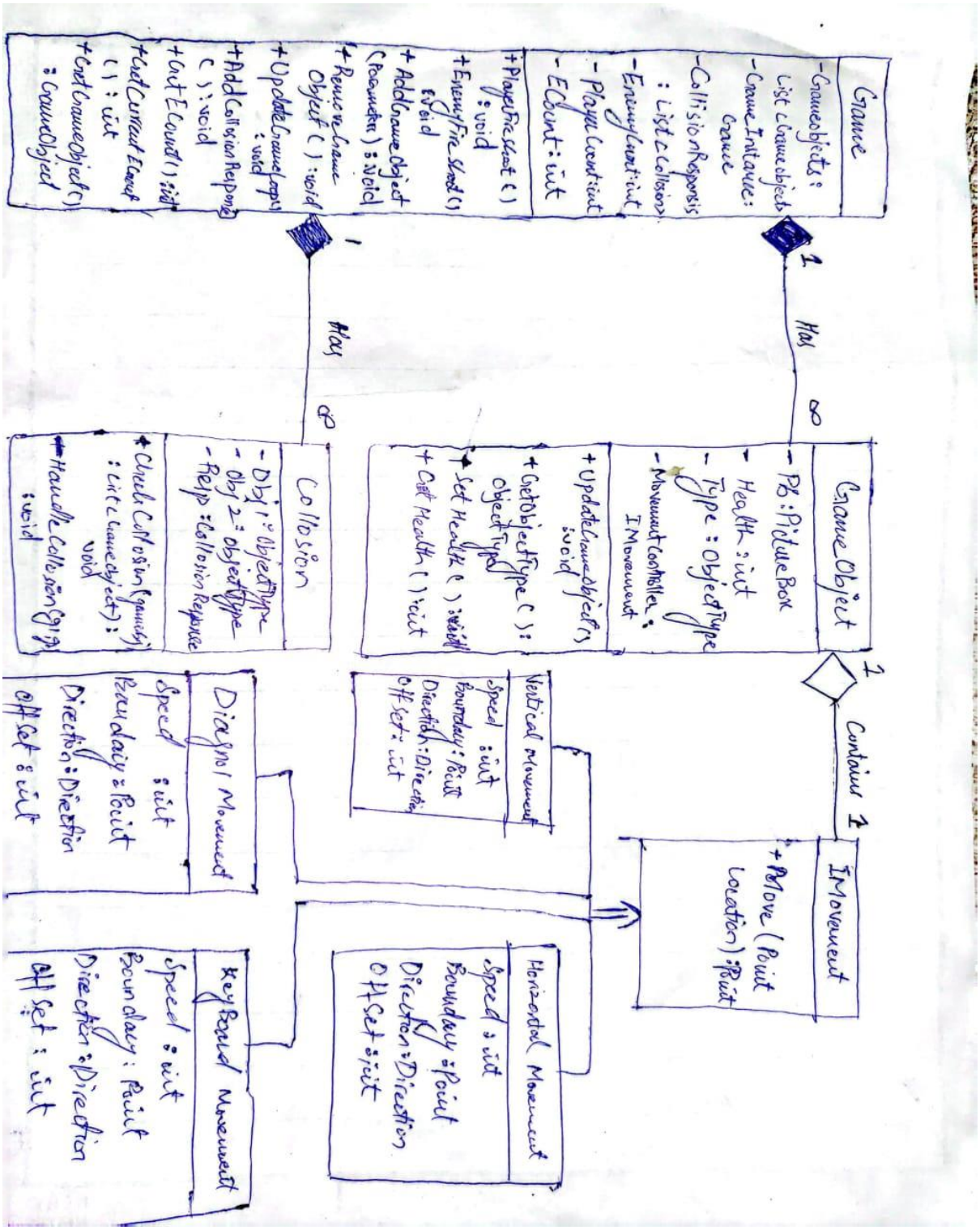


Fig.: Lost the Game

5. Class Diagram:



6. Framework Code:

6.1. Core:

- **Game:**

```
public class Game : IGame
{
    private Point boundary;
    private int offSetV;
    private int offSetH;
    private static List<Collision> collisionList = new List<Collision>();
    private static List<GameObject> gameObjList = new List<GameObject>();


    public static List<GameObject> GameObjList { get => gameObjList; set => gameObjList = value; }
    public static List<Collision> CollisionList { get => collisionList; set => collisionList = value; }
    public Point Boundary { get => boundary; set => boundary = value; }
    public int OffSetH { get => offSetH; set => offSetH = value; }
    public int OffSetV { get => offSetV; set => offSetV = value; }

    public event EventHandler OnBulletDel;
    public event EventHandler OnPlayerAdd;
    public event EventHandler OnProgressBarAdd;
    public event EventHandler OnPlayerDie;
    public event EventHandler OnPlayerCollideScore;
    public event EventHandler OnPlayerCollideHealth;
    public event EventHandler OnPlayerCollideEnemy;
    public event EventHandler OnPlayerScoreIncrease;
    public event EventHandler OnEnemyCollidePlayerBullet;
    public event EventHandler OnPlayerCollideEnemyBullet;
    public event EventHandler OnPlayerCollideBossBullet;

    public Game (Point boundary , int offSetV , int offSetH)
    {
        this.Boundary = boundary;
        this.offSetH = offSetH;
        this.offSetV = offSetV;
    }
    public void update ()
    {
        for (int i = 0 ; i < GameObjList.Count ; i++)
        {
            GameObjList[i].move();
            if (GameObjList[i].Type == objectTypes.player)
            {
                Player x = (Player)GameObjList[i];
                x.fireBullet();
                x.setPositionProgressBar();
                // x.rmvBullet(this);
            }
            if (GameObjList[i].Type == objectTypes.tank)
            {
                Player x = (Player)GameObjList[i];
                x.tankReset(Boundary , offSetH , offSetV);
```

```
                x.setPositionProgressBar();
            }
            if (GameObjList[i].Type == objectTypes.boss)
            {
                Player x = (Player)GameObjList[i];
                x.setPositionProgressBar();
            }
            detectCollision();


        }

    }
    public int numOfTanks ()
    {
        int count = 0;
        foreach (GameObject item in GameObjList)
        {
            if (item.Type == objectTypes.tank)
            {
                count++;
            }
        }
        return count;
    }
    public int numOfBoss ()
    {
        int count = 0;
        foreach (GameObject item in GameObjList)
        {
            if (item.Type == objectTypes.boss)
            {
                count++;
            }
        }
        return count;
    }
    public bool isWin ()
    {
        if (numOfBoss() == 0 && numOfTanks() == 0)
        {
            return true;
        }
        return false;
    }
    public bool checkPlayer ()
    {
        foreach (GameObject item in GameObjList)
        {
            if (item.Type == objectTypes.player)
            {
                return true;
            }
        }
        return false;
    }
    public bool isGameOver ()
    {
```

```
        if (!checkPlayer())
        {
            return true;
        }
        return false;
    }
    public bool isBossExist ()
    {
        foreach (GameObject item in GameObjList)
        {
            if (item.Type == objectTypes.boss)
            {
                return true;
            }
        }
        return false;
    }
    public void addGameObj (GameObject gO)
    {
        // GameObject gO = new GameObject(img , top , left , movement , type , mode
, height , width);
        if (gO.Type == objectTypes.player || gO.Type == objectTypes.tank || gO.Type
== objectTypes.boss)
        {
            Player x = (Player)gO;
            OnProgressBarAdd?.Invoke(x.HealthBar , EventArgs.Empty);
        }
        GameObjList.Add(gO);
        OnPlayerAdd?.Invoke(gO.Pb , EventArgs.Empty);
    }
    public void rmvGameObj (GameObject gO)
    {
        //GameObject gO = new GameObject(img , top , left , movement , type , mode ,
height , width);
        OnBulletDel?.Invoke(gO.Pb , EventArgs.Empty);
        GameObjList.Remove(gO);
    }
    public void RaisePlayerDieEvent (GameObject playerGameObject)
    {
        OnPlayerDie?.Invoke(playerGameObject.Pb , EventArgs.Empty);
    }
    public void RaiseOnPlayerCollideScoreEvent (GameObject gO)
    {
        OnPlayerCollideScore?.Invoke(gO.Pb , EventArgs.Empty);
        GameObjList.Remove(gO);
    }
    public void RaiseOnPlayerCollideHealthEvent (GameObject gO)
    {
        OnPlayerCollideHealth?.Invoke(gO.Pb , EventArgs.Empty);
        GameObjList.Remove(gO);
    }

    public void RaiseOnPlayerCollideEnemyBulletEvent (GameObject gO)
    {
        Player x = (Player)gO;
        if (x.HealthBar.Value >= 10)
        {
            x.HealthBar.Value -= 10;
```

```
        }
        else
        {
            OnPlayerCollideEnemyBullet?.Invoke(x , EventArgs.Empty);
            GameObjList.Remove(x);
        }
    }
    public void RaiseOnPlayerCollideBossBulletEvent (GameObject gO)
    {
        Player x = (Player)gO;
        if (x.HealthBar.Value >= 30)
        {
            x.HealthBar.Value -= 30;
        }
        else
        {
            OnPlayerCollideBossBullet?.Invoke(x , EventArgs.Empty);
            GameObjList.Remove(x);
        }
    }
    public void RaiseOnEnemyCollidePlayerBulletEvent (GameObject gO)
    {

        Player x = (Player)gO;
        if (x.HealthBar.Value >= 20)
        {
            x.HealthBar.Value -= 20;
            OnPlayerScoreIncrease?.Invoke(x , EventArgs.Empty);
        }
        else
        {
            foreach (GameObject blt in x.MyBullets)
            {
                GameObjList.Remove(blt);
            }
            OnEnemyCollidePlayerBullet?.Invoke(x , EventArgs.Empty);
            GameObjList.Remove(x);
        }
    }
    public void RaiseOnEnemyBossCollidePlayerBulletEvent (GameObject gO)
    {

        Player x = (Player)gO;
        if (x.HealthBar.Value >= 5)
        {
            x.HealthBar.Value -= 5;
            OnPlayerScoreIncrease?.Invoke(x , EventArgs.Empty);
        }
        else
        {
            foreach (GameObject blt in x.MyBullets)
            {
                GameObjList.Remove(blt);
            }
            OnEnemyCollidePlayerBullet?.Invoke(x , EventArgs.Empty);
            GameObjList.Remove(x);
        }
    }
```

```
    public void RaiseOnPlayerCollideEnemyEvent (GameObject gO)
    {

        Player x = (Player)gO;
        if (x.HealthBar.Value >= 15)
        {
            x.HealthBar.Value -= 15;
            x.Pb.Left = (boundary.X / 2);
            x.Pb.Top = boundary.Y - x.Pb.Height;
        }
        else
        {
            OnPlayerCollideEnemy?.Invoke(x , EventArgs.Empty);
            GameObjList.Remove(x);
        }
    }

    public void detectCollision ()
    {
        for (int x = 0 ; x < gameObjList.Count ; x++)
        {
            for (int y = 0 ; y < gameObjList.Count ; y++)
            {
                try
                {
                    if
(gameObjList[x].Pb.Bounds.IntersectsWith(gameObjList[y].Pb.Bounds))
                    {
                        foreach (Collision c in collisionList)
                        {
                            if (gameObjList[x].Type == c.G1 && gameObjList[y].Type ==
c.G2)
                            {
                                c.Behaviour.performPlayerAction(this               ,
gameObjList[x] , gameObjList[y]);
                                break;
                            }
                        }
                    }
                }
                catch(Exception e) { }

            }
        }
    }
    public void addCollision (Collision c)
    {
        CollisionList.Add(c);
    }
      }
```

- **GameObject:**

```
public class GameObject
 {
     private PictureBox pb;
     private IMovement movement;
     private objectTypes type;
```

```
    public GameObject (Image img , int top , int left , IMovement movement ,
objectTypes type , PictureBoxSizeMode mode , int height , int width)
    {
        Pb = new PictureBox();
        Pb.Image = img;
        Pb.BackColor = Color.Transparent;
        Pb.Height = height;
        Pb.Width = width;
        Pb.Top = top;
        Pb.Left = left;
        Pb.SizeMode = mode;
        this.Movement = movement;
        this.type = type;
    }

    public IMovement Movement { get => movement; set => movement = value; }
    public PictureBox Pb { get => pb; set => pb = value; }
    public objectTypes Type { get => type; set => type = value; }

    public void move ()
    {
        Pb.Location = Movement.move(Pb.Location);
    }

     }
```

- **Player:**

```
public class Player : GameObject
 {
    private ProgressBar healthBar;
    private int barLeft;
    private int barTop;
    private List<Bullet> myBullets;
    private Image bulletImg;
    public Player (Image bulletImg , Image img , int top , int left , IMovement
movement , objectTypes type , PictureBoxSizeMode mode , int height , int width) :
base(img , top , left , movement , type , mode , height , width)
    {
        HealthBar = new ProgressBar();
        this.MyBullets = new List<Bullet>();
        this.bulletImg = bulletImg;
    }

    public List<Bullet> MyBullets { get => myBullets; set => myBullets = value; }
    public Image BulletImg { get => bulletImg; set => bulletImg = value; }
    public ProgressBar HealthBar { get => healthBar; set => healthBar = value; }

    public void createHealthBar (int left , int top)
    {
        barLeft = left;
        barTop = top;
        HealthBar.Value = 100;
        HealthBar.Top = top;
        HealthBar.Left = left;
        HealthBar.Height /= 2;
    }
```

```
    public void setPositionProgressBar ()
    {
        HealthBar.Left = base.Pb.Left - 10;
        HealthBar.Top = base.Pb.Top + base.Pb.Height;
    }
    public void createBullet (Game g , IMovement movement , int top , int left ,
objectTypes type)
    {
        Bullet gO = new Bullet(BulletImg , top , left , movement , type ,
PictureBoxSizeMode.StretchImage , bulletImg.Height / 2 , bulletImg.Width / 2);
        MyBullets.Add(gO);
        g.addGameObj(gO);
    }
    public void fireBullet ()
    {
        foreach (Bullet b in MyBullets)
        {
            b.move();
        }
    }
    public void rmvBullet (Game g)
    {
        for (int i = 0 ; i < myBullets.Count ; i++)
        {
            if      ((myBullets[i].Pb.Location.Y      >      g.Boundary.Y      &&
myBullets[i].Movement.GetType() == typeof(Gravity)) || (myBullets[i].Pb.Location.Y <=
0 && myBullets[i].Movement.GetType() == typeof(GravityInvert)))
            {
                g.rmvGameObj(myBullets[i]);
                this.myBullets.Remove(myBullets[i]);

            }
        }
    }
    public void tankReset (Point Boundary , int offSetH , int offSetV)
    {
        if (base.Pb.Location.Y >= Boundary.Y)
        {
            Random rand = new Random();
            this.Pb.Top = 0;
            this.Pb.Left = rand.Next(offSetH , Boundary.X - offSetH);
        }
    }
      }
```

- **IGame:**

```
public interface IGame
{
    // void RaisePlayerHitEvent (GameObject obj);
    void RaisePlayerDieEvent (GameObject obj);
    void RaiseOnPlayerCollideScoreEvent (GameObject obj);
    void RaiseOnPlayerCollideEnemyBulletEvent (GameObject obj);
    void RaiseOnEnemyCollidePlayerBulletEvent (GameObject obj);
    void RaiseOnPlayerCollideEnemyEvent (GameObject obj);
    void RaiseOnPlayerCollideBossBulletEvent (GameObject obj);
    void RaiseOnEnemyBossCollidePlayerBulletEvent (GameObject obj);
    void RaiseOnPlayerCollideHealthEvent (GameObject obj);
```

```
    void rmvGameObj (GameObject gO);
         }
```

6.2. Enum:

- **objectTypes:**

```
public enum objectTypes
 {

player,tank,road,stairs,playerfire,tankFire,healthPill,scorePill,livesPill,boss,boss
Fire,
          }
```

6.3. Firing:

- **Bullet:**

```
public class Bullet : GameObject
 {
     public Bullet (Image img , int top , int left , IMovement movement , objectTypes
type , PictureBoxSizeMode mode , int height , int width) : base(img , top , left ,
movement , type , mode , height , width)
     {
     }

          }
```

6.4. HealthSystem:

- **ProgBar:**

```
public class ProgBar : GameObject
{
    private ProgressBar healthBar;

    public ProgBar (Image img , int top , int left , IMovement movement , objectTypes
type , PictureBoxSizeMode mode , int height , int width) : base(img , top , left ,
movement , type , mode , height , width)
    {
        healthBar = new ProgressBar();
        healthBar.Value = 100;
        healthBar.Top = top;
        healthBar.Left = left;
        healthBar.Height /= 2;

    }

    public ProgressBar HealthBar { get => healthBar; set => healthBar = value; }


          }
```

6.5. Movement:

- **Gravity:**

```
public class Gravity : IMovement
 {
```

```
    private int speed;
    public Gravity (int speed)
    {
        this.speed = speed;
    }
    public Point move (Point location)
    {
        location.Y += speed;
        return location;
    }
}
```

- **GravityInvert:**

```
public class GravityInvert : IMovement
 {
    private int speed;
    public GravityInvert (int speed)
    {
        this.speed = speed;
    }
    public Point move (Point location)
    {
        location.Y -= speed;
        return location;
    }
      }
```

- **Horizontal:**

```
public class Horizontal : IMovement
{

    private string direction;
    private Point boundary;
    private int speed;
    private int offSet;
    public Horizontal (int speed , Point boundary , string direction , int offSet)
    {
        this.speed = speed;
        this.boundary = boundary;
        this.direction = direction;
        this.offSet = offSet;
    }
    public Point move (Point location)
    {
        if (location.X <= 0)
        {
            direction = "Right";
        }
        else if (location.X + offSet >= boundary.X)
        {
            direction = "Left";
        }
        if (direction == "Left")
        {
            location.X -= speed;
        }
```

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

- **Vertical:**

```
public class Vertical : IMovement
{
    private string direction;
    private Point boundary;
    private int speed;
    private int offSet;
    public Vertical (int speed , Point boundary , string direction , int offSet)
    {
        this.speed = speed;
        this.boundary = boundary;
        this.direction = direction;
        this.offSet = offSet;
    }
    public Point move (Point location)
    {
        if (location.Y + offSet <= 0)
        {
            direction = "Down";
        }
        else if (location.Y + offSet < boundary.Y)
        {
            direction = "Up";
        }
        if (direction == "Up")
        {
            location.Y -= speed;
        }
        else if (direction == "Down")
        {
            location.Y += speed;
        }
        return location;
    }
          }
```

- **Keyboard:**

```
public class Keyboard : IMovement
 {
     private int speed;
     private int offSetH;
     private int offSetV;
     private System.Drawing.Point boundary;
     public Keyboard (int speed , System.Drawing.Point boundary , int offSetH , int
offSetV)
     {
         this.speed = speed;
         this.boundary = boundary;
         this.offSetH = offSetH;
```

```
        this.offSetV = offSetV;
    }
    public System.Drawing.Point move (System.Drawing.Point location)
    {
        if            (EZInput.Keyboard.IsKeyPressed(Key.RightArrow)            ||
EZInput.Keyboard.IsKeyPressed(Key.D))
        {
            if (location.X + offSetH < boundary.X)
            {
                location.X += speed;
            }
        }
        if             (EZInput.Keyboard.IsKeyPressed(Key.LeftArrow)            ||
EZInput.Keyboard.IsKeyPressed(Key.A))
        {
            if (location.X - offSetH > 0)
            {
                location.X -= speed;
            }
        }
        if              (EZInput.Keyboard.IsKeyPressed(Key.UpArrow)             ||
EZInput.Keyboard.IsKeyPressed(Key.W))
        {
            if (location.Y > 0)
            {
                location.Y -= speed;
            }
        }
        if             (EZInput.Keyboard.IsKeyPressed(Key.DownArrow)            ||
EZInput.Keyboard.IsKeyPressed(Key.S))
        {
            if (location.Y + offSetV < boundary.Y)
            {
                location.Y += speed;
            }

        }
        return location;
    }
      }
```

- **IMovement:**

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

6.6. Collisions:

- **BossBulletCollision:**

```
public class BossBulletCollision : ICollisionAction
{
    public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
    {
        GameObject chr;
        GameObject bullet;
```

```
        if (source1.Type == Enum.objectTypes.bossFire)
        {
            chr = source2;
            bullet = source1;
        }
        else
        {
            chr = source1;
            bullet = source2;
        }
        game.rmvGameObj(bullet);
        game.RaiseOnPlayerCollideBossBulletEvent(chr);
    }
          }
```

- **Collision:**

```
public class Collision
{
    private objectTypes g1;
    private objectTypes g2;
    private ICollisionAction behaviour;

    public Collision (objectTypes g1 , objectTypes g2 , ICollisionAction behaviour)
    {
        this.G1 = g1;
        this.G2 = g2;
        this.Behaviour = behaviour;
    }

    public objectTypes G1 { get => g1; set => g1 = value; }
    public objectTypes G2 { get => g2; set => g2 = value; }
    internal ICollisionAction Behaviour { get => behaviour; set => behaviour = value;
}
```

- **EnemyBulletCollision:**

```
public class EnemyBulletCollision : ICollisionAction
 {
     public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
     {
         GameObject chr;
         GameObject bullet;
         if (source1.Type == Enum.objectTypes.player)
         {
             chr = source1;
             bullet = source2;
         }
         else
         {
             chr = source2;
             bullet = source1;
         }
         game.rmvGameObj(bullet);
         game.RaiseOnPlayerCollideEnemyBulletEvent(chr);
```

```
     }
          }
```

- **PlayerBulletCollision:**

```
public class PlayerBulletCollision : ICollisionAction
{
    public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
    {
        GameObject chr;
        GameObject bullet;
        if (source1.Type == Enum.objectTypes.tank)
        {
            chr = source1;
            bullet = source2;
        }
        else
        {
            chr = source2;
            bullet = source1;
        }
        game.rmvGameObj(bullet);
        game.RaiseOnEnemyCollidePlayerBulletEvent(chr);
    }
}
```

- **PlayerBulletCollisionBoss:**

```
public  class PlayerBulletCollisionBoss : ICollisionAction
  {
      public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
      {
          GameObject chr;
          GameObject bullet;
          if (source1.Type == Enum.objectTypes.boss)
          {
              chr = source1;
              bullet = source2;
          }
          else
          {
              chr = source2;
              bullet = source1;
          }
          game.rmvGameObj(bullet);
          game.RaiseOnEnemyBossCollidePlayerBulletEvent(chr);
      }
  }
```

- **PlayerCollision:**

```
  public class PlayerCollision : ICollisionAction
  {
      public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
      {
```

```
          GameObject player;
          if (source1.Type == Enum.objectTypes.player)
          {
              player = source1;
          }
          else
          {
              player = source2;
          }
          game.RaiseOnPlayerCollideEnemyEvent(player);
      }
  }
```

- **PlayerCollisionHealth:**

```
  public class PlayerCollisionHealth : ICollisionAction
  {
      public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
      {
          GameObject player;
          if (source1.Type == Enum.objectTypes.healthPill)
          {
              player = source1;
          }
          else
          {
              player = source2;
          }
          game.RaiseOnPlayerCollideHealthEvent(player);
      }
  }
```

- **PlayerCollisionScore:**

```
public class PlayerCollisionScore : ICollisionAction
{
    public void performPlayerAction (IGame game , GameObject source1 , GameObject
source2)
    {
        GameObject player;
        if (source1.Type == Enum.objectTypes.scorePill)
        {
            player = source1;
        }
        else
        {
            player = source2;
        }
        game.RaiseOnPlayerCollideScoreEvent(player);
    }
}
```

- **ICollisionAction:**

```
public interface ICollisionAction
{
    void performPlayerAction (IGame game , GameObject g1 , GameObject g2);
```

```
}
```

7. Conclusion:

Escape Plan represents the culmination of creativity, technical expertise, and a passion for gaming development. With its dynamic gameplay, robust framework, and engaging features such as player control, enemy encounters, and a comprehensive health, lives, and score system, Escape Plan offers an immersive gaming experience that captivates players of all skill levels.

As the developer behind Escape Plan, I am committed to continuous improvement and innovation. In the future, I envision enhancing this game even further by introducing new levels, challenging obstacles, and additional gameplay mechanics. From refining the graphics to introducing multiplayer functionality or expanding the storyline, the possibilities for improvement are endless.