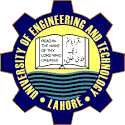
**Bullet Fest**



**Session 2023 – 2027**

**Submitted by:**

Usman Ali Ashraf 2023-CS-106

**Supervised by:**

Laeeq u Zaman Khan Niazi

Mam Maida Shahid

**Course:**

CSC-103 Object Oriented Programming

**Department of Computer Science**

**University of Engineering and Technology**

**Lahore Pakistan**

Contents

[1.1 About Freedom Fodge 2](#_Toc166037680)

[1.2 Description about characters: 2](#_Toc166037681)

[1.2.1 Player (Chichi Michi): 2](#_Toc166037682)

[1.2.2 Enemies (Zuca , Muca , Zamato ): 2](#_Toc166037683)

[1.2.3 Fires: 2](#_Toc166037684)

[1.3 OOP Concepts: 2](#_Toc166037685)

[1.3.1 Inheritance: 2](#_Toc166037686)

[1.3.2 Polymorphism: 2](#_Toc166037687)

[1.3.3 Interface: 2](#_Toc166037688)

[1.3.4 Association: 2](#_Toc166037689)

[1.3.5 Encapsulation 3](#_Toc166037690)

[1.4 Classes: 3](#_Toc166037691)

[1.5 Enums: 3](#_Toc166037692)

[1.5 Game Rules: 3](#_Toc166037693)

[1.6 WireFrames: 4](#_Toc166037694)

[1.6.1 Startup Page 4](#_Toc166037695)

[1.6.2 Game Page 4](#_Toc166037696)

[1.7 CRC Diagram: 5](#_Toc166037697)

[1.8 Code: 5](#_Toc166037698)

[1.8.1 Maze.cs 5](#_Toc166037699)

[1.8.2 Player.cs 6](#_Toc166037700)

[1.8.3 Enemy.cs 13](#_Toc166037701)

[1.8.4 HorizontalEnemy.cs 14](#_Toc166037702)

[1.8.5 VerticalEnemy.cs 17](#_Toc166037703)

[1.8.6 RandomEnemy.cs 20](#_Toc166037704)

[1.8.7 PlayerCollision.cs 22](#_Toc166037705)

[1.8.8 EnemyCollision.cs 24](#_Toc166037706)

[1.8.9 ICollision.cs 26](#_Toc166037707)

[1.8.10 IMovement.cs 26](#_Toc166037708)

[1.8.11 EnemyDirection.cs 26](#_Toc166037709)

[1.8.12 Gravity.cs 27](#_Toc166037710)

[1.8.13 GameObject.cs 28](#_Toc166037711)

[1.8.14 Fire.cs 28](#_Toc166037712)

[1.8.15 Game.cs 30](#_Toc166037713)

# **1.1 About Bullet Fest**

Dive into the shadows of confinement with **Bullet Fest** an enthralling prison escape game that pushes the boundaries of your intellect and resourcefulness. Unjustly confined within the unforgiving walls of a mysterious maximum-security facility, you must unravel the enigma surrounding your imprisonment and orchestrate a daring escape.

# **1.2 Description about characters:**

## **1.2.1 Player:**

Chichi Michi can move in four directions: up, down, left, and right. The player's Health increases when he collides with enemies or got collision with enemy fires. The score of him is increased when his fire collides with Enemy;

## **1.2.2 Enemies:**

There are total 7 enemies in my game. The enemy moves in horizontal, vertical, random motion. movement if Jerry is far. It act as a obstacle and uses as health drainage.

## **1.2.3 Fires:**

Fires move downward or in right direction within the maze, simulating a projectile. Fires collide with maze obstacles and may affect Chichi Michi's health.

* Top of Form

# **OOP Concepts:**

All the OOP Pillars are included in this game:

## **1.3.1 Inheritance:**

Inheritance is implemented in interfaces where an interface is inherited by other such as Vertical Movement Class inherits IMovement interface similarly PlayerVSEnemeyCollision Class implements ICollision Class. Moreover, all types of enemies are inherited by class Enemy.

## **1.3.2 Polymorphism:**

Polymorphism is implemented in some functions of Game Class where two different types of functions like movement function of parent class Enemy.It is also implemented in the form of other functions in other classes.

## **1.3.3 Interface:**

Two types of interfaces are implemented one is IMovement for movement class and other one is ICollision which detects the type of collisions.

## **1.3.4 Association:**

The classes are associated with each other in such a way that

* The relation between class “Game” and class “Player” is Composition because if there will be no game, there will be no player.
* The relation between class “Game” and class “Maze” is Composition because if there will be no game, there will be no maze.
* The relation between class “Game” and class “Enemy” is Composition because if there will be no game, there will be no enemies.
* The relation between class “Player” and class “Fire” is Composition because if there will be no player, there will be no fire.

## **Encapsulation**

In this Project, Encapsulation is used in all Bl Classes and some Dl classes also by making their attributes private.

# **1.4 Classes:**

There are total eight classes which have been implemented in this Project.

* Game
* Player
* Enemy
* Horizontal Enemy
* Vertical Enemy
* Random Enemy
* Maze
* EnemyCollision
* PlayerCollision
* Fire

# **1.5 Enums:**

There are total two classes which have been implemented in this Project.

* EnemyDirection
* ObjectType

# **1.5 Game Rules:**

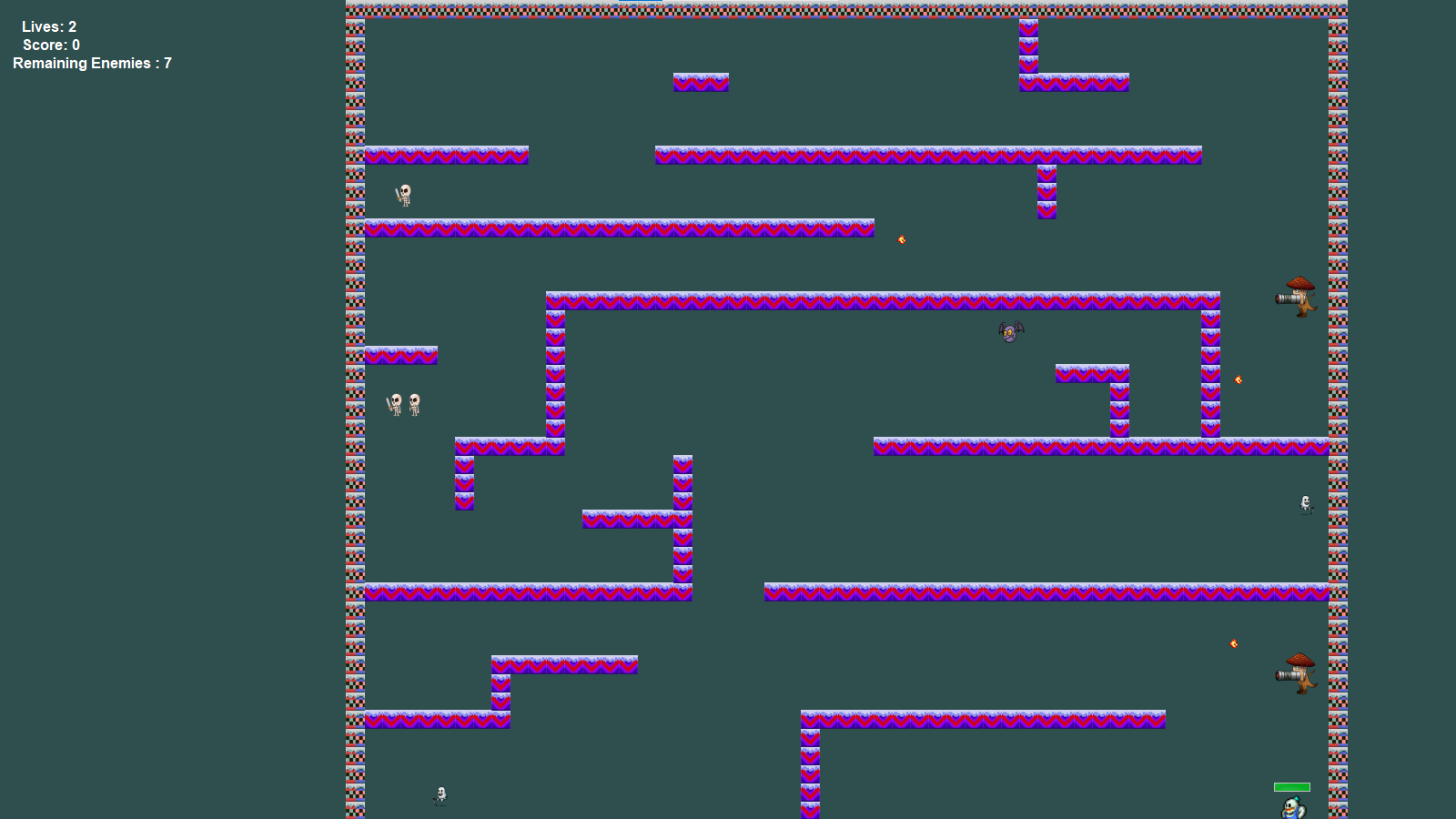
1. If Player Collides with the enemy it will decrease the player health.
2. If Player bullet hits the enemy it will kill the enemy.
3. Player has to kill all the enemies to win the game.
4. If player health reduces to zero without killing the enemy it will die and lose the game.
5. Some enemies require more than one bullet to kill them.

# **1.6 Wireframes:**

## **1.6.1 Startup Page**



## **1.6.2 Game Page**



# **1.7 CRC Diagram:**



# **1.8 Code:**

## **1.8.1 Maze.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.IO;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class Maze  {  private char[,] maze;  private int numRows;  private int numColumns;  private Form form;  private List<PictureBox> BoundaryPictureBoxes = new List<PictureBox>();  private List<PictureBox> InnerPictureBoxes = new List<PictureBox>();  public List<PictureBox> GetBoundaryPictureBoxes()  {  return BoundaryPictureBoxes;  }  public List<PictureBox> GetInnerPictureBoxes()  { return InnerPictureBoxes; }  public Maze(Form form, string filePath, char targetedChar, string imagePath, char targetedChar2, string imagePath2)  {  this.form = form;  DisplayMaze(filePath,targetedChar,imagePath,targetedChar2,imagePath2);  }  private void LoadMaze(string filePath)  {  string[] lines = File.ReadAllLines(filePath);  numRows = lines.Length;  numColumns = lines[0].Length;  maze = new char[numRows, numColumns];  for (int i = 0; i < numRows; i++)  {  for (int j = 0; j < numColumns; j++)  {  maze[i, j] = lines[i][j];  }  }  }  public void DisplayMaze(string filePath, char targetedChar, string imagePath, char targetedChar2, string imagePath2)  {  LoadMaze(filePath);  int pictureBoxWidth = 20;  int pictureBoxHeight = 20;  for (int i = 0; i < numRows; i++)  {  for (int j = 0; j < numColumns; j++)  {  if (maze[i, j] == targetedChar)  {  PictureBox pictureBox = new PictureBox();  pictureBox.Load(imagePath);  pictureBox.Width = pictureBox.Image.Width;  pictureBox.Height = pictureBox.Image.Height;  pictureBox.Location = new System.Drawing.Point(j \* pictureBoxWidth, i \* pictureBoxHeight);  form.Controls.Add(pictureBox);  BoundaryPictureBoxes.Add(pictureBox);  }  if (maze[i, j] == targetedChar2)  {  PictureBox pictureBox = new PictureBox();  pictureBox.Load(imagePath2);  pictureBox.Width = pictureBox.Image.Width;  pictureBox.Height = pictureBox.Image.Height;  pictureBox.Location = new System.Drawing.Point(j \* pictureBoxWidth, i \* pictureBoxHeight);  form.Controls.Add(pictureBox);  BoundaryPictureBoxes.Add(pictureBox);  }  }  }  }  }  } |

## **1.8.2 Player.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.IO;  using System.Windows.Forms;  using FreedomFodgeFrameWork.Interfaces;  namespace FreedomFodgeFrameWork  {  public class Player :IMovement  {  private Form form;  private int PlayerX;  private int PlayerY;  private string LeftImage1;  private string LeftImage2;  private string RightImage1;  private string RightImage2;  private string UpImage;  private string DownImage;  private string FireImage;  private PictureBox pictureBox;  private int PlayerSpeed;  private bool isImage1Visible = true;  private Game game;  private List<Fire> PlayerFires = new List<Fire>();  private int PlayerHealth;  private int BulletSpeed;  private int PlayerLives;  private ProgressBar playerHealthBar;  private int InitialHealth;  private Label playerLivesLabel;  public Player(Form form, Game game, int PlayerX, int PlayerY, int Speed, System.Drawing.Point Boundary, string LeftImage1, string LeftImage2, string RightImage1, string RightImage2, string UpImage, string DownImage, string FireImage, int playerHealth, int BulletSpeed, int PlayerLives)  {  this.form = form;  this.PlayerX = PlayerX;  this.PlayerY = PlayerY;  this.PlayerSpeed = Speed;  this.FireImage = FireImage;  this.LeftImage1 = LeftImage1;  this.LeftImage2 = LeftImage2;  this.RightImage1 = RightImage1;  this.RightImage2 = RightImage2;  this.UpImage = UpImage;  this.DownImage = DownImage;  this.game = game;  this.PlayerHealth = playerHealth;  this.InitialHealth = playerHealth;  this.BulletSpeed = BulletSpeed;  this.PlayerLives = PlayerLives;  DisplayPlayer(Boundary);  Movement();  InitializePlayerHealthBar();  InitializePlayerLivesLabel();  }  ICollision PlayerCollision = new PlayerCollision();  public int GetPlayerX() { return PlayerX; }  public int GetPlayerY() { return PlayerY; }  public int GetPlayerHealth() { return PlayerHealth; }  public void SetPlayerHealth(int health) { PlayerHealth = health; }  public void SetPlayerLives(int Lives) { PlayerLives = Lives; }  public PictureBox GetPictureBox() { return pictureBox; }  public List<Fire> GetPlayerFires()  {  return this.PlayerFires;  }  private void DisplayPlayer(System.Drawing.Point Boundary)  {  Image image = Image.FromFile(LeftImage1);  int pictureBoxWidth = image.Width;  int pictureBoxHeight = image.Height;  PlayerX = Math.Max(0, Math.Min(PlayerX, Boundary.X - pictureBoxWidth));  PlayerY = Math.Max(0, Math.Min(PlayerY, Boundary.Y - pictureBoxHeight));  pictureBox = new PictureBox();  pictureBox.Width = pictureBoxWidth + 10;  pictureBox.Height = pictureBoxHeight + 10;  pictureBox.Location = new System.Drawing.Point(PlayerX, PlayerY);  pictureBox.SizeMode = PictureBoxSizeMode.Zoom;  pictureBox.BackColor = Color.Transparent;  form.Controls.Add(pictureBox);  ShowImage(LeftImage1);  }  public void Movement()  {  form.KeyDown += HandleKeyDown;  form.KeyUp += HandleKeyUp;  }  private void HandleKeyDown(object sender, KeyEventArgs e)  {  switch (e.KeyCode)  {  case Keys.Left:  MoveLeft();  break;  case Keys.Right:  MoveRight();  break;  case Keys.Up:  MoveUp();  break;  case Keys.Down:  MoveDown();  break;  case Keys.Space:  GeneratePlayerFires();  RemoveInvisibleFires();  break;  }  }  private void RemoveInvisibleFires()  {  PlayerFires.RemoveAll(fire => !fire.IsVisible);  }  private void GeneratePlayerFires()  {  int fireX = pictureBox.Left + (pictureBox.Width / 2);  int fireY = pictureBox.Top + (pictureBox.Height / 2);  if (isImage1Visible)  {  PlayerFires.Add(new Fire("left", form, pictureBox, FireImage, fireX, fireY, game, BulletSpeed));  }  else  {  PlayerFires.Add(new Fire("right", form, pictureBox, FireImage, fireX, fireY, game, BulletSpeed));  }  }  private void HandleKeyUp(object sender, KeyEventArgs e)  {  switch (e.KeyCode)  {  case Keys.Up:  ShowDefaultImage();  break;  case Keys.Down:  ShowDefaultImage();  break;  }  }  public void LivesDecrement()  {  if (PlayerHealth <= 0)  {  PlayerHealth = InitialHealth;  PlayerLives--;  playerLivesLabel.Text = "Lives: " + PlayerLives;  if (PlayerLives == 0)  {  form.Hide();  }  }  }  private void MoveLeft()  {  ShowImage(LeftImage2);  Timer timer = new Timer();  timer.Interval = 40;  timer.Tick += (sender, e) =>  {  PlayerX -= PlayerSpeed;  playerHealthBar.Location = new Point(PlayerX, PlayerY - playerHealthBar.Height);  if (PlayerCollision.BoundaryWallCollision(game))  {  PlayerX += PlayerSpeed;  }  if (PlayerCollision.PlayerEnemyCollision(game))  {  PlayerX += PlayerSpeed;    }  UpdatePlayerPosition();  ShowImage(LeftImage1);  timer.Stop();  timer.Dispose();  };  timer.Start();  }  private void InitializePlayerHealthBar()  {  playerHealthBar = new ProgressBar();  playerHealthBar.Minimum = 0;  playerHealthBar.Maximum = Math.Max(1, PlayerHealth);  playerHealthBar.Value = PlayerHealth;  playerHealthBar.Width = pictureBox.Width;  playerHealthBar.Height = 10;  playerHealthBar.BackColor = Color.Red;  playerHealthBar.ForeColor = Color.Green;  playerHealthBar.Style = ProgressBarStyle.Continuous;  playerHealthBar.Visible = true;  playerHealthBar.Location = new Point(PlayerX, PlayerY - playerHealthBar.Height);  form.Controls.Add(playerHealthBar);  }  public void UpdatePlayerHealth(int newHealth)  {  PlayerHealth = newHealth;  playerHealthBar.Value = PlayerHealth;  }  private void MoveRight()  {  ShowImage(RightImage2);  Timer timer = new Timer();  timer.Interval = 40;  timer.Tick += (sender, e) =>  {  PlayerX += PlayerSpeed;  playerHealthBar.Location = new Point(PlayerX, PlayerY - playerHealthBar.Height);  if (PlayerCollision.BoundaryWallCollision(game))  {  PlayerX -= PlayerSpeed;  }  if (PlayerCollision.PlayerEnemyCollision(game))  {  PlayerX -= PlayerSpeed;    }  UpdatePlayerPosition();  ShowImage(RightImage1);  timer.Stop();  timer.Dispose();  };  timer.Start();  }  private void MoveUp()  {  ShowImage(UpImage);  Timer timer = new Timer();  timer.Interval = 40;  timer.Tick += (sender, e) =>  {  PlayerY -= PlayerSpeed;  playerHealthBar.Location = new Point(PlayerX, PlayerY - playerHealthBar.Height);  if (PlayerCollision.BoundaryWallCollision(game))  {  PlayerY += PlayerSpeed;  }  if (PlayerCollision.PlayerEnemyCollision(game))  {  PlayerY += PlayerSpeed;    }  UpdatePlayerPosition();  timer.Stop();  timer.Dispose();  };  timer.Start();  }  private void MoveDown()  {  ShowImage(DownImage);  Timer timer = new Timer();  timer.Interval = 40;  timer.Tick += (sender, e) =>  {  PlayerY += PlayerSpeed;  playerHealthBar.Location = new Point(PlayerX, PlayerY - playerHealthBar.Height);  if (PlayerCollision.BoundaryWallCollision(game))  {  PlayerY -= PlayerSpeed;  }  if (PlayerCollision.PlayerEnemyCollision(game))  {  PlayerY -= PlayerSpeed;    }  UpdatePlayerPosition();  timer.Stop();  timer.Dispose();  };  timer.Start();  }  private void ShowDefaultImage()  {  if (isImage1Visible)  ShowImage(LeftImage1);  else  ShowImage(RightImage1);  }  private void InitializePlayerLivesLabel()  {  playerLivesLabel = new Label();  playerLivesLabel.Font = new Font("Arial", 12, FontStyle.Bold);  playerLivesLabel.ForeColor = Color.Black;  playerLivesLabel.BackColor = Color.White;  playerLivesLabel.AutoSize = true;  playerLivesLabel.Location = new Point(form.ClientSize.Width - playerLivesLabel.Width - 10, 10); // Adjust location as needed  UpdatePlayerLivesLabel();  form.Controls.Add(playerLivesLabel);  }  private void UpdatePlayerLivesLabel()  {  playerLivesLabel.Text = "Lives: " + PlayerLives;  }  private void UpdatePlayerPosition()  {  int pictureBoxWidth = pictureBox.Width;  int pictureBoxHeight = pictureBox.Height;  PlayerX = Math.Max(0, Math.Min(PlayerX, form.ClientSize.Width - pictureBoxWidth));  PlayerY = Math.Max(0, Math.Min(PlayerY, form.ClientSize.Height - pictureBoxHeight));  pictureBox.Location = new System.Drawing.Point(PlayerX, PlayerY);  }  private void ShowImage(string imagePath)  {  pictureBox.Image = Image.FromFile(imagePath);  isImage1Visible = (imagePath == LeftImage1 || imagePath == LeftImage2);  }  }  } |

## **1.8.3 Enemy.cs**

|  |
| --- |
| using FreedomFodgeFrameWork.Interfaces;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class Enemy :IMovement  {  protected string imagePathLeft;  protected string imagePathRight;  protected Form form;  protected int xPosition;  protected int yPosition;  protected int EnemyHealth;  protected PictureBox pictureBox;  protected Random random;  protected Timer movementTimer;  protected int EnemySpeed;  protected Game game;  protected ICollision EnemyCollisions;  protected List<Fire> EnemyFires = new List<Fire>();  private Label RemainingEnemiesLabel;  public Enemy(Game game, int EnemySpeed, Form form, int xPosition, int yPosition,int EnemyHealth)  {  this.form = form;  this.xPosition = xPosition;  this.yPosition = yPosition;  this.EnemySpeed = EnemySpeed;  this.EnemyHealth = EnemyHealth;  this.game = game;  random = new Random();  EnemyCollisions = new EnemyCollision();  }  public int GetEnemyHealth()  {  return EnemyHealth;  }  public void SetEnemyHealth(int EnemyHealth)  {  this.EnemyHealth=EnemyHealth;  }  public PictureBox GetPictureBox()  {  return pictureBox;  }  public virtual void Movement()  {  }  }  } |

## **1.8.4 HorizontalEnemy.cs**

|  |
| --- |
| using FreedomFodgeFrameWork.Interfaces;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class HorizontalEnemy : Enemy  {  private EnemyDirection direction;  private string FireImage;  private List<PictureBox> pictureBoxes;  private long lastBulletTime;  public HorizontalEnemy(string imagePathLeft, string imagePathRight, Game game, string FireImage, int EnemySpeed, Form form, int xPosition, int yPosition,int EnemyHealth)  : base(game, EnemySpeed, form, xPosition, yPosition,EnemyHealth)  {  this.imagePathLeft = imagePathLeft;  this.imagePathRight = imagePathRight;  this.pictureBox = new PictureBox();  this.pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;  this.pictureBox.Width = 30;  this.FireImage = FireImage;  this.pictureBox.Height = 30;  pictureBox.BackColor = Color.Transparent;  direction = EnemyDirection.Left;  form.Controls.Add(this.pictureBox);  DisplayEnemy();  movementTimer = new Timer();  movementTimer.Interval = 50;  movementTimer.Tick += MovementTimer\_Tick;  movementTimer.Start();  pictureBoxes = new List<PictureBox>();  lastBulletTime = 0;  }  private ICollision Collision = new EnemyCollision();  public void DisplayEnemy()  {  if (pictureBox != null)  {  pictureBox.ImageLocation = imagePathLeft;  pictureBox.Location = new System.Drawing.Point(xPosition, yPosition);  }  }  private void MovementTimer\_Tick(object sender, EventArgs e)  {  Movement();  long currentTime = DateTime.Now.Ticks;  long elapsedTime = currentTime - lastBulletTime;  if (elapsedTime >= TimeSpan.FromSeconds(2).Ticks)  {  GenerateBullet();  lastBulletTime = currentTime;  }  }  public void GenerateBullet()  {  PictureBox bullet = new PictureBox();  bullet.SizeMode = PictureBoxSizeMode.StretchImage;  bullet.Width = 45;  bullet.Height = 45;  bullet.BackColor = Color.Transparent;  bullet.ImageLocation = FireImage;  bullet.Location = new Point(this.pictureBox.Left + (this.pictureBox.Width / 2), this.pictureBox.Bottom);  form.Controls.Add(bullet);  pictureBoxes.Add(bullet);  MoveFire(bullet);  }  public void MoveFire(PictureBox bullet)  {  Timer bulletTimer = new Timer();  bulletTimer.Interval = 1;  bulletTimer.Tick += (sender, e) =>  {  bullet.Top += 5;  if (bullet.Bottom > form.ClientSize.Height || WallFireCollision(bullet))  {  EraseBullet(bullet);  return;  }  if (bullet.Bounds.IntersectsWith(game.GetPlayer().GetPictureBox().Bounds))  {  EraseBullet(bullet);  int PlayerHealth = game.GetPlayer().GetPlayerHealth();  if (PlayerHealth >= 0)  {  PlayerHealth = PlayerHealth - 5;  game.GetPlayer().UpdatePlayerHealth(PlayerHealth);  game.GetPlayer().LivesDecrement();  }  }  };  bulletTimer.Start();  }  public void EraseBullet(PictureBox bullet)  {  form.Controls.Remove(bullet);  pictureBoxes.Remove(bullet);  }  public bool WallFireCollision(PictureBox firePictureBox)  {  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  foreach (PictureBox boundaryPictureBox in boundaryPictures)  {  if (firePictureBox.Bounds.IntersectsWith(boundaryPictureBox.Bounds))  {  return true;  }  }  return false;  }  public override void Movement()  {  switch (direction)  {  case EnemyDirection.Right:  pictureBox.Left -= EnemySpeed;  pictureBox.ImageLocation = imagePathRight;  break;  case EnemyDirection.Left:  pictureBox.Left += EnemySpeed;  pictureBox.ImageLocation = imagePathLeft;  break;  }  if (EnemyCollisions.BoundaryWallCollision(game))  {  if (direction == EnemyDirection.Left)  {  direction = EnemyDirection.Right;  }  else if (direction == EnemyDirection.Right)  {  direction = EnemyDirection.Left;  }  }  if (EnemyCollisions.PlayerEnemyCollision(game))  {  if (direction == EnemyDirection.Left)  {  direction = EnemyDirection.Right;  }  else if (direction == EnemyDirection.Right)  {  direction = EnemyDirection.Left;  }  }  }  }  } |

## **1.8.5 VerticalEnemy.cs**

|  |
| --- |
| using FreedomFodgeFrameWork.Interfaces;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class VerticalEnemy : Enemy  {  private EnemyDirection direction;  private string FireImage;  private List<PictureBox> pictureBoxes;  private long lastBulletTime;  public VerticalEnemy(string imagePath, Game game, int EnemySpeed,string FireImage, Form form, int xPosition, int yPosition, int EnemyHealth)  : base(game, EnemySpeed, form, xPosition, yPosition,EnemyHealth)  {  this.imagePathLeft = imagePath;  this.pictureBox = new PictureBox();  this.pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;  this.pictureBox.Width = 50;  this.pictureBox.Height = 50;  pictureBox.BackColor = Color.Transparent;  direction = EnemyDirection.Up;  this.FireImage= FireImage;  form.Controls.Add(this.pictureBox);  DisplayEnemy();  movementTimer = new Timer();  movementTimer.Interval = 50;  movementTimer.Tick += MovementTimer\_Tick;  movementTimer.Start();  pictureBoxes = new List<PictureBox>();  lastBulletTime = 0;  }  public void DisplayEnemy()  {  if (pictureBox != null)  {  pictureBox.ImageLocation = imagePathLeft;  pictureBox.Location = new System.Drawing.Point(xPosition, yPosition);  }  }  private void MovementTimer\_Tick(object sender, EventArgs e)  {  Movement();  long currentTime = DateTime.Now.Ticks;  long elapsedTime = currentTime - lastBulletTime;  if (elapsedTime >= TimeSpan.FromSeconds(2).Ticks)  {  GenerateBullet();  lastBulletTime = currentTime;  }  }  public void GenerateBullet()  {  PictureBox bullet = new PictureBox();  bullet.SizeMode = PictureBoxSizeMode.StretchImage;  bullet.Width = 10;  bullet.Height = 10;  bullet.BackColor = Color.Transparent;  bullet.ImageLocation = FireImage;  int bulletX = this.pictureBox.Left + (this.pictureBox.Width / 2);  int bulletY = this.pictureBox.Top;  bullet.Location = new Point(bulletX, bulletY);  form.Controls.Add(bullet);  pictureBoxes.Add(bullet);  MoveFire(bullet);  }  public void MoveFire(PictureBox bullet)  {  Timer bulletTimer = new Timer();  bulletTimer.Interval = 1;  bulletTimer.Tick += (sender, e) =>  {  bullet.Left -= 5;  if (bullet.Bottom > form.ClientSize.Height || WallFireCollision(bullet))  {  EraseBullet(bullet);  return;  }  if (bullet.Bounds.IntersectsWith(game.GetPlayer().GetPictureBox().Bounds))  {  EraseBullet(bullet);  int PlayerHealth = game.GetPlayer().GetPlayerHealth();  if (PlayerHealth >= 0)  {  PlayerHealth = PlayerHealth - 5;  game.GetPlayer().UpdatePlayerHealth(PlayerHealth);  game.GetPlayer().LivesDecrement();  }  }  };  bulletTimer.Start();  }  public void EraseBullet(PictureBox bullet)  {  form.Controls.Remove(bullet);  pictureBoxes.Remove(bullet);  }  public bool WallFireCollision(PictureBox firePictureBox)  {  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  foreach (PictureBox boundaryPictureBox in boundaryPictures)  {  if (firePictureBox.Bounds.IntersectsWith(boundaryPictureBox.Bounds))  {  return true;  }  }  return false;  }  public override void Movement()  {  switch (direction)  {  case EnemyDirection.Up:  pictureBox.Top -= EnemySpeed;  break;  case EnemyDirection.Down:  pictureBox.Top += EnemySpeed;  break;  }  if (EnemyCollisions.BoundaryWallCollision(game))  {  if (direction == EnemyDirection.Up)  {  direction = EnemyDirection.Down;  }  else if (direction == EnemyDirection.Down)  {  direction = EnemyDirection.Up;  }  }  if (EnemyCollisions.PlayerEnemyCollision(game))  {  if (direction == EnemyDirection.Up)  {  direction = EnemyDirection.Down;  }  else if (direction == EnemyDirection.Down)  {  direction = EnemyDirection.Up;  }  }  switch (direction)  {  case EnemyDirection.Up:  pictureBox.Top -= EnemySpeed;  break;  case EnemyDirection.Down:  pictureBox.Top += EnemySpeed;  break;  }  }  }  } |

## **1.8.6 RandomEnemy.cs**

|  |
| --- |
| using FreedomFodgeFrameWork.Interfaces;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class RandomEnemy : Enemy  {  EnemyDirection direction;  public RandomEnemy(string imagePathLeft, string imagePathRight, Game game, int EnemySpeed, Form form, int xPosition, int yPosition,int EnemyHealth)  : base(game, EnemySpeed, form, xPosition, yPosition,EnemyHealth)  {  this.imagePathLeft = imagePathLeft;  this.imagePathRight = imagePathRight;  this.pictureBox = new PictureBox();  this.pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;  pictureBox.BackColor = Color.Transparent;  this.pictureBox.Width = 30;  this.pictureBox.Height = 30;  form.Controls.Add(this.pictureBox);  DisplayEnemy();  movementTimer = new Timer();  movementTimer.Interval = 70;  movementTimer.Tick += MovementTimer\_Tick;  movementTimer.Start();  }  public void DisplayEnemy()  {  if (pictureBox != null)  {  pictureBox.ImageLocation = imagePathLeft;  pictureBox.Location = new System.Drawing.Point(xPosition, yPosition);  }  }  private void MovementTimer\_Tick(object sender, EventArgs e)  {  Movement();  }  public override void Movement()  {  direction = (EnemyDirection)random.Next(0, 4);  switch (direction)  {  case EnemyDirection.Up:  pictureBox.Top -= EnemySpeed;  break;  case EnemyDirection.Down:  pictureBox.Top += EnemySpeed;  break;  case EnemyDirection.Left:  pictureBox.Left -= EnemySpeed;  pictureBox.ImageLocation = imagePathLeft;  break;  case EnemyDirection.Right:  pictureBox.Left += EnemySpeed;  pictureBox.ImageLocation = imagePathRight;  break;  }  if (EnemyCollisions.BoundaryWallCollision(game)|| EnemyCollisions.PlayerEnemyCollision(game))  {  switch (direction)  {  case EnemyDirection.Up:  pictureBox.Top += EnemySpeed;  break;  case EnemyDirection.Down:  pictureBox.Top -= EnemySpeed;  break;  case EnemyDirection.Left:  pictureBox.Left += EnemySpeed;  break;  case EnemyDirection.Right:  pictureBox.Left -= EnemySpeed;  break;  }  }  }  }  } |

## **1.8.7 PlayerCollision.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using FreedomFodgeFrameWork.Interfaces;  namespace FreedomFodgeFrameWork  {  public class PlayerCollision :ICollision  {  public bool BoundaryWallCollision(Game game)  {  Player player = game.GetPlayer();  int playerX = player.GetPlayerX();  int playerY = player.GetPlayerY();  int playerWidth = player.GetPictureBox().Width;  int playerHeight = player.GetPictureBox().Height;  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  foreach (var boundaryPicture in boundaryPictures)  {  int boundaryX = boundaryPicture.Location.X;  int boundaryY = boundaryPicture.Location.Y;  int boundaryWidth = boundaryPicture.Width;  int boundaryHeight = boundaryPicture.Height;  if (playerY + playerHeight >= boundaryY &&  playerY <= boundaryY + boundaryHeight &&  playerX + playerWidth >= boundaryX &&  playerX <= boundaryX + boundaryWidth)  {  return true;  }  }  return false;  }  public bool PlayerEnemyCollision(Game game)  {  List<Enemy> EnemiesList = game.GetEnemies();  Player player = game.GetPlayer();  int playerX = player.GetPlayerX();  int playerY = player.GetPlayerY();  int playerWidth = player.GetPictureBox().Width;  int playerHeight = player.GetPictureBox().Height;  foreach (var enemy in EnemiesList)  {  PictureBox enemyPictureBox = enemy.GetPictureBox();  int enemyX = enemyPictureBox.Location.X;  int enemyY = enemyPictureBox.Location.Y;  int enemyWidth = enemyPictureBox.Width;  int enemyHeight = enemyPictureBox.Height;  if (playerY + playerHeight >= enemyY &&  playerY <= enemyY + enemyHeight &&  playerX + playerWidth >= enemyX &&  playerX <= enemyX + enemyWidth)  {  return true;  }  }  return false;  }  public bool FireCollision(Game game, PictureBox picturebox)  {  List<Enemy> enemiesList = game.GetEnemies();  foreach (Enemy enemy in enemiesList)  {  if (picturebox.Bounds.IntersectsWith(enemy.GetPictureBox().Bounds))  {  int health = enemy.GetEnemyHealth();  health--;  enemy.SetEnemyHealth(health);  if (health <= 0)  {  enemiesList.Remove(enemy);  enemy.GetPictureBox().Visible = false;  }  game.UpdateEnemyLabel();  return true;  }  }  game.SetEnemies(enemiesList);  game.UpdateEnemyLabel();  return false;  }  }  } |

## **1.8.8 EnemyCollision.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using FreedomFodgeFrameWork.Interfaces;  namespace FreedomFodgeFrameWork  {  public class EnemyCollision :ICollision  {  public bool BoundaryWallCollision(Game game)  {  List<Enemy> EnemiesList = game.GetEnemies();  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  foreach (var enemy in EnemiesList)  {  PictureBox enemyPictureBox = enemy.GetPictureBox();  int enemyX = enemyPictureBox.Location.X;  int enemyY = enemyPictureBox.Location.Y;  int enemyWidth = enemyPictureBox.Width;  int enemyHeight = enemyPictureBox.Height;  foreach (var boundaryPicture in boundaryPictures)  {  int boundaryX = boundaryPicture.Location.X;  int boundaryY = boundaryPicture.Location.Y;  int boundaryWidth = boundaryPicture.Width;  int boundaryHeight = boundaryPicture.Height;  if (enemyY + enemyHeight-2 >= boundaryY &&  enemyY <= boundaryY + boundaryHeight &&  enemyX + enemyWidth >= boundaryX &&  enemyX <= boundaryX + boundaryWidth)  {  return true;  }  }  }  return false;  }  public bool PlayerEnemyCollision(Game game)  {  List<Enemy> EnemiesList = game.GetEnemies();  Player player = game.GetPlayer();  int playerX = player.GetPlayerX();  int playerY = player.GetPlayerY();  int playerWidth = player.GetPictureBox().Width;  int playerHeight = player.GetPictureBox().Height;  foreach (var enemy in EnemiesList)  {  PictureBox enemyPictureBox = enemy.GetPictureBox();  int enemyX = enemyPictureBox.Location.X;  int enemyY = enemyPictureBox.Location.Y;  int enemyWidth = enemyPictureBox.Width;  int enemyHeight = enemyPictureBox.Height;  if (playerY + playerHeight >= enemyY &&  playerY <= enemyY + enemyHeight &&  playerX + playerWidth >= enemyX &&  playerX <= enemyX + enemyWidth)  {  int PlayerHealth = game.GetPlayer().GetPlayerHealth();  if (PlayerHealth >= 0)  {  PlayerHealth = PlayerHealth - 10;  game.GetPlayer().UpdatePlayerHealth(PlayerHealth);  game.GetPlayer().LivesDecrement();  }  return true;    }  }  return false;  }  public bool FireCollision(Game game, PictureBox picturebox)  {  Player player = game.GetPlayer();  if (picturebox.Bounds.IntersectsWith(player.GetPictureBox().Bounds))  {  return true;  }  return false;  }  }  } |

## **1.8.9 ICollision.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork.Interfaces  {  public interface ICollision  {  bool BoundaryWallCollision(Game game);  bool PlayerEnemyCollision(Game game);  bool FireCollision(Game game,PictureBox pictureBox);  }  } |

## **1.8.10 IMovement.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork.Interfaces  {  public interface IMovement  {  void Movement();  }  } |

## **1.8.11 EnemyDirection.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FreedomFodgeFrameWork  {    enum EnemyDirection  {  Up,  Down,  Left,  Right  }    } |

## **1.8.12 Gravity.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class Gravity  {  public static void ApplyGravity(PictureBox pictureBox, ref int X, ref int Y, Game game)  {  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  List<PictureBox> innerMazePictures = game.GetMaze().GetInnerPictureBoxes();  int fallingSpeed = 2;  Timer timer = new Timer();  timer.Interval = 10;  timer.Tick += (sender, e) => {    foreach (PictureBox boundaryPicture in boundaryPictures)  {  if (pictureBox.Bounds.IntersectsWith(boundaryPicture.Bounds))  {  timer.Stop();  return;  }  }  foreach (PictureBox innerMazePicture in innerMazePictures)  {  if (pictureBox.Bounds.IntersectsWith(innerMazePicture.Bounds))  {  timer.Stop();  return;  }  }  pictureBox.Top += fallingSpeed;    };  timer.Start();  X = pictureBox.Left;  Y = pictureBox.Top;  }  }  } |

## **1.8.13 GameObject.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FreedomFodgeFrameWork  {  public enum ObjectType  {  Wall,  Player,  Enemy  }    } |

## **1.8.14 Fire.cs**

|  |
| --- |
| using FreedomFodgeFrameWork.Interfaces;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FreedomFodgeFrameWork  {  public class Fire  {  private string direction;  private Form form;  private PictureBox pictureBox;  private string FireImage;  private int FireX;  private int FireY;  private Game game;  public Boolean IsVisible;  public int BulletSpeed;  public Fire(string direction, Form form, PictureBox pictureBox, string fireImage, int fireX, int fireY, Game game,int BulletSpeed)  {  this.direction = direction;  this.form = form;  this.pictureBox = pictureBox;  FireImage = fireImage;  FireX = fireX;  FireY = fireY;  this.game = game;  this.BulletSpeed = BulletSpeed;  GenerateBullet();  }  public PictureBox GetPictureBox(PictureBox pictureBox)  {  return pictureBox;  }  public void GenerateBullet()  {  PictureBox bullet = new PictureBox();  bullet.SizeMode = PictureBoxSizeMode.Zoom;  bullet.Width = 20;  bullet.Height = 20;  bullet.BackColor = Color.Transparent;  bullet.Image = Image.FromFile(FireImage);  if (direction == "left")  {  bullet.Location = new Point(FireX, FireY);  }  else if (direction == "right")  {  bullet.Location = new Point(FireX , FireY );  }  else if (direction == "down")  {  bullet.Location = new Point(FireX , FireY);  }  form.Controls.Add(bullet);    MoveBullet(bullet);  }  ICollision FireCollision = new PlayerCollision();  public void MoveBullet(PictureBox bullet)  {  Timer bulletTimer = new Timer();  bulletTimer.Interval = 2;  bulletTimer.Tick += (sender, e) =>  {  if (direction == "left")  {  bullet.Left -= BulletSpeed;  }  else if (direction == "right")  {  bullet.Left += BulletSpeed;  }  else if (direction == "down")  {  bullet.Top += BulletSpeed;  }  if (WallFireCollision(bullet))  {  EraseBullet(bullet);  return;  }  if (FireCollision.FireCollision(game, bullet))  {  EraseBullet(bullet);  return;  }  if (bullet.Left < 0 || bullet.Left > form.ClientSize.Width)  {  EraseBullet(bullet);  return;  }  };  bulletTimer.Start();  }  public void EraseBullet(PictureBox bullet)  {  bullet.Visible = false;  form.Controls.Remove(bullet);  IsVisible = false;    }  public bool WallFireCollision(PictureBox firePictureBox)  {  List<PictureBox> boundaryPictures = game.GetMaze().GetBoundaryPictureBoxes();  foreach (PictureBox boundaryPictureBox in boundaryPictures)  {  if (firePictureBox.Bounds.IntersectsWith(boundaryPictureBox.Bounds))  {  return true;  }  }  return false;  }  }  } |

## **1.8.15 Game.cs**

|  |
| --- |
| using FreedomFodgeFrameWork;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using FreedomFodgeFrameWork.Interfaces;  using System.Drawing;  namespace FreedomFodgeFrameWork  {  public class Game  {  private static Game instance;  private Maze Maze;  private Player Player;  private Form Form;  private Label RemainingEnemiesLabel;  private List<Enemy> Enemies = new List<Enemy>();  private Game(Form form)  {  this.Form = form;  InitializeEnemiesLabel();  }  public static Game GetInstance(Form form)  {  if (instance == null)  {  instance = new Game(form);  }  return instance;  }  public Maze GetMaze()  {  return this.Maze;  }  public Player GetPlayer()  {  return this.Player;  }  public void SetMaze(Maze maze)  {  this.Maze = maze;  }  public void SetPlayer(Player player)  {  this.Player = player;  }  public void AddEnemies(Enemy enemy)  { Enemies.Add(enemy); }  public List<Enemy> GetEnemies()  {  return Enemies;  }  public void SetEnemies(List<Enemy> Enemies)  {  this.Enemies = Enemies;  }  private void InitializeEnemiesLabel()  {  RemainingEnemiesLabel = new Label();  RemainingEnemiesLabel.Font = new Font("Arial", 12, FontStyle.Bold);  RemainingEnemiesLabel.ForeColor = Color.Black;  RemainingEnemiesLabel.BackColor = Color.White;  RemainingEnemiesLabel.AutoSize = true;  RemainingEnemiesLabel.Location = new Point(Form.ClientSize.Width - RemainingEnemiesLabel.Width - 80, 50);  UpdateEnemyLabel();  Form.Controls.Add(RemainingEnemiesLabel);  }  public void Update()  {  Player.Movement();  foreach (var enemy in Enemies)  {  enemy.Movement();  }  }  public void UpdateEnemyLabel()  {  RemainingEnemiesLabel.Text = "Remaining Enemies : " + Enemies.Count;  }  }  } |