**The Pirates**



Session: 2022 – 2026

**Submitted by:**

Ayesha Khalid

2022-CS-153

**Supervised by:**

Prof. Dr. Muhammad Awais Hassan

Department of Computer Sciences

**University of Engineering and Technology**

**Lahore Pakistan**

**Contents:**

[**Description:** 3](#_Toc141608513)

[**Game Characters Description:** 4](#_Toc141608514)

[**Player:** 4](#_Toc141608515)

[ **Tank :** 4](#_Toc141608516)

[**Enemies:** 4](#_Toc141608517)

[ **HR:** 4](#_Toc141608518)

[ **VK:** 4](#_Toc141608519)

[ **Helicopter:** 4](#_Toc141608520)

[ **RM:** 5](#_Toc141608521)

[**Game Objects Description:** 5](#_Toc141608522)

[ **The Player's Tank:** 5](#_Toc141608523)

[ **Heart:** 5](#_Toc141608524)

[ **Bomb:** 5](#_Toc141608525)

[ **Bullets:** 5](#_Toc141608526)

[ **The Enemys:** 5](#_Toc141608527)

[ **Walls:** 5](#_Toc141608528)

[**Levels:** 5](#_Toc141608529)

[**Rules & Instructions:** 6](#_Toc141608530)

[**Goal of the Game:** 6](#_Toc141608531)

[**OOP Concepts:** 6](#_Toc141608532)

[**Inheritance:** 6](#_Toc141608533)

[**Encapsulation:** 6](#_Toc141608534)

[**Abstraction:** 6](#_Toc141608535)

[**Association:** 6](#_Toc141608536)

[**Class Details:** 6](#_Toc141608537)

[**GameCell:** 6](#_Toc141608538)

[**Code**: 6](#_Toc141608539)

[**GameGrid:** 8](#_Toc141608540)

[**Code**: 8](#_Toc141608541)

[**GameObject:** 9](#_Toc141608542)

[**Code**: 9](#_Toc141608543)

[**GameObjectType:** 11](#_Toc141608544)

[**Code**: 11](#_Toc141608545)

[**GameDirection:** 11](#_Toc141608546)

[**Code**: 11](#_Toc141608547)

[**GamePlayer:** 11](#_Toc141608548)

[**Code**: 11](#_Toc141608549)

[**GameEnemy:** 13](#_Toc141608550)

[**Code**: 13](#_Toc141608551)

[**HorizontalEnemy:** 13](#_Toc141608552)

[**Code**: 13](#_Toc141608553)

[**VerticalEnemy:** 14](#_Toc141608554)

[**Code:** 14](#_Toc141608555)

[**RandomEnemy:** 16](#_Toc141608556)

[**Hellicopter:** 17](#_Toc141608557)

[**Code**: 17](#_Toc141608558)

[**Bullet:** 18](#_Toc141608559)

[**Code**: 18](#_Toc141608560)

[**Bomb:** 18](#_Toc141608561)

[**ImagerGiver:** 19](#_Toc141608562)

[**Code**: 19](#_Toc141608563)

[**GameCollision:** 20](#_Toc141608564)

[**Code**: 20](#_Toc141608565)

[**GameThings:** 21](#_Toc141608566)

[**Code**: 21](#_Toc141608567)

[**Wireframes:** 26](#_Toc141608568)

[**Class Diagram** 28](#_Toc141608569)

# 

# **Description:**

# Thomas is thin looking fragile boy who is very optimistic and has an old mother whom he has to feed. He used to do a job in a small coffee shop. Unfortunately, he lost his job because customers complained against him that he is never clean. Now he is jobless and living from hand to mouth. So now Thomas has to rob a bank. Thomas will get success in his plan but it will be his bad luck that a random guy who will be watching all this scene from a distance, will call the police.

# Then Thomas will be running to save his life. He won't even surrender because he knows that if police took him away then they will put him in lock for days and his old mother will die with hunger. So at any cost he will not want to get caught. On his way he will collide with many other people and steal their money as well in order to increase his money. There will be three police cars which will chase Thomas. Thomas is a clever boy and a fast runner so he knows that he will protect himself and the money from the police. Thomas has a gun and he will be able to fire at the police as well. He always loved to do the adventures and finally he will be fulfilling his desire by taking more and more risks. His mission will be to cross all the police cars in order to run away from the city. There will be a main gate at the other end of the from where he will escape. He will get tired while running and get water bottles to quench his thrust and a chance to increase his health which will get lost after getting hit by police fire. There will be only three health for Thomas and after that Thomas will die and game will be over. **Game Characters Description:**

## **Player:**

There is one human player in the Game.

## **Thomas :**

. Thomas is the main character of this game and is known for his smartness. He is adventurous and loves to take risks. Thomas is a brave boy who got distracted because of poverty. Thomas is a brave, determined, and has a never-say-die-spirit.

**Enemies:**The enemies in the game are:

## **Motor Police:**

The Motor police can move in horizontal direction and can fire only from its horizontal sides. If the fire hits the Thomas then Thomas will lose his one health cell.

## **Local Police:**

The Local police can only move in vertical direction and can fire only from the vertical sides. If the fire hits the Thomas then Thomas will lose his one health cell.

## **Highway Police:**

The Highway police is capable of moving in four directions and is capable of throwing bomb from any of his four sides. When the bomb will touch the Thomas he will go back to the starting point of the city and will lose his one health cell. Highway police will be chasing the Thomas.

# **Game Objects Description:**

Following are the Objects in the Game

## **Money:**

This is the primary object which is produced on the screen every time on random position. If the player eats it its health will be increased.

## **Bullets:**

Bullets are the primary object in the game that is produced by the player as well as the enemies but shapes are different.

## **The Enemys:**

The police is the primary obstacles that the player must defeat in order to progress through the game.

## **Walls:**

There are certain walls in the city that help the player to hide behind them and help to get cover from the police fire.

# **Rules & Instructions:**

The Thomas can hit the money in order to increase health .Thomas loses his health when got fired by the police. If Thomas hits the police he can kill them in 20 hits.

# **Goal of the Game:**

The goal of the game is to destroy all the police tanks to save thomas.

# **OOP Concepts:**

## **Inheritance:**

In this game, inheritance is used in many classes. As GamePlayer, GameEnemy , Bullet inherits the Class GameObject. Further VerticalPolice, HorizontalPolice and RandomPolice Classes inherits the Class GameEnemy.

## **Encapsulation:**

I have made all the parent classes attribute protected and the child and other classes’s attributes private. I have made getter and setter function to access and modify the values of attributes.

## **Abstraction:**

In my game, I have made the GameEnemy class abstract which have some abstract methods its child classes override the abstract functions in this class. It have abstract methods like move that all its children classes should implement.

## **Association:**

In my game, GameGrid is associated with the GameCell as the GameGrid has a 2d array of cells in it, so this class have one to many relationship with GameCell.

# **Class Details:**

## **GameCell:**

This the basic class as the game consists of the grid and grid is consisted on the array of the Cells.

### **Code**:

class GameCell

{

private int row;

private int col;

private GameObject currentGameObject;

private GameGrid grid;

private PictureBox pictureBox;

private const int width = 44;

private const int height = 40;

public int X // Getter & Setter of row

{

get

{

return row;

}

set

{

row = value;

}

}

public int Y // Getter & Setter of col

{

get

{

return col;

}

set

{

col = value;

}

}

public GameObject CurrentGameObject => currentGameObject; // Getter of currentGameObject

public PictureBox PictureBox // Getter & Setter of pictureBox

{

get

{

return pictureBox;

}

set

{

pictureBox = value;

}

}

public GameCell(int row, int col, GameGrid grid) // Constructor

{

this.row = row;

this.col = col;

pictureBox = new PictureBox();

pictureBox.Left = col \* width;

pictureBox.Top = row \* height;

pictureBox.Size = new Size(width, height);

pictureBox.SizeMode = PictureBoxSizeMode.Zoom;

pictureBox.BackColor = Color.Transparent;

this.grid = grid;

}

public void setGameObject(GameObject gameObject) // Setter of currentGameObject

{

currentGameObject = gameObject;

pictureBox.Image = gameObject.Image;

}

public GameCell nextCell(GameDirection direction) // Method nextCell

{

if (direction == GameDirection.Left && col > 0)

{

GameCell cell = grid.getCell(row, col - 1);

if (cell.CurrentGameObject.GameObjectType != 0)

{

return cell;

}

}

if (direction == GameDirection.Right && col < grid.Cols - 1)

{

GameCell cell2 = grid.getCell(row, col + 1);

if (cell2.CurrentGameObject.GameObjectType != 0)

{

return cell2;

}

}

if (direction == GameDirection.Up && row > 0)

{

GameCell cell3 = grid.getCell(row - 1, col);

if (cell3.CurrentGameObject.GameObjectType != 0)

{

return cell3;

}

}

if (direction == GameDirection.Down && row < grid.Rows - 1)

{

GameCell cell4 = grid.getCell(row + 1, col);

if (cell4.CurrentGameObject.GameObjectType != 0)

{

return cell4;

}

}

return this;

}

## **GameGrid:**

This the basic class as the game consists of the grid.

### **Code**:

class GameGrid

{

private GameCell[,] cells;

private int rows;

private int cols;

public int Rows // Getter and Setter of rows

{

get

{

return rows;

}

set

{

rows = value;

}

}

public int Cols // Getter and Setter of cols

{

get

{

return cols;

}

set

{

cols = value;

}

}

public GameGrid(string fileName, int rows, int cols) // Constructor

{

this.rows = rows;

this.cols = cols;

cells = new GameCell[rows, cols];

loadGrid(fileName);

}

public GameCell getCell(int x, int y) // Method getCell

{

return cells[x, y];

}

private void loadGrid(string fileName) // Method loadGrid

{

StreamReader streamReader = new StreamReader(fileName);

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

{

string text = streamReader.ReadLine();

for (int j = 0; j < cols; j++)

{

GameCell gameCell = new GameCell(i, j, this);

char displayCharacter = text[j];

GameObjectType gameObjectType = GameObject.getGameObjectType(displayCharacter);

Image gameObjectImage = ImageGiver.getGameObjectImage(displayCharacter);

GameObject gameObject = new GameObject(gameObjectType, gameObjectImage);

gameCell.setGameObject(gameObject);

cells[i, j] = gameCell;

}

}

streamReader.Close();

}

## **GameObject:**

This the basic class as the game consists of many game objects.

### **Code**:

class GameObject

{

private char displayCharacter;

private GameObjectType gameObjectType;

private GameCell currentCell;

private Image image;

public char DisplayCharacter // Getter & Setter of displayChar

{

get

{

return displayCharacter;

}

set

{

displayCharacter = value;

}

}

public GameObjectType GameObjectType // Getter & Setter of gameObjectType

{

get

{

return gameObjectType;

}

set

{

gameObjectType = value;

}

}

public GameCell CurrentCell // Getter & Setter of currentCell

{

get

{

return currentCell;

}

set

{

currentCell = value;

currentCell.setGameObject(this);

}

}

public Image Image // Getter & Setter of image

{

get

{

return image;

}

set

{

image = value;

}

}

public GameObject(GameObjectType type, Image image) // Constructer

{

gameObjectType = type;

Image = image;

}

public GameObject(GameObjectType type, char displayCharacter) // Constructer

{

gameObjectType = type;

this.displayCharacter = displayCharacter;

}

public static GameObjectType getGameObjectType(char displayCharacter) // Method

{

switch (displayCharacter)

{

case '#':

case '%':

case '|':

return GameObjectType.WALL;

case '.':

return GameObjectType.REWARD;

default:

return GameObjectType.NONE;

}

}

}

## **GameObjectType:**

This the basic enum class which consists of the different object types used in the game.

### **Code**:

enum GameObjectType

{

WALL,

PLAYER,

ENEMY,

HEART,

HENEMY,

VENEMY,

RENEMY,

REWARD,

BULLET,

BOMB,

NONE

}

## **GameDirection:**

This the basic enum class which consists of the four directions.

### **Code**:

enum GameDirection

{

Left,

Right,

Up,

Down

}

## **GamePlayer:**

This the class of the player of the game which inherits the class GameObject.

### **Code**:

class GamePlayer : GameObject

{

private static List<Bullet> bullets;

public GamePlayer(Image image, GameCell startCell) :base(GameObjectType.PLAYER, image)

{

base.CurrentCell = startCell;

bullets = new List<Bullet>();

}

public void move(GameCell gameCell)

{

base.CurrentCell = gameCell;

}

public void generateBullet()

{

Bullet bullet = new Bullet(ImageGiver.getPlayerBulletImage(),this.CurrentCell.nextCell(GameDirection.Right));

bullets.Add(bullet);

}

public void moveBullets()

{

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

{

GameCell currentCell = bullets[i].CurrentCell;

GameCell nextCell = bullets[i].nextCell(GameDirection.Right);

if (currentCell == nextCell)

{

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (GameCollision.isPlayerBulletCollideWithHorizontalEnemy(bullets[i].nextCell(GameDirection.Right)))

{

GameThings.addScore(1);

GameThings.decreaseHorizontalEnemyHealth(1);

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (GameCollision.isPlayerBulletCollideWithVerticalEnemy(bullets[i].nextCell(GameDirection.Right)))

{

GameThings.addScore(1);

GameThings.decreaseVerticalEnemyHealth(1);

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (GameCollision.isPlayerBulletCollideWithRandomEnemy(bullets[i].nextCell(GameDirection.Right)))

{

GameThings.addScore(2);

GameThings.decreaseRandomEnemyHealth(1);

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else

{

bullets[i].move(nextCell);

}

}

}

public void destroyBullets()

{

foreach(Bullet bullet in bullets)

{

bullet.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

bullets.Clear();

}

}

## **GameEnemy:**

This the abstract class of the enemy of the game which inherits the class GameObject.

### **Code**:

abstract class GameEnemy : GameObject

{

public GameEnemy(Image ghostImage,GameObjectType enemyType)

: base(enemyType, ghostImage)

{

}

public abstract List<Bullet> getBulletList();

public abstract GameCell nextCell();

public abstract void move(GameCell gameCell);

public abstract void generateBullet();

public abstract void moveBullets();

}

## **HorizontalEnemy:**

This the class of the horizontal enemy of the game which inherits the class GameEnemy, as GameEnemy is the abstract class so, it overrides the abstract methods.

### **Code**:

class HorizontalEnemy : GameEnemy

{

GameDirection direction = GameDirection.Left;

List<Bullet> bullets;

public HorizontalEnemy(Image ghostImage, GameCell startCell) : base(ghostImage, GameObjectType.HENEMY)

{

this.CurrentCell = startCell;

this.bullets = new List<Bullet>();

}

public override List<Bullet> getBulletList() { return bullets; }

public override void move(GameCell gameCell)

{

if (this.CurrentCell != null)

{

this.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

CurrentCell = gameCell;

}

public override GameCell nextCell()

{

GameCell nextCell = this.CurrentCell;

GameCell potentialNextCell = this.CurrentCell.nextCell(direction);

if (potentialNextCell == nextCell)

{

if (direction == GameDirection.Left)

{

direction = GameDirection.Right;

}

else if (direction == GameDirection.Right)

{

direction = GameDirection.Left;

}

}

else

{

nextCell = potentialNextCell;

}

return nextCell;

}

public override void generateBullet()

{

Bullet bullet = new Bullet(ImageGiver.getHorizontalEnemyBulletImage(), this.CurrentCell.nextCell(GameDirection.Left));

bullets.Add(bullet);

}

public override void moveBullets()

{

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

{

if (bullets[i].CurrentCell == bullets[i].nextCell(GameDirection.Left))

{

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (bullets[i].nextCell(GameDirection.Left).CurrentGameObject.GameObjectType == GameObjectType.PLAYER)

{

GameThings.decreasePlayerHealth(1);

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else

{

bullets[i].move(bullets[i].nextCell(GameDirection.Left));

}

}

}

}

## **VerticalEnemy:**

This the class of the vertical enemy of the game which inherits the class GameEnemy, as GameEnemy is the abstract class so, it overrides the abstract methods.

### **Code:**

class VerticalEnemy : GameEnemy

{

private GameDirection direction = GameDirection.Down;

private List<Bullet> bullets;

public VerticalEnemy(Image ghostImage, GameCell startCell): base(ghostImage, GameObjectType.VENEMY)

{

base.CurrentCell = startCell;

bullets = new List<Bullet>();

}

public override List<Bullet> getBulletList() { return bullets; }

public override void move(GameCell gameCell)

{

if (base.CurrentCell != null)

{

base.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

base.CurrentCell = gameCell;

}

public override GameCell nextCell()

{

GameCell gameCell = base.CurrentCell;

GameCell gameCell2 = base.CurrentCell.nextCell(direction);

if (gameCell2 == gameCell)

{

if (direction == GameDirection.Up)

{

direction = GameDirection.Down;

}

else if (direction == GameDirection.Down)

{

direction = GameDirection.Up;

}

}

else

{

gameCell = gameCell2;

}

return gameCell;

}

public override void generateBullet()

{

Bullet bullet = new Bullet(ImageGiver.getVerticalEnemyBulletImage(), this.CurrentCell.nextCell(GameDirection.Left));

bullets.Add(bullet);

}

public override void moveBullets()

{

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

{

if (bullets[i].CurrentCell == bullets[i].nextCell(GameDirection.Left))

{

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (bullets[i].nextCell(GameDirection.Left).CurrentGameObject.GameObjectType == GameObjectType.PLAYER)

{

GameThings.decreasePlayerHealth(1);

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else

{

bullets[i].move(bullets[i].nextCell(GameDirection.Left));

}

}

}

}

## **RandomEnemy:**

This the class of the vertical enemy of the game which inherits the class GameEnemy, as GameEnemy is the abstract class so, it overrides the abstract methods.

**Code**:

class RandomEnemy : GameEnemy

{

private GameDirection direction = GameDirection.Down;

List<Bullet> bullets;

public RandomEnemy(Image ghostImage, GameCell startCell)

: base(ghostImage, GameObjectType.RENEMY)

{

base.CurrentCell = startCell;

bullets = new List<Bullet>();

}

public override List<Bullet> getBulletList() { return bullets; }

public override void move(GameCell gameCell)

{

if (base.CurrentCell != null)

{

base.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

base.CurrentCell = gameCell;

}

public override GameCell nextCell()

{

int value = generateRandomNumber();

if (value == 0)

{

direction = GameDirection.Right;

}

else if (value == 1)

{

direction = GameDirection.Left;

}

else if (value == 2)

{

direction = GameDirection.Up;

}

else if (value == 3)

{

direction = GameDirection.Down;

}

GameCell gameCell = base.CurrentCell;

GameCell gameCell2 = base.CurrentCell.nextCell(direction);

return gameCell2;

}

public override void generateBullet()

{

Bullet bullet = new Bullet(ImageGiver.getRandomEnemyBulletImage(), this.CurrentCell.nextCell(GameDirection.Left));

bullets.Add(bullet);

}

public override void moveBullets()

{

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

{

if (bullets[i].CurrentCell == bullets[i].nextCell(GameDirection.Left))

{

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else if (bullets[i].nextCell(GameDirection.Left).CurrentGameObject.GameObjectType == GameObjectType.PLAYER)

{

GameThings.decreasePlayerHealth(1);

GameCell currentCell = this.CurrentCell;

bullets[i].CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

bullets.RemoveAt(i);

}

else

{

bullets[i].move(bullets[i].nextCell(GameDirection.Left));

}

}

}

public int generateRandomNumber()

{

Random rand = new Random();

return rand.Next(4);

}

}

## **Hellicopter:**

This the class of the hellicopter of the game which inherits the class RandomEnemy.

### **Code**:

class Hellicopter : RandomEnemy

{

List<Bomb> bombs;

public Hellicopter(Image ghostImage, GameCell startCell)

: base(ghostImage,startCell)

{

bombs = new List<Bomb>();

}

public void generateBomb()

{

Bomb bomb = new Bomb(ImageGiver.getBombImage(), this.CurrentCell.nextCell(GameDirection.Down));

bombs.Add(bomb);

}

public void destroyBombs()

{

foreach(Bomb bomb in bombs)

{

bomb.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

bombs.Clear();

}

}

## **Bullet:**

This class is bullet which inherits the GameObject.

### **Code**:

class Bullet : GameObject

{

public Bullet(Image image, GameCell startCell)

: base(GameObjectType.BULLET, image)

{

base.CurrentCell = startCell;

}

public void move(GameCell gameCell)

{

if (this.CurrentCell != null)

{

this.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

CurrentCell = gameCell;

}

public GameCell nextCell(GameDirection direction)

{

return this.CurrentCell.nextCell(direction);

}

}

## **Bomb:**

This class is bomb which inherits the GameObject.

**Code**:

class Bomb : GameObject

{

public Bomb(Image image, GameCell startCell)

: base(GameObjectType.BOMB, image)

{

base.CurrentCell = startCell;

}

}

## **ImagerGiver:**

This is the class made to get Images of different objects and the game Objects.

### **Code**:

class ImageGiver

{

public static GameObject getBlankGameObject()

{

return new GameObject(GameObjectType.NONE, Resources.simpleBox);

}

public static GameObject getHeartObject()

{

return new GameObject(GameObjectType.HEART, Resources.heart);

}

public static Image getPlayerImage()

{

return Resources.playerTank;

}

public static Image getVerticalEnemyImage()

{

return Resources.verticalEnemyTank;

}

public static Image getHorizontalEnemyImage()

{

return Resources.horizontalEnemyTank;

}

public static Image getRandomEnemyImage()

{

return Resources.randomEnemyTank;

}

public static Image getPlayerBulletImage()

{

return Resources.playerBullet;

}

public static Image getBombImage()

{

return Resources.bomb;

}

public static Image getHorizontalEnemyBulletImage()

{

return Resources.horizontalEnemyBullet;

}

public static Image getRandomEnemyBulletImage()

{

return Resources.randomEnemyBullet;

}

public static Image getVerticalEnemyBulletImage()

{

return Resources.verticalEnemyBullet;

}

public static Image getGameObjectImage(char displayCharacter)

{

Image result = Resources.simpleBox;

if (displayCharacter == '|' || displayCharacter == '%')

{

result = Resources.verticalWall;

}

if (displayCharacter == '#')

{

result = Resources.horizontalWall;

}

return result;

}

}

## **GameCollision:**

This the basic class which handles all the collisions of the game.

### **Code**:

class GameCollision

{

public static bool isEnemyCollideWithPlayer(GameEnemy ghost)

{

bool flag = false;

if (ghost.CurrentCell.CurrentGameObject.GameObjectType == GameObjectType.PLAYER)

{

flag = true;

}

return flag;

}

public static bool isPlayerCollideWithHeart(GameCell nextCell)

{

if (nextCell.CurrentGameObject.GameObjectType == GameObjectType.HEART)

{

return true;

}

return false;

}

public static bool isPlayerCollideWithBomb(GameCell nextCell)

{

if (nextCell.CurrentGameObject.GameObjectType == GameObjectType.BOMB)

{

return true;

}

return false;

}

public static bool isPlayerBulletCollideWithEnemy(GameCell potentialCell)

{

bool flag = false;

if (potentialCell.CurrentGameObject.GameObjectType == GameObjectType.ENEMY)

{

flag = true;

}

return flag;

}

public static bool isPlayerBulletCollideWithHorizontalEnemy(GameCell potentialCell)

{

bool flag = false;

if (potentialCell.CurrentGameObject.GameObjectType == GameObjectType.HENEMY)

{

flag = true;

}

return flag;

}

public static bool isPlayerBulletCollideWithVerticalEnemy(GameCell potentialCell)

{

bool flag = false;

if (potentialCell.CurrentGameObject.GameObjectType == GameObjectType.VENEMY)

{

flag = true;

}

return flag;

}

public static bool isPlayerBulletCollideWithRandomEnemy(GameCell potentialCell)

{

bool flag = false;

if (potentialCell.CurrentGameObject.GameObjectType == GameObjectType.RENEMY)

{

flag = true;

}

return flag;

}

}

## **GameThings:**

This class is made to handle the game and reduce the code in form1.cs

### **Code**:

class GameThings

{

private static int score = 0;

public char clue;

public int level = 1;

private static int playerHealth = 10;

private static int horizontalEnemyHealth = 10;

private static int verticalEnemyHealth = 10;

private static int randomEnemyHealth = 10;

private int timer = 0;

private Form gameGUI;

private GamePlayer player;

private GameGrid grid;

private List<GameEnemy> enemies;

private List<Hellicopter> hellicopters;

private bool controller;

private static int temp = 0;

public GameThings(Form gameGUI,char clue)

{

this.clue = clue;

this.gameGUI = gameGUI;

grid = new GameGrid("maze1.txt", 15, 35);

enemies = new List<GameEnemy>();

hellicopters = new List<Hellicopter>();

printMaze(grid);

GameCell cell = grid.getCell(8, 10);

player = new GamePlayer(ImageGiver.getPlayerImage(), cell);

controller = true;

if(clue == 'R')

{

if(temp == 0)

{

player.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

readFromFile("data.txt");

}

controller = false;

temp++;

}

if(controller)

{

score = 0;

level = 1;

playerHealth = 10;

verticalEnemyHealth = 10;

horizontalEnemyHealth = 10;

randomEnemyHealth = 10;

temp++;

}

}

public void addHellicopter()

{

Hellicopter hellicopter = new Hellicopter(Resources.Hellicopter, grid.getCell(10, 16));

hellicopters.Add(hellicopter);

}

public void helicoptersFunctionality()

{

foreach(var i in hellicopters)

{

i.move(i.nextCell());

if (timer % 15 == 0)

i.generateBomb();

}

}

public void destroyBombs()

{

foreach (var i in hellicopters)

{

i.destroyBombs();

}

}

public void addHorizontalEnemyInList()

{

clearEnemyList();

HorizontalEnemy he = new HorizontalEnemy(ImageGiver.getHorizontalEnemyImage(), grid.getCell(6, 24));

addEnemyInList(he);

}

public void addRandomEnemyInList()

{

clearEnemyList();

RandomEnemy re = new RandomEnemy(ImageGiver.getRandomEnemyImage(), grid.getCell(6, 21));

addEnemyInList(re);

}

public void clearEnemyList()

{

foreach(var i in enemies)

{

i.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

foreach (Bullet bullet in i.getBulletList())

{

bullet.CurrentCell.setGameObject(ImageGiver.getBlankGameObject());

}

}

enemies.Clear();

}

public GamePlayer getPlayer()

{

return player;

}

public int getScore()

{

return score;

}

public int getPlayerHealth()

{

return playerHealth;

}

public int getHorizontalEnemyHealth()

{

return horizontalEnemyHealth;

}

public int getVerticalEnemyHealth()

{

return verticalEnemyHealth;

}

public int getRandomEnemyHealth()

{

return randomEnemyHealth;

}

public int getTimer()

{

return timer;

}

public List<GameEnemy> GetEnemies()

{

return enemies;

}

public void Timer()

{

timer++;

}

public GameCell getCell(int x, int y)

{

return grid.getCell(x, y);

}

public void addEnemyInList(GameEnemy enemy)

{

enemies.Add(enemy);

}

private void printMaze(GameGrid grid)

{

for (int i = 0; i < grid.Rows; i++)

{

for (int j = 0; j < grid.Cols; j++)

{

GameCell cell = grid.getCell(i, j);

gameGUI.Controls.Add(cell.PictureBox);

}

}

}

public static void addScore(int increment)

{

score += increment;

}

public void increaseLevel()

{

level++;

}

public static void decreasePlayerHealth(int decrement)

{

playerHealth -= decrement;

}

public static void decreaseHorizontalEnemyHealth(int decrement)

{

horizontalEnemyHealth -= decrement;

}

public static void decreaseVerticalEnemyHealth(int decrement)

{

verticalEnemyHealth -= decrement;

}

public static void decreaseRandomEnemyHealth(int decrement)

{

randomEnemyHealth -= decrement;

}

public void produceHeartRandomly()

{

if (timer % 16 == 0 && playerHealth < 5)

{

Random rand = new Random();

int x = rand.Next(14);

int y = rand.Next(30);

GameCell cell = grid.getCell(x, y);

if (cell.CurrentGameObject.GameObjectType == GameObjectType.NONE)

{

cell.setGameObject(ImageGiver.getHeartObject());

}

}

}

public void storeInFile(string path)

{

if (File.Exists(path))

{

StreamWriter file = new StreamWriter(path);

file.WriteLine(player.CurrentCell.X + "," + player.CurrentCell.Y);

file.WriteLine(score + "," + playerHealth );

if (level == 1)

file.WriteLine(level + "," + verticalEnemyHealth);

else if (level == 2)

file.WriteLine(level + "," + horizontalEnemyHealth);

else if (level == 3)

file.WriteLine(level + "," + randomEnemyHealth);

file.Flush();

file.Close();

}

}

public void readFromFile(string path)

{

int line = 1;

string record;

if (File.Exists(path))

{

StreamReader file = new StreamReader(path);

while ((record = file.ReadLine()) != null)

{

string[] splittedRecord = record.Split(',');

if(line == 1)

{

int X = int.Parse(splittedRecord[0]);

int Y = int.Parse(splittedRecord[1]);

player.CurrentCell = grid.getCell(X, Y);

}

if(line == 2)

{

score = int.Parse(splittedRecord[0]);

playerHealth = int.Parse(splittedRecord[1]);

}

if(line == 3)

{

level = int.Parse(splittedRecord[0]);

if (level == 1)

verticalEnemyHealth = int.Parse(splittedRecord[1]);

else if (level == 2)

horizontalEnemyHealth = int.Parse(splittedRecord[1]);

else if (level == 3)

randomEnemyHealth = int.Parse(splittedRecord[1]);

}

line++;

}

}

}

}

# **Wireframes:**

These are the wireframes of the game.



Game Interface

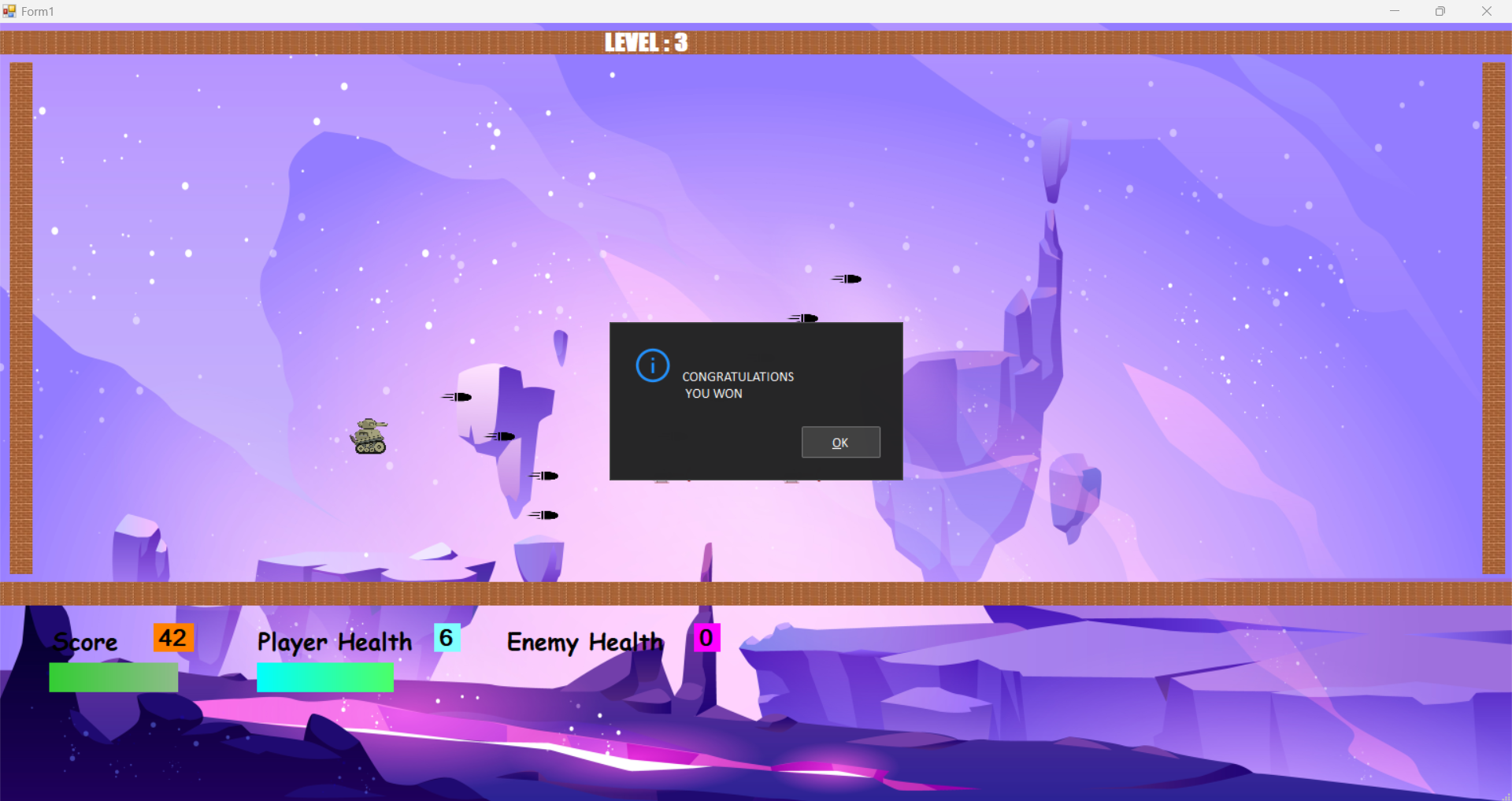


Level 1



Level 2





You Won

# **Class Diagram**