**Tank Wars**

****

Session: 2022 – 2026

**Submitted by:**

Mohammad Abdullah 2022-CS-155

**Supervised by:**

Prof. Dr. Muhammad Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table of Contents**

**Introduction………………………………………………………………………………...……3**

**Images…………………………………………………………………………………………...…3**

**Class Responsibility Collaboration Card…………………………………………...…9**

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

**Code……………………………………………………………………………………………….11**

**Conclusion……........……………………………………………………………..……........…..40**

**Project Link**

**https://youtu.be/uwTlOU4sttY**

* **Introduction**
* **Story**

Your city has been attacked by the enemy forces. You are outnumbered by the opposition. The army has been defeated. The protection and safety of the whole city and its people depend on your shoulders now. You will have to safeguard the city. Gather your courage and wisdom to fight the enemies and kill them to free your city from the shackles of defeat and insult.

* **Objectives**

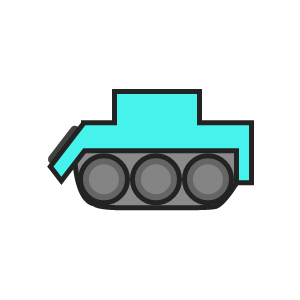
The objectives of the game is to defend your city from the onslaught of the opposing army by using your skills, wisdom and courage. Each tank defeated brings you closer to the victory. Your mission is to destroy the enemies while not putting yourself down.

* **Challenges**

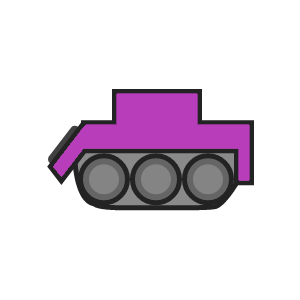
You will face numerous challenges while playing this game. You are outnumbered by the enemy forces. You will also have to tackle all the enemies within the full capacity of your health. You will have to face different kinds of enemies with different kinds of movements. Will you rise to the challenge and become the hero the city needs, Commander?

* **Images**
  + **Characters**

****

**Figure 1 : Player**

**Figure 2 : Vertical Enemy**

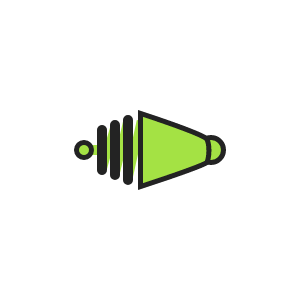
****

**Figure 3 : Horizontal Enemy**

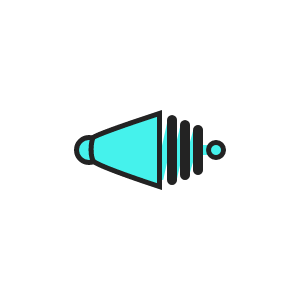
**Figure 4 : Random Enemy**

**Figure 5 : Smart Enemy**

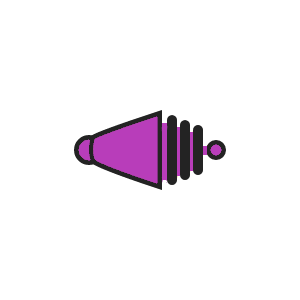
* + **Bullets**

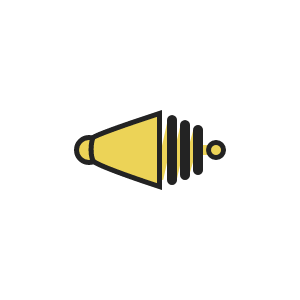
****

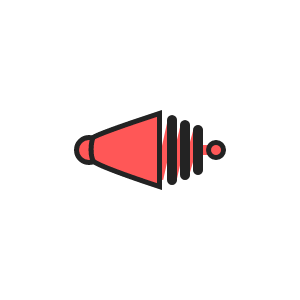
**Figure 6 : Player Bullet**

******

**Figure 7 : Vertical Enemy Bullet**

****

****Figure 8 : Horizontal Enemy Bullet**

**Figure 9 : Random Enemy Bullet**

**Figure 10 : Smart Enemy Bullet**

* + **Wireframes**

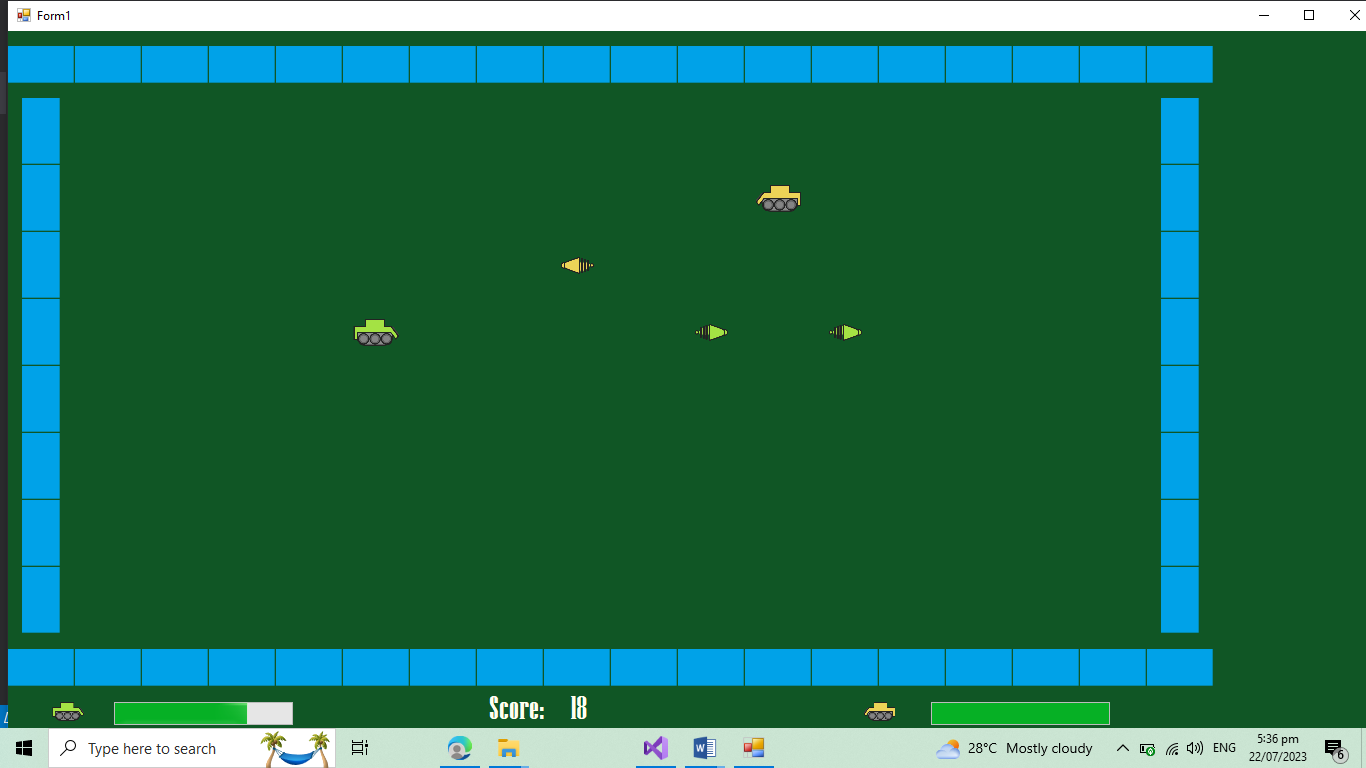
**Figure 11 : Main Menu**



**Figure 12 : Fight with 1st Enemy (Vertical)**

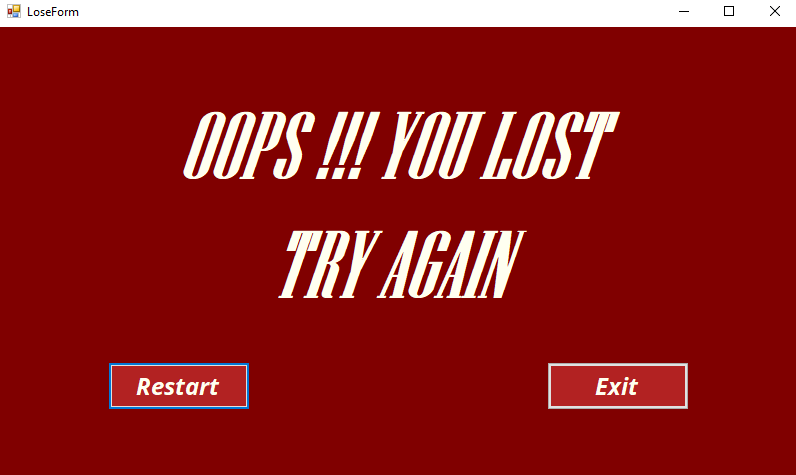
****

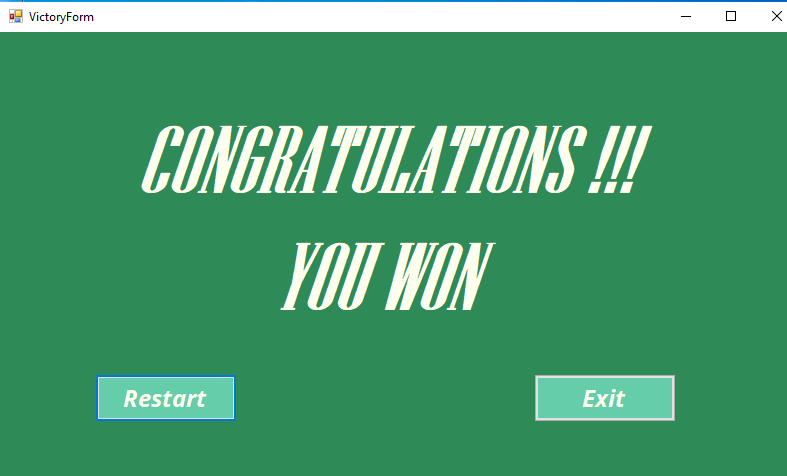
**Figure 13 : Fight with 2nd Enemy (Horizontal)**

******

****Figure 14 : Fight with 3rd Enemy (Random)**

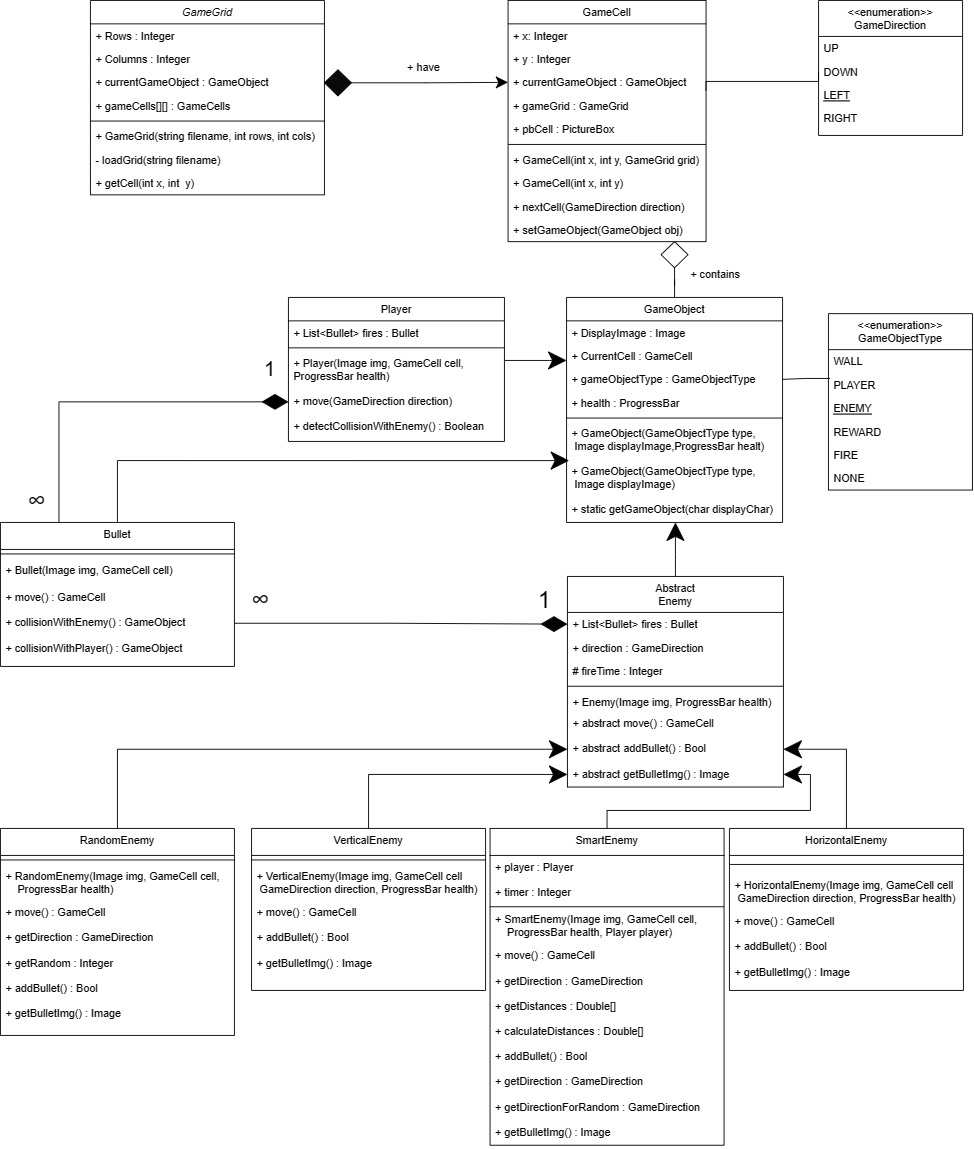
**Figure 15 : Fight with 4th Enemy (Smart)**

******

**Figure 16 : Defeat**

**Figure 17 : Victory**

* **Class Responsibility Collaboration Card**

****

* **Object-Oriented Programming**
  + **Association**

Association is the relation between classes of objects that allows one object to cause another to perform an action on its behalf. The association is used in the game framework in the class GameCell where grid’s association is provided to it because a cell should know about its surroundings.

* + **Composition**

Composition means that the lifetime of an object of a specific class cannot exist outside the lifetime of container object. Composition is used at three places. GameGrid contains GameCells and bullets’ lists are contained inside the player and enemies class. GameGrid cannot exist without GameCells, so composition is applied here. Same goes with bullets, as the respective bullets should not exist without the lifetime of respective class.

* + **Inheritance**

Inheritance is the extension of a class. Inheritance is used at two places. One is the extension of GameObject. It is extended by Player and Enemy. Player and Enemy exist as GameObjects and they require additional functionality to work, so inheritance is applied here. Secondly, Enemy is further extended to Horizontal, Vertical, Random and Smart Enemy. All these enemies have different moving functions and they are related to the Enemy class. So, that is the reason inheritance is applied here.

* + **Polymorphism**

Polymorphism is the use of a function in different ways. In this particular framework, polymorphism is applied in the enemy and its child classes. There is a abstract function of move in the enemy class whose functionality is provided by the child classes. So it is decided at the run-time that which object of Enemy will move according to which function. It is not decided at the compile-time

* + **Abstraction**

Abstraction is the hiding of the unnecessary details. The enemy class is made an abstract class because it contains an abstract function of move. Its functionality must be provided by its heirs. Its functionality cannot be provided by the enemy class. That is the reason, this class is made an abstract class. This is how abstraction has been used in the framework

* **Code**
  + **Game Grid**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

public class GameGrid

{

public

int Rows;

public

int Cols;

public

GameObject currentGameObject;

public

GameCell[, ] gameCells;

public GameGrid(string filename, int rows, int cols)

{

gameCells = new GameCell[rows, cols];

this.Rows = rows;

this.Cols = cols;

loadGrid(filename);

}

private void loadGrid(string filename)

{

StreamReader file = new StreamReader(filename);

string record;

for (int row = 0; row < this.Rows; row++)

{

record = file.ReadLine();

for (int cols = 0; cols < this.Cols; cols++)

{

GameCell cell = new GameCell(row, cols, this);

GameObjectType type = GameObject.getGameObjectType(record[cols]);

Image image = Game.getImage(record[cols]);

GameObject gameObject = new GameObject(type, image);

cell.setGameObject(gameObject);

gameCells[row, cols] = cell;

}

}

file.Close();

}

public GameCell getCell(int x, int y)

{

if (gameCells[x, y] != null)

{

return gameCells[x, y];

}

else

{

return null;

}

}

}

}

* + **Game Cell**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

namespace Game\_Framework.GL

{

public class GameCell

{

public int X;

public int Y;

public GameGrid grid;

public GameObject CurrentGameObject;

PictureBox pictureBox;

const int width = 67;

const int height = 67;

public PictureBox PictureBox

{

get = > pictureBox;

set = > pictureBox = value;

}

public GameCell(int x, int y, GameGrid grid)

{

this.Y = y;

this.X = x;

this.grid = grid;

pictureBox = new PictureBox();

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

pictureBox.Top = x \* width;

pictureBox.Left = y \* height;

pictureBox.SizeMode = PictureBoxSizeMode.Zoom;

pictureBox.BackColor = Color.Transparent;

pictureBox.Visible = true;

}

public GameCell(int x, int y)

{

this.X = x;

this.Y = y;

}

public GameCell nextCell(GameDirection direction)

{

if (direction == GameDirection.Up)

{

if (this.X > 0)

{

GameCell ncell = grid.getCell(X - 1, Y);

if (ncell.CurrentGameObject.type != GameObjectType.WALL)

{

return ncell;

}

}

}

if (direction == GameDirection.Down)

{

if (this.X < grid.Rows - 1)

{

GameCell ncell = grid.getCell(X + 1, Y);

if (ncell.CurrentGameObject.type != GameObjectType.WALL)

{

return ncell;

}

}

}

if (direction == GameDirection.Left)

{

if (this.Y > 0)

{

GameCell ncell = grid.getCell(X, Y - 1);

if (ncell.CurrentGameObject.type != GameObjectType.WALL)

{

return ncell;

}

}

}

if (direction == GameDirection.Right)

{

GameCell ncell = grid.getCell(X, Y + 1);

if (this.Y < grid.Cols - 1)

{

if (ncell.CurrentGameObject.type != GameObjectType.WALL)

{

return ncell;

}

}

}

return this; // if can not return next cell return its own reference

}

public PictureBox getPbCell()

{

return pictureBox;

}

public PictureBox returnPbCell()

{

return pictureBox;

}

public Image getImage()

{

return CurrentGameObject.DisplayImage;

}

public void setGameObject(GameObject gameObject)

{

CurrentGameObject = gameObject;

pictureBox.Image = gameObject.DisplayImage;

}

}

}

* + **Game Object**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

public class GameObject

{

public Image DisplayImage;

private GameCell currentCell;

public GameObjectType type;

public ProgressBar health;

public GameObject(GameObjectType type, Image displayImage, ProgressBar health)

{

this.type = type;

DisplayImage = displayImage;

this.health = health;

}

public GameObject(GameObjectType type, Image displayImage)

{

this.type = type;

DisplayImage = displayImage;

}

public static GameObjectType getGameObjectType(char displayCharacter)

{

if (displayCharacter == '|' || displayCharacter == '\*')

{

return GameObjectType.WALL;

}

if (displayCharacter == '.')

{

return GameObjectType.REWARD;

}

return GameObjectType.NONE;

}

public GameCell CurrentCell

{

get = > currentCell;

set

{

currentCell = value;

currentCell.setGameObject(this);

}

}

}

}

* + **Player**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

class Player : GameObject

{

public List<Bullet> playerFires = new List<Bullet>();

public Player(Image img, GameCell startCell, ProgressBar health) : base(GameObjectType.PLAYER, img, health)

{

CurrentCell = startCell;

}

public GameCell move(GameDirection direction)

{

GameCell currentCell = this.CurrentCell;

GameCell nextCell = currentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

return nextCell;

public bool detectCollisionWithEnemy()

{

if (CurrentCell.CurrentGameObject.type == GameObjectType.ENEMY)

{

return true;

}

return false;

}

}

}

* + **Enemy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

namespace Game\_Framework.GL

{

public abstract class Enemy : GameObject

{

protected int fireTime = 0;

public List<Bullet> Fires = new List<Bullet>();

public Enemy(Image display, ProgressBar health) : base(GameObjectType.ENEMY, display, health)

{

this.health = health;

}

public GameDirection direction;

public abstract GameCell move();

public abstract bool addBullet();

public abstract Image getBulletImage();

}

}

}

return nextCell;

}

public override Image getBulletImage()

{

return Properties.Resources.fire\_purple;

}

public override bool addBullet()

{

if (fireTime == 15)

{

fireTime = 0;

return true;

}

else

{

fireTime++;

return false;

}

}

}

}

* + **Horizontal Enemy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

class HorizontalEnemy : Enemy

{

public HorizontalEnemy(Image character, GameCell startCell, GameDirection direction, ProgressBar health) : base(character, health)

{

this.CurrentCell = startCell;

this.direction = direction;

}

public override GameCell move()

{

GameCell currentCell = this.CurrentCell;

GameCell nextCell = CurrentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (nextCell == currentCell)

{

if (direction == GameDirection.Left)

{

direction = GameDirection.Right;

}

else if (direction == GameDirection.Right)

{

direction = GameDirection.Left;

}

}

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

return nextCell;

}

public override Image getBulletImage()

{

return Properties.Resources.fire\_purple;

}

public override bool addBullet()

{

if (fireTime == 15)

{

fireTime = 0;

return true;

}

else

{

fireTime++;

return false;

}

}

}

}

* + **Vertical Enemy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

class VerticalEnemy : Enemy

{

public VerticalEnemy(Image character, GameCell startCell, GameDirection direction, ProgressBar health) : base(character, health)

{

this.CurrentCell = startCell;

this.direction = direction;

}

public override GameCell move()

{

GameCell currentCell = this.CurrentCell;

GameCell nextCell = CurrentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (nextCell == currentCell)

{

if (direction == GameDirection.Up)

{

direction = GameDirection.Down;

}

else if (direction == GameDirection.Down)

{

direction = GameDirection.Up;

}

}

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

return nextCell;

}

public override Image getBulletImage()

{

return Properties.Resources.fire\_cyan;

}

public override bool addBullet()

{

if (fireTime == 20)

{

fireTime = 0;

return true;

}

else

{

fireTime++;

return false;

}

}

}

}

* + **Random Enemy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

class RandomEnemy : Enemy

{

public RandomEnemy(Image character, GameCell startCell, ProgressBar health) : base(character, health)

{

this.CurrentCell = startCell;

}

public override GameCell move()

{

direction = getDirection();

GameCell currentCell = this.CurrentCell;

GameCell nextCell = CurrentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

return nextCell;

}

public int getRandom()

{

Random r = new Random();

int value = r.Next(4);

return value;

}

public GameDirection getDirection()

{

int value = getRandom();

if (value == 0)

{

return GameDirection.Left;

}

else if (value == 1)

{

return GameDirection.Right;

}

else if (value == 2)

{

return GameDirection.Up;

}

else

{

return GameDirection.Down;

}

}

public override Image getBulletImage()

{

return Properties.Resources.fire\_yellow;

}

public override bool addBullet()

{

if (fireTime == 10)

{

fireTime = 0;

return true;

}

else

{

fireTime++;

return false;

}

}

}

}

* + **Smart Enemy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Framework.GL

{

class SmartEnemy : Enemy

{

public Player player;

int timer = 0;

public SmartEnemy(Image character, GameCell startCell, Player player, ProgressBar health) : base(character, health)

{

this.CurrentCell = startCell;

this.player = player;

}

public override GameCell move()

{

if (timer == 5)

{

direction = getDirection();

GameCell currentCell = this.CurrentCell;

GameCell nextCell = CurrentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

timer = 0;

return nextCell;

}

else

{

direction = getDirectionForRandom();

GameCell currentCell = this.CurrentCell;

GameCell nextCell = CurrentCell.nextCell(direction);

this.CurrentCell = nextCell;

if (currentCell != nextCell)

{

currentCell.setGameObject(Game.getBlankGameObject());

}

timer++;

return nextCell;

}

}

public GameDirection getDirection()

{

double[] distance = new double[4];

distance = getDistances();

if (distance[0] <= distance[1] && distance[0] <= distance[2] && distance[0] <= distance[3])

{

return GameDirection.Up;

}

else if (distance[1] <= distance[0] && distance[1] <= distance[2] && distance[1] <= distance[3])

{

return GameDirection.Down;

}

else if (distance[2] <= distance[0] && distance[2] <= distance[1] && distance[2] <= distance[3])

{

return GameDirection.Left;

}

else if (distance[3] <= distance[0] && distance[3] <= distance[1] && distance[3] <= distance[2])

{

return GameDirection.Right;

}

return GameDirection.Down;

}

public double[] getDistances()

{

double[] list = new double[4];

list[0] = calculateDistance(CurrentCell.X - 1, CurrentCell.Y, player.CurrentCell.X, player.CurrentCell.Y); // Up

list[1] = calculateDistance(CurrentCell.X + 1, CurrentCell.Y, player.CurrentCell.X, player.CurrentCell.Y); // Down

list[2] = calculateDistance(CurrentCell.X, CurrentCell.Y - 1, player.CurrentCell.X, player.CurrentCell.Y); // Left

list[3] = calculateDistance(CurrentCell.X, CurrentCell.Y + 1, player.CurrentCell.X, player.CurrentCell.Y); // Right

return list;

}

static double calculateDistance(int X, int Y, int pX, int pY)

{

return Math.Sqrt(Math.Pow((pX - X), 2) + Math.Pow((pY - Y), 2));

}

public int getRandom()

{

Random r = new Random();

int value = r.Next(4);

return value;

}

public GameDirection getDirectionForRandom()

{

int value = getRandom();

if (value == 0)

{

return GameDirection.Left;

}

else if (value == 1)

{

return GameDirection.Right;

}

else if (value == 2)

{

return GameDirection.Up;

}

else

{

return GameDirection.Down;

}

}

public override Image getBulletImage()

{

return Properties.Resources.fire\_red;

}

public override bool addBullet()

{

if (fireTime == 5)

{

fireTime = 0;

return true;

}

else

{

fireTime++;

return false;

}

}

}

}

* + **Bullet**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Game\_Framework.GL

{

public class Bullet : GameObject

{

public Bullet(Image img, GameCell startCell) : base(GameObjectType.FIRE, img)

{

CurrentCell = startCell;

}

public GameCell move(GameDirection direction)

{

GameCell currentCell = this.CurrentCell;

GameCell nextCell = currentCell.nextCell(direction);

this.CurrentCell = nextCell;

currentCell.setGameObject(Game.getBlankGameObject());

return nextCell;

}

public GameObject collisionWithEnemy()

{

GameCell nxtCell = CurrentCell.nextCell(GameDirection.Right);

if (nxtCell.CurrentGameObject.type == GameObjectType.ENEMY)

{

Enemy ghost = (Enemy)nxtCell.CurrentGameObject;

CurrentCell.setGameObject(Game.getBlankGameObject());

if (ghost.health.Value != 0)

{

CurrentCell.setGameObject(Game.getBlankGameObject());

ghost.health.Value -= 10;

Game.addScore();

}

if (ghost.health.Value == 10)

{

nxtCell.setGameObject(Game.getBlankGameObject());

ghost.health.Value -= 10;

}

return nxtCell.CurrentGameObject;

}

return null;

}

public GameObject collisionWithPlayer()

{

GameCell nxtCell = CurrentCell.nextCell(GameDirection.Left);

if (nxtCell.CurrentGameObject.type == GameObjectType.PLAYER)

{

Player player = (Player)nxtCell.CurrentGameObject;

CurrentCell.setGameObject(Game.getBlankGameObject());

if (player.health.Value != 0)

{

CurrentCell.setGameObject(Game.getBlankGameObject());

player.health.Value -= 5;

}

if (player.health.Value == 5)

{

nxtCell.setGameObject(Game.getBlankGameObject());

player.health.Value -= 5;

}

return nxtCell.CurrentGameObject;

}

return null;

}

}

}

* + **Game**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

namespace Game\_Framework.GL

{

public class Game

{

static int score = 0;

public static void addScore()

{

score++;

}

public static int getScore()

{

return score;

}

public static GameObject getBlankGameObject()

{

GameObject blank = new GameObject(GameObjectType.NONE, Properties.Resources.simple\_box);

return blank;

}

public static Image getImage(char character)

{

Image img = Game\_Framework.Properties.Resources.simple\_box;

if (character == '\*')

{

img = Game\_Framework.Properties.Resources.horizontal;

}

else if (character == '|')

{

img = Game\_Framework.Properties.Resources.vertical;

}

else if (character == 'P')

{

img = Game\_Framework.Properties.Resources.player;

}

else if (character == 'V')

{

img = Game\_Framework.Properties.Resources.enemy\_cyan;

}

else if (character == 'H')

{

img = Game\_Framework.Properties.Resources.enemy\_purple;

}

else if (character == 'R')

{

img = Game\_Framework.Properties.Resources.enemy\_yellow;

}

else if (character == 'S')

{

img = Game\_Framework.Properties.Resources.enemy\_red;

}

return img;

}

}

}

* + **Game Form**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Game\_Framework.GL;

using EZInput;

namespace Game\_Framework

{

public partial class GameForm : Form

{

GameGrid grid = new GameGrid("Maze.txt", 10, 18);

Player player;

VerticalEnemy vEnemy;

HorizontalEnemy hEnemy;

RandomEnemy rEnemy;

SmartEnemy sEnemy;

Enemy enemy;

bool flag;

bool collision;

int enemyCount;

public GameForm()

{

InitializeComponent();

}

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

{

printMaze(grid);

Image playerImg = Game.getImage('P');

GameCell startPlayerCell = grid.getCell(5, 2);

player = new Player(playerImg, startPlayerCell, playerHealthBar);

timerGameLoop.Enabled = true;

flag = false;

enemyCount = 0;

}

private void printMaze(GameGrid grid)

{

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

{

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

{

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

cell.PictureBox.Visible = true;

Controls.Add(cell.PictureBox);

}

}

}

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

{

if (Keyboard.IsKeyPressed(Key.UpArrow))

{

player.move(GameDirection.Up);

}

if (Keyboard.IsKeyPressed(Key.DownArrow))

{

player.move(GameDirection.Down);

}

if (Keyboard.IsKeyPressed(Key.LeftArrow))

{

player.move(GameDirection.Left);

}

if (Keyboard.IsKeyPressed(Key.RightArrow))

{

player.move(GameDirection.Right);

}

if (Keyboard.IsKeyPressed(Key.Space))

{

Image fireImg = Properties.Resources.Fire\_Player;

Bullet fire = new Bullet(fireImg, player.CurrentCell.nextCell(GameDirection.Right));

player.playerFires.Add(fire);

}

enemy = getEnemy();

if (enemy != null)

{

enemy.move();

addEnemyBullet(enemy);

moveEnemyFires(enemy);

}

moveFires();

int score = Game.getScore();

lblScore.Text = score.ToString();

collision = player.detectCollisionWithEnemy();

if (collision == true || playerHealthBar.Value <= 0)

{

LoseForm form = new LoseForm();

form.Show();

this.Hide();

timerGameLoop.Enabled = false;

}

flag = detectVictory();

if (flag == true)

{

VictoryForm form = new VictoryForm();

form.Show();

this.Hide();

timerGameLoop.Enabled = false;

}

}

private bool detectVictory()

{

if (sEnemyHealthBar.Value == 0)

{

return true;

}

return false;

}

private void moveFires()

{

for (var i = 0; i < player.playerFires.Count; i++)

{

Bullet fire = player.playerFires[i];

fire.move(GameDirection.Right);

if (fire.collisionWithEnemy() != null)

{

player.playerFires.Remove(fire);

}

}

}

private void moveEnemyFires(Enemy enemy)

{

for (var i = 0; i < enemy.Fires.Count; i++)

{

Bullet fire = enemy.Fires[i];

fire.move(GameDirection.Left);

if (fire.collisionWithPlayer() != null)

{

enemy.Fires.Remove(fire);

}

}

}

private Enemy getEnemy()

{

if (enemyCount == 0)

{

Image verticalEnemyImg = Game.getImage('V');

GameCell verticalEnemyCell = grid.getCell(4, 15);

vEnemy = new VerticalEnemy(verticalEnemyImg, verticalEnemyCell, GameDirection.Up, vEnemyHealthBar);

SetLblPbAndHealthBarVisibilityFalse();

pbEnemyCyanLbl.Visible = true;

vEnemyHealthBar.Visible = true;

enemyCount++;

}

else if (enemyCount == 1)

{

if (vEnemyHealthBar.Value > 0)

{

return vEnemy;

}

else

{

enemyCount++;

}

}

else if (enemyCount == 2)

{

Image horizontalEnemyImg = Game.getImage('H');

GameCell horizontalEnemyCell = grid.getCell(6, 10);

hEnemy = new HorizontalEnemy(horizontalEnemyImg, horizontalEnemyCell, GameDirection.Left, hEnemyHealthBar);

SetLblPbAndHealthBarVisibilityFalse();

pbEnemyPurpleLbl.Visible = true;

hEnemyHealthBar.Visible = true;

enemyCount++;

}

else if (enemyCount == 3)

{

if (hEnemyHealthBar.Value > 0)

{

return hEnemy;

}

else

{

enemyCount++;

}

}

else if (enemyCount == 4)

{

Image randomEnemyImg = Game.getImage('R');

GameCell randomEnemyCell = grid.getCell(2, 13);

rEnemy = new RandomEnemy(randomEnemyImg, randomEnemyCell, rEnemyHealthBar);

SetLblPbAndHealthBarVisibilityFalse();

pbEnemyYellowLbl.Visible = true;

rEnemyHealthBar.Visible = true;

enemyCount++;

}

else if (enemyCount == 5)

{

if (rEnemyHealthBar.Value > 0)

{

return rEnemy;

}

else

{

enemyCount++;

}

}

else if (enemyCount == 6)

{

Image smartEnemyImg = Game.getImage('S');

GameCell smartEnemyCell = grid.getCell(5, 13);

sEnemy = new SmartEnemy(smartEnemyImg, smartEnemyCell, player, sEnemyHealthBar);

SetLblPbAndHealthBarVisibilityFalse();

sEnemyHealthBar.Visible = true;

pbEnemyRedLbl.Visible = true;

enemyCount++;

}

else if (enemyCount == 7)

{

if (sEnemyHealthBar.Value > 0)

{

return sEnemy;

}

else

{

enemyCount++;

}

}

return null;

}

public void addEnemyBullet(Enemy enemy)

{

if (enemy.addBullet())

{

Bullet bullet = new Bullet(enemy.getBulletImage(), enemy.CurrentCell.nextCell(GameDirection.Left));

enemy.Fires.Add(bullet);

}

}

private void SetLblPbAndHealthBarVisibilityFalse()

{

pbEnemyCyanLbl.Visible = false;

pbEnemyYellowLbl.Visible = false;

pbEnemyPurpleLbl.Visible = false;

pbEnemyRedLbl.Visible = false;

sEnemyHealthBar.Visible = false;

vEnemyHealthBar.Visible = false;

hEnemyHealthBar.Visible = false;

rEnemyHealthBar.Visible = false;

}

}

}

* + **Defeat Form**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

public partial class LoseForm : Form

{

public LoseForm()

{

InitializeComponent();

}

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

{

GameForm form = new GameForm();

form.Show();

this.Hide();

}

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

{

this.Close();

}

}

}

* + **Victory Form**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

public partial class VictoryForm : Form

{

public VictoryForm()

{

InitializeComponent();

}

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

{

GameForm form = new GameForm();

form.Show();

this.Hide();

}

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

{

this.Close();

}

}

}

* + **Main Form**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

public partial class MainMenu : Form

{

public

MainMenu()

{

InitializeComponent();

}

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

{

GameForm form = new GameForm();

form.Show();

this.Hide();

}

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

{

this.Close();

}

}

}

* + **Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Game\_Framework

{

static class Program

{

[STAThread] static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainMenu());

}

}

}

* **Conclusion**

I have made a game framework for Tank Wars game in which tanks fight for the ultimate battle. This framework can be used to add new dimensions to this game using your creativity. This framework has been built keeping in mind the maintainability and scalability of the code. This framework used all the basic and core concepts of object-oriented programming. This game is made using beautiful and appealing assets. I hope that you would enjoy playing and modifying the game.