Final Game Project

Session: 2022 – 2026

Submitted by:

OmarBaig 2022-CS-43

Submitted To:

Prof. Dr. Muhammad Awais Hassan

Department of Computer Science

University of Engineering and Technology Lahore Pakistan

Table Of Content

Project Documentation: Alien Hunt

Introduction

The Alien Hunt project is an implementation of the classic hunting game using C# and Windows Forms. The main objective of the game is to control the hero character to kill the aliens who has attacked his territory and have powerful voids that may give him the unbearable damage and may lead the hero to death it will be challenging for the hero to kill the aliens, who have futuristic weapons. The main character has to kill them while avoiding collisions with them. The game consists of a grid-based game world, where each cell in the grid represents a game object, such as walls, enemies, the player , and fire objects. The game offers interactive gameplay and engaging challenges as the player navigates through the maze-like environment.

OOP Concepts

The project incorporates several Object-Oriented Programming (OOP) concepts to achieve flexibility and code reusability:

1. Encapsulation: The classes in the project encapsulate the data and behavior related to their respective game objects. For example, the **`GameCell`** class encapsulates the properties and methods specific to each cell in the grid, such as setting the game object and obtaining the next cell based on the current direction.

2. Inheritance: The **`GameObject`** class serves as the base class for various game objects, such as walls, food, enemies, and the player. The **`GameCell`** class inherits from the **`GameObject`** class and contains additional information about its position on the grid.

3. Polymorphism: The **`MovingObject`** abstract class represents game objects that can move (such as the player and enemies). It provides abstract methods for movement, allowing different types of moving objects to implement their specific movement logic.

Design Pattern Implementation

The project follows a modular design, dividing functionality into separate classes to ensure maintainability and separation of concerns:

1. Game Class: The `**Game**` class contains static methods to create various game objects, such as blank objects, and player and enemy images. It provides a central point to access these resources throughout the game.

2. GameCell Class: The `**GameCell**` class represents a cell in the game grid. It encapsulates the logic for setting and retrieving the current game object in the cell and calculating the next cell based on the current direction. Each `**GameCell**` instance is associated with a `**PictureBox**`, which allows for displaying the game objects graphically.

3. GameObject Class: The `**GameObject**` class represents the base class for all game objects. It contains properties and methods common to all game objects, such as the object type and the associated image.

4. GameGrid Class: The `**GameGrid**` class represents the entire grid-based game world. It reads the game layout from a text file and creates `**GameCell**` objects accordingly. This class is responsible for loading the game environment and providing access to individual cells.

5. MovingObject Class: The `**MovingObject**` abstract class represents game objects that can move (such as the player and enemies). It inherits from the `**GameObject**` class and provides abstract methods for movement, allowing individual moving objects to implement their specific movement logic.

6. Player Class: The `**Player**` class represents the player-controlled character. It inherits from the `**MovingObject**` class and implements the movement logic for the player.

7. Enemy Class: The `**Enemy**` class represents the enemies in the game. It inherits from the `**MovingObject**` class and implements the movement logic for the enemies. The `**randomEnemyI**` interface is used to support random movement for enemies.

8. Fire Class: The `**Fire**` class represents the fire objects in the game. It inherits from the `**MovingObject**` class and implements the movement logic for the fire objects.

Class Details

Game Class

The `**Game**` class provides static methods to create various game objects:

- `**getBlankGameObject**()`: Returns a blank `**GameObject**` representing an empty cell on the game grid.

- `**getGameObjectImage**(char **displayCharacter**)`: Returns the appropriate image for a given character representing different game objects. The method maps specific characters to corresponding images, such as walls, food, enemies, and the player.

GameCell Class

The `**GameCell**` class represents a cell in the game grid:

- `**GameCell**(int row, int col, **GameGrid grid**)`: Constructor to create a `**GameCell**` instance at the specified row and column coordinates, associated with the given `**GameGrid**`.

- `**setGameObject**(**GameObject gameObject**)`: Sets the current game object in the cell and updates the `**PictureBox**` image accordingly.

- `**nextCell**(**GameDirection direction**)`: Returns the next cell in the specified direction from the current cell, considering the grid boundaries and any obstacles (walls) in the way.

- Properties: Provides access to the row, column, current game object, and `**PictureBox**` of the cell.

GameObject Class

The `**GameObject**` class represents the base class for all game objects:

- `**GameObject**(**GameObjectType** type, **Image** image)`: Constructor to create a `**GameObject**` instance with the specified game object type and associated image.

- `**GameObject**(**GameObjectType** type, **char** displayCharacter)`: Constructor to create a `**GameObject**` instance with the specified game object type and display character.

- `**getGameObjectType**(**char** displayCharacter)`: Static method to determine the `**GameObjectType**` based on the provided display character.

- **Properties**: Provides access to the display character and game object type.

GameGrid Class

The `**GameGrid**` class represents the entire grid-based game world:

- `**GameGrid**(**string** fileName, **int** rows, **int** cols)`: Constructor to create a `**GameGrid**` instance by loading the game layout from

a text file and creating `**GameCell**` objects for each cell.

- `**getCell**(int x, int y)`: Returns the `**GameCell**` instance at the specified row and column coordinates.

- Properties: Provides access to the number of rows and columns in the grid.

MovingObject Class

The `**MovingObject**` abstract class represents game objects that can move (such as the player and enemies):

- `**MovingObject**(Image image, GameCell startCell, GameObjectType gameObjectType)`: Constructor to create a `**MovingObject**` instance with the specified image, starting cell, and game object type.

- `**move**(GameDirection direction)`: Abstract method to be implemented by subclasses for movement in the specified direction.

- `**move**()`: Abstract method to be implemented by subclasses for movement.

Player Class

The `**Player**` class represents the player-controlled hero:

- `**Player**(Image image, GameCell startCell, int PlayerLife)`: Constructor to create a `**Player**` instance with the specified image, starting cell, and initial number of lives.

- `**move**(GameDirection direction)`: Implements movement for the player in the specified direction. It calls the base class's `**move**(GameDirection direction)` method.

- `**move**()`: Overrides the base class's `**move**()` method with an empty implementation.

- `**getCurrentCell**()`: Returns the current cell of the player.

- `**getPlayerLife**()`: Returns the current number of lives of the player.

- `**setPlayerLife**(int i)`: Updates the number of lives of the player.

Enemy Class

The `**Enemy**` class represents the enemies in the game:

- `**Enemy**(Image image, GameCell startCell, int lifes)`: Constructor to create an `Enemy` instance with the specified image, starting cell, and initial number of lives.

- `**move**()`: Implements movement for the enemy. The movement is based on the current direction, and the enemy changes direction if there is an obstacle in the way.

- `**randomMove**()`: Implements random movement for the enemy. The enemy randomly chooses one of the available directions to move.

- `**move**(GameDirection direction)`: Overrides the base class's `**move**(GameDirection direction)` method with an empty implementation since enemies use the default `**move**()` method for movement.

- `**removeEnemyFromList**(int i)`: Removes the enemy at the specified index from the list of enemies.

- `**setEnemyLife**(int life)`: Updates the number of lives of the enemy.

- `**getEnemyLife**()`: Returns the current number of lives of the enemy.

Fire Class

The `**Fire**` class represents the fire objects in the game:

- `**Fire(**Image image, GameCell startCell)`: Constructor to create a `**Fire**` instance with the specified image and starting cell.

- `**move**(GameDirection direction)`: Implements movement for the fire objects in the specified direction. The fire objects move only one cell in the specified direction.

- `**move**()`: Overrides the base class's `**move**()` method with an empty implementation since fire objects use the `**move**(GameDirection direction)` method for movement.

- `**getFlag**()`: Returns the flag status of the fire object.

- `**setFlag**(bool flag)`: Sets the flag status of the fire object.

Collisions Class

The `**Collisions**` class contains static methods to handle collision detection:

- `**collision**(GameCell playerNextCell)`: Checks if the player's next cell collides with any enemy.

- `**collisionWithFire**()`: Checks if any enemy's cell collides with any fire object. If a collision occurs, reduces the enemy's life and removes the fire object from the list.

- `**collisionWithFireOfPlayer**(Player p)`: Checks if the player's cell collides with any fire object. If a collision occurs, reduces the player's life and removes the fire object from the list.

- `**enemyLifeCheck**()`: Checks the life of each enemy and removes enemies with life <= 0 from the list.

- `**playerLifeCheck**(Player p)`: Checks the life of the player. Returns `true` if the player's life is <= 0, indicating that the game should end.

Form1 Class (Game Form)

The `Form1` class represents the main form of thegame. It handles various game events and user inputs:

- `**player**`: Instance of the player-controlled character.

- `**enemy**`, `**enemy2**`, `**enemy3**`: Instances of the enemy characters.

- `**grid**`: Instance of the game grid that represents the entire game world.

- `**fire**`: Instance of the fire object.

- `**Form1_Load**(object sender, EventArgs e)`: Event handler method to initialize the game environment, create player and enemy objects, and load the maze layout from a text file.

- `**printMaze**(GameGrid grid)`: Helper method to display the game grid on the form.

- `**updateScore**()`: Updates the score on the form by increasing it by 5.

- `**playerHealth**()`: Updates the player's health display on the form and checks if the player's life is <= 0 to end the game.

- `**Enemy1Health**()`, `

Enemy2Health()`, `**bossEnemyHealth**()`: Updates the health displays of the enemies.

- `gameLoop_**Tick**(object sender, EventArgs e)`: Event handler method for the game loop timer. Handles player movement, enemy movement, fire generation, and collision detection during the game.

- `**allFireMovementCommonFunction**()`: Helper method to handle movement and collision checks for all fire objects (both player and enemy fires).

- `**moveEnemy**()`: Helper method to move all enemies and perform random movement for the boss enemy.

- `**collsion**(GameCell playerNextCell)`: Helper method to check for collisions between the player and enemies based on the player's next cell.

- `**gameEnd**()`: Helper method to end the game and show the game end form.

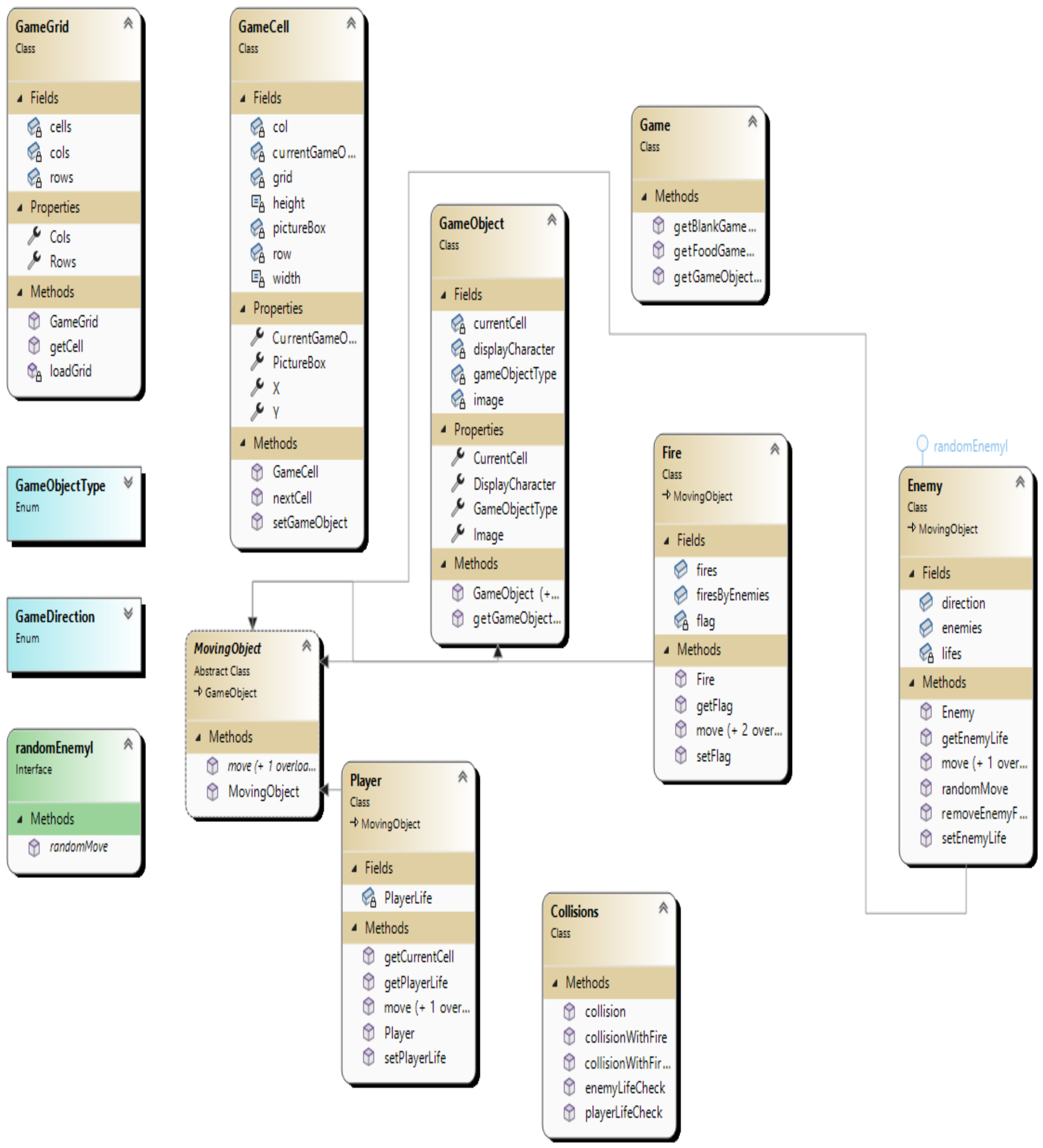
- `**generateFire**()`: Helper method to generate a new fire object when the player presses the 'F' key.

- `**fireTimer**_Tick(object sender, EventArgs e)`: Event handler method for the fire timer. Generates fire objects by enemies at regular intervals.

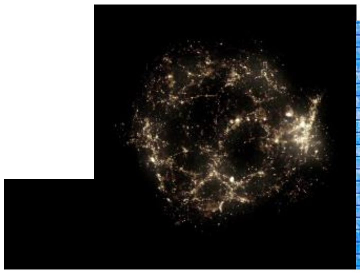
Conclusion

The Game project is a well-organized and object-oriented implementation of the classic hunting game using C# and Windows Forms. It utilizes various OOP concepts, such as encapsulation, inheritance, and polymorphism, abstraction as well as interface to achieve flexibility, code reusability, and maintainability. The implementation follows a modular design pattern, dividing functionality into separate classes and methods to ensure separation of concerns and easy maintenance. The game provides an engaging gameplay experience with player control, enemy movement, and collision detection, making it a fun and interactive game for users to enjoy.

CRC-CARD


GameGrid
Class
Fields
cells
cols
rows
Properties
Cols
Rows
Methods
GameGrid
getCell
loadGrid
GameCell
Class
Fields
col
currentGameO...
grid
height
pictureBox
row
width
Properties
CurrentGameO...
PictureBox
X
Y
Methods
GameCell
nextCell
setGameObject
Game
Class
Methods
getBlankGame...
getFoodGame...
getGameObject...
GameObject
Class
Fields
currentCell
displayCharacter
gameObjectType
image
Properties
CurrentCell
DisplayCharacter
GameObjectType
Image
Methods
GameObject (+...
getGameObject...
GameObjectType
Enum
GameDirection
Enum
randomEnemyI
Interface
Methods
randomMove
MovingObject
Abstract Class
→ GameObject
Methods
move (+ 1 overloa...
MovingObject
Player
Class
→ MovingObject
Fields
PlayerLife
Methods
getCurrentCell
getPlayerLife
move (+ 1 over...
Player
setPlayerLife
Fire
Class
→ MovingObject
Fields
fires
firesByEnemies
flag
Methods
Fire
getFlag
move (+ 2 over...
setFlag
randomEnemyI
Enemy
Class
→ MovingObject
Fields
direction
enemies
lifes
Methods
Enemy
getEnemyLife
move (+ 1 over...
randomMove
removeEnemyF...
setEnemyLife
Collisions
Class
Methods
collision
collisionWithFire
collisionWithFir...
enemyLifeCheck
playerLifeCheck

Assets Images:



Code:

```
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 PacMan.GameGL;

using EZInput;

using PacManGUI.GameGL;

using System.Threading;


namespace PacManGUI

{

   public partial class Form1 : Form

   {
```

```
Player player;

Enemy enemy;

Enemy enemy2;

Enemy enemy3;

GameGrid grid;

private Fire fire;


public Form1()

{

   InitializeComponent();

}

private void Form1_Load(object sender, EventArgs e)

{

    grid = new GameGrid("maze.txt", 35, 70);


   //Loading Images


   Image playerImage = Game.getGameObjectImage('p');

   Image enemyImage = Game.getGameObjectImage('1');

   Image enemyImage2 = Game.getGameObjectImage('2');

   Image enemyImage3 = Game.getGameObjectImage('3');


   //Initializing Coordinates


   GameCell startCellPlayer = grid.getCell(23,30);

   GameCell startCellEnemy = grid.getCell(30, 30);

   GameCell startCellEnemy2 = grid.getCell(33, 30);

   GameCell startCellEnemy3 = grid.getCell(10, 30);


   // calling Objects/Players


   player = new Player(playerImage, startCellPlayer,10);

   enemy = new Enemy(enemyImage, startCellEnemy, 5);
```

```
    enemy2 = new Enemy(enemyImage2, startCellEnemy2, 5);
    enemy3 = new Enemy(enemyImage3, startCellEnemy3, 15);
    Enemy.enemies.Add(enemy);
    Enemy.enemies.Add(enemy2);
    Enemy.enemies.Add(enemy3);
    printMaze(grid);
}



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);
            this.Controls.Add(cell.PictureBox);
        }
    }
}

void updateScore()
{
    int x = int.Parse(Scorelabel1.Text);
    x += 5;
    Scorelabel1.Text =x.ToString();
}
void playerHealth()
{
    label3.Text =player.getPlayerLife().ToString();
    if(player.getPlayerLife() <= 0)
    {
        gameEnd();
    }
}
void Enemy1Health()
{
    label8.Text = enemy.getEnemyLife().ToString();
}
void Enemy2Health()
{
    label6.Text = enemy2.getEnemyLife().ToString();
}
void bossEnemyHealth()
{
    label4.Text = enemy3.getEnemyLife().ToString();
}

private void gameLoop_Tick(object sender, EventArgs e)
{
    moveEnemy();


    Collisions.enemyLifeCheck();
    GameCell playerNextCell;
    allFireMovementCommonFunction();
    if (Keyboard.IsKeyPressed(Key.LeftArrow)) {
        playerNextCell =  player.move(GameDirection.Left);
        collsion(playerNextCell);
    }
    if (Keyboard.IsKeyPressed(Key.RightArrow)){
        playerNextCell = player.move(GameDirection.Right);
        collsion(playerNextCell);
    }
    if (Keyboard.IsKeyPressed(Key.UpArrow)){
        playerNextCell = player.move(GameDirection.Up);
        collsion(playerNextCell);
    }
    if (Keyboard.IsKeyPressed(Key.DownArrow)){
        playerNextCell = player.move(GameDirection.Down);
        collsion(playerNextCell);
    }
    if (Keyboard.IsKeyPressed(Key.F))
    {
        generateFire();
    }

}



public void allFireMovementCommonFunction()
{
    for (int i = 0; i < Fire.fires.Count; i++)
    {
        Fire.fires[i].move(GameDirection.Right);
        Fire.firesByEnemies[i].move(GameDirection.Left);
        if (Collisions.collisionWithFire())
        {
            updateScore();
            Enemy1Health();
            Enemy2Health();
            bossEnemyHealth();
        }
    }
    for (int i = 0; i < Fire.firesByEnemies.Count; i++)
    {
        Fire.firesByEnemies[i].move(GameDirection.Left);
        if (Collisions.collisionWithFireOfPlayer(player))
        {
            playerHealth();
        }
    }
}

public void moveEnemy()
{
    for(int i = 0; i < Enemy.enemies.Count; i++)
    {
        if (Enemy.enemies[i] == enemy3)
        {
            Enemy.enemies[i].randomMove();
        }
        GameCell g = Enemy.enemies[i].move();
    }
}
public void collsion(GameCell playerNextCell)
{
    if(Collisions.collision(playerNextCell))
    {
        gameEnd();
    }
}

void gameEnd()
{
    GameEnd g = new GameEnd();
    this.Hide();
    g.Show();
}
public void generateFire()
{
```

```
            Image fireImage = Game.getGameObjectImage('f');
            GameCell fireStartCell = player.getCurrentCell();
            fireStartCell = grid.getCell(fireStartCell.X, fireStartCell.Y + 1);
            fire = new Fire(fireImage, fireStartCell);
            Fire.fires.Add(fire);
        }
        private void fireTimer_Tick(object sender, EventArgs e)
        {
            foreach(Enemy en in Enemy.enemies)
            {
                GameCell cell = en.CurrentCell;
                Image fireImage = Game.getGameObjectImage('v');
                if(en.direction == GameDirection.Right)
                {
                    cell = grid.getCell(cell.X, cell.Y - 1);
                }
                else if(en.direction == GameDirection.Left)
                {
                    cell = grid.getCell(cell.X, cell.Y - 2 );
                }
                fire = new Fire(fireImage, cell);
                Fire.firesByEnemies.Add(fire);
            }
        }
    }
}
using PacMan.GameGL;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace PacManGUI.GameGL
{
   internal interface randomEnemyI
   {
       GameCell randomMove();
   }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using PacManGUI.GameGL;

namespace PacMan.GameGL
{
    class Player : MovingObject
   {
       int PlayerLife;
       public Player(Image image,GameCell startCell, int PlayerLife) :base (image , startCell, GameObjectType.PLAYER) {
           this.CurrentCell = startCell;
           this.PlayerLife = PlayerLife;
       }
       public override GameCell move(GameDirection direction) {
           GameCell currentCell = this.CurrentCell;
           GameCell nextCell= currentCell.nextCell(direction);
           if (currentCell != nextCell) {
               currentCell.setGameObject(Game.getBlankGameObject());
           }
           this.CurrentCell = nextCell;
           return nextCell;
```

```
        }

        public override GameCell move()
        {
            return null;
        }

        public GameCell getCurrentCell()
        {
            return this.CurrentCell;
        }

        public int getPlayerLife()
        {
            return this.PlayerLife;
        }

        public void setPlayerLife(int i)
        { this.PlayerLife = i; }
    }

}
using PacMan.GameGL;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PacManGUI.GameGL
{
    abstract class MovingObject : GameObject
    {
        public MovingObject(Image image, GameCell startCell, GameObjectType gameObjectType) : base(gameObjectType, image)
        {
            this.CurrentCell = startCell;
        }

        public abstract GameCell move(GameDirection direction);
        public abstract GameCell move();
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PacMan.GameGL
{
   public enum GameObjectType
    {
        WALL,
        PLAYER,
        ENEMY,
        REWARD,
        Fire,
        NONE
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
namespace PacMan.GameGL
{
  public class GameObject
  {
    char displayCharacter;
    GameObjectType gameObjectType;
    GameCell currentCell;
    Image image;
    public GameObject(GameObjectType type, Image image)
    {
      this.gameObjectType = type;
      this.Image = image;
    }
    public GameObject(GameObjectType type, char displayCharacter)
    {
      this.gameObjectType = type;
      this.displayCharacter = displayCharacter;
    }

    public static GameObjectType getGameObjectType(char displayCharacter) {

      if (displayCharacter == '|' || displayCharacter == '%' || displayCharacter == '#') {
        return GameObjectType.WALL;
      }

      if (displayCharacter == '.') {
        return GameObjectType.REWARD;
      }
      return GameObjectType.NONE;
    }
```

```
        public char DisplayCharacter { get => displayCharacter; set => displayCharacter = value; }

        public GameObjectType GameObjectType { get => gameObjectType; set => gameObjectType = value; }

        public GameCell CurrentCell {

            get => currentCell;

            set{

                currentCell = value;

                currentCell.setGameObject(this);

            }

        }

        public Image Image { get => image; set => image = value; }

    }

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace PacMan.GameGL

{

   public enum GameDirection

    {

        Left,

        Right,

        Up,

        Down

    }

}

using System.Windows.Forms;

using System.Drawing;

namespace PacMan.GameGL

{
```

```
public class GameCell
{
    int row;
    int col;
    GameObject currentGameObject;
    GameGrid grid;
    PictureBox pictureBox;
    const int width = 20;
    const int height = 20;
    public GameCell(int row, int col,GameGrid grid) {
        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) {
        currentGameObject = gameObject;
        pictureBox.Image = gameObject.Image;
    }
    public GameCell nextCell(GameDirection direction)
    {
        if (direction == GameDirection.Left) {
            if (this.col > 0) {
                GameCell ncell = grid.getCell(row, col-1);
                if (ncell.CurrentGameObject.GameObjectType != GameObjectType.WALL) {
                    return ncell;
                }
            }
```

```
}
if (direction == GameDirection.Right)
{
    if (this.col < grid.Cols-1)
    {
        GameCell ncell = grid.getCell(this.row, this.col+1);
        if (ncell.CurrentGameObject.GameObjectType != GameObjectType.WALL)
        {
            return ncell;
        }
    }
}

if (direction == GameDirection.Up)
{
    if (this.row > 0)
    {
        GameCell ncell = grid.getCell(this.row-1, this.col);
        if (ncell.CurrentGameObject.GameObjectType != GameObjectType.WALL)
        {
            return ncell;
        }
    }
}

if (direction == GameDirection.Down)
{
    if (this.row < grid.Rows - 1)
    {
        GameCell ncell = grid.getCell(this.row+1, this.col);
        if (ncell.CurrentGameObject.GameObjectType != GameObjectType.WALL)
        {
            return ncell;
```

```
                }
              }
            }
            return this; // if can not return next cell return its own reference
        }
        public int X { get => row; set => row = value; }
        public int Y { get => col; set => col = value; }
        public GameObject CurrentGameObject { get => currentGameObject;}
        public PictureBox PictureBox { get => pictureBox; set => pictureBox = value; }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
namespace PacMan.GameGL
{
    public class Game
    {
        public static GameObject getBlankGameObject(){
            GameObject blankGameObject = new GameObject(GameObjectType.NONE,
PacManGUI.Properties.Resources.simplebox);
            return blankGameObject;
        }
        public static GameObject getFoodGameObject()
        {
            GameObject food = new GameObject(GameObjectType.NONE, PacManGUI.Properties.Resources.pallet);
            return food;
        }
        public static Image getGameObjectImage(char displayCharacter)
        {
```

```
Image img = PacManGUI.Properties.Resources.simplebox;
if (displayCharacter == '|' || displayCharacter == '%')
{
   img = PacManGUI.Properties.Resources.wall;
}
if (displayCharacter == 'v' || displayCharacter == 'V')
{
   img = PacManGUI.Properties.Resources._void;
}
if (displayCharacter == '#')
{
   img = PacManGUI.Properties.Resources.wall;
}
if (displayCharacter == '.')
{
   img = PacManGUI.Properties.Resources.pallet;
}
if (displayCharacter == 'P' || displayCharacter == 'p') {
   img = PacManGUI.Properties.Resources.player;
}
if (displayCharacter == '1' || displayCharacter == '1')
{
   img = PacManGUI.Properties.Resources.enemyA;
}
if (displayCharacter == '2' || displayCharacter == '2')
{
   img = PacManGUI.Properties.Resources.enemyB;
}
if (displayCharacter == '3' || displayCharacter == '3')
{
   img = PacManGUI.Properties.Resources.bossEnemy;
}
if (displayCharacter == 'f' || displayCharacter == 'f')
```

```
            {
                img = PacManGUI.Properties.Resources.fire;
            }
            return img;
        }
    }
}
using PacMan.GameGL;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PacManGUI.GameGL
{
    internal class Fire: MovingObject
    {
        public static List<Fire> fires = new List<Fire>();
        public static List<Fire> firesByEnemies = new List<Fire>();
        private bool flag = false;
        public Fire(Image image, GameCell startCell) : base(image,startCell, GameObjectType.Fire)
        {
            this.CurrentCell = startCell;
        }

        public override GameCell move (GameDirection direction)
        {
            GameCell currentCell = this.CurrentCell;
            GameCell nextCell = currentCell.nextCell(direction);
    if (currentCell != nextCell)

    {

        currentCell.setGameObject(Game.getBlankGameObject());

    }

    if (currentCell == nextCell)

    {

        currentCell.setGameObject(Game.getBlankGameObject());

        Fire.fires.Remove(this);

        return null;

    }

    this.CurrentCell = nextCell;

    return nextCell;

}

public  GameCell move(GameDirection direction, GameCell gamecell)

{

    if(flag == true)

    {

        GameCell currentCell = this.CurrentCell;

        GameCell nextCell = currentCell.nextCell(direction);


        if (currentCell != nextCell)

        {

            currentCell.setGameObject(Game.getBlankGameObject());

        }

        if (currentCell == nextCell)

        {

            currentCell.setGameObject(Game.getBlankGameObject());

            Fire.fires.Remove(this);

            return null;

        }

        this.CurrentCell = nextCell;

        return nextCell;

    }
```

```
            return null;


        }
        public override GameCell move()
        {
            return null;
        }


        public bool getFlag()
        {
            return flag;
        }
        public void setFlag(bool flag) { this.flag = flag; }
    }
}
using PacMan.GameGL;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace PacManGUI.GameGL
{
    class Enemy: MovingObject,randomEnemyI
    {
        public GameDirection direction;
        private int lifes;
        public static List<Enemy> enemies = new List<Enemy>();
        public Enemy(Image image, GameCell startCell,int lifes) : base(image, startCell, GameObjectType.ENEMY)
        {
            this.CurrentCell = startCell;
    this.direction = GameDirection.Left;

    this.lifes = lifes;

}


public override GameCell move()

{

    GameCell currentCell = this.CurrentCell;

    GameCell nextCell = currentCell.nextCell(direction);

    if (nextCell == currentCell || nextCell.CurrentGameObject.GameObjectType == GameObjectType.ENEMY)

    {

        if (direction == GameDirection.Left)

        {

            direction = GameDirection.Right;

        }

        else if (direction == GameDirection.Right)

        {

            direction = GameDirection.Left;

        }

    }

    currentCell.setGameObject(Game.getBlankGameObject());

    this.CurrentCell = nextCell;


    return nextCell;

}


public GameCell randomMove()

{

    GameDirection[] directionArr = { GameDirection.Left, GameDirection.Right, GameDirection.Up, GameDirection.Down };

    Random random = new Random();

    direction = directionArr[random.Next(directionArr.Length)];

    GameCell currentCell = this.CurrentCell;

    GameCell nextCell = currentCell.nextCell(direction);
```

```
            }

            if (nextCell == currentCell || nextCell.CurrentGameObject.GameObjectType == GameObjectType.ENEMY ||
nextCell.CurrentGameObject.GameObjectType == GameObjectType.WALL)

            {

                while ((nextCell == currentCell || nextCell.CurrentGameObject.GameObjectType == GameObjectType.ENEMY ||
nextCell.CurrentGameObject.GameObjectType == GameObjectType.WALL))

                {

                    direction = directionArr[random.Next(directionArr.Length)];

                    nextCell = currentCell.nextCell(direction);

                }


            }


            currentCell.setGameObject(Game.getBlankGameObject());

            this.CurrentCell = nextCell;

            return nextCell;

        }

        public override GameCell move(GameDirection direction)

        {

            return null;

        }


        public void removeEnemyFromList(int i)

        {

            enemies.RemoveAt(i);

        }

        public void setEnemyLife(int life)

        {

            this.lifes = life;

        }

        public int getEnemyLife()

        {

            return lifes;

        }
```

```
    }
}
using PacMan.GameGL;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;

namespace PacManGUI.GameGL
{
    internal class Collisions
    {
        public static bool collision(GameCell playerNextCell)
        {
            foreach(Enemy e in  Enemy.enemies)
            {
                if (e.CurrentCell.PictureBox.Bounds.IntersectsWith(playerNextCell.PictureBox.Bounds))
                {
                    return true;
                }
            }
            return false;
        }
        public static bool collisionWithFire()
        {
            for (int i = Fire.fires.Count - 1; i >= 0; i--)
            {
                foreach (Enemy e in Enemy.enemies)
                {
```

```
            if (e.CurrentCell.PictureBox.Bounds.IntersectsWith(Fire.fires[i].CurrentCell.PictureBox.Bounds))
            {
                e.setEnemyLife(e.getEnemyLife()-1);
                Fire.fires.RemoveAt(i);
                return true;
            }
        }
    }
    return false;
}
public static bool collisionWithFireOfPlayer(Player p)
{
    for (int i = Fire.firesByEnemies.Count - 1; i >= 0; i--)
    {

        if (p.CurrentCell.PictureBox.Bounds.IntersectsWith(Fire.firesByEnemies[i].CurrentCell.PictureBox.Bounds))
        {
            p.setPlayerLife(p.getPlayerLife() - 1);
            Fire.firesByEnemies.RemoveAt(i);
           return true;
        }

    }
    return false;
}

public static void enemyLifeCheck()
{
    for (int i = 0; i < Enemy.enemies.Count; i++)
    {
        if (Enemy.enemies[i].getEnemyLife() <= 0)
        {
            Enemy.enemies.RemoveAt(i);
```

```
                }
            }
        }
        public bool playerLifeCheck(Player p)
        {
            if (p.getPlayerLife() <= 0)
            {
                return true;
            }
            return false;
        }
    }
}
```