Fire Buddy

October 10, 2023

Session: 2022 – 2026

Submitted by:

Muhammad Wali Ahmad 2022-CS-65

Supervised by:

Ma'am Maida Shahid

Department of Computer Science

University of Engineering and Technology

Lahore, Pakistan

Contents

1 Short Description of Game:

Burning with the ambition of world domination, King Kratos of the Kingdom of Sparta with his companions abducted the beautiful princess of Sweetland as the beginning of the invasion and imprisoned them deep inside the castle of the Kingdom of Sparta. Sweetland, who lost the princess who controls the magic of fire, begins too cold little by little. At this rate, the Sweetland will soon disappear! Tom, the fireman, felt unprecedented resentment at being robbed of the princess in front of him. He vowed to rescue the princess from the hands of the villains, and set out for his country that is Sweetland, where the enemies were waiting. The home of the enemies, Sparta Castle, is a 2-store stone tower. A stone-breathing monster is roaming, and Kratos is worried about the fire and built the castle out of stone. It's more like a fortress than a castle because it's durable first, so it's not a very elegant castle. The fireman, who are not good at rocky places, tried to infiltrate the rocky castle of Kratos and his companions with courage.

2 Game Characters Description:

Player

There is one human player in the Game.

Tom:
Tom is the main character in the game and is known for his fireballs. He is brave and loves to save his motherland, always searching the way to save his country from the enemy. Tom is brave, determined, and has a never-say-die spirit. He is the hero of the game, admired for his bravery and determination in the face of danger. Its health is 100.

Enemies

There are 4 enemies in the game.

Botchan:
Botchan is one of the four monsters in the game and is known for his vertical movement in the game. Its health is 50.
Titchi:
Titchi is another monster and is known for his horizontal movement in the game. Its health is also 50 like Botchna.
Sarada:
Sarada is one of the four monsters. It moves randomly when the player is close which reduce the health of Tom. Its health is 100.
Kratos:

Kratos is one of the four monsters in the game and is known for being aggressive and difficult to shake. He is always chasing Tom through the maze, trying to catch him at every turn. Kratos is fast and relentless, making him one of the most dangerous foes that Tom must face. Its health is 200.

3 Game Objects Description:

Following are the Objects in the Game

Door:
A Door, also known as Next Level, is an object used in the Fire Buddy game. When Tom Touches a Door, it causes the Tom to go into Next level, but it only used when all ghost is dead.
Score boosters:
Small Dollars are called "score boosters". When Tom collects a score booster its score increases by 10.
Walls:
Walls are the barriers in the game which the Tom and the monsters cannot cross. In game these are represented by bricks

4 Rules and Interactions:

Following are the rules and interaction of the game:

- Tom loses a life if he collides with any of the monsters.
- If Tom eats score booster, then the Score will increase by 10.
- Score increases when the Tom kills monsters by firing fireballs.
- When the fireballs hit the monster the health of the monster decreases and score increases
- When enemy fire on Tom then his health also decreases.
- When Tom touches the princess in last level then game finished.

5 Goal of the Game:

The goal of the game is to kill all of the monster that have been put across the maze while saving Tom lives and touches the door for next level. In the next level Tom fight with Kratos which is the king of Sparta. After beating him, the cage of princess opened and Tom catch her and game finished.

6 Wireframes of the Game:

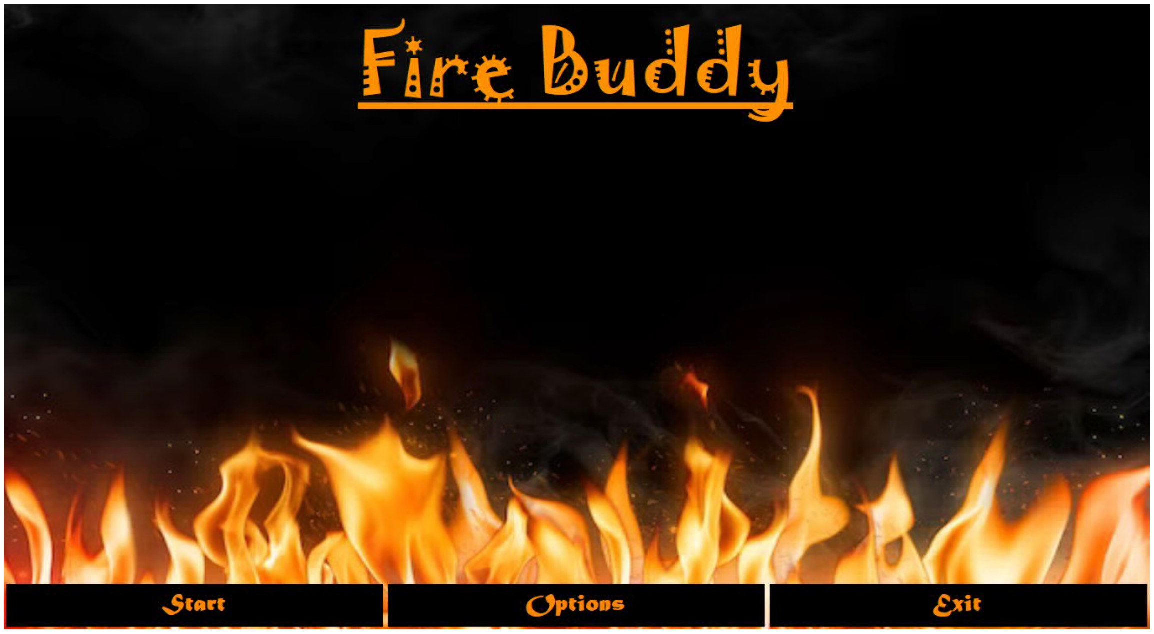


Figure 1: Main Menu

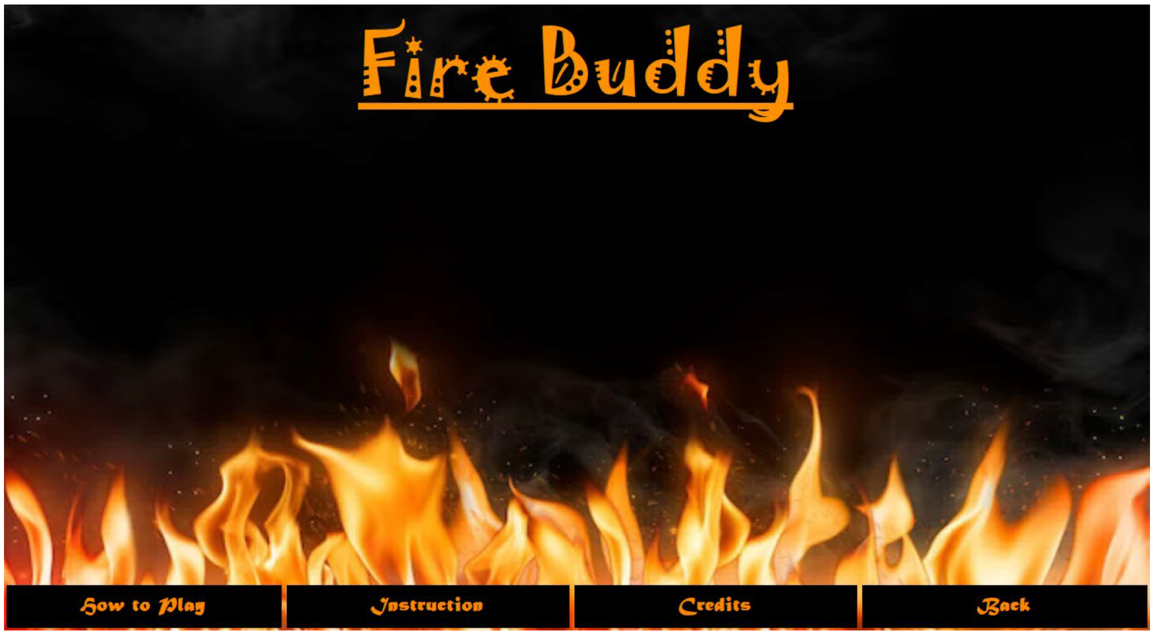
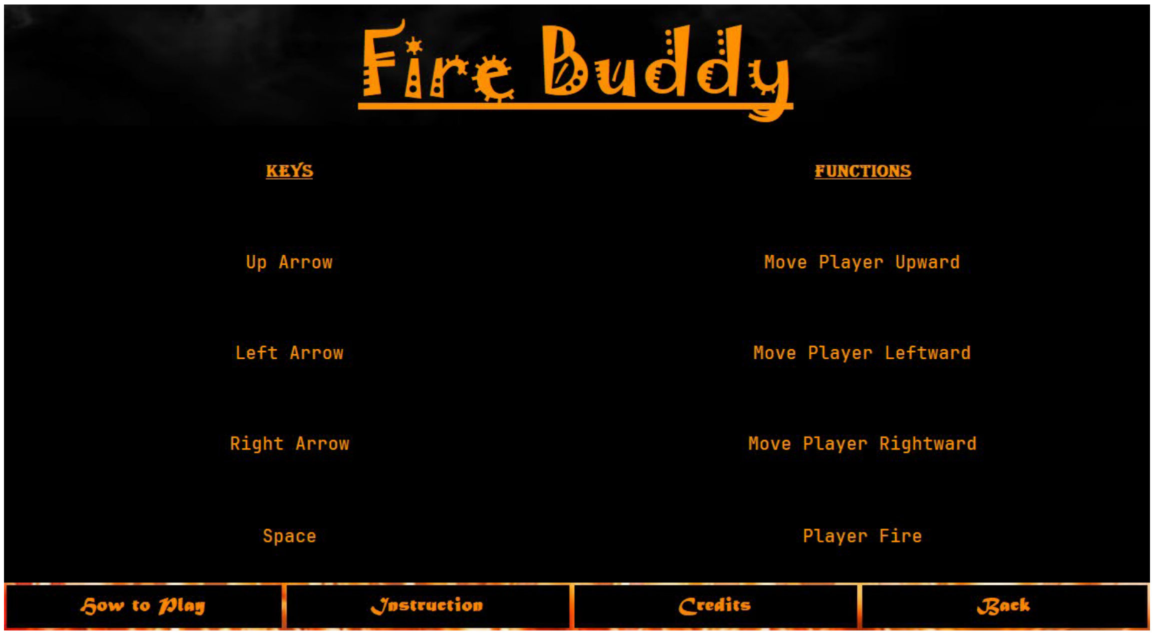
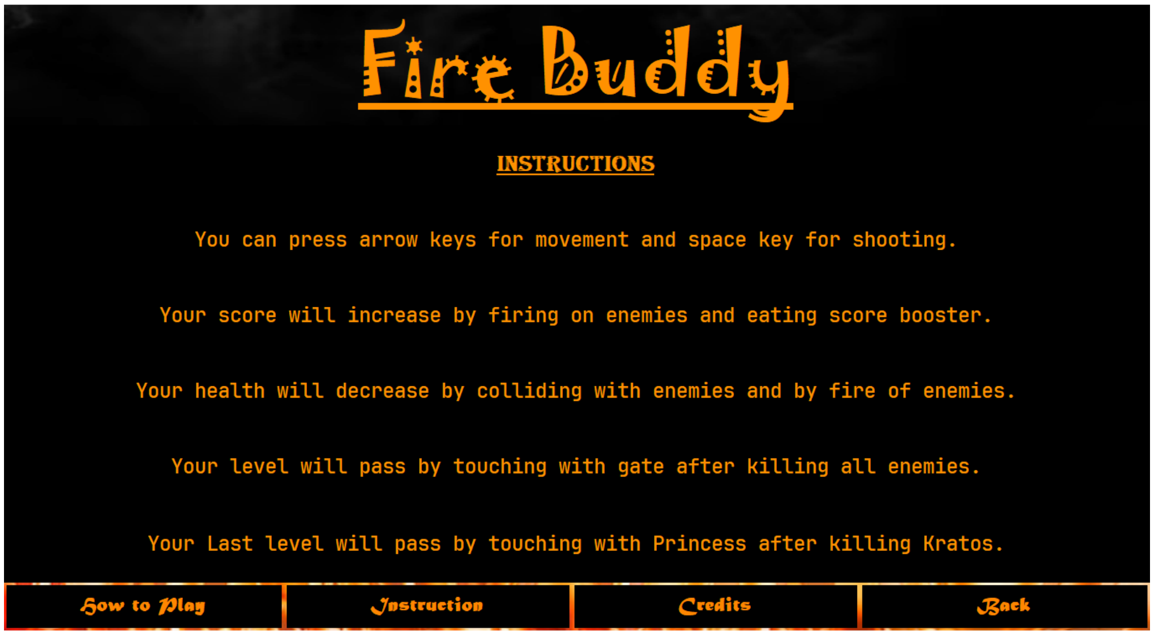


Figure 2: Options



KEYS	FUNCTIONS
Up Arrow	Move Player Upward
Left Arrow	Move Player Leftward
Right Arrow	Move Player Rightward
Space	Player Fire

Figure 3: How to Play



INSTRUCTIONS

You can press arrow keys for movement and space key for shooting.

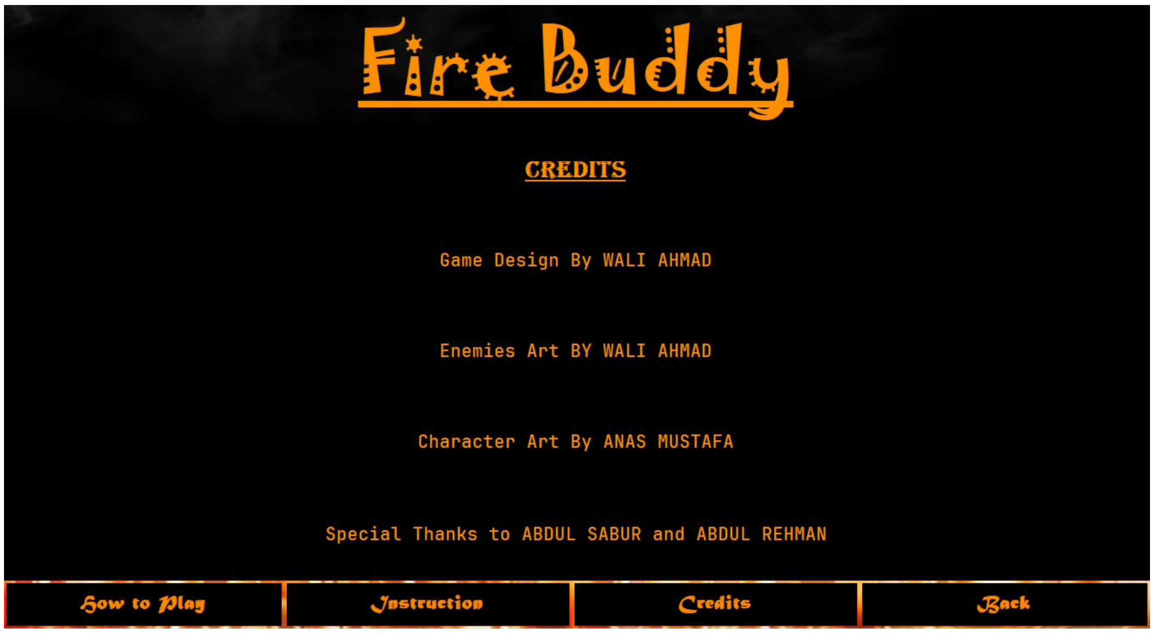
Your score will increase by firing on enemies and eating score booster.

Your health will decrease by colliding with enemies and by fire of enemies.

Your level will pass by touching with gate after killing all enemies.

Your Last level will pass by touching with Princess after killing Kratos.

Figure 4: Instructions



Fire Buddy

CREDITS

Game Design By WALI AHMAD

Enemies Art BY WALI AHMAD

Character Art By ANAS MUSTAFA

Special Thanks to ABDUL SABUR and ABDUL REHMAN

Figure 5: Credits

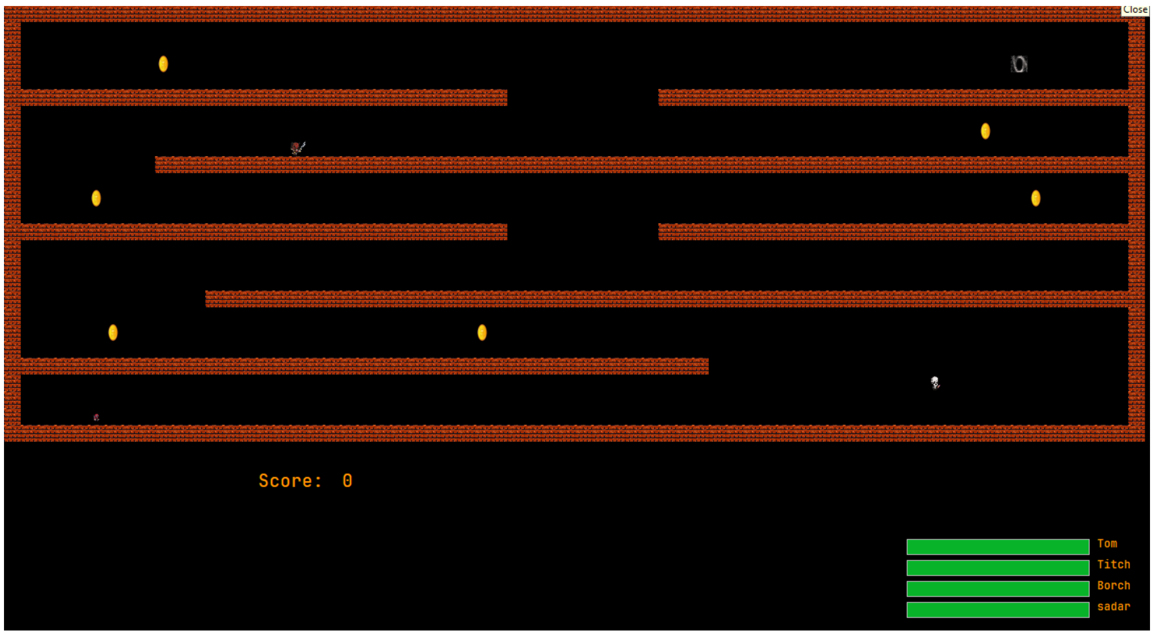
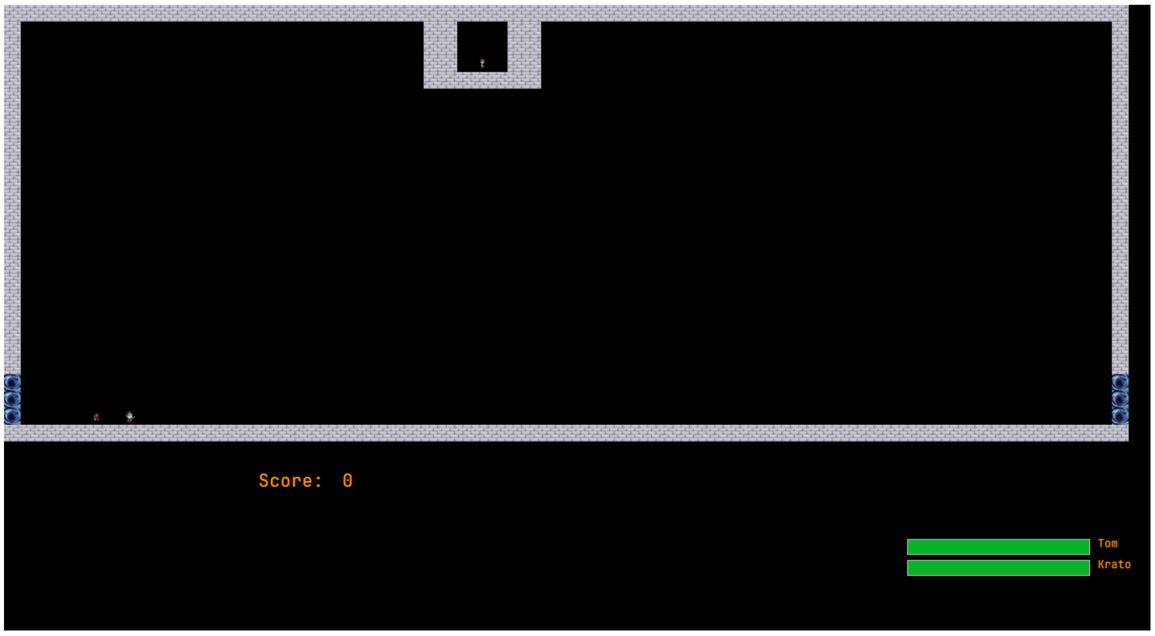


Figure 6: Level 1
Figure 7: Level 2



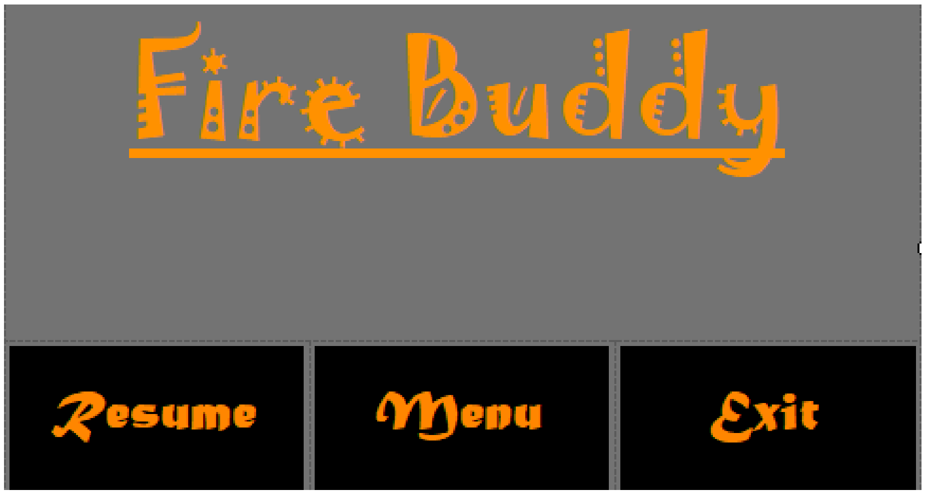


Figure 8: Resume

7 Complete Code:

- **GL Classes:**

Game:

```
public class Game
    {
        public Player player;
        GameGrid grid;
        public List<Ghost> ghosts;
        public List<Fire> Fires = new List<Fire>();
        int score = 0;
        Form gameGUI;
        public Game(string path, Form gameGUI, int X, int
            Y)
        {
            this.gameGUI = gameGUI;
            grid = new GameGrid(path, X, Y);
            Image playerImage = getGameObjectImage('P');
            ghosts = new List<Ghost>();
            GameCell startCell = grid.getCell(24, 5);
            player = new Player(playerImage, startCell);
            printMaze(grid);

        }
        public GameCell getCell(int x, int y)
        {
            return grid.getCell(x, y);
        }
        public void addGhost(Ghost ghost)
        {
            ghosts.Add(ghost);
        }
        public void deleteGhost(Ghost ghost)
        {
            ghosts.Remove(ghost);
        }
        public void addFire(Fire f)
        {
            Fires.Add(f);
        }
        public void RemoveFire()
        {
            for (int i = 0; i < Fires.Count; i++)
            {
                if (Fires[i].Stopped)
                {
                    Fires[i].CurrentCell.setGameObject(
                        getBlankGameObject());
                    Fires.RemoveAt(i);
                }
            }
        }
        public Player getPacManPlayer()
        {
            return player;
        }
        public void addScorePoints(int points)
        {
            this.score = score + points;
        }
        public int getScore()
        {
            return score;
        }
        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);
                    gameGUI.Controls.Add(cell.PictureBox)
                        ;
                }

            }
        }

        public static GameObject getBlankGameObject()
        {
            GameObject blankGameObject = new GameObject(
                GameObjectType.NONE, Properties.Resources.
                simplebox);
            return blankGameObject;
        }
        public Image getVerticalGhostImage()
        {
            return Properties.Resources.VM;
        }

        public Image getHorizontalGhostImage()
        {
            return Properties.Resources.HM;
        }
```

```
public Image getRandomGhostImage()
{
    return Properties.Resources.FM;
}
public Image getSmartGhostImage()
{
    return Properties.Resources.SM;
}

public static Image getGameObjectImage(char
   displayCharacter)
{

    Image img = Properties.Resources.simplebox;


    if (displayCharacter == '#')
    {
        img = Properties.Resources.wall1;
    }

    if (displayCharacter == '$')
    {
        img = Properties.Resources.coin;
    }

    if (displayCharacter == 'P' ||
       displayCharacter == 'p')
    {
        img = Properties.Resources.Player;
    }

    if (displayCharacter == 'O')
    {
        img = Properties.Resources.Hole;
    }
    if (displayCharacter == 'W')
    {
        img = Properties.Resources.wall3;
    }
    if (displayCharacter == '%')
    {
        img = Properties.Resources.wall2;
    }
    if (displayCharacter == 'Q')
```

```
                {
                    img = Properties.Resources.Princess;
                }
                return img;
            }
        }

            GameCell:

    public class GameCell
        {
            int row;
            int col;
            GameObject currentGameObject;
            GameGrid grid;
            PictureBox pictureBox;
            const int width = 20;
            const int height = 20;

            public GameCell()
            {

            }
            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.
                    StretchImage;
                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 && col >
                    0)
```

```
{
    GameCell cell = grid.getCell(row, col -
        1);
    if (cell.CurrentGameObject.GameObjectType
         == GameObjectType.NONE || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.PLAYER || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.HM || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.VM|| cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.RM || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.SM || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.REWARD || cell.
        CurrentGameObject.GameObjectType ==
        GameObjectType.HOLE)
    {
         return cell;
    }
}

if (direction == GameDirection.Right && col <
     grid.Cols - 1)
{
    GameCell cell2 = grid.getCell(row, col +
        1);
    if (cell2.CurrentGameObject.
        GameObjectType == GameObjectType.NONE
         || cell2.CurrentGameObject.
        GameObjectType == GameObjectType.
        PLAYER || cell2.CurrentGameObject.
        GameObjectType == GameObjectType.HM ||
         cell2.CurrentGameObject.
        GameObjectType == GameObjectType.VM ||
         cell2.CurrentGameObject.
        GameObjectType == GameObjectType.RM ||
         cell2.CurrentGameObject.
        GameObjectType == GameObjectType.SM ||
         cell2.CurrentGameObject.
        GameObjectType == GameObjectType.
        REWARD || cell2.CurrentGameObject.
        GameObjectType == GameObjectType.HOLE)
    {
        return cell2;
    }
}

if (direction == GameDirection.Up && row > 0)
{
    GameCell cell3 = grid.getCell(row - 1,
        col);
    if (cell3.CurrentGameObject.
        GameObjectType == GameObjectType.NONE
        || cell3.CurrentGameObject.
        GameObjectType == GameObjectType.
        PLAYER || cell3.CurrentGameObject.
        GameObjectType == GameObjectType.HM ||
         cell3.CurrentGameObject.
        GameObjectType == GameObjectType.VM ||
         cell3.CurrentGameObject.
        GameObjectType == GameObjectType.RM ||
         cell3.CurrentGameObject.
        GameObjectType == GameObjectType.SM ||
         cell3.CurrentGameObject.
        GameObjectType == GameObjectType.
        REWARD || cell3.CurrentGameObject.
        GameObjectType == GameObjectType.HOLE)
    {
        return cell3;
    }
}

if (direction == GameDirection.Down && row <
    grid.Rows - 2)
{
    GameCell cell4 = grid.getCell(row + 1,
        col);
    if (cell4.CurrentGameObject.
        GameObjectType == GameObjectType.NONE
        || cell4.CurrentGameObject.
        GameObjectType == GameObjectType.
        PLAYER || cell4.CurrentGameObject.
        GameObjectType == GameObjectType.HM ||
         cell4.CurrentGameObject.
        GameObjectType == GameObjectType.VM ||
         cell4.CurrentGameObject.
        GameObjectType == GameObjectType.RM ||
         cell4.CurrentGameObject.
        GameObjectType == GameObjectType.SM ||
                cell4.CurrentGameObject.
               GameObjectType == GameObjectType.
               REWARD || cell4.CurrentGameObject.
               GameObjectType == GameObjectType.HOLE)
            {
                return cell4;
            }
        }

        return this;
    }
    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; }
}
```

GameGrid:

```
public class GameGrid
{
    GameCell[,] cells;
    int rows;
    int cols;

    public GameGrid(string fileName, int rows, int
        cols)
    {
        //Numbers of rows and cols should load from
            the text file
        this.rows = rows;
        this.cols = cols;
        cells = new GameCell[rows, cols];
        this.loadGrid(fileName);
    }
    public GameCell getCell(int x, int y)
    {
        return cells[x, y];
    }
    public int Rows { get => rows; set => rows =
        value; }
    public int Cols { get => cols; set => cols =
        value; }
        void loadGrid(string fileName)
        {

            StreamReader fp = new StreamReader(fileName);
            string record;
            for (int row = 0; row < this.rows; row++)
            {
                record = fp.ReadLine();
                for (int col = 0; col < this.cols; col++)
                {
                    GameCell cell = new GameCell(row, col
                        , this);
                    char displayCharacter = record[col];
                    GameObjectType type = GameObject.
                        getGameObjectType(displayCharacter
                        );
                    Image displayIamge = Game.
                        getGameObjectImage(
                        displayCharacter);
                    GameObject gameObject = new
                        GameObject(type, displayIamge);
                    cell.setGameObject(gameObject);
                    cells[row, col] = cell;
                }
            }

            fp.Close();
        }
    }

        GameObject:

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 == '#')
     {
          return GameObjectType.WALL;
     }

     if (displayCharacter == '$')
     {
          return GameObjectType.REWARD;
     }
     if (displayCharacter == 'O')
     {
          return GameObjectType.HOLE;
     }
     if (displayCharacter == 'Q')
     {
          return GameObjectType.QUEEN;
     }
     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; }
    }

        GameDirection:

public enum GameDirection
    {
        Left,
        Right,
        Up,
        Down
    }

        GameObjectType:

public enum GameObjectType
    {
        WALL,
        PLAYER,
        QUEEN,
        HM,
        VM,
        RM,
        SM,
        REWARD,
        HOLE,
        FIRE,
        NONE
    }

        Ghost:

public abstract class Ghost : GameObject
    {
        public Ghost(Image ghostImage,GameObjectType type
           ) : base(type, ghostImage)
        {
        }

        public abstract GameCell nextCell();
        public abstract void move(GameCell gameCell);
    }

        Player:
```

```
public class Player : GameObject
    {
        public Player(Image image, GameCell startCell) :
            base(GameObjectType.PLAYER, image)
        {
            this.CurrentCell = startCell;
        }

        public void move(GameCell gameCell)
        {
            CurrentCell = gameCell;
        }

    }

        Fire:

public class Fire : GameObject
    {
        GameDirection direction;
        bool stopped = false;
        public Fire(Image img, GameCell startCell,
            GameDirection direction) : base(GameObjectType
            .FIRE, img)
        {
            this.direction = direction;
            base.CurrentCell = startCell;
        }

        public bool Stopped { get => stopped; set =>
            stopped = value; }

        public void move(GameCell gameCell)
        {
            if (base.CurrentCell != null)
            {
                base.CurrentCell.setGameObject(Game.
                    getBlankGameObject());
            }
            base.CurrentCell = gameCell;

        }
        public GameCell nextCell()
        {
            GameCell gameCell = base.CurrentCell;
            GameCell gameCell2 = base.CurrentCell.
                nextCell(direction);
            if (gameCell2 == gameCell || gameCell2.
                CurrentGameObject.GameObjectType ==
                GameObjectType.REWARD)
            {
                Stopped = true;
            }
            else
            {
                gameCell = gameCell2;
            }
            return gameCell;
        }
    }

        CollisionDetection:

public class CollisionDetection
    {

        public bool isGhostCollideWithPlayer(Ghost ghost)
        {
            bool flag = false;
            if (ghost.CurrentCell.CurrentGameObject.
                GameObjectType == GameObjectType.PLAYER)
            {
                flag = true;
            }
            return flag;
        }
        public bool isGhostCollideWithBullet(Ghost ghost)
        {
            bool flag = false;
            if (ghost.nextCell().CurrentGameObject.
                GameObjectType == GameObjectType.FIRE)
            {
                flag = true;
            }
            return flag;
        }
        public bool isPlayerCollideWithCoin(GameCell
            potentialCell)
        {
            bool flag = false;
            if (potentialCell.CurrentGameObject.
                GameObjectType == GameObjectType.REWARD)
            {
                flag = true;
            }
            return flag;

        }
        public bool isHMCollideWithBullet(Fire f)
        {
            bool flag = false;
            if (f.nextCell().CurrentGameObject.
                GameObjectType == GameObjectType.HM)
            {
                f.Stopped = true;
                flag = true;
            }
            return flag;
        }
        public bool isVMCollideWithBullet(Fire f)
        {
            bool flag = false;
            if (f.nextCell().CurrentGameObject.
                GameObjectType == GameObjectType.VM)
            {
                f.Stopped = true;
                flag = true;
            }
            return flag;
        }
        public bool isRMCollideWithBullet(Fire f)
        {
            bool flag = false;
            if (f.nextCell().CurrentGameObject.
                GameObjectType == GameObjectType.RM)
            {
                f.Stopped = true;
                flag = true;
            }
            return flag;
        }
        public bool isSMCollideWithBullet(Fire f)
        {
            bool flag = false;
            if (f.nextCell().CurrentGameObject.
                GameObjectType == GameObjectType.SM)
```

```
            {
                f.Stopped = true;
                flag = true;
            }
            return flag;
        }
    }

        HorizontalGhost:

public class HorizontalGhost : Ghost
    {
        private GameDirection direction = GameDirection.
            Left;
        public HorizontalGhost(Image ghostImage, GameCell
             startCell)
            : base(ghostImage,GameObjectType.HM)
        {
            base.CurrentCell = startCell;
        }

        public override void move(GameCell gameCell)
        {
            if (base.CurrentCell != null)
            {
                base.CurrentCell.setGameObject(Game.
                    getBlankGameObject());
            }

            base.CurrentCell = gameCell;
        }

        public override GameCell nextCell()
        {
            GameCell gameCell = base.CurrentCell;
            GameCell gameCell2 = base.CurrentCell.
                nextCell(direction);
            if (gameCell2 == gameCell)
            {
                if (direction == GameDirection.Left)
                {
                    direction = GameDirection.Right;
                }
                else if (direction == GameDirection.Right
                    )
                {
                    direction = GameDirection.Left;
                }
            }
            else
            {
                gameCell = gameCell2;
            }

            return gameCell;
        }
    }

        VerticalGhost:

public class VerticalGhost : Ghost
    {
        GameDirection direction = GameDirection.Down;

        public VerticalGhost(Image ghostImage, GameCell
            startCell) : base(ghostImage,GameObjectType.VM
            )
        {
            this.CurrentCell = startCell;
        }

        public override void move(GameCell gameCell)
        {
            if (this.CurrentCell != null)
            {
                this.CurrentCell.setGameObject(Game.
                    getBlankGameObject());

            }
            CurrentCell = gameCell;
        }

        public override GameCell nextCell()
        {

            GameCell nextCell = this.CurrentCell;

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

            if (potentialNextCell == nextCell)
            {
                if (direction == GameDirection.Up)
                {
                    direction = GameDirection.Down;
                }
                else if (direction == GameDirection.Down)
                {
                    direction = GameDirection.Up;
                }
            }
            else
            {
                nextCell = potentialNextCell;
            }
            return nextCell;

        }

    }
```

RandomGhost:

```
public class RandomGhost : Ghost
    {
        private GameDirection direction = GameDirection.
            Left;
        public RandomGhost(Image ghostImage, GameCell
            startCell)
            : base(ghostImage, GameObjectType.RM)
        {
            base.CurrentCell = startCell;
        }

        public override void move(GameCell gameCell)
        {

            if (CurrentCell != null)
            {
                base.CurrentCell.setGameObject(Game.
                    getBlankGameObject());
            }

            base.CurrentCell = gameCell;

        }

        public override GameCell nextCell()
        {
            GameDirection[] Select = new GameDirection[]
                { GameDirection.Down, GameDirection.Up,
                GameDirection.Right, GameDirection.Left };
            Random number = new Random();
            int i = number.Next(4);
            this.direction = Select[i];
            GameCell gameCell = base.CurrentCell;
            GameCell gameCell2 = base.CurrentCell.
                nextCell(direction);
            if (gameCell2 == gameCell)
            {
                int j = number.Next(4);
                this.direction = Select[j];
            }
            else
            {
                gameCell = gameCell2;
            }

            return gameCell;
        }
    }

        SmartGhost:

public class SmartGhost : Ghost
    {
        private GameDirection direction = GameDirection.
            Left;

        Game g;
        public SmartGhost(Image ghostImage, GameCell
            startCell, Game g)
            : base(ghostImage,GameObjectType.SM)
        {
            base.CurrentCell = startCell;
            this.g = g;
        }
        private GameDirection SetDirection()
        {
            int x = g.getPacManPlayer().CurrentCell.X;
            int y = g.getPacManPlayer().CurrentCell.Y;
            if (x > CurrentCell.X)
            {
                direction = GameDirection.Down;
            }
            else if (y > CurrentCell.Y)
            {
                direction = GameDirection.Right;
            }
            else if (x < CurrentCell.X)
            {
                direction = GameDirection.Up;
            }
            else if (y < CurrentCell.Y)
            {
                direction = GameDirection.Left;
            }
            return direction;
        }
        public override void move(GameCell gameCell)
        {
            if (base.CurrentCell != null)
            {
                base.CurrentCell.setGameObject(Game.
                    getBlankGameObject());
            }

            base.CurrentCell = gameCell;
        }

        public override GameCell nextCell()
        {
            GameCell gameCell2 = base.CurrentCell.
                nextCell(SetDirection());

            return gameCell2;
        }
    }
```

- **UI Classes:**

```
        Main:

public partial class Main : Form
    {
        public Main()
        {
            InitializeComponent();
            setStart(new Start());
            WindowState = FormWindowState.Maximized;
        }
        public void setLevel(Level1 form)
        {
            panel_main.Controls.Clear();
            form.TopLevel = false;
            form.FormBorderStyle = FormBorderStyle.None;
            panel_main.Controls.Add(form);
            form.Dock = DockStyle.Fill;
            panel_main.Tag = form;
            form.menu_btn += Form_menu_btn;
            form.nextlvl_btn += Form_nextlvl_btn;
            form.BringToFront();
            form.Show();
        }
        public void setLevel2(Level2 form)
        {
            panel_main.Controls.Clear();
            form.TopLevel = false;
            form.FormBorderStyle = FormBorderStyle.None;
            panel_main.Controls.Add(form);
            form.Dock = DockStyle.Fill;
            panel_main.Tag = form;
            form.menu_btn += Form_menu_btn1;
            form.BringToFront();
            form.Show();
        }

        private void Form_menu_btn1(object sender,
           EventArgs e)
        {

            panel_main.Controls.Clear();
            setStart(new Start());
        }

        private void Form_nextlvl_btn()
        {
            setLevel2(new Level2());
        }

        private void Form_menu_btn(object sender,
           EventArgs e)
        {
            panel_main.Controls.Clear();
            setStart(new Start());
```

```
        }

        public void setStart(Start form)
        {
            panel_main.Controls.Clear();
            form.TopLevel = false;
            form.FormBorderStyle = FormBorderStyle.None;
            panel_main.Controls.Add(form);
            form.Dock = DockStyle.Fill;
            panel_main.Tag = form;
            form.BringToFront();
            form.start_btn += Form_start_btn;
            form.Show();
        }

        private void Form_start_btn(object sender,
           EventArgs e)
        {
            if (((Start)sender).IsClick)
            {
                setLevel(new Level1());
            }
        }
    }

        Start:

public partial class Start : Form
    {
        public event EventHandler start_btn;

        private bool isClick;
        public bool IsClick
        {
            get => isClick;
        }

        public Start()
        {
            InitializeComponent();
            setMusic();
            option_btns.Hide();
        }
         private void setMusic()
        {
```

```
    SoundPlayer snd = new SoundPlayer(Properties.
       Resources.background);
    snd.PlayLooping();
}
private void btn_exit_Click(object sender,
   EventArgs e)
{
    Environment.Exit(0);
}

private void btn_start_Click(object sender,
   EventArgs e)
{
    isClick = true;
    start_btn?.Invoke(this, e);
    this.Close();
}

private void btn_opt_Click(object sender,
   EventArgs e)
{
    btns.Hide();
    option_btns.Show();
}

private void back_btn_Click(object sender,
   EventArgs e)
{
    sub_panel.Hide();
    option_btns.Hide();
    btns.Show();
}
public void setForm(Form form)
{
    sub_panel.Controls.Clear();
    form.TopLevel = false;
    form.FormBorderStyle = FormBorderStyle.None;
    sub_panel.Controls.Add(form);
    form.Dock = DockStyle.Fill;
    sub_panel.Tag = form;
    form.BringToFront();
    form.Show();
}

private void play_btn_Click(object sender,
   EventArgs e)
```

```
        {
            setForm(new HowtoPlay());
            sub_panel.Show();
        }

        private void instruction_btn_Click(object sender,
            EventArgs e)
        {
            setForm(new Instruction());
            sub_panel.Show();
        }

        private void credit_btn_Click(object sender,
           EventArgs e)
        {
            setForm(new Credits());
            sub_panel.Show();
        }
    }

        Instruction:

public partial class Instruction : Form
    {
        public Instruction()
        {
            InitializeComponent();
        }
    }

        HowToPlay:

public partial class HowtoPlay : Form
    {
        public HowtoPlay()
        {
            InitializeComponent();
        }
    }

        Credits:

public partial class Credits : Form
    {
        public Credits()
        {
            InitializeComponent();
        }
    }

        Level1:

public partial class Level1 : Form
    {
        Game game;
        GameDirection direct = GameDirection.Right;
        CollisionDetection collider;
        char movementStatus = 'd';
        int jumpCount = 0;
        public event EventHandler menu_btn;
        public event Action nextlvl_btn;
        VerticalGhost g1;
        HorizontalGhost g2;
        RandomGhost g3;
        public Level1()
        {
            InitializeComponent();
            game = new Game("Maze1.txt", this, 26, 67);
            collider = new CollisionDetection();
        }

        private void Level1_Load(object sender, EventArgs
             e)
        {
            g1 = new VerticalGhost(game.
                getVerticalGhostImage(), game.getCell(22,
                55));
            g2 = new HorizontalGhost(game.
                getHorizontalGhostImage(), game.getCell
                (16, 14));
            g3 = new RandomGhost(game.getRandomGhostImage
                (), game.getCell(8, 16));

            game.addGhost(g1);
            game.addGhost(g2);
            game.addGhost(g3);
        }

        private void GameLoop_Tick(object sender,
            EventArgs e)
        {
            movePlayer();
            moveGhosts();
            moveDown();
            MoveFire();
            game.RemoveFire();
            if (Keyboard.IsKeyPressed(Key.Escape))
            {
                setMenu();
            }
            checkLevel();
            showScore();
        }
        private void checkLevel()
        {
            if (game.player.CurrentCell.nextCell(direct).
                CurrentGameObject.GameObjectType ==
                GameObjectType.HOLE)
            {
                nextlvl_btn?.Invoke();
                this.Close();
            }
        }
        private void MoveFire()
        {

            foreach (Fire f in game.Fires)
            {
                if (collider.isHMCollideWithBullet(f))
                {
                    HM.Value -= 20;
                    if (HM.Value <= 0)
                    {
                        game.deleteGhost(g2);
                    }
                }
                if (collider.isVMCollideWithBullet(f))
                {
                    VM.Value -= 20;
                    if (VM.Value <= 0)
                    {
                        game.deleteGhost(g1);
                    }
                }
                if (collider.isRMCollideWithBullet(f))
                {
                    RM.Value -= 10;
                    if (RM.Value <= 0)
                    {
                        game.deleteGhost(g3);
                    }
                }
                f.move(f.nextCell());
            }
        }
        private void GhostDelete()
        {

        }

        public void setMenu()
        {
            GameLoop.Stop();
            Menu form = new Menu();
            form.TopLevel = false;
            this.Controls.Add(form);
            form.StartPosition = FormStartPosition.
                CenterScreen;
            form.BringToFront();
            form.resume_btn += Form_resume_btn;
            form.menu_btn += Form_menu_btn;
            form.Show();
        }

        private void Form_menu_btn(object sender,
            EventArgs e)
        {
            this.Close();
            menu_btn?.Invoke(this, e);
            this.Close();
        }

        private void Form_resume_btn(object sender,
            EventArgs e)
        {
            GameLoop.Start();
        }

        public void moveGhosts()
        {
            foreach (Ghost g in game.ghosts)
            {
                if (collider.isGhostCollideWithPlayer(g))
                {
                    if (player.Value > 0)
                    {
                        player.Value -= 10;
                        if(player.Value == 0)
                        {

                        MessageBox.Show("Game over");
                        Environment.Exit(0);
                        }
                    }
                }

                g.move(g.nextCell());


            }
        }

        private void showScore()
        {
            score.Text = game.getScore().ToString();
        }
        private void movePlayer()
        {
            Player player = game.getPacManPlayer();
            GameCell potentialNewCell = player.
                CurrentCell;
            GameCell currentCell = player.CurrentCell;
            if (Keyboard.IsKeyPressed(Key.LeftArrow))
            {
                potentialNewCell = player.CurrentCell.
                    nextCell(GameDirection.Left);
                direct = GameDirection.Left;
                movementStatus = 'd';
            }
            if (Keyboard.IsKeyPressed(Key.RightArrow))
            {
                potentialNewCell = player.CurrentCell.
                    nextCell(GameDirection.Right);
                direct = GameDirection.Right;
                movementStatus = 'd';
            }
            if (Keyboard.IsKeyPressed(Key.UpArrow))
            {
                movementStatus = 'u';
            }
        if (Keyboard.IsKeyPressed(Key.Space))
        {
            GameCell NewCell = player.CurrentCell.
                nextCell(direct);
            Fire f = new Fire(Properties.Resources.
                bullet, NewCell, direct);
            game.addFire(f);
        }
        if (collider.isPlayerCollideWithCoin(
            potentialNewCell))
        {
            game.addScorePoints(1);
        }
        currentCell.setGameObject(Game.
            getBlankGameObject());
        player.move(potentialNewCell);
}
private void moveDown()
{
    Player player = game.player;
    GameCell potentialNewCell = player.
        CurrentCell;
    GameCell nextCell;
    if (movementStatus == 'd')
    {
        nextCell = player.CurrentCell.nextCell(
            GameDirection.Down);
        if (nextCell != potentialNewCell)
        {
            GameCell currentCell = player.
                CurrentCell;
            currentCell.setGameObject(Game.
                getBlankGameObject());
            player.move(nextCell);
        }

    }
    if (movementStatus == 'u')
    {
        for (int i = 0; i < 3; i++)
        {
            potentialNewCell = player.CurrentCell
                .nextCell(GameDirection.Up);
            jumpCount++;
            GameCell currentCell = player.
                CurrentCell;
```

```
                    currentCell.setGameObject(Game.
                        getBlankGameObject());
                    player.move(potentialNewCell);
                }

                if (jumpCount == 3)
                {
                    movementStatus = 'd';

                }
            }
        }
    }

        Level2:

public partial class Level2 : Form
    {

        Game game;
        GameDirection direct = GameDirection.Right;
        CollisionDetection collider;
        char movementStatus = 'd';
        int jumpCount = 0;
        SmartGhost g4;
        public event EventHandler menu_btn;
        public Level2()
        {
            InitializeComponent();
            game = new Game("Maze2.txt", this, 26, 67);
            collider = new CollisionDetection();
        }

        private void Level2_Load(object sender, EventArgs
             e)
        {
            g4 = new SmartGhost(game.getSmartGhostImage()
                , game.getCell(8, 16), game);
            game.addGhost(g4);
        }
        private void MoveFire()
        {

            foreach (Fire f in game.Fires)
            {
                if (collider.isSMCollideWithBullet(f))
        {
            SM.Value -= 20;
            if (SM.Value <= 0)
            {
                game.deleteGhost(g4);
            }
        }
        f.move(f.nextCell());
    }
}
public void setMenu()
{
    GameLoop.Stop();
    Menu form = new Menu();
    form.TopLevel = false;
    this.Controls.Add(form);
    form.StartPosition = FormStartPosition.
        CenterScreen;
    form.BringToFront();
    form.resume_btn += Form_resume_btn;
    form.menu_btn += Form_menu_btn;
    form.Show();
}
private void Form_menu_btn(object sender,
    EventArgs e)
{
    this.Close();
    menu_btn?.Invoke(this, e);
    this.Close();
}

private void Form_resume_btn(object sender,
    EventArgs e)
{
    GameLoop.Start();
}

public void moveGhosts()
{
    foreach (Ghost g in game.ghosts)
    {
        if (collider.isGhostCollideWithPlayer(g))
        {
            if (player.Value > 0)
            {
                player.Value -= 10;
                if (player.Value == 0)
                {

                    MessageBox.Show(”Game over”);
                    Environment.Exit(0);
                }
            }
        }

        g.move(g.nextCell());

    }
}
private void showScore()
{
    score.Text = game.getScore().ToString();
}
private void movePlayer()
{
    Player player = game.getPacManPlayer();
    GameCell potentialNewCell = player.
       CurrentCell;
    GameCell currentCell = player.CurrentCell;
    if (Keyboard.IsKeyPressed(Key.LeftArrow))
    {
        potentialNewCell = player.CurrentCell.
           nextCell(GameDirection.Left);
        direct = GameDirection.Left;
        movementStatus = ’d’;
    }
    if (Keyboard.IsKeyPressed(Key.RightArrow))
    {
        potentialNewCell = player.CurrentCell.
           nextCell(GameDirection.Right);
        direct = GameDirection.Right;
        movementStatus = ’d’;
    }
    if (Keyboard.IsKeyPressed(Key.UpArrow))
    {
        movementStatus = ’u’;
    }
    if (Keyboard.IsKeyPressed(Key.Space))
    {
        GameCell NewCell = player.CurrentCell.
           nextCell(direct);
        Fire f = new Fire(Properties.Resources.
                bullet , NewCell, direct);
            game.addFire(f);
        }
        if (collider.isPlayerCollideWithCoin(
            potentialNewCell))
        {
            game.addScorePoints(1);
        }
        currentCell.setGameObject(Game.
            getBlankGameObject());
        player.move(potentialNewCell);
    }
    private void moveDown()
    {
        Player player = game.player;
        GameCell potentialNewCell = player.
            CurrentCell;
        GameCell nextCell;
        if (movementStatus == 'd')
        {
            nextCell = player.CurrentCell.nextCell(
                GameDirection.Down);
            if (nextCell != potentialNewCell)
            {
                GameCell currentCell = player.
                    CurrentCell;
                currentCell.setGameObject(Game.
                    getBlankGameObject());
                player.move(nextCell);
            }

        }
        if (movementStatus == 'u')
        {
            for (int i = 0; i < 3; i++)
            {
                potentialNewCell = player.CurrentCell
                    .nextCell(GameDirection.Up);
                jumpCount++;
                GameCell currentCell = player.
                    CurrentCell;
                currentCell.setGameObject(Game.
                    getBlankGameObject());
                player.move(potentialNewCell);
            }
```

```
                if (jumpCount == 3)
                {
                    movementStatus = 'd';

                }
            }
        }

        private void GameLoop_Tick(object sender,
            EventArgs e)
        {
            movePlayer();
            moveGhosts();
            moveDown();
            MoveFire();
            game.RemoveFire();
            if (Keyboard.IsKeyPressed(Key.Escape))
            {
                setMenu();
            }
            showScore();
        }
    }

        Menu:

public partial class Menu : Form
    {
        public event EventHandler resume_btn;

        private bool isClick;
        public bool IsClick
        {
            get => isClick;
        }
        public event EventHandler menu_btn;

        private bool isClick2;
        public bool IsClick2
        {
            get => isClick2;
        }

        public Menu()
        {
            InitializeComponent();
        }

        private void btn_resume_Click(object sender,
            EventArgs e)
        {
             isClick = true;
             resume_btn?.Invoke(this, e);
             this.Close();
        }

        private void btn_exit_Click(object sender,
            EventArgs e)
        {
             MessageBox.Show("Game Over");
             Environment.Exit(0);
        }

        private void btn_menu_Click(object sender,
            EventArgs e)
        {
             isClick2 = true;
             menu_btn?.Invoke(this, e);
             this.Close();
        }
    }
```

8 CONCLUSION:

In conclusion, I am delighted to present the Fire Buddy game, a captivating WinForms experience that merges thrilling gameplay with the innovative use of classes like GL (Game Logic) and UI (User Interface). This project aimed to create an immersive gaming adventure that allows players to become the ultimate fire-wielding hero, combating challenges and igniting excitement.

8.1 Summarization and Achievements:

Fire Buddy is a technology-driven WinForms game designed to provide players with an adrenaline-pumping experience, filled with fiery action and strategic challenges. By incorporating the principles of object-oriented programming, the game's GL and UI classes form the backbone of the seamless and engaging gameplay.

Throughout the development process, several notable achievements have been accomplished. The game boasts a visually striking and user-friendly UI that draws players into a world of flames and excitement. The GL class, meticulously crafted, ensures smooth and responsive gameplay, making every fire-wielding action feel empowering.

In Fire Buddy, players embark on a quest to rescue the kingdom from darkness, facing various enemies, puzzles, and obstacles along the way. The GL class plays a vital role in orchestrating these challenges, offering a mix of entertaining and progressively difficult levels.

Furthermore, the game includes an array of fire-based abilities and power-ups that players can unlock and utilize strategically, adding depth and replay ability to the gaming experience. The seamless integration of these features through the GL and UI classes contributes to an immersive and enjoyable journey.

8.2 Challenges:

The creation of Fire Buddy was not without its fair share of challenges. Implementing complex fire physics and animations while ensuring optimal performance demanded meticulous attention to detail and optimization. Balancing the difficulty curve to keep the gameplay challenging yet not overwhelming required iterative testing and fine-tuning.

Incorporating a responsive UI that adapts to different screen sizes and resolutions proved to be a challenging task, but it was crucial for delivering a polished and accessible gaming experience.

8.3 Lessons Learned:

Throughout the project, I gained valuable insights that will undoubtedly shape my future endeavors. The use of classes like GL and UI demonstrated the importance of modularity and separation of concerns. This approach significantly improved code maintainability and allowed for a more efficient development process.

Effective planning and requirement gathering were key to developing a coherent and engaging game. The GL class, as the core of the game's logic, required careful consideration to ensure it accommodated present and future features seamlessly.

Continuous testing and user feedback played a vital role in identifying and addressing issues early on. Regular iterations and improvements were necessary to deliver a polished final product.

In conclusion, the Fire Buddy game project has been a thrilling and rewarding experience. The integration of classes like GL (Game Logic) and UI (User Interface) has paved the way for an exciting gaming adventure that players will enjoy. I am proud of the achievements and valuable lessons learned, and I look forward to exploring new possibilities in future game development projects."

9 Video Profile:

Video Link