

**Lab Assignment**

**Submitted By:**

Ramsha Kokab

FA21-BCS-010

**Course Instructor:**

Mr. Syed Bilal Haider Bukhari

**Course:**

CC-lab

**Date:**

13th Dec, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

**Q: design a Domain-Specific Language (DSL) in C# to define and generate gameplay elements like ships, enemy waves, power-ups, and levels for a dynamic galaxy shooter game?"**

***Deliverables:***

**1**. DSL Specification: Document the syntax and rules of the DSL, including examples.

**2**. Parser Implementation: A C# program that parses DSL scripts into intermediate objects or data structures.

**3**. Interpreter: Code to execute the DSL by generating game objects and logic dynamically.

**4**. Game Prototype: A working galaxy shooter game that incorporates elements described in the DSL.

**5**. Test Scripts: Example DSL scripts that demonstrate various gameplay configurations and behaviors.

1. Code Documentation: Clear comments and instructions explaining how the DSL parser and interpreter work.
2. **DSL Specification:**

The DSL defines a simple syntax to describe gameplay elements. Below is an example:

Level 1 {

Ship {

Name: "Falcon";

Speed: 10;

Health: 100;

}

EnemyWave {

Count: 5;

Type: "Alien Fighter";

Speed: 8;

SpawnInterval: 2;

}

PowerUp {

Type: "Shield";

Duration: 10;

SpawnLocation: (100, 200);

}

}

#### Syntax Rules:

* **Levels**: Begin with the keyword Level, followed by a number, and enclosed in {}.
* **Ships**: Define ships using the Ship keyword with attributes like Name, Speed, and Health.
* **Enemy Waves**: Use EnemyWave with properties Count, Type, Speed, and SpawnInterval.
* **Power-ups**: Use PowerUp with attributes Type, Duration, and SpawnLocation.

1. **DSL Parser Implementation:**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

public class GalaxyShooterParser

{

    public GameData Parse(string dsl)

    {

        var gameData = new GameData();

        var levelMatches = Regex.Matches(dsl, @"Level\s+(\d+)\s\*{([^}]\*)}");

        foreach (Match levelMatch in levelMatches)

        {

            var level = new Level { Number = int.Parse(levelMatch.Groups[1].Value) };

            var content = levelMatch.Groups[2].Value;

            // Parse ships

            var shipMatches = Regex.Matches(content, @"Ship\s\*{([^}]\*)}");

            foreach (Match shipMatch in shipMatches)

            {

                var ship = new Ship();

                ParseAttributes(shipMatch.Groups[1].Value, ship);

                level.Ships.Add(ship);

            }

            // Parse enemy waves

            var waveMatches = Regex.Matches(content, @"EnemyWave\s\*{([^}]\*)}");

            foreach (Match waveMatch in waveMatches)

            {

                var wave = new EnemyWave();

                ParseAttributes(waveMatch.Groups[1].Value, wave);

                level.EnemyWaves.Add(wave);

            }

            // Parse power-ups

            var powerUpMatches = Regex.Matches(content, @"PowerUp\s\*{([^}]\*)}");

            foreach (Match powerUpMatch in powerUpMatches)

            {

                var powerUp = new PowerUp();

                ParseAttributes(powerUpMatch.Groups[1].Value, powerUp);

                level.PowerUps.Add(powerUp);

            }

            gameData.Levels.Add(level);

        }

        return gameData;

    }

    private void ParseAttributes(string attributes, dynamic obj)

    {

        var matches = Regex.Matches(attributes, @"(\w+):\s\*([^;]+);");

        foreach (Match match in matches)

        {

            var key = match.Groups[1].Value;

            var value = match.Groups[2].Value;

            var prop = obj.GetType().GetProperty(key);

            if (prop.PropertyType == typeof(int))

                prop.SetValue(obj, int.Parse(value));

            else if (prop.PropertyType == typeof(string))

                prop.SetValue(obj, value.Trim('"'));

            else if (prop.PropertyType == typeof((int, int)))

                prop.SetValue(obj, ParseTuple(value));

        }

    }

    private (int, int) ParseTuple(string value)

    {

        var values = value.Trim('(', ')').Split(',');

        return (int.Parse(values[0]), int.Parse(values[1]));

    }

}

// Data models

public class GameData

{

    public List<Level> Levels { get; set; } = new();

}

public class Level

{

    public int Number { get; set; }

    public List<Ship> Ships { get; set; } = new();

    public List<EnemyWave> EnemyWaves { get; set; } = new();

    public List<PowerUp> PowerUps { get; set; } = new();

}

public class Ship

{

    public string Name { get; set; }

    public int Speed { get; set; }

    public int Health { get; set; }

}

public class EnemyWave

{

    public int Count { get; set; }

    public string Type { get; set; }

    public int Speed { get; set; }

    public int SpawnInterval { get; set; }

}

public class PowerUp

{

    public string Type { get; set; }

    public int Duration { get; set; }

    public (int, int) SpawnLocation { get; set; }

}

1. **GameInterpreter with MonoGame Integration:**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using System;

using System.Collections.Generic;

public class Game1 : Game

{

    private GraphicsDeviceManager \_graphics;

    private SpriteBatch \_spriteBatch;

    private List<Ship> ships = new List<Ship>();

    private List<EnemyWave> enemyWaves = new List<EnemyWave>();

    private List<PowerUp> powerUps = new List<PowerUp>();

    private Texture2D shipTexture;

    private Texture2D enemyTexture;

    private Texture2D powerUpTexture;

    public Game1()

    {

        \_graphics = new GraphicsDeviceManager(this);

        Content.RootDirectory = "Content";

        IsMouseVisible = true;

    }

    protected override void Initialize()

    {

        base.Initialize();

    }

    protected override void LoadContent()

    {

        \_spriteBatch = new SpriteBatch(GraphicsDevice);

        // Load textures

        shipTexture = Content.Load<Texture2D>("ship");

        enemyTexture = Content.Load<Texture2D>("enemy");

        powerUpTexture = Content.Load<Texture2D>("powerUp");

        // Parse DSL and create game objects

        string dsl = @"

            Level 1 {

                Ship {

                    Name: ""Falcon"";

                    Speed: 10;

                    Health: 100;

                }

                EnemyWave {

                    Count: 5;

                    Type: ""Alien Fighter"";

                    Speed: 8;

                    SpawnInterval: 2;

                }

                PowerUp {

                    Type: ""Shield"";

                    Duration: 10;

                    SpawnLocation: (100, 200);

                }

            }

        ";

        var parser = new GalaxyShooterParser();

        var gameData = parser.Parse(dsl);

        // Interpret the DSL and create game objects

        var interpreter = new GameInterpreter(this);

        interpreter.Execute(gameData);

    }

    protected override void Update(GameTime gameTime)

    {

        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))

            Exit();

        // Update ships

        foreach (var ship in ships)

        {

            ship.Position.X += ship.Speed \* (float)gameTime.ElapsedGameTime.TotalSeconds;

        }

        // Update power-ups (logic can be extended as needed)

        base.Update(gameTime);

    }

    protected override void Draw(GameTime gameTime)

    {

        GraphicsDevice.Clear(Color.CornflowerBlue);

        \_spriteBatch.Begin();

        // Draw ships

        foreach (var ship in ships)

        {

            \_spriteBatch.Draw(shipTexture, ship.Position, Color.White);

        }

        // Draw enemies

        foreach (var wave in enemyWaves)

        {

            foreach (var enemy in wave.Enemies)

            {

                \_spriteBatch.Draw(enemyTexture, enemy.Position, Color.Red);

            }

        }

        // Draw power-ups

        foreach (var powerUp in powerUps)

        {

            \_spriteBatch.Draw(powerUpTexture, new Vector2(powerUp.SpawnLocation.Item1, powerUp.SpawnLocation.Item2), Color.Green);

        }

        \_spriteBatch.End();

        base.Draw(gameTime);

    }

    public void AddShip(Ship ship)

    {

        ships.Add(ship);

    }

    public void AddEnemyWave(EnemyWave wave)

    {

        enemyWaves.Add(wave);

    }

    public void AddPowerUp(PowerUp powerUp)

    {

        powerUps.Add(powerUp);

    }

}

public class GameInterpreter

{

    private Game1 \_game;

    public GameInterpreter(Game1 game)

    {

        \_game = game;

    }

    public void Execute(GameData gameData)

    {

        foreach (var level in gameData.Levels)

        {

            Console.WriteLine($"Loading Level {level.Number}...");

            foreach (var ship in level.Ships)

            {

                Console.WriteLine($"Spawning Ship: {ship.Name}, Speed: {ship.Speed}, Health: {ship.Health}");

                \_game.AddShip(ship);

            }

            foreach (var wave in level.EnemyWaves)

            {

                Console.WriteLine($"Spawning Enemy Wave: {wave.Type}, Count: {wave.Count}");

                \_game.AddEnemyWave(wave);

            }

            foreach (var powerUp in level.PowerUps)

            {

                Console.WriteLine($"Placing Power-Up: {powerUp.Type} at {powerUp.SpawnLocation}");

                \_game.AddPowerUp(powerUp);

            }

        }

    }

}

// Classes for ships, enemies, and power-ups

public class Ship

{

    public string Name { get; set; }

    public int Speed { get; set; }

    public int Health { get; set; }

    public Vector2 Position { get; set; } = Vector2.Zero;

}

public class EnemyWave

{

    public int Count { get; set; }

    public string Type { get; set; }

    public int Speed { get; set; }

    public int SpawnInterval { get; set; }

    public List<Enemy> Enemies { get; set; } = new List<Enemy>();

}

public class Enemy

{

    public Vector2 Position { get; set; } = Vector2.Zero;

    public int Speed { get; set; }

}

public class PowerUp

{

    public string Type { get; set; }

    public int Duration { get; set; }

    public (int, int) SpawnLocation { get; set; }

}

1. **Game Prototypes**

* **Levels:**

using System;

using System.Collections.Generic;

public class Level {

  public int LevelNumber { get; set; }

  public List<string> Ships { get; set; }

  public List<int> EnemyWaves { get; set; }

  public List<string> PowerUps { get; set; }

  public Level(int levelNumber, List<string> ships, List<int> enemyWaves, List<string> powerUps) {

    LevelNumber = levelNumber;

    Ships = ships;

    EnemyWaves = enemyWaves;

    PowerUps = powerUps;

  }

  public void Display() {

    Console.WriteLine($"Level: {LevelNumber}");

    Console.WriteLine($"Ships: {string.Join(", ", Ships)}");

    Console.WriteLine($"Enemy Waves: {string.Join(", ", EnemyWaves)}");

    Console.WriteLine($"Power-Ups: {string.Join(", ", PowerUps)}");

  }

}

public class LevelPrototype {

  static void Main() {

    var level = new Level(1, new List<string> { "Fighter", "Bomber" }, new List<int> { 1, 2 }, new List<string> { "Shield", "SpeedBoost" });

    level.Display();

  }

}

* **Ship:**

using System;

using System.Collections.Generic;

public class Ship {

  public string Name { get; set; }

  public int Health { get; set; }

  public int Speed { get; set; }

  public List<string> Weapons { get; set; }

  public Ship(string name, int health, int speed, List<string> weapons) {

    Name = name;

    Health = health;

    Speed = speed;

    Weapons = weapons;

  }

  public void Display() {

    Console.WriteLine($"Ship: {Name}, Health: {Health}, Speed: {Speed}, Weapons: {string.Join(", ", Weapons)}");

  }

}

public class ShipPrototype {

  static void Main() {

    var ship = new Ship("Fighter", 100, 10, new List<string> { "Laser", "Missile" });

    ship.Display();

  }

}

* **Enemy wave:**

using System;

using System.Collections.Generic;

public class Enemy {

  public string Type { get; set; }

  public int Count { get; set; }

  public Enemy(string type, int count) {

    Type = type;

    Count = count;

  }

  public void Display() {

    Console.WriteLine($"Enemy Type: {Type}, Count: {Count}");

  }

}

public class EnemyWave {

  public int WaveNumber { get; set; }

  public List<Enemy> Enemies { get; set; }

  public EnemyWave(int waveNumber, List<Enemy> enemies) {

    WaveNumber = waveNumber;

    Enemies = enemies;

  }

  public void Display() {

    Console.WriteLine($"Wave Number: {WaveNumber}");

    foreach (var enemy in Enemies) {

      enemy.Display();

    }

  }

}

public class EnemyWavePrototype {

  static void Main() {

    var enemies = new List<Enemy> {

      new Enemy("SmallShip", 5),

      new Enemy("Boss", 1)

    };

    var wave = new EnemyWave(1, enemies);

    wave.Display();

  }

}

* **Power up:**

using System;

public class PowerUp {

  public string Type { get; set; }

  public int Duration { get; set; }

  public PowerUp(string type, int duration) {

    Type = type;

    Duration = duration;

  }

  public void Display() {

    Console.WriteLine($"Power-Up: {Type}, Duration: {Duration} seconds");

  }

}

public class PowerUpPrototype {

  static void Main() {

    var powerUp = new PowerUp("Shield", 10);

    powerUp.Display();

  }

}

1. **Test Scripts:**

Level 1 {

Ship {

Name: "Interceptor";

Speed: 15;

Health: 120;

}

EnemyWave {

Count: 10;

Type: "Drone";

Speed: 12;

}

PowerUp {

Type: "Double Damage";

SpawnLocation: (400, 500);

}

}

