
CSE 1325 Project Documentation

# Elemental Battle Game

Students names, surnames, IDs:

1. Angel Montalvo, 1002148576
2. Sam Trinh, 1002186851

Mentor: Marika Apostolova

**Object-oriented programming CSE 1325**

**Student declaration:**

*We declare that:*

- *We understand what is meant by plagiarism*
- *The implication of plagiarism has been explained to me by our professor*
- *This assignment is all team own work and we have acknowledged any use of the published and unpublished works of other people.*

**1 Student** name, surname, ID and **signature: Angel Montalvo, 1002148576, A.M.**
**2 Student** name, surname, ID and **signature: Sam Trinh, 1002186851, S.T**

**Date:11/23/2024**

| | | |
|---|---|---|
| | **Total number of pages including this cover page** | 15 |
| **Class Code / Group** | CSE 1325 | |
| **Lecturer's Name** | MARIKA APOSTOLOVA | |

# Table of Contents

1. **Project Introduction**

   This game aimed to create a Java-based console game where the user selects an element (Water, Fire, Light, Moon, Ice, or Grass). After the player selects their character and attacks, they fight a random elemental enemy. The game includes elemental resistances and a turn-based battle system where characters take turns attacking until one is defeated. Afterward, the user can choose whether to play again or terminate the game.

2. **Project Description**

   **Objective:** This game intends to provide the user with a fun and enjoyable gameplay experience. The primary objective of the game is to defeat the enemy elemental character by strategically using attacks, taking into account the cooldown system and elemental resistances. Players must plan their attacks wisely to overcome the unique strengths and weaknesses of the opposing element.

   **Need:** Video games are important to everyone nowadays, no matter the age. Although this game doesn't use a complex engine or doesn't have graphics for the user, we believe this game could be fun for any user playing it. It emphasizes strategic thinking, as players must consider the strengths and weaknesses of their chosen element and manage attack cooldowns effectively. With its simple yet engaging mechanics, it offers replayability and a fun experience for all Java developers.

   **Process:**

   Character Selection: Players start by selecting one of six elemental characters: Water, Fire, Light, Moon, Ice, or Grass. Each element has distinct attributes and attacks. After the player chooses their desired element, a random number generator selects an opponent with one of the 6 elemental options.

   Battle Sequence: Players are allowed four different options during their turn. Their primary attack does the most damage and incorporates a cool-down timer after being used. Their secondary attack is weaker than the primary, however, this attack does not have a cool-down. Their third option is to block the enemy's attack on that turn, reducing normal damage taken by 50%. The final option is to heal for a percentage of the user's health, placing a cooldown on this option as well. After the user chooses their option, the program uses a random number generator to dictate the enemy's move for that turn. At the end of both the player's and the enemy's turns, an updated interface of the user's/enemies' health.

   Victory / Defeat: Once the enemy is defeated, the amount of total XP earned populates the console, and the user is given the option of moving on to the next enemy, or saving their

progress. If the user decides to save their progress, the next time the program is populated the user will load into the level they left off on.

Tools Used: The program was created using Java and uses standard Java libraries, random number generation, and basic logic implementation.

## 3. Project Architecture

1. **Game Loop:** The game is built on two main loops: the interface loop, which allows the user to load their game or create a new player, and the fightRounds loop, which includes using your character to battle the opposing enemy element.
    a. Battle Logic: There are numbers that dictate how powerful an element is. Each element has different stats and should have the same "power level" overall.
    b. Game States: The player can load their save file if they have played before and have chosen to save. Each major stat is loaded for the player and can pick up where they have left off.
    c. Character Creation: The character will have different stats based on the element. Health is how much damage a character can take. Primary Attack is how much their main attack can deal. Secondary damage is how much damage their secondary attack can deal. Heal Strength is how much their heal can restore Health.
2. **Player:** The player character distinguishes itself by being controlled by the user. The user can choose between four options and decide what to do. Having only four options is lackluster, but we also have a leveling system that keeps the user wanting to keep on playing.
    a. Leveling System: The leveling system requires the player to beat the first enemy to reveal itself. The EXP gain is based on the monster's MaxHealth. The player deals more damage and can take more damage with each level.
    b. Turn-Based Action: The user can choose between four actions: Primary Attack, Secondary Attack, Defend, and Heal. Primary Attack has a 1-turn cooldown but deals more damage. Secondary Attack deals less damage but has no cooldown. Defend makes the enemy deal 50% of the damage they would deal for 1 turn. Heal is based on HealStrength and restores a certain amount of health but has a cooldown of 2 turns.

3. **Enemy:** The enemy character is a noncontrollable character that scales more per round, can have different stats per loop, but has randomized actions to balance out their higher stats.
    a. Scaling Factor: The enemy gains a scaling factor of 0.1 per round and modifies their MaxHealth per round. The enemy also gains more attack per round by a flat amount. This allows the enemy to take and deal more damage, making their randomized actions more daunting.
    b. Different Per Loop: The enemy changes their elements and thus has different stats per round. The enemy will scale appropriately based on the round still.
    c. Randomized Actions: The enemy takes it turn based on a random number generator between 1-4. The enemy can do what the player can do but without any strategy.
4. **Game State Management:** The game ends with the user saving their game and exiting out. However, the user can also save their player character and how far they have gotten.
    a. Saves every Important Stat: The game saves the enemy damage and health, the player level and element, and how far the player has gone.
    b. Loads every Stat Saved: When booting up the game again or simply trying to retry a round over and over, the game loads up where the player has gone and simply picks up where the player left off.

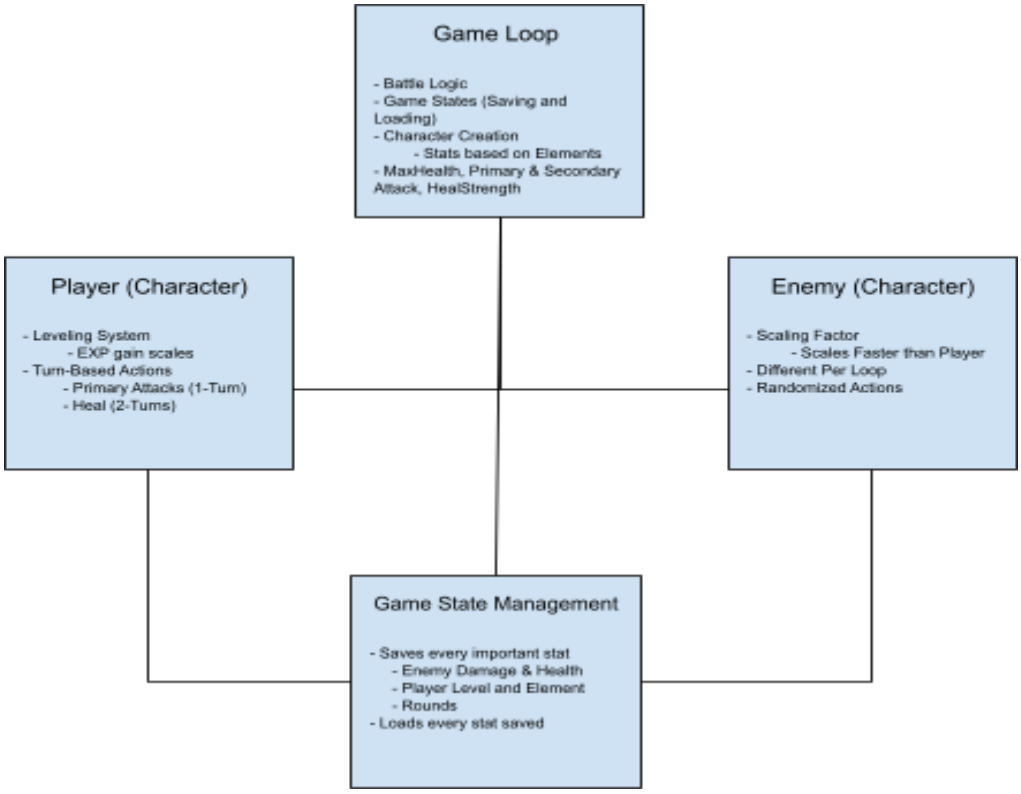


*Figure 1 System architecture*

## 4. UML Class Diagram

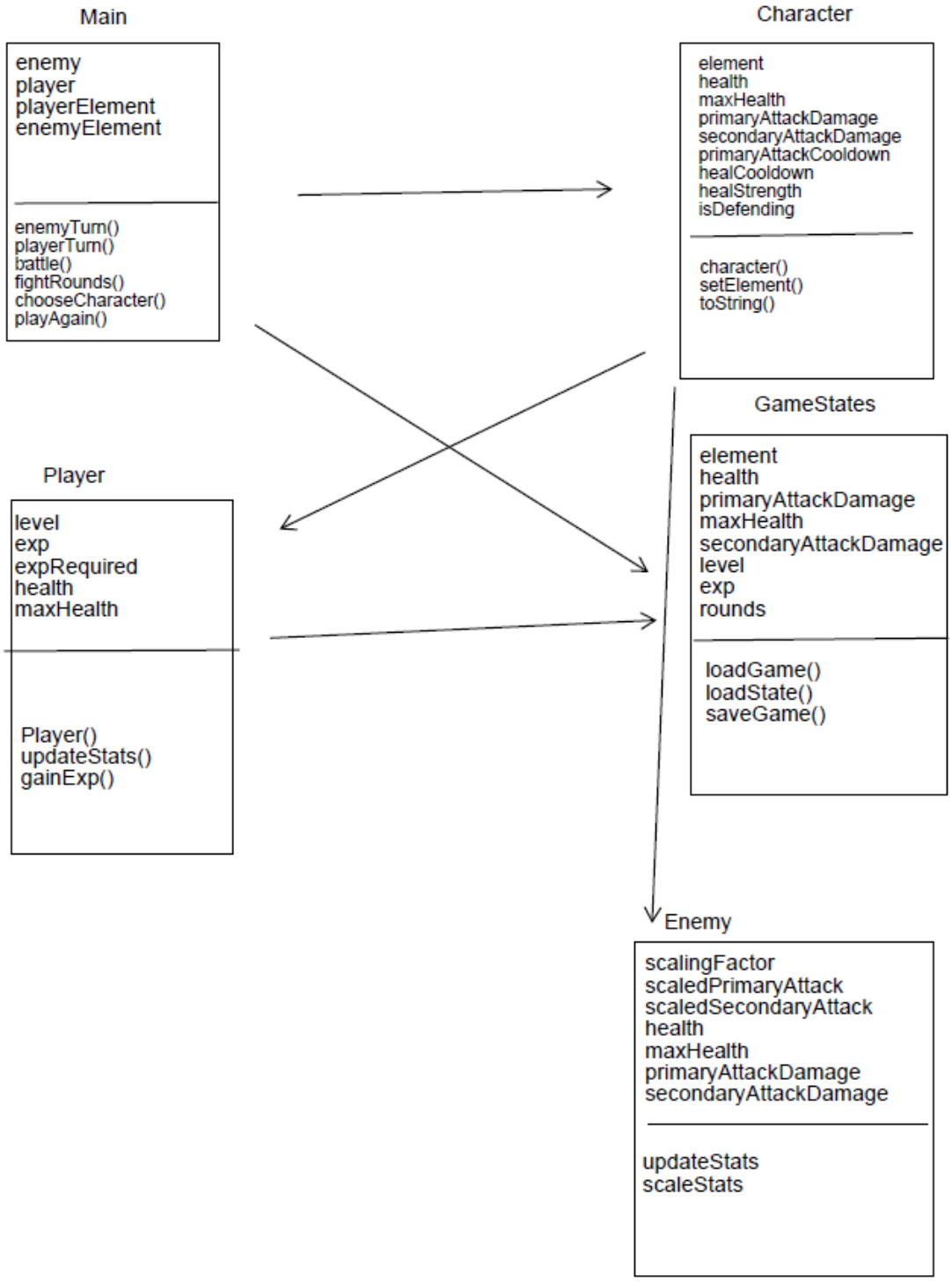

*Figure 2 UML class diagram*

## 5. OOP Concepts Used

Classes: a class is a blueprint or a template for creating objects. It encapsulates data and methods (functions) that define the behavior of objects of that class. We have 2 sub-classes in our code, that share the parent class “character”. The parent class is used so that the player’s character and enemy characters’ attacks can level up properly, and deal the proper amount of damage depending on the round.

Dynamic behavior: Dynamic behavior refers to an object's ability to change its behavior or state based on the current situation or context at runtime, often achieved through mechanisms like polymorphism and dynamic binding. This concept applies to our cool-down system, changing dynamically over time and displaying how object states evolve over time.

## 6. Code Description

**The first key feature of our project is the player class:**

```
class Player extends Character
{
   int level;
   int exp;
   int expRequired;
   public Player(String element) {
      super(element);
      this.level = 1;
      this.expRequired = 100 * level;
      this.exp = 0;
      this.maxHealth = this.maxHealth + 5*(level-1);
   }
   @Override
   public void updateStats() {
      while (exp >= expRequired) {
         exp -= expRequired;
         level++;
         expRequired = 100 * level;
         this.maxHealth += 5 * (level-1);
         this.health = this.maxHealth;
         primaryAttackDamage += 2;
```

```
            secondaryAttackDamage += 1;

            System.out.println("Player leveled up to level " + level + "!");
        }
        System.out.println("Player EXP: " + exp + "/" + expRequired);
    }

    public void gainExp(int amount) {
        exp += amount;
        System.out.println("Player gained " + amount + " EXP!");
        updateStats();
    }
}
```

This class is important because it allows the user's character to gain XP, which boosts the character's stats allowing them a more enjoyable gaming experience. This is a key feature in our program because if the user's character remained at the same level while the monster difficulty increased, the battles would become too hard for the user, discouraging them from playing the game long term.

**The second key feature in our program is our "save file" option:**

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;

class GameStates
{
  public static void SaveGame(String fileName,
                              String element,
                              int health,
                              int maxHealth,
                              int primaryAttackDamage,
                              int secondaryAttackDamage,
                              float scalingFactor,
                              int scaledPrimaryAttack,
                              int scaledSecondaryAttack,
                              int level,
                              int exp,
                              int expRequired,
                              int rounds)
```

```
{
    try (FileWriter saveState = new FileWriter(fileName)) {
        saveState.write(element + "\n");
        saveState.write(health + "\n");
        saveState.write(maxHealth + "\n");
        saveState.write(primaryAttackDamage + "\n");
        saveState.write(secondaryAttackDamage + "\n");
        saveState.write(scalingFactor + "\n");
        saveState.write(scaledPrimaryAttack + "\n");
        saveState.write(scaledSecondaryAttack + "\n");
        saveState.write(level + "\n");
        saveState.write(exp + "\n");
        saveState.write(expRequired + "\n");
        saveState.write(rounds + "\n");
        System.out.println("Game state saved successfully!");
    } catch (IOException error) {
        System.out.println("Error saving game state: " + error.getMessage());
    }
}
public static int LoadGame(String filename, Player player, Enemy enemy)
{
  int rounds = 1;
  BufferedReader loadState = null;
  try
  {
    loadState = new BufferedReader(new FileReader(filename));
    player.element = loadState.readLine();
    player.health = Integer.parseInt(loadState.readLine());
    player.maxHealth = Integer.parseInt(loadState.readLine());
    player.primaryAttackDamage = Integer.parseInt(loadState.readLine());
    player.secondaryAttackDamage = Integer.parseInt(loadState.readLine());
    enemy.scalingFactor = Float.parseFloat(loadState.readLine());
    enemy.scaledPrimaryAttack = Integer.parseInt(loadState.readLine());
    enemy.scaledSecondaryAttack = Integer.parseInt(loadState.readLine());
    player.level = Integer.parseInt(loadState.readLine());
    player.exp = Integer.parseInt(loadState.readLine());
    player.expRequired = Integer.parseInt(loadState.readLine());
    rounds = Integer.parseInt(loadState.readLine());
    System.out.println("Rounds loaded: " + rounds);
  } catch (IOException error) {
      System.out.println("Error loading game state: " + error.getMessage());
```

```
    }
    return rounds;
  }
}
```

This class uses file handling to save the user's process in case they want to stop playing. This feature might be the most important aspect to a gamer because it allows for them to truy enjoy the game without the worry of losing their progress dangling over their head. We understand emergencies are bound to happen while gaming, and it is important the user not have to stress about saving their game just because they have to go deal with something else.

**The third key feature in this program is the elemental switch case:**

```
public void setElement(String element) {
        this.element = element;
        switch (element){
        case "Water":
           this.primaryAttackDamage = 15;
           this.secondaryAttackDamage = 10;
           this.health = 105;
           this.maxHealth = 105;
           this.healStrength = 20;
           break;
        case "Fire":
           this.primaryAttackDamage = 30;
           this.secondaryAttackDamage = 10;
           this.health = 75;
           this.maxHealth = 75;
           this.healStrength = 5;
           break;
        case "Ice":
           this.primaryAttackDamage = 15;
           this.secondaryAttackDamage = 5;
           this.health = 140;
           this.maxHealth = 140;
           this.healStrength = 20;
           break;
        case "Grass":
           this.primaryAttackDamage = 15;
           this.secondaryAttackDamage = 10;
           this.maxHealth = 100;
           this.health = 100;
```

```
            this.healStrength = 15;
            break;
            case "Moon":
            this.primaryAttackDamage = 20;
            this.secondaryAttackDamage = 5;
            this.health = 105;
            this.maxHealth = 105;
            this.healStrength = 15;
            break;
            case "Light":
            this.primaryAttackDamage = 35;
            this.secondaryAttackDamage = 15;
            this.health = 70;
            this.maxHealth = 70;
            this.healStrength = 5;
        }
    }
```

This feature is very important to the game because it is what differentiates it from other turn-based games. In the gaming industry, it is bound for every successful game to have a knockoff. While it is true that our game isn't the first to incorporate nature elements into it, we believe that adding resistances and balancing attack damage based on ur element choice is a unique touch that other games have not used.

## 7. System Testing

```
Welcome to the Enhanced Elemental Battle Game!
Choose your element and face an enemy with unique strengths and weaknesses!
Elements: Water, Fire, Light, Moon, Ice, Grass
Each element has unique stats and abilities. Let the battle begin...

Choose your character by typing the name of the element:
1. Water
2. Fire
3. Light
4. Moon
5. Ice
6. Grass
light

Your character: Light Element (Health: 70)
Enemy character: Ice Element (Health: 100)


========== CURRENT STATS ==========
Player: Light      | Health:  70/70
Enemy:  Ice        | Health: 100/100
====================================

It's your turn!
Choose your action:
1. Primary Attack (0 damage) [Cooldown: Ready]
2. Secondary Attack (25 damage)
3. Defend (Reduce incoming damage by 50%)
4. Heal (Restore 5 health) [Cooldown: Ready]
2

You used your Secondary Attack!
100 -> 75

It's the enemy's turn!

The enemy uses heal!
The enemy heals for 20 health!
75 -> 95

========== CURRENT STATS ==========
Player: Light      | Health:  70/70
Enemy:  Ice        | Health:  95/100
====================================
```

Figure 3.1

This Screenshot displays the opening message of the program. It introduces the user to the game and displays all of their options for elemental characters. Additionally, it shows the users' combat options and shows the option picked by the enemy with an updated console interface.

```
You defeated the enemy!
Player gained 130 EXP!
Player EXP: 160/300
Enemy stats scaled with factor 1.4!

The enemy has changed its element!
New enemy character: Moon Element (Health: 140)

This is now round: 5

========== CURRENT STATS ==========
Player: Light      | Health:   1/85
Enemy:  Moon       | Health: 140/140
====================================

It's your turn!
Choose your action:
1. Primary Attack (39 damage) [Cooldown: Ready]
2. Secondary Attack (17 damage)
3. Defend (Reduce incoming damage by 50%)
4. Heal (Restore 5 health) [Cooldown: Ready]
2

You used your Secondary Attack!
140 -> 123

It's the enemy's turn!

The enemy uses their Secondary Attack!
You received 5 damage!
1 -> 0

You were defeated by the enemy!

Would you like to play again? (yes/no)
```

Figure 3.2

This screenshot shows the leveling system the game uses. When an enemy is defeated, they gain EXP that stacks towards the next level, ultimately increasing the players' attack damage. At the end, it shows the user was defeated, and gives them the option of starting a new campaign or terminating the program.

## 8. Group members

Angel - This admin came up with the idea to make a turn-based battle game, and also with the idea of incorporating elements into the program. He created the Replit used to collaborate on the assignment and created most of the core functions, such as the “chooseCharacter”, “enemyTurn”, and “playerTurn” functions

Sam - This admin was responsible for fine-tuning the app. Sam came up with the idea to incorporate a leveling system into the game and also came up with the idea to scale enemy strength based on the round of the game. Some of the functions created by Sam were the “updateStats”, “gameStates”, and “gainExp”.

## 9. Conclusion and future works

The main objective of this game was to create a fun and interactive Java program. By looking through the program’s code, and looking through the documentation any coder can understand the implementation. To improve this game, we can implement varying difficulties to the enemy such as making an easy, medium, or hard mode. We were also thinking about adding items to the game, which wouldn’t take up a player's turn but could only be used once a round.