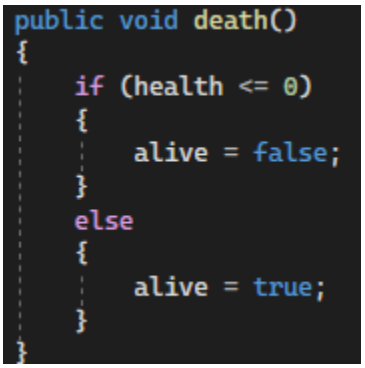
Term Project: Monster Fighter

Christopher Batts & Cuodi Cromartie

ABSTRACT



```
public void death()
{
    if (health <= 0)
    {
        alive = false;
    }
    else
    {
        alive = true;
    }
}
```

INTRODUCTION

Monster Fighter is our term project for COP4368. This game is a turn-based combat game where the player must fight and defeat 4 monster enemies. The player is given 3 options during combat, attack (Which deals a random number of damage depending on the user's current level), block (Which limits the amount of damage dealt by the enemies following attack), and heal (Which instantly grants the player 10 more health points). The enemy attacks after every player turn.

1. DEPENDENCY, ASSOCIATION, & INHERITANCE

Dependency is when an object uses another unrelated object to perform a task. Monster hunter uses this form of object collaboration in both the hero class, and the monster class. The hero class has a method named **attack(monster monster1)** .

```
// DEPENDENCY
public void attack(monster monster1)
{
    Random random = new Random();
    if (this.lvl == 1)
    {
        monster1.health = monster1.health - random.Next(3, 5);
    }
    if (this.lvl == 2)
    {
        monster1.health = monster1.health - random.Next(5, 7);

    }
    if (this.lvl == 3)
    {
        monster1.health = monster1.health - random.Next(5, 13);

    }
    if (this.lvl == 4)
    {
        monster1.health = monster1.health - random.Next(6, 15);
    }
}
```

(UserControl1.xaml.cs Line 28-50)

The attack method takes a monster object as a parameter to allow the hero object to do damage to the monster object and lower its health. The monster class also has a version of the same method, **attack(hero hero1)**

```
// DEPENDENCY
public void attack(hero hero1)
{
    Random random = new Random();
    if (lvl == 1)
    {
        hero1.health = hero1.health - random.Next(3, 5);
    }
    if (lvl == 2)
    {
        hero1.health = hero1.health - random.Next(3, 8);

    }
    if (lvl == 3)
    {
        hero1.health = hero1.health - random.Next(5, 10);

    }
    if (lvl == 4)
    {
        hero1.health = hero1.health - random.Next(7, 17);
    }
}
```

(UserControl1.xaml.cs Line 95-116)

accepting a hero object as a parameter. Both of these functions rely on another unrelated object to carry out their task i.e. dependency.

Association is utilized in Monster Fighter with the healthPotion class. Both the hero class and the monster class have a healthPotion object as one of their properties. This healthPotion object has a set amount of health points in it that are accessed through the **heal()** method

```
// AGGREGATION + COMPISITION
public Boolean heal()
{
    if (potion.getHealth() > 0)
    {
        health = health + 10;
        potion.setHealth(potion.getHealth() - 10);
        return true;
    }
    else
    {
        return false;
    }

}
```

(UserControl1.xaml.cs Line 64-77)

in the hero class. This is a "*has a"* relationship. The hero class *has a* health potion, this is aggregation. The potion is also essentially useless outside of the context of the hero class, making this a form of composition as well.

Inheritance is achieved in Monster Fighter by having the monster class inherit the hero class.

```
// INHERITANCE
public class monster : hero
```

(UserControl1.xaml.cs Line 80)

Since the monster class is responsible for performing all of the same actions as the hero class (player), the monster class inherits all of the properties and methods from the hero class. This allows the monster to perform the same actions as the player, with some alterations for gameplay purposes.

2. CLASS LIBRARY

Monster Fighter uses a class library to reference the hero class, monster class, and healthPotion class. This was achieved by creating all three classes in a WPF User Control Library project in Visual Studio, building the solution, and adding it as a reference in the Monster Fighter WPF project.

```
using MonsterLib;
```

(Form1.cs Line 10)

3. ACCESS MODIFIERS

Monster fighter contains a read only property in the healthPotion class for the variable health. This variable is read only and utilizes get and set methods.

```
public class healthPotion
{
    private int health = 20;
    public healthPotion(int health1)
    {
        health = health1;
    }
    public int getHealth()
    {
        return health;
    }
    public void setHealth(int newHealth)
    {
        health = newHealth;
    }
}
```

(UserControl1.xaml.cs Line 119-134)

4. EXCEPTION HANDLING

Due to the nature of this project, there are not many opportunities to run into exceptions. However, exception handling has been implemented for the initial “Enter your name” screen. If a user somehow enters invalid characters, too many characters, or no characters at all, the program can catch this and display an error message.

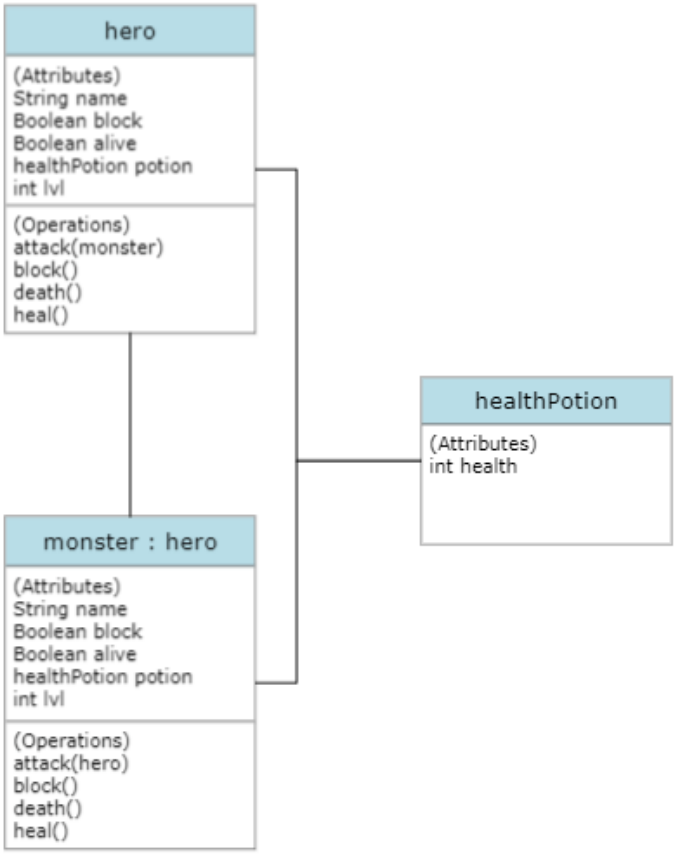
```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        hero.name = textBox1.Text;
    }
    catch (System.Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    if (hero.name.Length > 25)
    {
        label2.Text = "Name is too long. Please re-enter";
        label2.Visible = true;
        label2.BackColor = Color.Red;
        textBox1.Text = "";
    }
    if (hero.name.Length == 0)
    {
        label2.Text = "Please enter a name";
        label2.Visible = true;
        label2.BackColor = Color.Red;
        textBox1.Text = "";
    }
```

(Form1.cs Lines 33-56)

5. TESTING

In depth and detailed testing of Monster Hunter has been done and is documented in the Test_Scenario 1-5 test files included in this projects folder. Testing has been done for each stage of the applications runtime. Starting with naming your hero, to the end of the final in-game battle, all options and functions of the application have been completed and any oversights initially found have been fixed. Testing of this application brought great improvements to stability and quality of life for the end user.

7. UML DIAGRAM



8. USER MANUAL

We have created a user manual (Monster Fighter - User Manual.pdf) that explains the rules of the game as well as the instructions to run the game and view the code the game is composed of. This manual consists of a setup section, an introduction, and a combat section, along with screenshots of the game while it is running.