University “Politehnica” of Bucharest

Faculty of Electronics, Telecommunications, and Information Technology

-A fighting game between 2 characters-

Object Oriented Programming

- **Student:** Gheorghe Gheorghe-Theodor
- **Group:** 412G

<u>Introduction</u>

The current application is a video game featuring a thrilling battle between two characters. To provide players with diverse gaming experiences, the game is structured into two distinct gamemodes. Users can easily select their desired gamemode from the application's interface when launching the game. <u>The first gamemode</u>, named 'Story Mode', involves the player's customizable character fighting against three villains controlled by the computer. In order to complete this gamemode, the player must emerge victorious in battles against all three opponents. <u>In the second gamemode</u>, named "Player vs Player," two players will face each other using their customizable characters. The players will have the ability to introduce their character's attack, defense, and special attack mode, following specific rules that ensure a fair and balanced fight.

The program was built using two main classes: Character and Villain. The Character class is the main class, from which the Villain class inherits its elements. Both classes have a set of attributes, constructors, setters, and destructors, as well as additional methods where needed. The Character class allows the user to create their own customizable characters with attack and defensive attributes, which can then be used to play in both gamemodes. On the other hand, the Villain class is a derived class from the Character class, and has pre-set attributes such as names and weapons.

Program Structure

The build application is structured as follows:

1. A base class called **“Character”** describes a character in the video game and has the following attributes.

   a. This class has the following attributes:

   i.name: the character's name

   ii. health: the character's health

   iii. attack_points: the character's attack points

   iv. defense_points: the character's defense points

   v. special_attack_name: the name of the character's special attack

   b.The “**Character**” class contains the following methods:

   i. Setters to give values to the attributes

   ii. Getters to return the values of the attributes

   iii. Implicit and Parameters Constructors to initialize the attributes with default or given values

   c.The “**Character**” class contains the following functions:

   i. **close_attack:** This function takes two parameters: attacker and target, both of which are of type Character. It reduces the attacker's attack points by 20% and the target's health by 10%.

   ii. **far_attack:** attacker and target, which are instances of the Character class. This method reduces the attacker's attack points by 30% and also reduces the target's health by 20%.

   iii. **special_attack:** Takes two parameters: attacker and target, which are of type Character. This type of attack cost 30 attack points and generate a random damage between 0 and 40 HP to the target, making the game more unpredictable and more fun. 😊

iv. **receive_health**: Takes one parameter: receiver, which is type of Character. This type of ability cost 50 defensive points and give to the receiver 20 HP;

v. **display stats:** takes one parameter which is a Character type and display his name and abilities.

vi. **poison_attack:** takes two parameters: attacker and target which are Character type elements.If this attack is executed, it costs the attacker 50 attack points and there is a 20% chance of it happening. If it does happen, the target's health is reduced by 85 points.

2. A class called **"Villain"** which inherits the elements of "**Character**".

a. The class contains as attributes:

i. name: the villain's name;

ii. **health**: the character's health;

iii. **attack_points:** the character's attack points;

iv. **defense_points:** the character's defense points;

v. **special_attack_name:** the name of the character's special attack;

vi. **weapon1 and weapon2**: the name of the abilities or the name of the weapons used by the villain.

b. The "**Villain**" class contains the following methods:

i. Setters to give values to the attributes;

ii. Implicit and Parameters Constructors to initialize the attributes with default or given values.

b. The "**Villain**" class contains the following functions:

i. **attack1:** This function takes two parameters: attacker which is the type of Villain and target which is the type of. It reduces the villan's attack points by 10% and the target's health by 15%.

ii. **attack2:** This function takes two parameters: attacker which is the type of Villain and target which is the type of Character. It reduces the villan's attack points by 30% and the target's health by 20%.

iii. **special_attack:** This function takes two parameters: attacker which is the type of Villain and target which is the type of Character. This type of attack cost 30 attack points and generate a random between 0 and 40 HP to the target.

iv. **receive_health:** Takes one parameter: receiver, which is type of Character. This type of ability cost 50 defensive points and give to the receiver 20 HP;

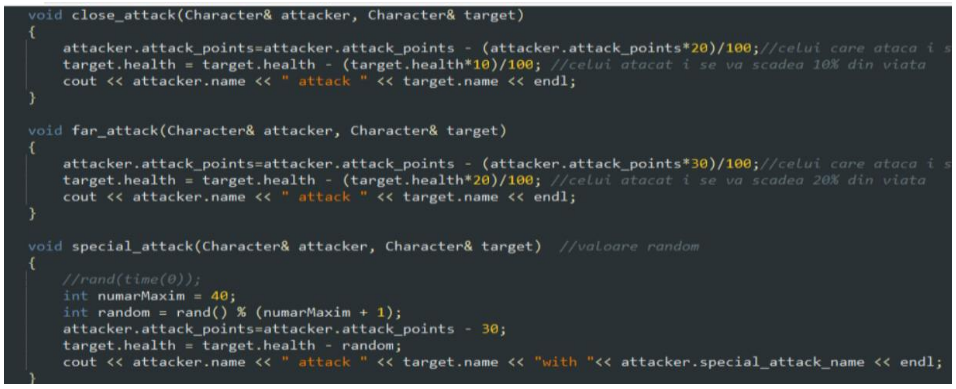
v.**display_stats:** the role to display the name and abilities of the villain.

A closer look at the functions

In the context of the specific program mentioned earlier, the functions used serve to perform specific actions related to the game mechanics. For example, the function that reduces the attacker's attack points and the target's health is used during combat to calculate the damage done by an attack.

Other functions in the program may be used to initialize character attributes, handle input/output, or control game flow. Each function has a specific role and purpose within the program, contributing to the overall functionality of the game.

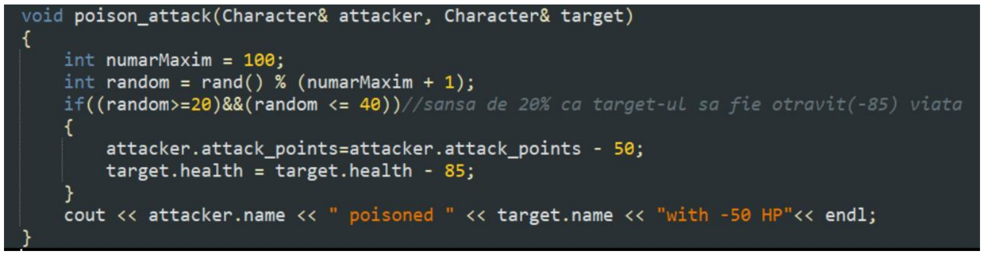
The attack functions have the following structure:



```
void close_attack(Character& attacker, Character& target)
{
    attacker.attack_points=attacker.attack_points - (attacker.attack_points*20)/100;//celui care ataca i s
    target.health = target.health - (target.health*10)/100; //celui atacat i se va scadea 10% din viata
    cout << attacker.name << " attack " << target.name << endl;
}

void far_attack(Character& attacker, Character& target)
{
    attacker.attack_points=attacker.attack_points - (attacker.attack_points*30)/100;//celui care ataca i s
    target.health = target.health - (target.health*20)/100; //celui atacat i se va scadea 20% din viata
    cout << attacker.name << " attack " << target.name << endl;
}

void special_attack(Character& attacker, Character& target)  //valoare random
{
    //rand(time(0));
    int numarMaxim = 40;
    int random = rand() % (numarMaxim + 1);
    attacker.attack_points=attacker.attack_points - 30;
    target.health = target.health - random;
    cout << attacker.name << " attack " << target.name << "with "<< attacker.special_attack_name << endl;
}
```

The purpose of the random value generation function in the game is to introduce an element of unpredictability and excitement. By generating a random value for the amount of damage dealt by the attacker using “The Special Attack” to the target, the game becomes more dynamic and engaging for the players. It also adds an element of strategy, as players must consider the potential range of damage they may deal or receive.

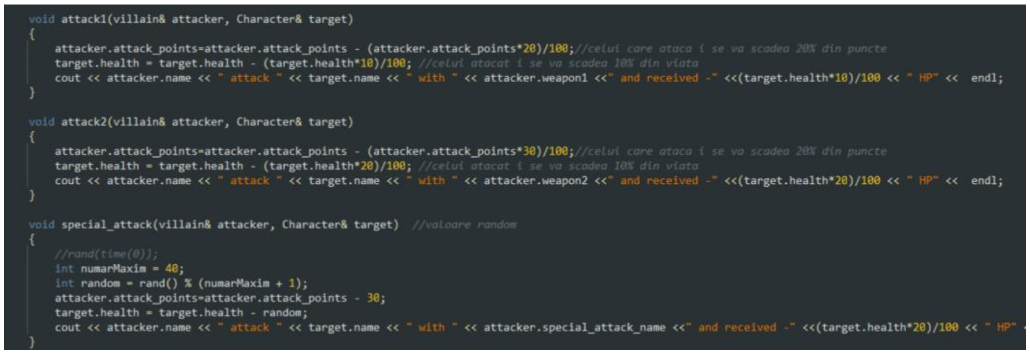
Another exciting and important function that adds a dynamic mechanic to the game is the following:



```cpp
void poison_attack(Character& attacker, Character& target)
{
    int numarMaxim = 100;
    int random = rand() % (numarMaxim + 1);
    if((random>=20)&&(random <= 40))//sansa de 20% ca target-ul sa fie otravit(-85) viata
    {
        attacker.attack_points=attacker.attack_points - 50;
        target.health = target.health - 85;
    }
    cout << attacker.name << " poisoned " << target.name << "with -50 HP"<< endl;
}
```

It has a 20% chance of occurring and has a high cost in attack points, but deals a significant amount of damage to the target.

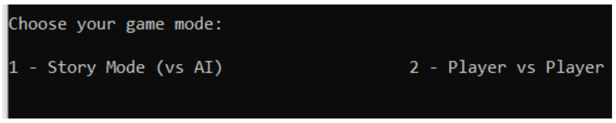
For the “Villan” class, the logic behind is the same as the one used for the main class:



```cpp
void attack1(villain& attacker, Character& target)
{
    attacker.attack_points=attacker.attack_points - (attacker.attack_points*20)/100;//celui care ataca i se va scadea 20% din puncte
    target.health = target.health - (target.health*10)/100; //celui atacat i se va scadea 10% din viata
    cout << attacker.name << " attack " << target.name << " with " << attacker.weapon1 <<" and received -" <<(target.health*10)/100 << " HP" <<  endl;
}

void attack2(villain& attacker, Character& target)
{
    attacker.attack_points=attacker.attack_points - (attacker.attack_points*30)/100;//celui care ataca i se va scadea 20% din puncte
    target.health = target.health - (target.health*20)/100; //celui atacat i se va scadea 10% din viata
    cout << attacker.name << " attack " << target.name << " with " << attacker.weapon2 <<" and received -" <<(target.health*20)/100 << " HP" <<  endl;
}

void special_attack(villain& attacker, Character& target)  //valoare random
{
    //rand(time(0));
    int numarMaxim = 40;
    int random = rand() % (numarMaxim + 1);
    attacker.attack_points=attacker.attack_points - 30;
    target.health = target.health - random;
    cout << attacker.name << " attack " << target.name << " with " << attacker.special_attack_name <<" and received -" <<(target.health*20)/100 << " HP"
}
```

<u>Now, let’s RUN our code !</u>

If we are running our code, we will have to choose between 2 game modes:



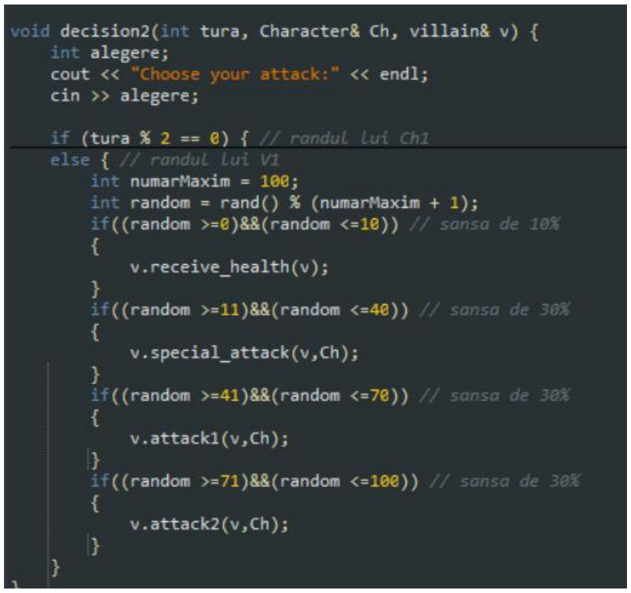
```
Choose your game mode:

1 - Story Mode (vs AI)                    2 - Player vs Player
```

- **STORY MODE:** In this game mode, you will face computer-controlled three opponents who make their own decisions.

🤔 How are the decisions taken by the computer ?

->The game uses an algorithm to determine the decisions made by the computer-controlled villains. The algorithm generates a random number between 1 and 5 each time the villain needs to make a decision, and the function of his class will be called.

The algorithm looks like in the picture below:



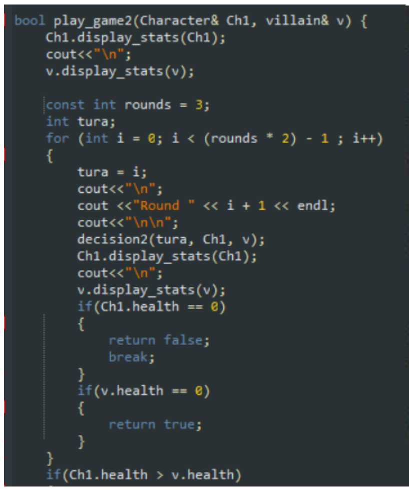
```cpp
void decision2(int tura, Character& Ch, villain& v) {
    int alegere;
    cout << "Choose your attack:" << endl;
    cin >> alegere;

    if (tura % 2 == 0) { // randul lui Ch1
    else { // randul lui V1
        int numarMaxim = 100;
        int random = rand() % (numarMaxim + 1);
        if((random >=0)&&(random <=10)) // sansa de 10%
        {
            v.receive_health(v);
        }
        if((random >=11)&&(random <=40)) // sansa de 30%
        {
            v.special_attack(v,Ch);
        }
        if((random >=41)&&(random <=70)) // sansa de 30%
        {
            v.attack1(v,Ch);
        }
        if((random >=71)&&(random <=100)) // sansa de 30%
        {
            v.attack2(v,Ch);
        }
    }
```

It has a:

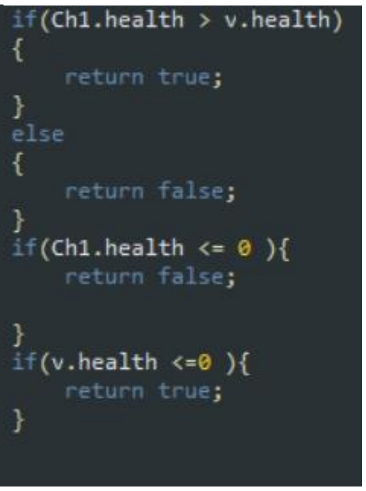
- 10% chance to attempt to heal itself
- 30% chance to attack using one of the three methods available.

The winner between your created character and the villain is determined by a Boolean function that takes into account several logical conditions. If your character satisfies these conditions and wins against the villain, the function will return TRUE. Otherwise, it will return FALSE.

The code of the function is the following:

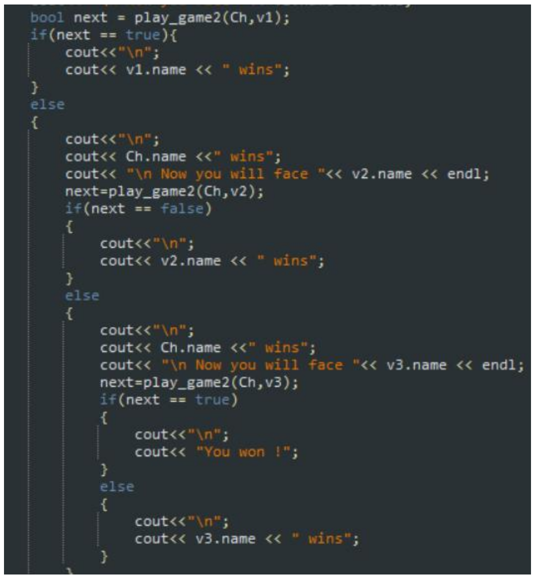


```cpp
bool play_game2(Character& Ch1, villain& v) {
    Ch1.display_stats(Ch1);
    cout<<"\n";
    v.display_stats(v);

    const int rounds = 3;
    int tura;
    for (int i = 0; i < (rounds * 2) - 1 ; i++)
    {
        tura = i;
        cout<<"\n";
        cout <<"Round " << i + 1 << endl;
        cout<<"\n\n";
        decision2(tura, Ch1, v);
        Ch1.display_stats(Ch1);
        cout<<"\n";
        v.display_stats(v);
        if(Ch1.health == 0)
        {
            return false;
            break;
        }
        if(v.health == 0)
        {
            return true;
        }
    }
    if(Ch1.health > v.health)
```



```cpp
if(Ch1.health > v.health)
{
    return true;
}
else
{
    return false;
}
if(Ch1.health <= 0 ){
    return false;

}
if(v.health <=0 ){
    return true;
}
```

Now after the fight between your character and the first villain is over, the program needs to take some decisions, based on the function mentioned before. If you win against the first villan, you have to fight against the next one, and the decision is made in the following manner(using logical operators):



```
bool next = play_game2(Ch,v1);
if(next == true){
    cout<<"\n";
    cout<< v1.name << " wins";
}
else
{
    cout<<"\n";
    cout<< Ch.name <<" wins";
    cout<< "\n Now you will face "<< v2.name << endl;
    next=play_game2(Ch,v2);
    if(next == false)
    {
        cout<<"\n";
        cout<< v2.name << " wins";
    }
    else
    {
        cout<<"\n";
        cout<< Ch.name <<" wins";
        cout<< "\n Now you will face "<< v3.name << endl;
        next=play_game2(Ch,v3);
        if(next == true)
        {
            cout<<"\n";
            cout<< "You won !";
        }
        else
        {
            cout<<"\n";
            cout<< v3.name << " wins";
        }
```

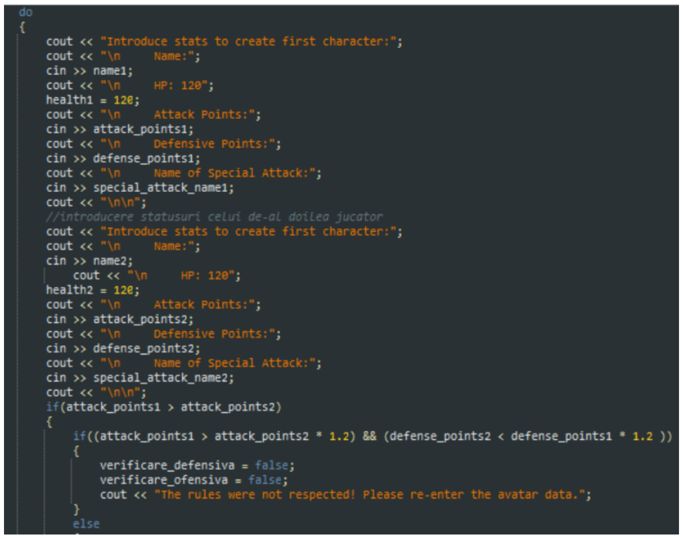
If you are winning against all of the three villains, you **WIN** the game !

- **Player vs Player:** in this game mode ,you will play against your friend, and also this game mod you have to risk and have fun. 😄

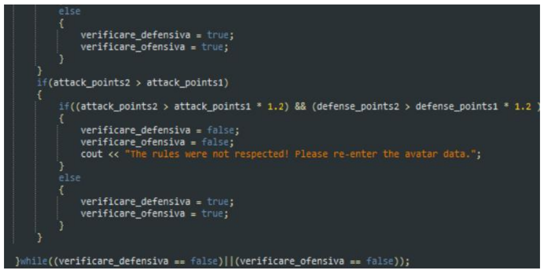
In this game mode, players have the freedom to customize their own characters, allowing for a more personalized and engaging gameplay experience, but they have to respect some **RULES**:

If one player has higher defensive points than the other, they will need to allocate fewer attack points. Otherwise, the players will have to re-enter their data. The difference between the same type of points cannot vary by more than 20%. The minimum allowed value for both, attack and defense points is 80.

If the attack and the defensive points don't respect the rule, the players will have to re-entry the data. This was made in the following piece of code:



```
do
{
    cout << "Introduce stats to create first character:";
    cout << "\n     Name:";
    cin >> name1;
    cout << "\n     HP: 120";
    health1 = 120;
    cout << "\n     Attack Points:";
    cin >> attack_points1;
    cout << "\n     Defensive Points:";
    cin >> defense_points1;
    cout << "\n     Name of Special Attack:";
    cin >> special_attack_name1;
    cout << "\n\n";
    //introducere statusuri celui de-al doilea jucator
    cout << "Introduce stats to create first character:";
    cout << "\n     Name:";
    cin >> name2;
        cout << "\n     HP: 120";
    health2 = 120;
    cout << "\n     Attack Points:";
    cin >> attack_points2;
    cout << "\n     Defensive Points:";
    cin >> defense_points2;
    cout << "\n     Name of Special Attack:";
    cin >> special_attack_name2;
    cout << "\n\n";
    if(attack_points1 > attack_points2)
    {
        if((attack_points1 > attack_points2 * 1.2) && (defense_points2 < defense_points1 * 1.2 ))
        {
            verificare_defensiva = false;
            verificare_ofensiva = false;
            cout << "The rules were not respected! Please re-enter the avatar data.";
        }
        else
```



```
        else
        {
            verificare_defensiva = true;
            verificare_ofensiva = true;
        }
    }
    if(attack_points2 > attack_points1)
    {
        if((attack_points2 > attack_points1 * 1.2) && (defense_points2 > defense_points1 * 1.2 )
        {
            verificare_defensiva = false;
            verificare_ofensiva = false;
            cout << "The rules were not respected! Please re-enter the avatar data.";
        }
        else
        {
            verificare_defensiva = true;
            verificare_ofensiva = true;
        }
    }

}while((verificare_defensiva == false)||(verificare_ofensiva == false));
```

After every round, all the status for both players will be displayed on the screen.

Conclusion

Overall, my project seems to be a fun and engaging game that involves creating custom characters and battling against various villains with different attributes and weapons. The use of classes, functions, and randomization adds an element of unpredictability to the game, making each playthrough unique. The ability to customize your character's attack and defense points adds a strategic element to the game, and the implementation of different attack types and special abilities adds variety to the battles.

The algorithm used for the villain's decision-making process adds an element of challenge to the game, as the player never knows what attack the villain will use. The implementation of a Boolean function to determine the winner of each battle and the progression to the next villain adds a sense of achievement and progression to the game.

I hope you enjoy and have fun playing the game! ☺