МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Лабораторная работа № 3

Выполнил студент группы М3203:

Грохотов Владислав Геннадиевич

САНКТ-ПЕТЕРБУРГ

2019

**Задание:**

Определить класс **BattleUnitsStack**. Этот класс должен описывать всю необходимую информацию для участия стека в сражении (следует учесть возможность воскрешения, изменения параметров и т.п.). Экземпляр должен формироваться на основе “походного” аналога. Реализовать все необходимые методы для управления параметрами.

Определить класс **BattleArmy**. Этот класс должен описывать всю необходимую информацию о “боевой” армии (содержит “боевые” стеки). Следует учесть возможность добавления стеков (например, магия “Призвать элементалей” - добавляет стек элементалей). Максимальное количество стеков в боевой армии - девять. Реализовать все необходимые методы для управления армией.

Определить класс **Battle**. Это основной класс движка, все взаимодействие должно происходить через него. Экземпляр создается на основании двух боевых армий (при желании можно сделать боевку не для двух, а произвольного количества армий). Класс должен предоставлять информацию о текущем ходе (какой юнит ходит, какие действия доступны) и возможность совершить действие. Также необходимо получать информацию о статусе сражения: завершено или нет, кто победил.

Пока реализацию данного класса можно полностью не делать (то, что связано с информацией о ходе и совершении действия), достаточно будет продемонстрировать интерфейс с “заглушками”. Эти заглушки заменятся на нормальную реализацию в следующей лабе.

**Ход рассуждений:**

В стаке юнитов было решено ввести поле HP – здоровье всего стака, чтобы было удобнее наносить урон стаку и уже по оставшемуся здоровью высчитывать оставшееся количество.

Для учета потерь введено поле StartAmount.

Для учета количества стаков в армии введена новая константа в файл Config.cs и добавлен метод NumberOfAliveStacks.

Задана проверка, жив ли стак или армия.

В классе Battle реализованы методы, проверяющие, закончилось ли сражение, кто победил.

Механизмы защиты данных скопирован с походного аналога.

**Код:**

1. BattleUnitsStack.cs

using System.Collections.Generic;

using lab2;

namespace lab3

{

public class BattleUnitsStack

{

public Unit UnitType { get; }

public int StartAmount { get; }

public int Amount => ( this.Hp / (int)(this.UnitType.HitPoints));

public int Hp { get; }

public bool IsAlive => this.Hp > 0;

public List<Effects> Effects;

public BattleUnitsStack(UnitsStack unitsStack)

{

this.UnitType = unitsStack.UnitType;

this.StartAmount = unitsStack.Amount;

this.Hp = unitsStack.Amount \* (int) (unitsStack.UnitType.HitPoints);

}

public BattleUnitsStack Clone()

{

return new BattleUnitsStack(new UnitsStack(this.UnitType.Clone(), this.Amount));

}

}

}

1. BattleArmy.cs

using System;

using System.Collections.Generic;

using lab2;

namespace lab3

{

public class BattleArmy

{

private readonly List<BattleUnitsStack> \_stacksList;

public string ArmyName{ get; }

public List<BattleUnitsStack> StacksList

{

get

{

var newStacksList = new List<BattleUnitsStack>();

\_stacksList.ForEach((stack) => newStacksList.Add(stack.Clone()));

return newStacksList;

}

}

public int NumberOfAliveStacks

{

get

{

int number = 0;

foreach (var stack in this.StacksList)

{

if (stack.IsAlive)

number += 1;

}

return number;

}

}

public void AddStack(BattleUnitsStack battleUnitsStack)

{

if (this.NumberOfAliveStacks < Config.MAX\_ARMY\_NUMBER)

\_stacksList.Add(battleUnitsStack);

else

throw new ArgumentException("To much BattleStacks");

}

public bool IsArmyAlive()

{

if (this.NumberOfAliveStacks > 0)

return true;

else

return false;

}

public BattleArmy(List<BattleUnitsStack> stacksList, string name)

{

if (stacksList.Count > Config.MAX\_ARMY\_NUMBER)

{

throw new ArgumentException("To much Stacks");

}

var newStacksList = new List<BattleUnitsStack>();

stacksList.ForEach((stack) => newStacksList.Add(stack.Clone()));

this.\_stacksList = newStacksList;

ArmyName = name;

}

public override string ToString()

{

string result = $"Army {ArmyName} :\n";

foreach (var stack in \_stacksList)

{

result += stack.ToString();

}

return result;

}

public BattleArmy Clone()

{

return new BattleArmy(this.StacksList, this.ArmyName);

}

}

}

1. Battle.cs

using System;

using lab2;

namespace lab3

{

class Battle

{

private BattleArmy FirstBattleArmy;

private BattleArmy SecondBattleArmy;

public bool HasBattleEnded => !FirstBattleArmy.IsArmyAlive() || !SecondBattleArmy.IsArmyAlive();

public BattleArmy WhoWin()

{

if (FirstBattleArmy.IsArmyAlive())

{

return FirstBattleArmy;

}

else

{

return SecondBattleArmy;

}

}

public Battle(BattleArmy firstBattleArmy, BattleArmy secondBattleArmy)

{

this.FirstBattleArmy = firstBattleArmy.Clone();

this.SecondBattleArmy = secondBattleArmy.Clone;

}

public void StartBattle()

{

while (!HasBattleEnded)

{

///

}

}

}

}