МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №14
по дисциплине: Объектно-ориентированное программирование
Тема: Тестирование. Знакомство с TDD. Тесты как способ
формирования архитектуры

Выполнил: студент группы ПВ-223
Дмитриев А.А.
Проверил:
Черников С.В.

Белгород 2024 г.

Цель работы: знакомство с понятием тестирования. Получение практических навыков для написания модульных тестов.

Задание:
Разработать программу на языке программирования Python и модульные тесты с использованием библиотеки pytest в соответствии с вариантом задания.
Задание на разработку ПО в лабораторной состоит из двух частей (начальные условия задачи, финальные условия задачи). В Таблице 1 приведены условия задач.

Вариант 2:

Разработать подсистему для стрельбы из игрового оружия. «Игровое оружие» может выполнять стрельбу, и имеет возможность отображать себя на экран	Теперь оружие получает способность перезарядки. Появляется такой вид оружие, как нож.

Код (Тесты 1):

```
from pytest import *

from begin.item import *
from begin.player import Player
from begin.window import *


@mark.parametrize("players_team_first, players_team_second",
                  [
                      (
                              [Player("C", 150, AK74(), (6, 7))],
                              [Player("A", 90, AK74(), (0, 0)),
                               Player("B", 100, TT(), (0, 2))]
                      )
                  ], )
def test_add_players(players_team_first, players_team_second):
    window = Window()
    env = window.get_environment()

    for player in players_team_first:
        env.add_player_to_team_first(player)

    for player in players_team_second:
        env.add_player_to_team_second(player)

    assert env.team_first == players_team_first
    assert env.team_second == players_team_second


@mark.parametrize("player, result",
                  [
                      (
                              Player("C", 150, AK74(), (6, 7)),
                              f"Pl: C HP: 150",
                      ),
                      (
                              Player("E", 100, TT(), (6, 7)),
                              f"Pl: E HP: 100",
                      )
                  ])
def test_display_player(player, result):
    assert player.display() == result


@mark.parametrize("player, result",
                  [
                      (
                              Player("C", 150, AK74(), (6, 7)),
                              "AK74",
                      ),
                      (
                              Player("E", 100, TT(), (6, 7)),
                              "TT",
                      )
                  ])
def test_display_item_of_player(player, result):
    assert player.item.display() == result


@mark.parametrize("attacker, defender, result",
                  [
                      (
                              Player("C", 150, AK74(), (6, 7)),
                              Player("E", 100, TT(), (6, 7)),
                              60,
                      ),
```

```
                    (
                            Player("C", 150, AK74(), (6, 7)),
                            Player("E", 10, TT(), (6, 7)),
                            0,
                    )
                ])
def test_attack(attacker, defender, result):
    attacker.use_on(defender)
    assert defender.hp == result
```

Код (Исходник 1):

```
class DisplayBehavior:
    def display(self) -> str: pass


class DisplayAK74(DisplayBehavior):
    def display(self):
        return "AK74"


class DisplayTT(DisplayBehavior):
    def display(self):
        return "TT"

class Environment:
    def __init__(self):
        self.team_first = []
        self.team_second = []

        self.battleMap = []

    @staticmethod
    def distance_to(player, target):
        return math.sqrt((player.coord[0] - target.coord[0]) ** 2 + (player.coord[1] -
target.coord[1]) ** 2)

    @staticmethod
    def comp_hp(player, delta_hp):
        player.hp += delta_hp

    def add_player_to_team_first(self, player):
        self.add_player_to_team(player, self.team_first)

    def add_player_to_team_second(self, player):
        self.add_player_to_team(player, self.team_second)

    @staticmethod
    def add_player_to_team(player, team: list):
        team.append(player)

    def remove_player_from_team_first(self, player):
        self.remove_player_from_team(player, self.team_first)

    def remove_player_from_team_second(self, player):
        self.remove_player_from_team(player, self.team_second)

    @staticmethod
    def remove_player_from_team(player, team: list):
        team.remove(player)

    def move_player(self, player, coord: tuple):
        pass

class Item(UseBehavior, DisplayBehavior):
    def __init__(self, use_behavior, display_behavior):
        self.use_behavior = use_behavior
```

```
        self.display_behavior = display_behavior

    def use_on(self, player, target):
        self.use_behavior.use_on(player, target)

    def display(self):
        return self.display_behavior.display()


class AK74(Item):
    def __init__(self):
        super().__init__(DistanceAttack(), DisplayAK74())


class TT(Item):
    def __init__(self):
        super().__init__(DistanceAttack(), DisplayTT())


class Direction:
    UP = (0, -1)
    DOWN = (0, 1)
    LEFT = (-1, 0)
    RIGHT = (1, 0)


class Player(DisplayBehavior):
    def __init__(self, name: str, hp: int, item: Item, coord: tuple):
        self.name = name
        self.hp = hp
        self.item = item
        self.coord = coord

    def display(self):
        return f"Pl: {self.name} HP: {self.hp}"

    def use_on(self, target):
        self.item.use_on(self, target)

    def move(self, direction: Direction, steps):
        pass


class UseBehavior:

    def use_on(self, player, target): pass


class DistanceAttack(UseBehavior):
    def use_on(self, player, target):
        if Environment.distance_to(player, target) < 10:
            Environment.comp_hp(target, -40 if target.hp > 40 else -target.hp)

from begin.environment import Environment


class Window:
    def __init__(self):
        self.environment = Environment()
        # some logic for displaying

    def get_environment(self):
        return self.environment

    def display_to_window(self, object):
        pass
```

Код (Тесты 2):

```
from pytest import *

from finish.item import *
from finish.player import Player
from finish.window import *


@mark.parametrize("players_team_first, players_team_second",
                  [
                      (
                              [Player("C", 150, AK74(), (6, 7))],
                              [Player("A", 90, AK74(), (0, 0)),
                               Player("B", 100, TT(), (0, 2))]
                      )
                  ], )
def test_add_players(players_team_first, players_team_second):
    window = Window()
    env = window.get_environment()

    for player in players_team_first:
        env.add_player_to_team_first(player)

    for player in players_team_second:
        env.add_player_to_team_second(player)

    assert env.team_first == players_team_first
    assert env.team_second == players_team_second


@mark.parametrize("player, result",
                  [
                      (
                              Player("C", 150, AK74(), (6, 7)),
                              f"Pl: C HP: 150",
                      ),
                      (
                              Player("E", 100, TT(), (6, 7)),
                              f"Pl: E HP: 100",
                      )
                  ])
def test_display_player(player, result):
    assert player.display() == result


@mark.parametrize("player, result",
                  [
                      (
                              Player("C", 150, AK74(), (6, 7)),
                              "AK74",
                      ),
                      (
                              Player("E", 100, TT(), (6, 7)),
                              "TT",
                      )
                  ])
def test_display_item_of_player(player, result):
    assert player.item.display() == result


@mark.parametrize("player, target, result",
                  [
```

```
                    (
                            Player("C", 150, AK74(), (6, 7)),
                            Player("E", 100, TT(), (6, 7)),
                            60,
                    ),
                    (
                            Player("C", 150, AK74(), (6, 7)),
                            Player("E", 10, TT(), (6, 7)),
                            0,
                    ),
                    (
                            Player("C", 150, AK74(), (0, 0)),
                            Player("E", 10, TT(), (20, 20)),
                            10,
                    )
                ])
def test_attack(player, target, result):
    player.use_on(target)
    assert target.hp == result


@mark.parametrize("player, target, result",
                [(
                        Player("C", 150, Knife(), (7, 7)),
                        Player("E", 100, AK74(), (7, 7)),
                        0
                )]
                )
def test_knife_attack(player, target, result):
    player.use_on(target)
    assert target.hp == result


@mark.parametrize("player, result",
                [(
                        Player("C", 150, Knife(), (6, 7)),
                        "Knife"
                )]
                )
def test_knife_display(player, result):
    assert player.item.display() == result


@mark.parametrize("player",
                [(
                        Player("A", 150, AK74(), (6, 7))
                )]
                )
def test_reload(player):
    temp_player = Player("C", 100, AK74(), (6, 7))
    while player.item.ammunition_behavior.curr_ammo > 0:
        player.use_on(temp_player)

    other_player = Player("D", 100, Knife(), (6, 7))
    player.use_on(other_player)

    assert player.item.ammunition_behavior.curr_ammo == 0
    assert other_player.hp == 100

    player.reload()
    player.use_on(other_player)

    assert player.item.ammunition_behavior.curr_ammo ==
player.item.ammunition_behavior.max_ammo - 1
    assert other_player.hp != 100
```

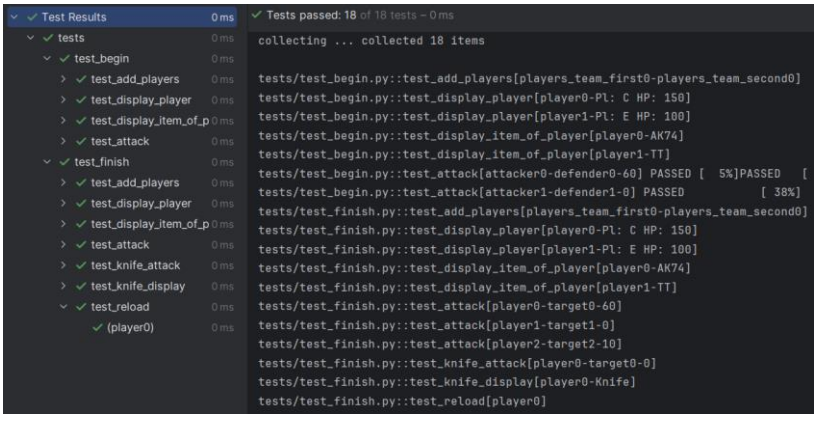
Код (Исходник 2):

```
class AmmunitionBehavior:

    def spend_ammo(self) -> bool: pass

    def reload(self): pass


class AmmunitionOn30(AmmunitionBehavior):
    curr_ammo = 30
    max_ammo = 30

    def spend_ammo(self):
        if self.curr_ammo > 0:
            self.curr_ammo -= 1
            return True

        return False

    def reload(self):
        self.curr_ammo = self.max_ammo


class AmmunitionOn8(AmmunitionBehavior):
    curr_ammo = 8
    max_ammo = 8

    def spend_ammo(self):
        if self.curr_ammo > 0:
            self.curr_ammo -= 1
            return True

        return False

    def reload(self):
        self.curr_ammo = self.max_ammo


class NoAmmunition(AmmunitionBehavior):
    def spend_ammo(self):
        return True

    def reload(self):
        pass

class DisplayBehavior:
    def display(self) -> str: pass


class DisplayAK74(DisplayBehavior):
    def display(self):
        return "AK74"


class DisplayTT(DisplayBehavior):
    def display(self):
        return "TT"


class DisplayKnife(DisplayBehavior):
    def display(self):
        return "Knife"


class Environment:
```

```
    def __init__(self):
        self.team_first = []
        self.team_second = []

        self.battleMap = []

    @staticmethod
    def distance_to(player, target):
        return math.sqrt((player.coord[0] - target.coord[0]) ** 2 + (player.coord[1] -
target.coord[1]) ** 2)

    @staticmethod
    def comp_hp(player, delta_hp):
        player.hp += delta_hp

    def add_player_to_team_first(self, player):
        self.add_player_to_team(player, self.team_first)

    def add_player_to_team_second(self, player):
        self.add_player_to_team(player, self.team_second)

    @staticmethod
    def add_player_to_team(player, team: list):
        team.append(player)

    def remove_player_from_team_first(self, player):
        self.remove_player_from_team(player, self.team_first)

    def remove_player_from_team_second(self, player):
        self.remove_player_from_team(player, self.team_second)

    @staticmethod
    def remove_player_from_team(player, team: list):
        team.remove(player)

    def move_player(self, player, coord: tuple):
        pass
from finish.ammunition_behavior import AmmunitionBehavior, AmmunitionOn30,
AmmunitionOn8, NoAmmunition
from finish.use_behavior import DistanceAttack, UseBehavior, MeleeAttack
from finish.display_behavior import DisplayAK74, DisplayTT, DisplayBehavior,
DisplayKnife


class Item(UseBehavior, DisplayBehavior, AmmunitionBehavior):
    def __init__(self, use_behavior, display_behavior, ammunition_behavior):
        self.use_behavior = use_behavior
        self.display_behavior = display_behavior
        self.ammunition_behavior = ammunition_behavior

    def use_on(self, player, target):
        if self.ammunition_behavior.spend_ammo():
            self.use_behavior.use_on(player, target)

    def display(self):
        return self.display_behavior.display()

    def reload(self):
        self.ammunition_behavior.reload()


class AK74(Item):
    def __init__(self):
        super().__init__(DistanceAttack(), DisplayAK74(), AmmunitionOn30())


class TT(Item):
```

```
    def __init__(self):
        super().__init__(DistanceAttack(), DisplayTT(), AmmunitionOn8())


class Knife(Item):
    def __init__(self):
        super().__init__(MeleeAttack(), DisplayKnife(), NoAmmunition())


class Direction:
    UP = (0, -1)
    DOWN = (0, 1)
    LEFT = (-1, 0)
    RIGHT = (1, 0)


class Player(DisplayBehavior):
    def __init__(self, name: str, hp: int, item: Item, coord: tuple):
        self.name = name
        self.hp = hp
        self.item = item
        self.coord = coord

    def display(self):
        return f"Pl: {self.name} HP: {self.hp}"

    def use_on(self, target):
        self.item.use_on(self, target)

    def reload(self):
        self.item.reload()

    def move(self, direction: Direction, steps):
        pass


class UseBehavior:

    def use_on(self, player, target): pass


class DistanceAttack(UseBehavior):
    def use_on(self, player, target):
        if Environment.distance_to(player, target) < 10:
            Environment.comp_hp(target, -40 if target.hp > 40 else -target.hp)


class MeleeAttack(UseBehavior):
    def use_on(self, player, target):
        if Environment.distance_to(player, target) <= 1:
            Environment.comp_hp(target, -200 if target.hp > 200 else -target.hp)
```

Пример работы:



Вывод: В ходе лабораторной работы познакомились с понятием тестирования. Получили практические навыки для написания модульных тестов.