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

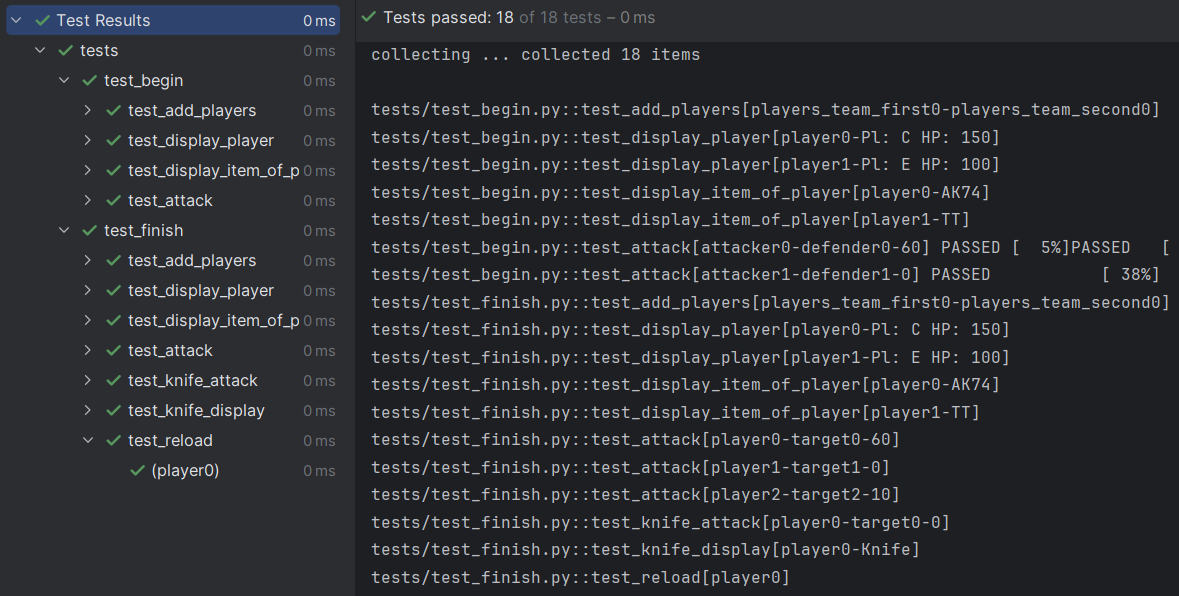
class DisplayBehavior:  
 def display(self) -> str: pass  
  
  
class DisplayAK74(DisplayBehavior):  
 def display(self):  
 return "AK74"  
  
  
class DisplayTT(DisplayBehavior):  
 def display(self):  
 return "TT"  
  
  
class DisplayKnife(DisplayBehavior):  
 def display(self):  
 return "Knife"  
  
  
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.