Федеральное государственное автономное образовательное учреждение высшего профессионального образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Лабораторная работа №2

по курсу:

«Технологии программирования»

по теме:

«Основы ООП»

Студент: Князев Иван Викторович
Группа: БИВТ-23-8
Преподаватель: Гласов Александр Владимирович

Москва 2024

Цель работы

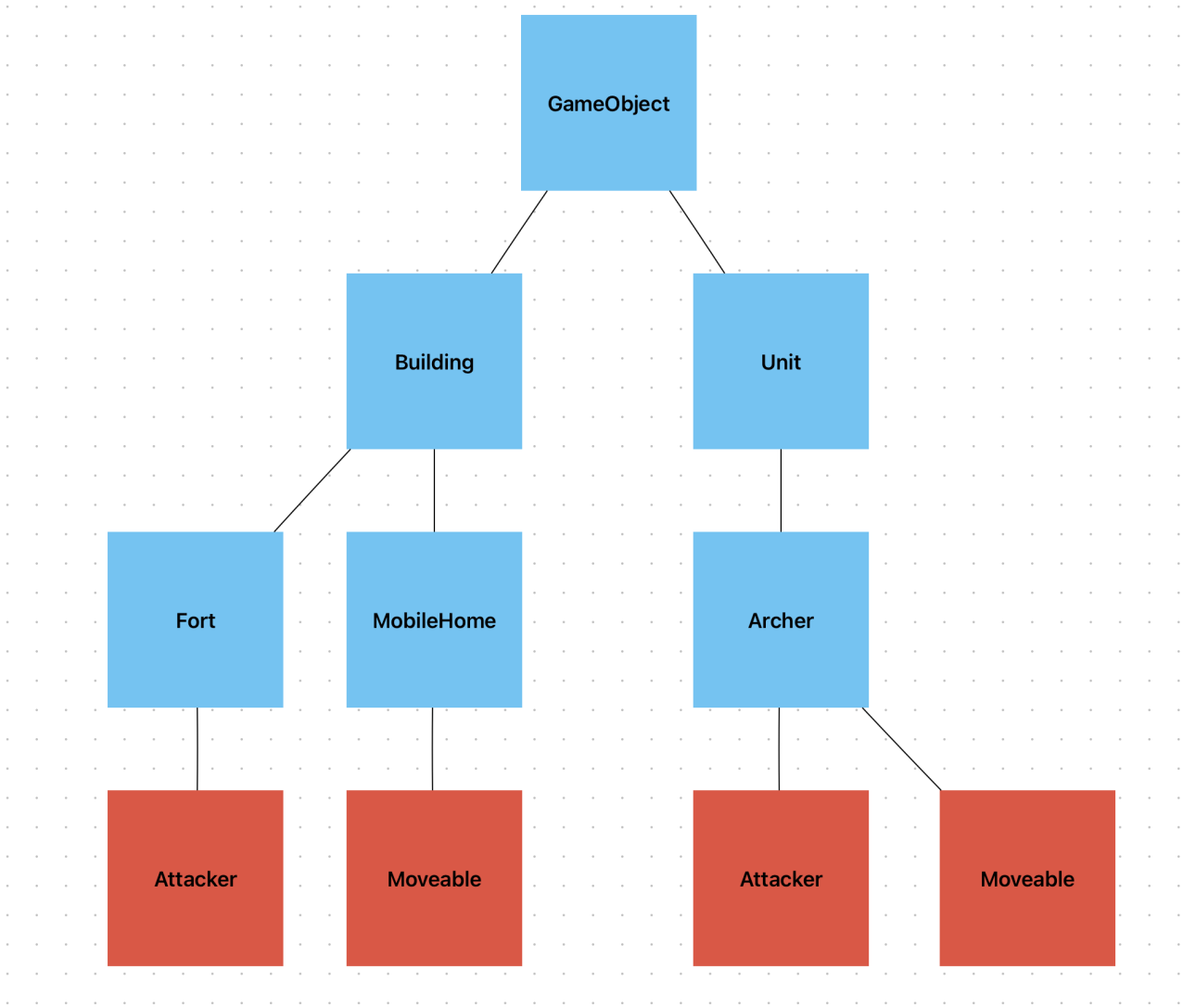
Изучить и освоить основы ООП (Объектно Ориентированного Программирования).

Порядок выполнения роботы:

1) Продумать реализацию всех компонентов согласно Техническому Заданию;
2) Выбрать ЯП (язык программирования) для разработки;
3) Реализовать заданный функционал;
4) Написать основные тесты для проверки корректности работы системы;
5) Запушить проект в удалённый репозиторий.

Ход работы:

Необходимо было реализовать иерархию классов по следующей структуре:



Также было решено сделать все классы кроме Fort, MobileHome и Archer *абстрактными* для того, чтобы подчеркнуть заложенную в данную схему функциональность. Так как интерфейсов в синтаксисе Python нет, то было принято решение реализовать их пи помощи *полностью абстрактных* классов.

Для реализации был выбран ЯП **Python** и следующая структура проекта:

```
.
├── game_app
│   ├── abs_objects.py
│   ├── base_abs_object.py
│   ├── __init__.py
│   ├── interfaces.py
│   └── objects.py
├── main.py
├── README.md
└── Князев_Лаба_2.docx
```

Точка входа — main.py, файлы из директории game_app отвечают за реализацию всех классов по заданной иерархии.

Затем были последовательно реализованы все необходимые компоненты, рассмотренные ниже.

1. Базовый абстрактный класс **GameObject**.

game_app > base_abs_object.py > ...

```python
from abc import ABC, abstractmethod


class GameObject(ABC):

    id_counter: int = 0 # static field for count ID numbrs of objects

    @abstractmethod
    def __init__(self, name: str, X: float = 0, Y: float = 0) -> None:
        GameObject.id_counter += 1
        self.id: int = GameObject.id_counter
        self.name: str = name
        self.X: float = X
        self.Y: float = Y

    def getId(self) -> int:
        return self.id

    def getName(self) -> str:
        return self.name

    def getX(self) -> float:
        return self.X

    def getY(self) -> float:
        return self.Y

    @abstractmethod
    def __repr__(self) -> str:
        object = "GameObject(id={0}, name={1}, X={2}, Y={3})".format(
            self.id, self.name, self.X, self.Y)
        return object

```

2. Абстрактные классы **Unit, Building**.

game_app > abs_objects.py > ...

```python
from abc import ABC, abstractmethod
from game_app.base_abs_object import GameObject


class Unit(GameObject, ABC):
    @abstractmethod
    def __init__(self, name: str, X: float, Y: float,
                 is_alive: bool = True, hp: float = 100) -> None:
        super().__init__(name, X, Y)
        self.is_alive: bool = is_alive
        self.hp: float = hp

    def isAlive(self) -> bool:
        return self.is_alive

    def getHp(self) -> float:
        return self.hp

    def receiveDamage(self, damage: float) -> None:
        if damage <= self.hp:
            self.hp -= damage
            self.is_alive = True if self.hp > 0 else False
        else:
            raise ValueError("Damage of more hp!")


class Building(GameObject, ABC):
    @abstractmethod
    def __init__(self, name: str, X: float, Y: float,
                 is_build: bool = False) -> None:
        super().__init__(name, X, Y)
        self.is_build: bool = is_build

    def isBuilt(self) -> bool:
        return self.is_build
```

3. Интерфейсы **Attacker, Movable**.

game_app > interfaces.py > Moveable

```python
from typing import Union
from abc import ABC, abstractmethod
from game_app.abs_objects import Unit, Building


class Attacker(ABC):
    @abstractmethod
    def attack(self, unit: Unit) -> None:
        """Method to attack a unit and deal damage to their hp"""
        raise NotImplementedError(
                'Implement attack() in %s.' % (self.__class__.__name__))


class Moveable(ABC):
    @abstractmethod
    def move(self, diffX: int, diffY: int) -> None:
        """Method to move a unit or a building"""
        raise NotImplementedError(
                'Implement move() in %s.' % (self.__class__.__name__))
```

4. Классы **Archer, Fort, MobileHouse**.

README.md | main.py | objects.py

game_app > objects.py > MobileHouse > __init__

```python
from game_app.interfaces import Attacker, Moveable
from game_app.abs_objects import Unit, Building

# Set damage variables
archer_damage = 24.0
fort_damage = 19.0


class Archer(Unit, Attacker, Moveable):
    def __init__(self, name: str, X: float, Y: float,
                 is_alive: bool = True, hp: float = 100) -> None:
        super().__init__(name, X, Y, is_alive, hp)

    def attack(self, unit: Unit) -> None:
        unit.receiveDamage(damage=archer_damage)

    def move(self, diffX: int = 5, diffY: int = 5) -> None:
        self.X += diffX
        self.Y += diffY

    def __repr__(self) -> str:
        object = super().__repr__()
        object = "Archer({0}, is_alive={1}, hp={2})".format(
            object[11:-1], self.is_alive, self.hp)
        return object


class Fort(Building, Attacker):
    def __init__(self, name: str, X: float, Y: float,
                 is_build: bool = False) -> None:
        super().__init__(name, X, Y, is_build)

    def attack(self, unit: Unit) -> None:
        unit.receiveDamage(damage=fort_damage)

    def __repr__(self) -> str:
        object = super().__repr__()
        object = "Fort({0}, is_build={1})".format(object[11:-1], self.is_build)
        return object


class MobileHouse(Building, Moveable):
    def __init__(self, name: str, X: float, Y: float,
                 is_build: bool = False) -> None:
        super().__init__(name, X, Y, is_build)

    def move(self, diffX: int = 3, diffY: int = 3) -> None:
        self.X += diffX
        self.Y += diffY

    def __repr__(self) -> str:
        object = super().__repr__()
        object = "MobileHouse({0}, is_build={1})".format(
            object[11:-1], self.is_build)
        return object
```

5. Тестовое приложение в **main.py**.

main.py > test_objects

```python
from game_app.objects import Archer, Fort, MobileHouse


def print_tests_data(header: str, objects: list) -> None:
    print("[TEST]", header)
    print("-" * 70)
    print(*objects, sep="\n")
    print("-" * 70, "\n")


def test_objects() -> None:
    archer_1 = Archer(name="archer_1", X=1, Y=1)
    archer_2 = Archer(name="archer_2", X=1, Y=1)
    fort = Fort(name="fort_1", X=12, Y=15, is_build=True)
    mobile_house = MobileHouse(name="mobile_house_1", X=14, Y=20, is_build=True)
    print_tests_data("At the start:", [archer_1, archer_2, fort, mobile_house])

    archer_1.attack(archer_2)
    archer_2.move(30, 40)
    print_tests_data("After move and attack 'archer_2' from archers:",
                     [archer_1, archer_2])

    fort.attack(archer_1)
    print_tests_data("After attack 'archer_1' from fort:",
                     [archer_1, archer_2])

    mobile_house.move(10, 20)
    print_tests_data("After move mobile_house:", [mobile_house])

    tests_of_methods_1 = [
        f"getId() -> int: {archer_1.getId()}",
        f"getName() -> str: {archer_1.getName()}",
        f"getX() -> int: {archer_1.getX()}",
        f"getY() -> int: {archer_1.getY()}",
        f"isAlive() -> bool: {archer_1.isAlive()}",
        f"getHp() -> float: {archer_1.getHp()}"
    ]
    print_tests_data(
        "Methods of the 'archer_1' object:", tests_of_methods_1)

    tests_of_methods_2 = [
        f"getId() -> int: {fort.getId()}",
        f"getName() -> str: {fort.getName()}",
        f"getX() -> int: {fort.getX()}",
        f"getY() -> int: {fort.getY()}",
        f"isBuilt() -> bool: {fort.isBuilt()}"
    ]
    print_tests_data("Methods of the 'fort' object:",
                     tests_of_methods_2)

    for attack in range(10):
        try:
            fort.attack(archer_2)
            print_tests_data(f"After attack 'archer_2' #{
                             attack+1} from fort:", [archer_2])
        except ValueError as err:
            print(f"Exception: type=ValueError, msg='{err}'")
            break


if __name__ == "__main__":
    test_objects()
```

Также в корневой папке был создан файл **README.md**, в который была добавлена инструкция для запуска проекта и вывод данных о работе тестового приложения из терминала.

После в GitHub был создан новый репозиторий, в который в последующем запушен весь код.

Репозиторий располагается по ссылке:
https://github.com/Ivan-Knyazev/Lab_2_programming_technologies/tree/main