1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. ««Рефакторинг программы»
2. по дисциплине «Объектно-ориентированное программирование»
3. Выполнил
4. студент гр. 5131001/20503 Черникова В. М.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Кубрин Г. С.

<*подпись*>

1. Санкт-Петербург
2. 2024

# Цель работы

Целью работы – освоение особенностей этапов анализа предметной области и проектирования архитектуры программного обеспечения в объектной модели.

# Задачи

В рамках выполнения лабораторной работы необходимо решить следующие задачи:

⎯ анализ предметной области заданной программы, реализованной в процедурной парадигме программирования с выделением ключевых абстракций;

⎯ анализ недостатков архитектуры и кода исходного проекта;

⎯ проектирование новой архитектуры программы в объектной парадигме программирования (рефакторинг) с использованием UML-диаграммы классов (см. UML.docx);

⎯ программная реализация новой архитектуры на языке программирования Python или C++;

⎯ реализовать unit-тесты для основного функционала программы.

# условие

До начала выполнения лабораторной работы исходная программа, для которой будет выполнятся рефакторинг, должна быть согласовано. Выбранная программа должна удовлетворять следующим требованиям:

⎯ программа должна быть реализована в процедурной парадигме;

⎯ в рамках предметной области программы должна быть возможность выделить не менее четырёх клюевых абстракций (классов);

⎯ программа должна содержать не менее 300 строк кода;

⎯ язык программирования исходного проекта – Python, Си (проекты на других языках программирования могут быть использованы только в случае индивидуального согласования с преподавателем).

В случае отсутствия подходящего проекта может быть использован один из предложенных в приложение 1 вариантов.

При выполнения лабораторной работы необходимо соблюдать следующие условия:

1. Итоговая архитектура проекта в объектной парадигме должна содержать не менее пяти классов (с учётом ключевых абстракций предметной области и механизмов реализации).

2. В случае использования языка программирования Python все методы должны содержать аннотацию типов данных (механизм type hinting);

3. Должно быть разработано не менее пяти unit-тестов.

4. Unit-тесты должны покрывать ключевое поведение основных классов программы.

# Ход работы

Работа состояла из нескольких этапов:

1. Декомпозиция и реализация необходимого минимума функций
2. Декомпозиция дополнительных методов и операторов и их реализация
3. Выполнение обертки класса для python

Итак, в результате прохождения этих шагов, были получены следующие результаты.

Первым делом было необходимо определить количество и приблизительный функционал основных классов, их наименование. В итоге получились следующие: InterfaceGraphics, Button, InterfaceField, MyDriver, MyGame, InterfaceEvent. Важно понимать, что изнчальный проект (<https://github.com/domarp-j/minesweeper>) был реализован с графикой библиотеки pygame, но для реализации тестов была написана консольная версия.

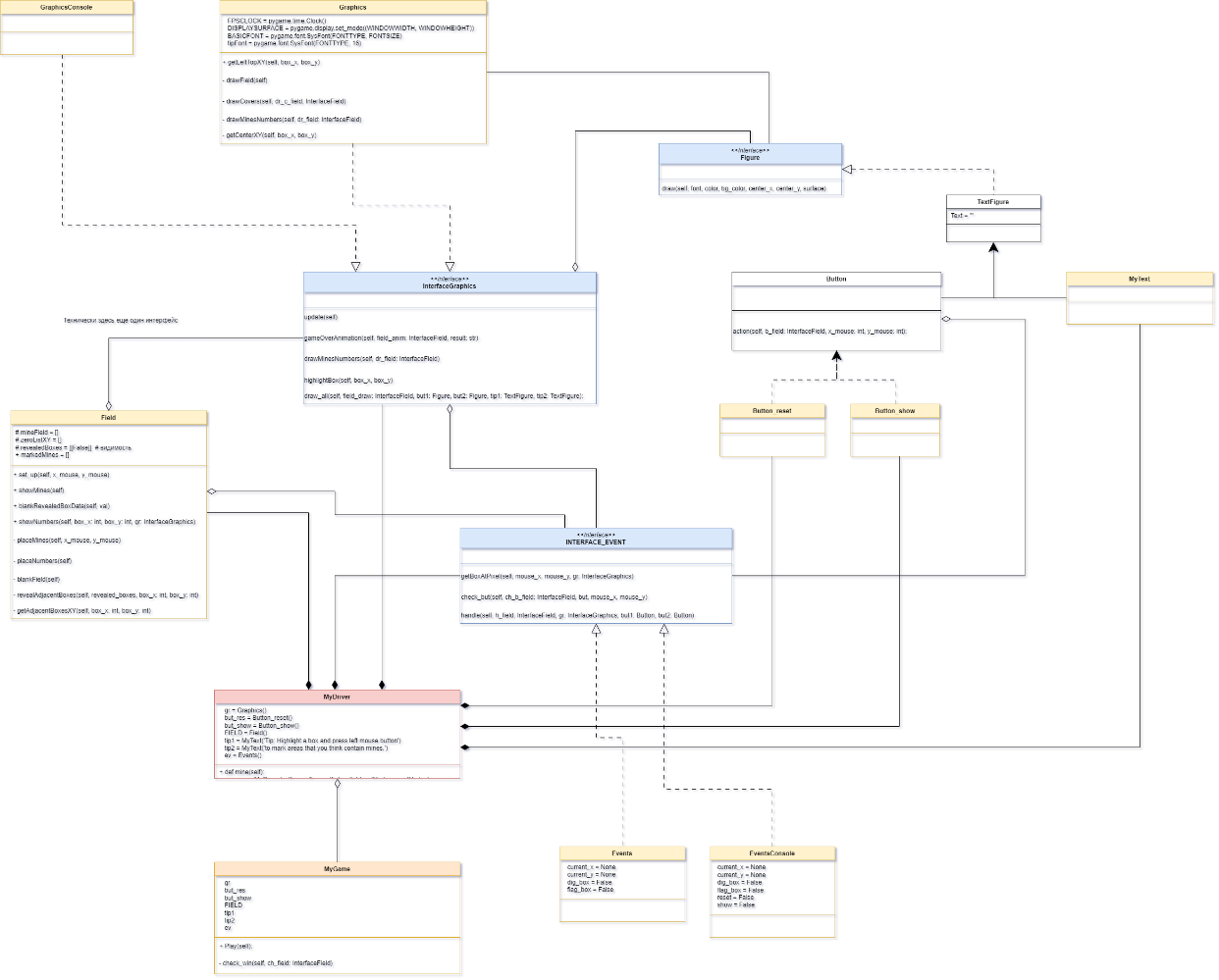


Рисунок 1 - UML-диаграмма проекта

|  |  |
| --- | --- |
| **Название класса** | **Функционал** |
| Field | Поле – хранит параметры поля и его копии, которые отвечают за логику игры. |
| InterfaceEvent | Интерфейс, который реализует взаимодействие игрока с полем |
| Button | Класс, от которого наследуются ButtonReset и ButtonShow – рестарт игры и показать все поле |
| InterfaceGraphics | Интерфейс, который отрисовывет поле игроку |
| MyGame | Бизнес-логика игры |
| MyDriver | Класс, который собирает классы реализации и запускает с ними игру |

После выполнения основной части работы было написано несколько тестов, чтобы проверить работу логики игры и класса событий.

|  |  |
| --- | --- |
| **Название теста** | **Требования теста** |
| test\_ | При попадании игрока на мину происходит проигрыш |
| test\_field | Правильная генерация поля |
| test\_dig | При копании клетка устанавливается в состояние «вскопать» |
| test\_flag | При копании клетка устанавливается в состояние «помечена» |
| test\_show | При «нажатии» на кнопку поле полностью станет видимым |

Тесты с консольным вводом были реализованы с помощью фикстуры monkeypatch, которая имитирует консольный ввод с помощью строки. Данный подход считается нестабильным, но для решения текущей задачи этого достаточно.

# выводы

В результате выполнения лабораторной работы были получены основные навыки создания грамотной декомпозиции классов. Также был получен опыт в написании теста, с помощью модуля pytest, их отладке, запуске и решении проблем, связанных с особенностью тестирования.