

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА** - **Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Математического обеспечения и стандартизации информационных технологий (МОСИТ)

**Отчёт по практической работе №3**

**по дисциплине**

**«Тестирование и верификация программного обеспечения»**

Выполнил студент группы: ИКБО-04-22 Егоров Л.А. *(Ф. И.О.студента)*

Принял ассистент \_\_Петрова А.А.\_\_ *(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024

# Содержание

[1 Использование методологии TDD 3](#_Toc2)

[1.1 Определение персонального варианта 3](#_Toc3)

[1.2 Разработка модуля тестирования 3](#_Toc4)

[1.3 Запуск тестирования 13](#_Toc5)

[1.4 Реализация классов и методов 14](#_Toc6)

[1.4.1 Класс TicTacToe 14](#_Toc7)

[1.4.2 Класс Transition 16](#_Toc8)

[1.4.3 Класс GameState 16](#_Toc9)

[1.4.4 Класс GameMode 17](#_Toc10)

[1.5 Повторный запуск тестирования 17](#_Toc11)

[2 Использование методологии BDD 18](#_Toc12)

[2.1 Описание сценариев BDD 18](#_Toc13)

[2.2. Автоматизация сценариев BDD 20](#_Toc14)

[2.3 Запуск тестов 22](#_Toc15)

[Заключение 25](#_Toc16)

[Приложение 26](#_Toc17)

# 1 Использование методологии TDD

## 1.1 Определение персонального варианта

Шифр студенческого билета: 22И1128, поэтому персональный вариант: 28 - 25 = 3

Задание персонального варианта: Игра «Крестики-нолики».

## 1.2 Разработка модуля тестирования

Первый шаг в методологии TDD (Test-Driven Development) — это создание тестов, которые описывают ожидаемое поведение функций программы. Тестирование будет реализовано с использованием библиотеки «unittest» для языка Python.

Для модуля «Крестики-нолики» нам потребуется класс, который будет взаимодействовать с пользователем посредством меню, предлагая начать игру или выбрать режим. Также для программы понадобятся классы-перечисления: для конечного автомата состояний игры и для возможных режимов игры. Основной класс может включать в себя поля, отвечающие за состояние игры, текущий режим и за игровое поле.

Описание тестов для модуля «Крестики-нолики»:

Все тесты направлены на проверку правильной работы меню и правильного хода игры в зависимости от ввода пользователей. Тесты проверяют:

Точность перехода между состояниями программы.

Корректная обработка любого ввода от пользователя.

Правильный вывод игрового поля.

Корректная проверка условия выигрыша или ничьи.

Все тесты были разделены на четыре категории, и по ним же распределены по четырём классам:

**Тест TestChangeState:**

**Цель:** убедиться, что программа корректно переходит между состояниями и не допускает некорректный ввод в каждом из состояний.

1. test\_menu\_state

* Описание: проверка на то, что программа при запуске находится в состоянии «Меню»
* Ожидаемый результат: состояние программы - «Меню»

1. test\_change\_to\_input

* Описание: проверка перехода в состояние ввода режима игры
* Ожидаемый результат: после ввода пользователем единицы программа должна ожидать от пользователя ввод режима игры

1. test\_input\_game\_mode

* Описание: проверка возврата в меню после ввода режима игры
* Ожидаемый результат: программа должна вернуться в состояние меню после ввода режима игры или выдать ошибку при некорректном вводе

1. test\_start\_game

* Описание: проверка перехода в режим игры
* Ожидаемый результат: после ввода двойки в меню должна начаться игра

1. test\_change\_player

* Описание: после хода каждого игрока ход должен передаваться другому игроку
* Ожидаемый результат: после начала игры ход у первого игрока, он ходит, и ход передаётся второму игроку

На Рисунке 1.2.1 представлен тестовый класс TestChangeState.



Рисунок 1.2.1 – Тестовый класс TestChangeState

**Тест TestPrintBoard:**

**Цель:** убедиться, что программа выведет корректное игровое поле в любой момент игры и при любом режиме игры.

1. test\_print\_classic

* Описание: проверка корректного вывода игрового поля в начале стандартной игры
* Ожидаемый результат: программа вернёт строку с корректно построенным полем

1. test\_print\_extended

* Описание: проверка корректного вывода игрового поля в начале расширенной игры
* Ожидаемый результат: программа вернёт строку с корректно построенным полем 5\*5

1. test\_print\_classic\_with\_symbols

* Описание: проверка корректного вывода игрового поля в ходе обычной игры
* Ожидаемый результат: программа вернёт строку с корректно построенным полем, содержащим на себе символы «x» и «o» в корректных местах

1. test\_print\_extended\_with\_symbols

* Описание: проверка корректного вывода игрового поля в ходе расширенной игры
* Ожидаемый результат: программа вернёт строку с корректно построенным полем, содержащим на себе символы «x» и «o» в корректных местах

На Рисунках 1.2.2 - 1.2.3 представлен тестовый класс TestPrintBoard.

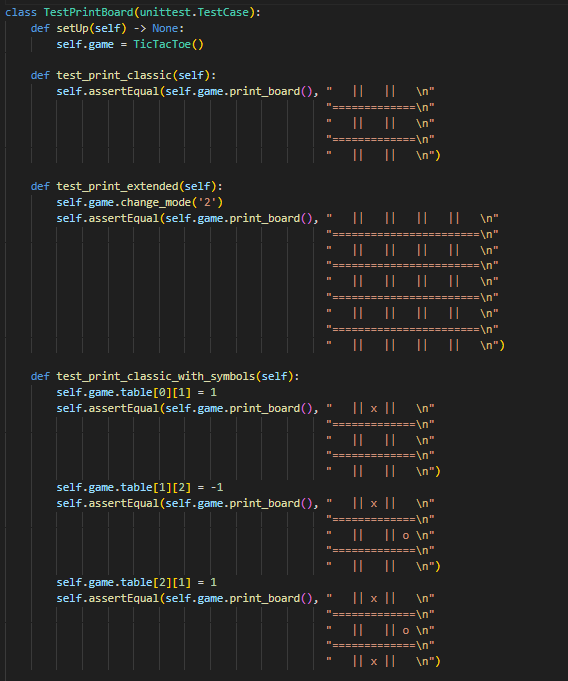


Рисунок 1.2.2 – Тестовый класс TestPrintBoard

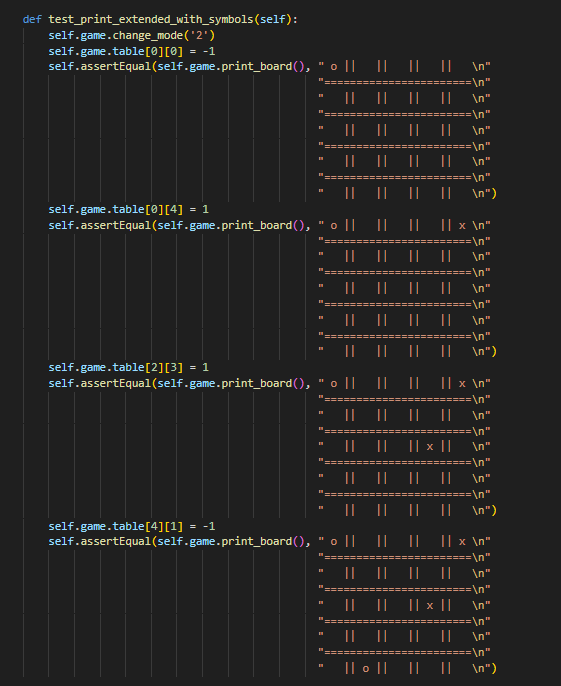


Рисунок 1.2.3 – Тестовый класс TestPrintBoard

**Тест TestInputPosition:**

**Цель:** проверить корректность обработки пользовательского ввода в разных режимах игры.

1. test\_input\_classic

* Описание: проверка корректности ввода позиции игроками в стандартной игре
* Ожидаемый результат: программа должна сохранять заполненные ячейки на игровом поле в случае корректного ввода или выводить сообщение об ошибке в случае некорректного ввода

1. test\_input\_extended

* Описание: проверка корректности ввода позиции игроками в стандартной игре
* Ожидаемый результат: программа должна сохранять заполненные ячейки на игровом поле в случае корректного ввода или выводить сообщение об ошибке в случае некорректного ввода

На Рисунке 1.2.4 представлен тестовый класс TestInputPosition.

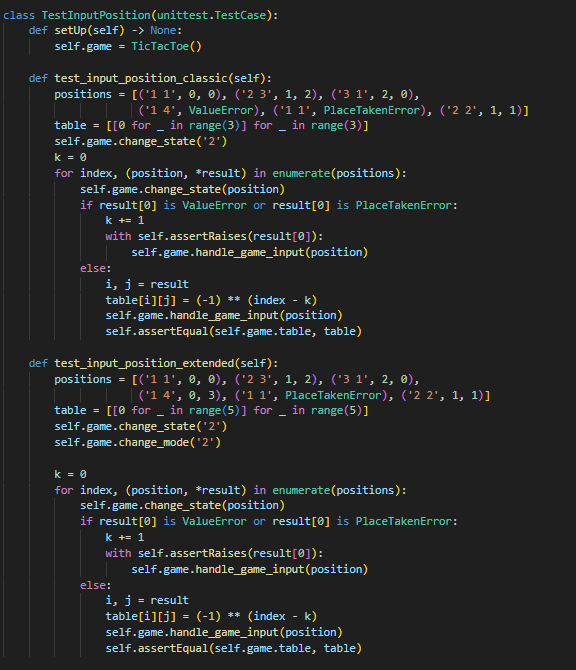


Рисунок 1.2.4 – Тестовый класс TestInputPosition

**Тест TestCheckGameEnd:**

**Цель:** проверить, чтобы программа корректно определяла случаи выигрыша, ничьи в игре и не заканчивала игру по непредсказуемому сценарию.

1. mock\_playing

* Описание: вспомогательная функция для проверки результата игры в текущий момент времени, принимает в качестве параметра ожидаемый результат
* Ожидаемый результат: функция проверки конца игры должна выдать то же значение, что и в переданном параметре

1. test\_check\_without\_win

* Описание: группа тестов, когда игра ещё продолжается.
* Ожидаемый результат: во всех тестах функция проверки конца игры должна вернуть 0.

1. test\_check\_first\_win

* Описание: группа тестов, где первый игрок выиграл.
* Ожидаемый результат: во всех тестах функция проверки конца игры должна вернуть 1.

1. test\_check\_second\_win

* Описание: группа тестов, где второй игрок выиграл.
* Ожидаемый результат: во всех тестах функция проверки должна вернуть -1.

1. test\_check\_draw

* Описание: группа тестов, где игра завершилась ничьёй.
* Ожидаемый результат: во всех тестах функция проверки должна вернуть 2.

На Рисунках 1.2.5 - 1.2.6 представлен тестовый класс TestCheckGameEnd.

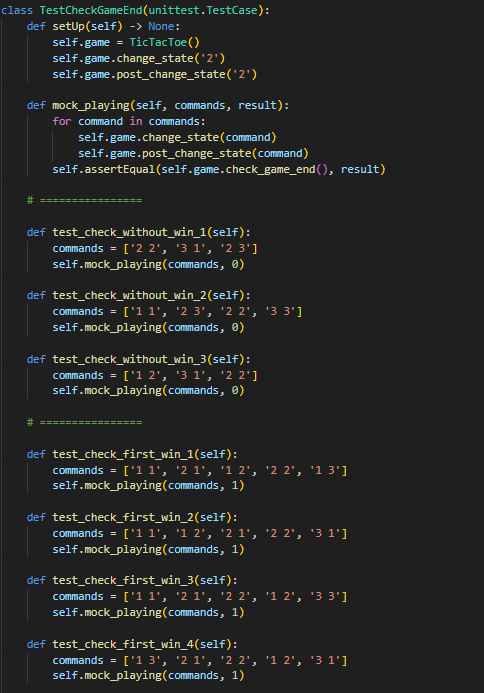


Рисунок 1.2.5 – Тестовый класс TestCheckGameEnd

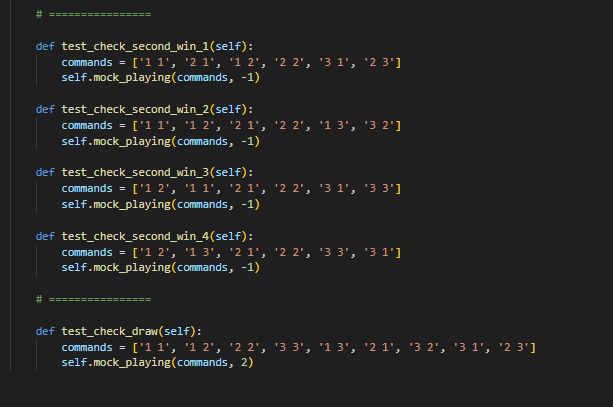


Рисунок 1.2.6 – Тестовый класс TestCheckGameEnd

## 1.3 Запуск тестирования

На данном этапе необходимо было запустить разработанные тесты. Так как класса и методов, описывающих работу программы «Крестики-нолики», еще не существует, все тесты провалились. На Рисунке 1.3.1 представлены результаты тестирования.

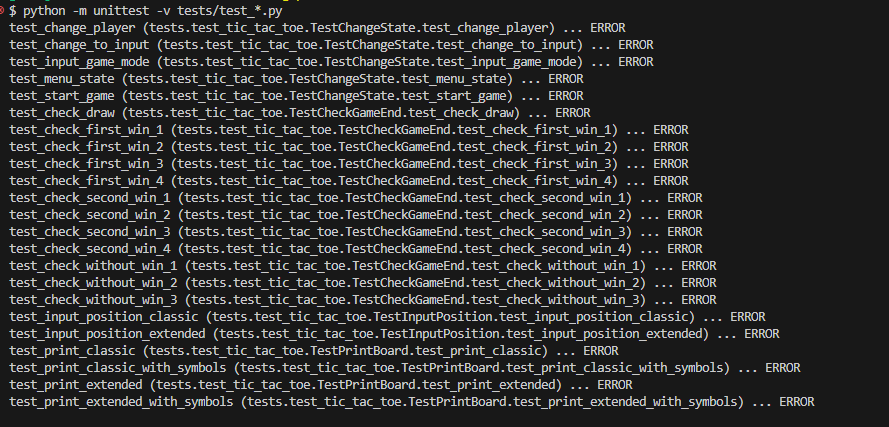


Рисунок 1.3.1 – Результаты тестирования

## 1.4 Реализация классов и методов

### 1.4.1 Класс TicTacToe

Является основным классом в программе, во время выполнения программы создаётся единственный экземпляр данного класса. Код этого класса представлен в Листинге А.2.

1. Метод \_\_init\_\_:

Создаваемые поля:

* state – текущее состояние игры
* previous\_state – предыдущее состояние игры
* mode – режим игры
* table – игровое поле
* transitions – список возможных переходов между состояниями

1. Метод change\_state(self, input\_command)

**Описание:** первично обрабатывает введённую пользователем команду и проверяет, можно ли в текущем состоянии такую команду обработать – если да, то совершает переход в другое состояние, если нет – то вызывает исключение.

**Параметры:** input\_command: str – введённая пользователем команда

**Возвращаемое значение:** отсутствует

1. Метод print\_board(self)

**Описание:** метод проходит по полю table текущего объекта, собирая строку игрового поля для вывода пользователям.

**Параметры:** отсутствуют

**Возвращаемое значение:** строка с игровым полем

1. Метод change\_mode(self, input\_command)

**Описание:** вызывается в случае изменения режима игры и обрабатывает переход, включая изменение размера игрового поля.

**Параметры:** input\_command: str – введённая пользователем команда

**Возвращаемое значение:** отсутствует.

1. Метод handle\_game\_input(self, position)

**Описание:** обрабатывает ввод от игроков во время игры. Если введённая позиция некорректна или введена позиция ячейки, которая уже занята, то функция вызывает исключение.

**Параметры:** position: str – введённая пользователем позиция

**Возвращаемое значение:** отсутствует.

1. Метод post\_change\_state(self, input\_command)

**Описание:** дальнейшая обработка перехода между состояниями.

**Параметры:** input\_command: str – введённая пользователем команда

**Возвращаемое значение:** отсутствует.

1. Метод check\_game\_end(self)

**Описание:** проверяет, нет ли выигрышной или ничейной ситуации в игре и возвращает код состояния игры.

**Параметры:** отсутствуют

**Возвращаемое значение:** 0 – если игра должна продолжаться, 1 – если выиграл первый игрок, -1 – если выиграл второй игрок, 2 – если игра закончилась ничьёй.

1. Метод create\_prompt(self)

**Описание:** создаёт строку, которая будет предлагать пользователю ввести нужную команду в зависимости от состояния

**Параметры:** отсутствуют.

**Возвращаемое значение:** строка, предлагающая ввести команду

1. Метод clear\_table(self)

**Описание:** очищает игровую таблицу после окончания игры.

**Параметры:** отсутствуют.

**Возвращаемое значение:** отсутствует.

1. Метод game\_cycle(self)

**Описание:** метод, запускающий игру в бесконечном цикле и прогоняющий все методы класса.

**Параметры:** отсутствуют

**Возвращаемое значение:** отсутствует.

### 1.4.2 Класс Transition

Предназначен для описания переходов в конечном автомате. Код этого класса представлен в Листинге А.2.

1. Метод \_\_init\_\_(self, current\_state, \_input, next\_state)

**Описание:** конструктор, создающий переход между состояниями в конечном автомате.

**Параметры:** current\_state: GameState – первоначальное состояние; \_input – либо строка, либо функция, по которой осуществляется переход в новое состояние; next\_state: GameState – новое состояние

**Возвращаемое значение:** отсутствует.

1. Метод \_\_call\_\_(self, \_input)

**Описание:** метод, возвращающий новое состояние, если параметр \_input подходит под текущий переход.

**Параметры:** \_input – строка перехода в новое состояние

**Возвращаемое значение:** GameState, если переход существует или None в случае отсутствия.

### 1.4.3 Класс GameState

Класс-перечисление, предназначенный для описания состояний конечного автомата. Состоит из пяти сущностей: MENU (состояние меню), INPUT\_MODE (состояние ввода режима игры), FIRST\_PLAYER (состояние хода первого игрока), SECOND\_PLAYER (состояние хода второго игрока) и EXIT (состояние завершения программы). Код представлен в листинг А.2.

### 1.4.4 Класс GameMode

Класс-перечисление, предназначенный для описания режимов игры. Состоит из двух сущнностей: CLASSIC (стандартный режим, 3\*3 клетки) и EXTENDED (расширенный, 5\*5 клеток). Код представлен в Листинге А.2.

## 1.5 Повторный запуск тестирования

Теперь необходимо повторно запустить тесты и убедиться, что все методы класса проходят их успешно.

На Рисунке 1.5.1 представлены результаты тестирования.

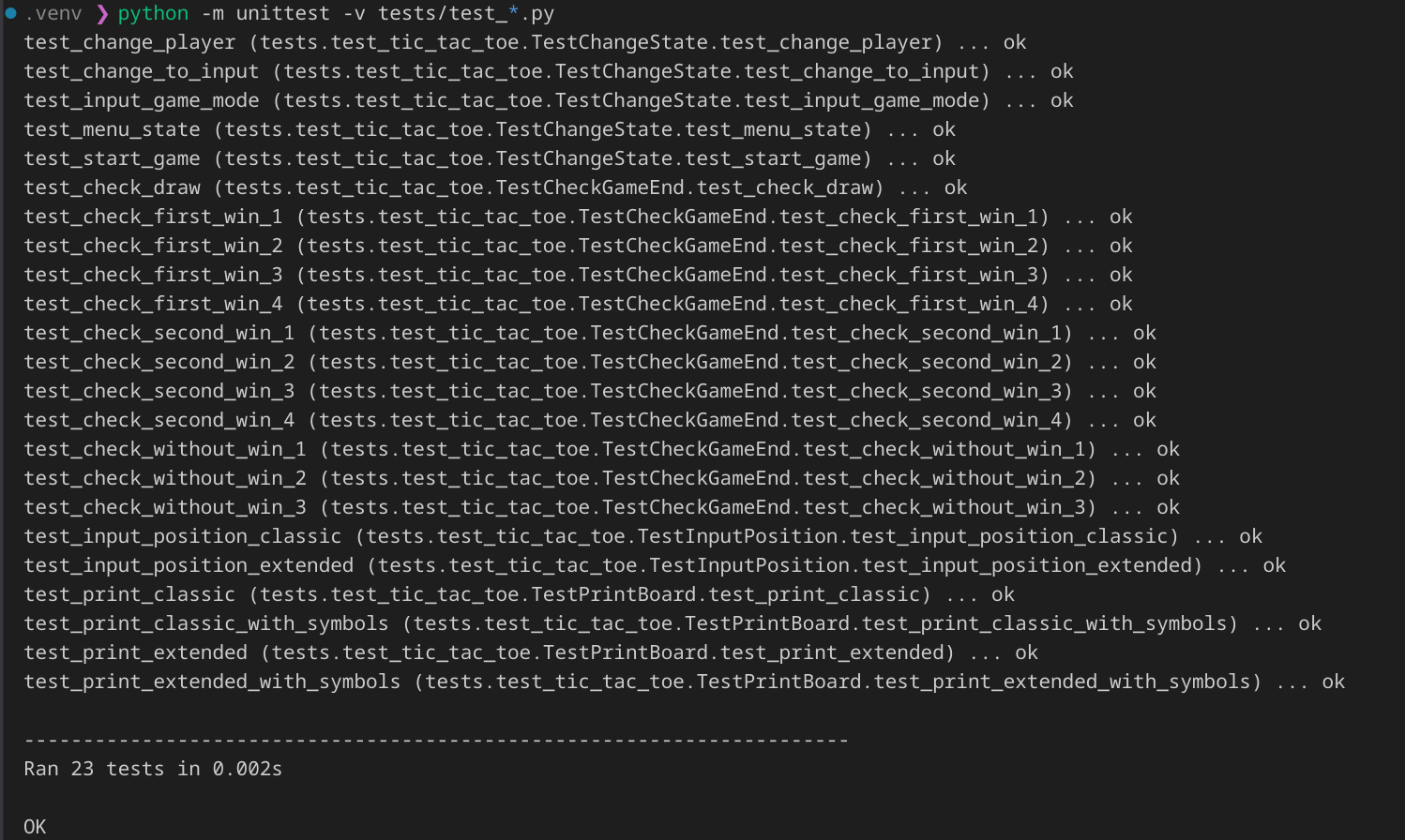


Рисунок 1.5.1 – Результаты успешного тестирования

# 2 Использование методологии BDD

## 2.1 Описание сценариев BDD

На первом этапе создаётся описание сценариев BDD для функциональности программы «Крестики-нолики». На Рисунках 2.1.1 - 2.1.2 приведены сценарии для игры. Был написан код на языке Gherkin по пути features/tic\_tac\_toe.feature в директории проекта.

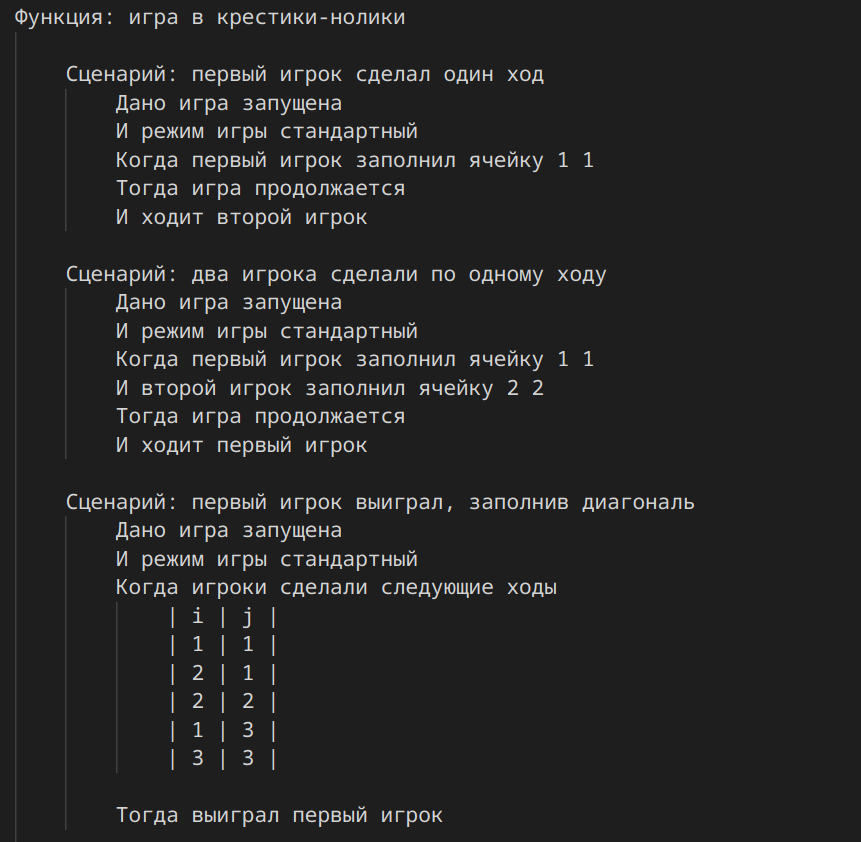


Рисунок 2.1.1 – Сценарии на языке Gherkin

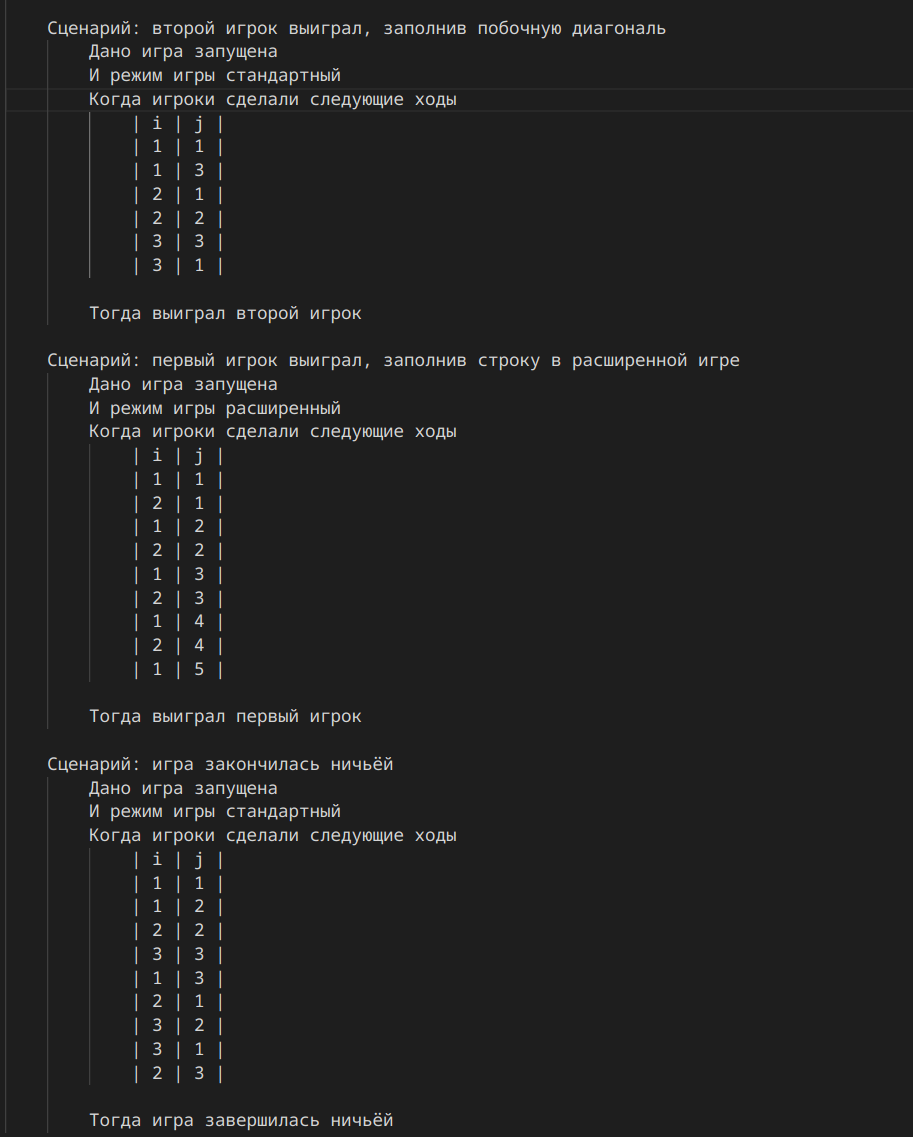


Рисунок 2.1.2 – Сценарии на языке Gherkin

Каждый сценарий описывает различные ситуации, которые могут возникнуть во время игры.

## 2.2. Автоматизация сценариев BDD

Далее по пути features/steps/tic\_tac\_toe\_step.py был реализован код для проведения автоматизированного тестирования с использованием библиотеки Behave для Python.

Аннотации @given, @when, и @then используются в библиотеке Behave для реализации BDD (Behavior-Driven Development) тестов. Эти аннотации привязывают шаги из сценариев, написанных на языке Gherkin, к Python-коду, который выполняет соответствующие действия.

**@given** — **Дано** (описывает начальное состояние): это шаг, который описывает начальные условия, контекст или окружение, в котором начинается тест.

**@when** — **Когда** (описывает действие, которое выполняет пользователь): этот шаг описывает действие пользователя или системы. Это может быть ввод данных, взаимодействие с интерфейсом или вызов метода.

**@then** — **Тогда** (описывает ожидаемый результат): этот шаг определяет ожидаемый результат после выполнения действия. Это могут быть проверки или утверждения, которые гарантируют, что система работает правильно.

На Рисунках 2.2.1 - 2.2.2 представлено тестирование для сценариев.



Рисунок 2.2.1 – Тесты для сценариев



Рисунок 2.2.2 – Тесты для сценариев

## 2.3 Запуск тестов

Тестирование сценариев запущено командой, представленной в Листинге 2.3.1. Важно при запуске указать в параметрах русский язык, чтобы модуль behave мог считать описания, написанные на русском.

Листинг 2.3.1 – Команда запуска behave

behave —lang=ru

В результате все сценарии успешно прошли тестирование. Результаты тестирования представлены на Рисунках 2.3.1 - 2.3.2.

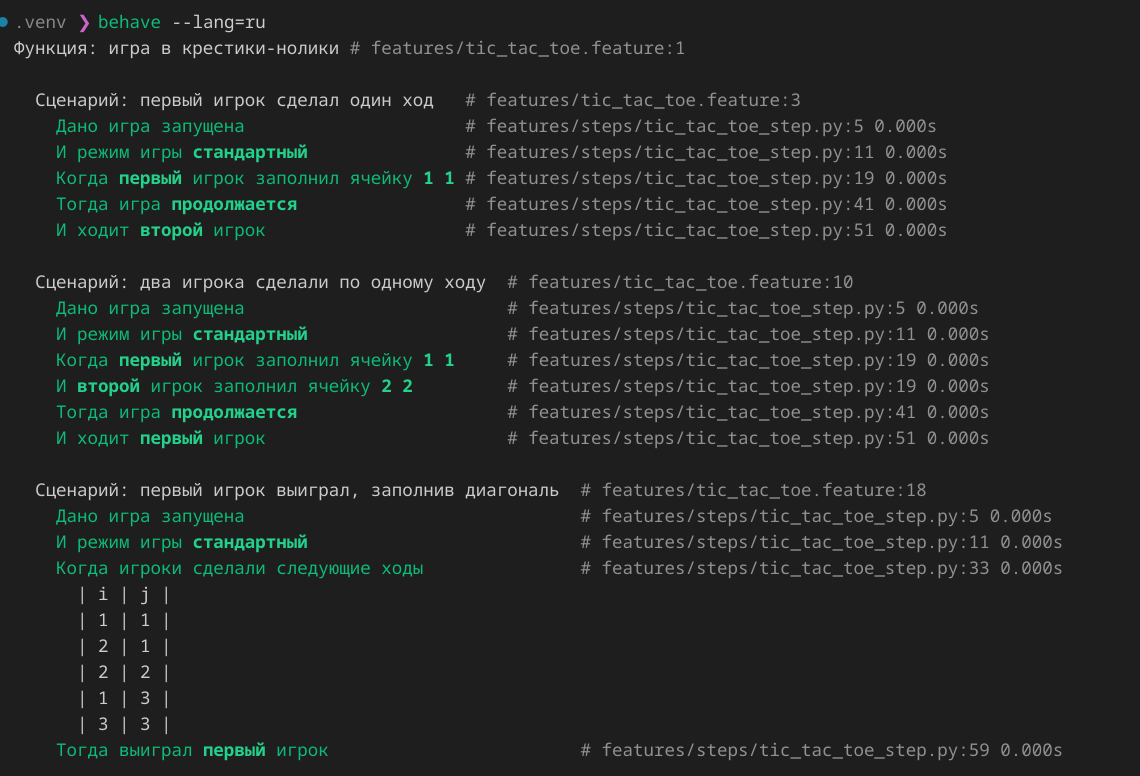


Рисунок 2.3.1 – Успешное прохождение тестов

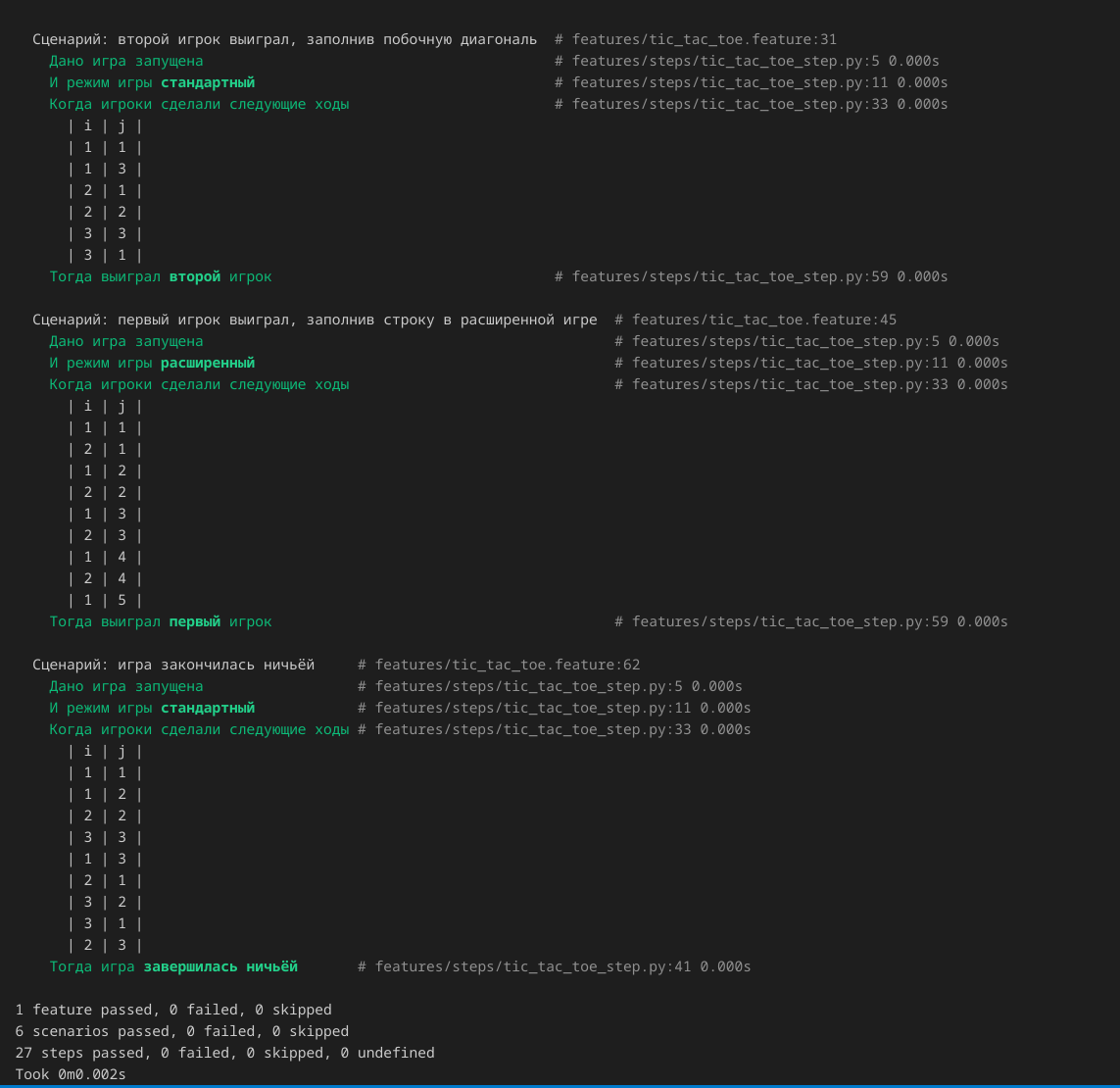


Рисунок 2.3.2 - Успешное прохождение тестов

# Заключение

В ходе выполнения практической работы были изучены методологии TDD (Test-Driven Development) и BDD (Behavior-Driven Development), которые подразумевают создание тестов для функционала ПО до создания самой функциональности. Так, были написаны тесты и сценарии согласно данным подходам, а затем реализован программный продукт, методы которого успешно проходят все написанные тесты.

# Приложение

Листинг А.1 – Код реализации юнит-тестов

import unittest

from tic\_tac\_toe import TicTacToe, GameState, PlaceTakenError

class TestChangeState(unittest.TestCase):

def setUp(self) -> None:

self.game = TicTacToe()

def test\_menu\_state(self) -> None:

self.assertEqual(self.game.state, GameState.MENU)

def test\_change\_to\_input(self) -> None:

self.game.change\_state('1')

self.assertEqual(self.game.state, GameState.INPUT\_MODE)

def test\_input\_game\_mode(self) -> None:

possible\_inputs = [('1', GameState.MENU), ('2', GameState.MENU),

('0', ValueError), ('asjdgs', ValueError)]

for \_input, result in possible\_inputs:

self.game.state = GameState.INPUT\_MODE

if result is ValueError:

with self.assertRaises(ValueError):

self.game.change\_state(\_input)

else:

self.game.change\_state(\_input)

self.assertEqual(self.game.state, result)

def test\_start\_game(self) -> None:

self.game.change\_state('2')

self.assertEqual(self.game.state, GameState.FIRST\_PLAYER)

def test\_change\_player(self):

self.game.change\_state('2')

self.game.change\_state('1 1')

self.assertEqual(self.game.state, GameState.SECOND\_PLAYER)

self.game.change\_state('4 3')

self.assertEqual(self.game.state, GameState.FIRST\_PLAYER)

with self.assertRaises(ValueError):

self.game.change\_state('6 6')

class TestPrintBoard(unittest.TestCase):

def setUp(self) -> None:

self.game = TicTacToe()

def test\_print\_classic(self):

self.assertEqual(

self.game.print\_board(), " || || \n"

"=============\n"

" || || \n"

"=============\n"

" || || \n")

def test\_print\_extended(self):

Продолжение Листинга А.1

self.game.change\_mode('2')

self.assertEqual(

self.game.print\_board(), " || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n")

def test\_print\_classic\_with\_symbols(self):

self.game.table[0][1] = 1

self.assertEqual(

self.game.print\_board(), " || x || \n"

"=============\n"

" || || \n"

"=============\n"

" || || \n")

self.game.table[1][2] = -1

self.assertEqual(

self.game.print\_board(), " || x || \n"

"=============\n"

" || || o \n"

"=============\n"

" || || \n")

self.game.table[2][1] = 1

self.assertEqual(

self.game.print\_board(), " || x || \n"

"=============\n"

" || || o \n"

"=============\n"

" || x || \n")

def test\_print\_extended\_with\_symbols(self):

self.game.change\_mode('2')

self.game.table[0][0] = -1

self.assertEqual(

self.game.print\_board(), " o || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n")

self.game.table[0][4] = 1

self.assertEqual(

self.game.print\_board(), " o || || || || x \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n")

self.game.table[2][3] = 1

self.assertEqual(

Продолжение Листинга А.1

self.game.print\_board(), " o || || || || x \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || x || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || || \n")

self.game.table[4][1] = -1

self.assertEqual(

self.game.print\_board(), " o || || || || x \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || || || x || \n"

"=======================\n"

" || || || || \n"

"=======================\n"

" || o || || || \n")

class TestInputPosition(unittest.TestCase):

def setUp(self) -> None:

self.game = TicTacToe()

def test\_input\_position\_classic(self):

positions = [('1 1', 0, 0), ('2 3', 1, 2), ('3 1', 2, 0),

('1 4', ValueError), ('1 1', PlaceTakenError),

('2 2', 1, 1)]

table = [[0 for \_ in range(3)] for \_ in range(3)]

self.game.change\_state('2')

k = 0

for index, (position, \*result) in enumerate(positions):

self.game.change\_state(position)

if result[0] is ValueError or result[0] is PlaceTakenError:

k += 1

with self.assertRaises(result[0]):

self.game.handle\_game\_input(position)

else:

i, j = result

table[i][j] = (-1)\*\*(index - k)

self.game.handle\_game\_input(position)

self.assertEqual(self.game.table, table)

def test\_input\_position\_extended(self):

positions = [('1 1', 0, 0), ('2 3', 1, 2), ('3 1', 2, 0),

('1 4', 0, 3), ('1 1', PlaceTakenError), ('2 2', 1, 1)]

table = [[0 for \_ in range(5)] for \_ in range(5)]

self.game.change\_state('2')

self.game.change\_mode('2')

k = 0

for index, (position, \*result) in enumerate(positions):

self.game.change\_state(position)

if result[0] is ValueError or result[0] is PlaceTakenError:

k += 1

with self.assertRaises(result[0]):

self.game.handle\_game\_input(position)

else:

Продолжение Листинга А.1

i, j = result

table[i][j] = (-1)\*\*(index - k)

self.game.handle\_game\_input(position)

self.assertEqual(self.game.table, table)

class TestCheckGameEnd(unittest.TestCase):

def setUp(self) -> None:

self.game = TicTacToe()

self.game.change\_state('2')

self.game.post\_change\_state('2')

def mock\_playing(self, commands, result):

for command in commands:

self.game.change\_state(command)

self.game.post\_change\_state(command)

self.assertEqual(self.game.check\_game\_end(), result)

# ================

def test\_check\_without\_win\_1(self):

commands = ['2 2', '3 1', '2 3']

self.mock\_playing(commands, 0)

def test\_check\_without\_win\_2(self):

commands = ['1 1', '2 3', '2 2', '3 3']

self.mock\_playing(commands, 0)

def test\_check\_without\_win\_3(self):

commands = ['1 2', '3 1', '2 2']

self.mock\_playing(commands, 0)

# ================

def test\_check\_first\_win\_1(self):

commands = ['1 1', '2 1', '1 2', '2 2', '1 3']

self.mock\_playing(commands, 1)

def test\_check\_first\_win\_2(self):

commands = ['1 1', '1 2', '2 1', '2 2', '3 1']

self.mock\_playing(commands, 1)

def test\_check\_first\_win\_3(self):

commands = ['1 1', '2 1', '2 2', '1 2', '3 3']

self.mock\_playing(commands, 1)

def test\_check\_first\_win\_4(self):

commands = ['1 3', '2 1', '2 2', '1 2', '3 1']

self.mock\_playing(commands, 1)

# ================

def test\_check\_second\_win\_1(self):

commands = ['1 1', '2 1', '1 2', '2 2', '3 1', '2 3']

self.mock\_playing(commands, -1)

def test\_check\_second\_win\_2(self):

commands = ['1 1', '1 2', '2 1', '2 2', '1 3', '3 2']

self.mock\_playing(commands, -1)

Окончание Листинга А.1

def test\_check\_second\_win\_3(self):

commands = ['1 2', '1 1', '2 1', '2 2', '3 1', '3 3']

self.mock\_playing(commands, -1)

def test\_check\_second\_win\_4(self):

commands = ['1 2', '1 3', '2 1', '2 2', '3 3', '3 1']

self.mock\_playing(commands, -1)

# ================

def test\_check\_draw(self):

commands = [

'1 1', '1 2', '2 2', '3 3', '1 3', '2 1', '3 2', '3 1', '2 3'

]

self.mock\_playing(commands, 2)

Листинг А.2 – Код реализации программы «Крестики-нолики»

from enum import Enum

from collections.abc import Callable

import re

class PlaceTakenError(Exception):

pass

class GameState(Enum):

MENU = 1

INPUT\_MODE = 2

FIRST\_PLAYER = 3

SECOND\_PLAYER = 4

EXIT = 5

class GameMode(Enum):

CLASSIC = 1

EXTENDED = 2

class Transition:

def \_\_init\_\_(self, current\_state: GameState,

\_input: str | Callable[[str], bool], next\_state: GameState):

self.current\_state = current\_state

self.input = \_input

self.next\_state = next\_state

def \_\_call\_\_(self, \_input: str) -> GameState | None:

if type(self.input) is str and self.input == \_input:

return self.next\_state

elif callable(self.input) and self.input(\_input):

return self.next\_state

return None

class TicTacToe:

def \_\_init\_\_(self):

Продолжение Листинга А.2

self.previous\_state: GameState | None = None

self.state = GameState.MENU

self.mode = GameMode.CLASSIC

self.table = [[0 for \_ in range(3)] for \_ in range(3)]

self.transitions = [

Transition(GameState.MENU, '1', GameState.INPUT\_MODE),

Transition(GameState.MENU, '2', GameState.FIRST\_PLAYER),

Transition(GameState.INPUT\_MODE, lambda x: re.match(r'[1-2]', x),

GameState.MENU),

Transition(GameState.FIRST\_PLAYER,

lambda x: re.match(r'[1-5] [1-5]', x),

GameState.SECOND\_PLAYER),

Transition(GameState.SECOND\_PLAYER,

lambda x: re.match(r'[1-5] [1-5]', x),

GameState.FIRST\_PLAYER),

Transition(GameState.MENU, '3', GameState.EXIT)

]

def change\_state(self, input\_command: str):

next\_state = None

for transition in self.transitions:

if transition.current\_state == self.state:

next\_state = transition(input\_command)

if next\_state is not None:

self.previous\_state = self.state

self.state = next\_state

break

if next\_state is None:

raise ValueError('Введена неправильная команда')

def print\_board(self):

if self.mode == GameMode.CLASSIC:

n = 3

elif self.mode == GameMode.EXTENDED:

n = 5

board\_lines = []

for i in range(n):

board\_line = []

for j in range(n):

elem = self.table[i][j]

elem = "x" if elem == 1 else "o" if elem == -1 else " "

cell = f" {elem} "

board\_line.append(cell)

board\_line = "||".join(board\_line)

board\_line += "\n"

board\_lines.append(board\_line)

border = "=" \* (3 \* n + 2 \* (n - 1)) + "\n"

board = border.join(board\_lines)

return board

def change\_mode(self, input\_command: str):

if input\_command == '1':

self.mode = GameMode.CLASSIC

n = 3

elif input\_command == '2':

self.mode = GameMode.EXTENDED

n = 5

Продолжение Листинга А.2

self.table = [[0 for \_ in range(n)] for \_ in range(n)]

def handle\_game\_input(self, position: str):

if self.mode == GameMode.CLASSIC:

n = 3

elif self.mode == GameMode.EXTENDED:

n = 5

if self.previous\_state == GameState.FIRST\_PLAYER:

elem = 1

elif self.previous\_state == GameState.SECOND\_PLAYER:

elem = -1

i, j = map(lambda x: int(x) - 1, position.split())

if i in range(n) and j in range(n) and self.table[i][j] == 0:

self.table[i][j] = elem

else:

self.state = self.previous\_state

if i not in range(n) or j not in range(n):

raise ValueError('Введены некорректные координаты')

elif self.table[i][j] != 0:

raise PlaceTakenError('Место уже занято')

def post\_change\_state(self, input\_command: str):

match self.previous\_state:

case GameState.INPUT\_MODE:

self.change\_mode(input\_command)

case GameState.FIRST\_PLAYER:

self.handle\_game\_input(input\_command)

case GameState.SECOND\_PLAYER:

self.handle\_game\_input(input\_command)

case GameState.MENU:

pass

def check\_game\_end(self):

if self.mode == GameMode.CLASSIC:

n = 3

elif self.mode == GameMode.EXTENDED:

n = 5

if self.previous\_state == GameState.FIRST\_PLAYER:

elem = 1

elif self.previous\_state == GameState.SECOND\_PLAYER:

elem = -1

row\_win, col\_win, diag1\_win, diag2\_win = 0, 0, 0, 0

if any([all(cell == 1 for cell in row) for row in self.table]) or \

any([all(cell == -1 for cell in row) for row in self.table]):

row\_win = elem

if any(all(cell == 1 for cell in col) for col in zip(\*self.table)) or \

any(all(cell == -1 for cell in col) for col in zip(\*self.table)):

col\_win = elem

if all(self.table[i][i] == 1 for i in range(n)) or \

all(self.table[i][i] == -1 for i in range(n)):

diag1\_win = elem

if all(self.table[i][n - i - 1] == 1 for i in range(n)) or \

all(self.table[i][n - i - 1] == -1 for i in range(n)):

diag2\_win = elem

Продолжение Листинга А.2

if any(elem == 1 for elem in [row\_win, col\_win, diag1\_win, diag2\_win]):

result = 1

elif any(elem == -1

for elem in [row\_win, col\_win, diag1\_win, diag2\_win]):

result = -1

elif all(self.table[i][j] != 0 for i in range(n) for j in range(n)):

result = 2

else:

result = 0

if result != 0:

self.state = GameState.MENU

self.clear\_table()

return result

def create\_prompt(self):

match self.state:

case GameState.MENU:

return ('1) Изменить режим игры\n2) Запустить игру\n'

'3) Выйти из игры\n')

case GameState.INPUT\_MODE:

return ('Введите режим игры:\n'

'1) Классический (3 x 3)\n'

'2) Расширенный (5 x 5)\n')

case GameState.FIRST\_PLAYER:

return 'Первый игрок, введите позицию: '

case GameState.SECOND\_PLAYER:

return 'Второй игрок, введите позицию: '

def clear\_table(self):

if self.mode == GameMode.CLASSIC:

n = 3

elif self.mode == GameMode.EXTENDED:

n = 5

self.table = [[0 for \_ in range(n)] for \_ in range(n)]

def game\_cycle(self):

error = False

while True:

if self.state in [GameState.FIRST\_PLAYER,

GameState.SECOND\_PLAYER] and \

not error:

print(self.print\_board())

input\_command = input(self.create\_prompt())

try:

self.change\_state(input\_command)

error = False

except ValueError as e:

print(e)

error = True

continue

if self.state == GameState.EXIT:

break

try:

self.post\_change\_state(input\_command)

error = False

Окончание Листинга А.2

except (ValueError, PlaceTakenError) as e:

print(e)

error = True

continue

if self.state in [GameState.FIRST\_PLAYER, GameState.SECOND\_PLAYER]:

result = self.check\_game\_end()

if result != 0:

print(self.print\_board())

if result == 1:

print('Победил первый игрок!')

elif result == -1:

print('Победил второй игрок!')

elif result == 2:

print('Победила дружба)')

def main():

game = TicTacToe()

game.game\_cycle()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Листинг А.3 – Сценарии на языке Gherkin

Функция: игра в крестики-нолики

Сценарий: первый игрок сделал один ход

Дано игра запущена

И режим игры стандартный

Когда первый игрок заполнил ячейку 1 1

Тогда игра продолжается

И ходит второй игрок

Сценарий: два игрока сделали по одному ходу

Дано игра запущена

И режим игры стандартный

Когда первый игрок заполнил ячейку 1 1

И второй игрок заполнил ячейку 2 2

Тогда игра продолжается

И ходит первый игрок

Сценарий: первый игрок выиграл, заполнив диагональ

Дано игра запущена

И режим игры стандартный

Когда игроки сделали следующие ходы

| i | j |

| 1 | 1 |

| 2 | 1 |

| 2 | 2 |

| 1 | 3 |

| 3 | 3 |

Тогда выиграл первый игрок

Сценарий: второй игрок выиграл, заполнив побочную диагональ

Дано игра запущена

И режим игры стандартный

Когда игроки сделали следующие ходы

Окончание Листинга А.3

| i | j |

| 1 | 1 |

| 1 | 3 |

| 2 | 1 |

| 2 | 2 |

| 3 | 3 |

| 3 | 1 |

Тогда выиграл второй игрок

Сценарий: первый игрок выиграл, заполнив строку в расширенной игре

Дано игра запущена

И режим игры расширенный

Когда игроки сделали следующие ходы

| i | j |

| 1 | 1 |

| 2 | 1 |

| 1 | 2 |

| 2 | 2 |

| 1 | 3 |

| 2 | 3 |

| 1 | 4 |

| 2 | 4 |

| 1 | 5 |

Тогда выиграл первый игрок

Сценарий: игра закончилась ничьёй

Дано игра запущена

И режим игры стандартный

Когда игроки сделали следующие ходы

| i | j |

| 1 | 1 |

| 1 | 2 |

| 2 | 2 |

| 3 | 3 |

| 1 | 3 |

| 2 | 1 |

| 3 | 2 |

| 3 | 1 |

| 2 | 3 |

Тогда игра завершилась ничьёй

Листинг А.4 – Код реализации тестирования сценариев

from behave import given, then, when

from tic\_tac\_toe import TicTacToe, GameState, PlaceTakenError

@given('игра запущена')

def step\_game\_launched(context):

context.game = TicTacToe()

context.game.change\_state('2')

@given('режим игры {mode}')

def step\_game\_mode(context, mode):

if mode == 'стандартный':

context.game.change\_mode('1')

elif mode == 'расширенный':

context.game.change\_mode('2')

Окончание Листинга А.4

@when('{player} игрок заполнил ячейку {i} {j}')

def step\_cell\_filled(context, player, i, j):

position = f'{i} {j}'

context.is\_failed = False

try:

context.game.change\_state(position)

except ValueError:

context.is\_failed = True

try:

context.game.post\_change\_state(position)

except (ValueError, PlaceTakenError):

context.is\_failed = True

@when('игроки сделали следующие ходы')

def step\_many\_turns(context):

for row in context.table:

position = f"{row['i']} {row['j']}"

context.game.change\_state(position)

context.game.post\_change\_state(position)

@then('игра {status}')

def step\_game\_status(context, status):

if status == 'продолжается':

assert context.game.state in [

GameState.FIRST\_PLAYER, GameState.SECOND\_PLAYER

]

elif status == 'завершилась ничьёй':

assert context.game.check\_game\_end() == 2

@then('ходит {player} игрок')

def step\_current\_player(context, player):

if player == 'первый':

assert context.game.state == GameState.FIRST\_PLAYER

elif player == 'второй':

assert context.game.state == GameState.SECOND\_PLAYER

@then('выиграл {player} игрок')

def step\_winner(context, player):

if player == 'первый':

assert context.game.check\_game\_end() == 1

elif player == 'второй':

assert context.game.check\_game\_end() == -1