|  |
| --- |
|  |
| 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

Отчет по практической работе

по дисциплине «Тестирование и верификация ПО»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы **ИКБО-11-22** | Гришин А.В. |
| **Проверил:** | Мельников Д.А. |

2024 г.

**Оглавление**

[МЕТОДОЛОГИЯ TDD 3](#_Toc180434213)

[**ЭТАП №1. СОЗДАНИЕ ТЕСТА.** 3](#_Toc180434214)

[**ЭТАП №2. ЗАПУСК ТЕСТОВ.** 4](#_Toc180434215)

[**ЭТАП №3. РЕАЛИЗАЦИЯ ФУНКЦИИ.** 4](#_Toc180434216)

[**ЭТАП №4. ПОВТОРНЫЙ ЗАПУСК ТЕСТОВ.** 6](#_Toc180434217)

[МЕТОДОЛОГИЯ BDD 6](#_Toc180434218)

[**ЭТАП №1. ОПИСАНИЕ СЦЕНАРИЕВ BDD.** 6](#_Toc180434219)

[**ЭТАП №2. АВТОМАТИЗАЦИЯ СЦЕНАРИЕВ BDD.** 7](#_Toc180434220)

[**ЭТАП №3. ЗАПУСК ТЕСТОВ.** 8](#_Toc180434221)

[Заключение: 8](#_Toc180434222)

Тема работы согласно варианту 03 – Игра «Крестики-нолики»

# МЕТОДОЛОГИЯ TDD

TDD, или Test-Driven Development (Разработка, управляемая тестами), - это методология разработки программного обеспечения, которая подразумевает создание тестов для функциональности ПО до того, как эта функциональность будет фактически реализована. TDD представляет собой циклический процесс, который помогает разработчикам создавать высококачественное, надежное ПО.

## **ЭТАП №1. СОЗДАНИЕ ТЕСТА.**

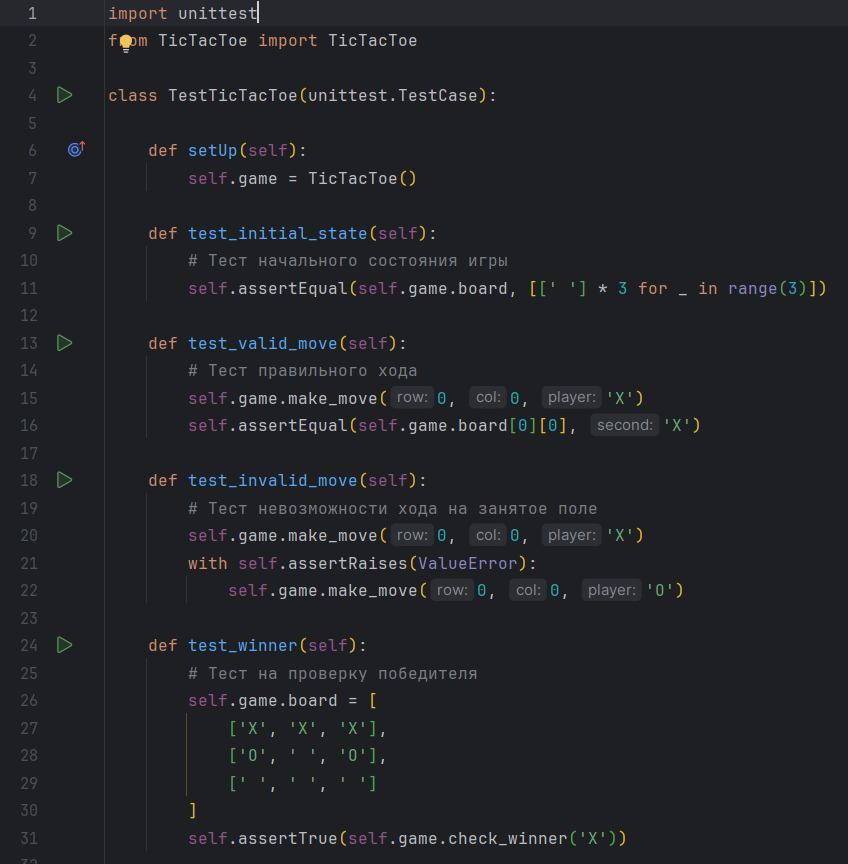


Рисунок 1 – Примеры тестов методологии TDD

*Таблица 1 – Листинг тестов методологии TDD*

|  |
| --- |
| import unittest from TicTacToe import TicTacToe  class TestTicTacToe(unittest.TestCase):   def setUp(self):  self.game = TicTacToe()   def test\_initial\_state(self):  # Тест начального состояния игры  self.assertEqual(self.game.board, [[' '] \* 3 for \_ in range(3)])   def test\_valid\_move(self):  # Тест правильного хода  self.game.make\_move(0, 0, 'X')  self.assertEqual(self.game.board[0][0], 'X')   def test\_invalid\_move(self):  # Тест невозможности хода на занятое поле  self.game.make\_move(0, 0, 'X')  with self.assertRaises(ValueError):  self.game.make\_move(0, 0, 'O')   def test\_winner(self):  # Тест на проверку победителя  self.game.board = [  ['X', 'X', 'X'],  ['O', ' ', 'O'],  [' ', ' ', ' ']  ]  self.assertTrue(self.game.check\_winner('X'))   def test\_draw(self):  # Тест на ничью  self.game.board = [  ['X', 'O', 'X'],  ['X', 'X', 'O'],  ['O', 'X', 'O']  ]  self.assertTrue(self.game.is\_draw()) |

## **ЭТАП №2. ЗАПУСК ТЕСТОВ.**

На этом этапе запускаются тесты. Так как функции еще не существуют, все они провалятся.

## **ЭТАП №3. РЕАЛИЗАЦИЯ ФУНКЦИИ.**

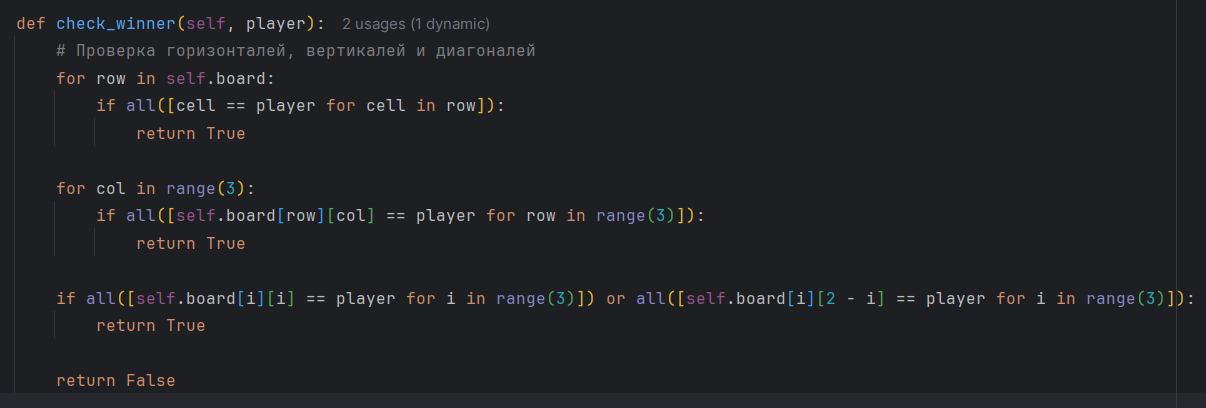


Рисунок 2 – Пример функции

*Таблица 2 – Листинг программы “TicTacToe”*

|  |
| --- |
| class TicTacToe:  def \_\_init\_\_(self):  self.board = [[' '] \* 3 for \_ in range(3)]  self.current\_player = 'X'   def display\_board(self):  for row in self.board:  print('|'.join(row))  print('-' \* 5)   def make\_move(self, row, col, player):  if self.board[row][col] != ' ':  raise ValueError("Cell is already taken")  self.board[row][col] = player   def check\_winner(self, player):  # Проверка горизонталей, вертикалей и диагоналей  for row in self.board:  if all([cell == player for cell in row]):  return True   for col in range(3):  if all([self.board[row][col] == player for row in range(3)]):  return True   if all([self.board[i][i] == player for i in range(3)]) or all([self.board[i][2 - i] == player for i in range(3)]):  return True   return False   def is\_draw(self):  return all(cell != ' ' for row in self.board for cell in row)   def switch\_player(self):  self.current\_player = 'O' if self.current\_player == 'X' else 'X'   def main():  game = TicTacToe()  print("Welcome to Tic-Tac-Toe!")  game.display\_board()   while True:  try:  row = int(input(f"Player {game.current\_player}, enter the row (0, 1, or 2): "))  col = int(input(f"Player {game.current\_player}, enter the column (0, 1, or 2): "))   if row not in range(3) or col not in range(3):  print("Invalid input. Please enter numbers between 0 and 2.")  continue   game.make\_move(row, col, game.current\_player)  game.display\_board()   if game.check\_winner(game.current\_player):  print(f"Player {game.current\_player} wins!")  break   if game.is\_draw():  print("It's a draw!")  break   game.switch\_player()  except ValueError as e:  print(e)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## **ЭТАП №4. ПОВТОРНЫЙ ЗАПУСК ТЕСТОВ.**

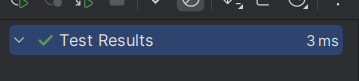
****

Рисунок 3 – Успешное выполнение тестов



Рисунок 4 – Общее количество тестов

# МЕТОДОЛОГИЯ BDD

**BDD, или Behavior Driven Development** (Разработка, ориентированная на поведение), - это методология разработки программного обеспечения, которая сосредотачивается на описании поведения программы с точки зрения её пользователей и интересующих сторон. BDD представляет собой эволюцию техники TDD (Test-Driven Development), в которой акцент делается на спецификациях поведения и участии бизнес-аналитиков и представителей заказчика в процессе разработки.

## **ЭТАП №1. ОПИСАНИЕ СЦЕНАРИЕВ BDD.**

На первом этапе создаётся описание сценариев BDD для функциональности игры.

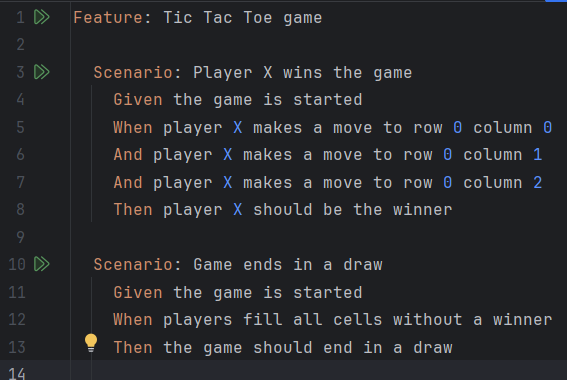


Рисунок 5 – Описание сценариев

## **ЭТАП №2. АВТОМАТИЗАЦИЯ СЦЕНАРИЕВ BDD.**

На следующем шаге создаются автоматизированные тесты для каждого сценария, используя фреймворк для тестирования, который поддерживает BDD, в нашем случае это Behave для Python.

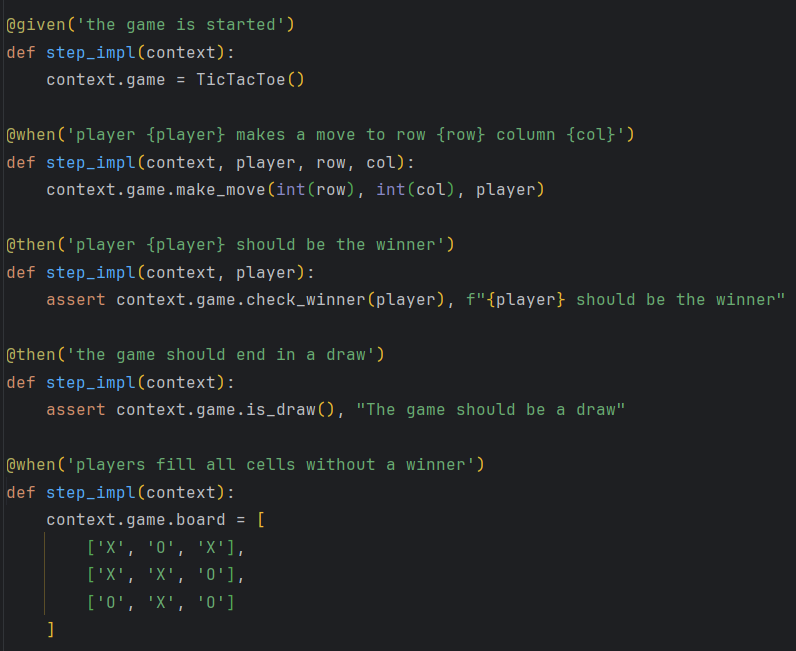


Рисунок 6 – Описание тестов

*Таблица 3 – Листинг тестов методологии BDD*

|  |
| --- |
| from behave import given, when, then from TicTacToe import TicTacToe  @given('the game is started') def step\_impl(context):  context.game = TicTacToe()  @when('player {player} makes a move to row {row} column {col}') def step\_impl(context, player, row, col):  context.game.make\_move(int(row), int(col), player)  @then('player {player} should be the winner') def step\_impl(context, player):  assert context.game.check\_winner(player), f"{player} should be the winner"  @then('the game should end in a draw') def step\_impl(context):  assert context.game.is\_draw(), "The game should be a draw"  @when('players fill all cells without a winner') def step\_impl(context):  context.game.board = [  ['X', 'O', 'X'],  ['X', 'X', 'O'],  ['O', 'X', 'O']  ] |

## **ЭТАП №3. ЗАПУСК ТЕСТОВ.**



Рисунок 7 – Успешное выполнение тестов



Рисунок 8 – Общее количество тестов

# Заключение:

В ходе работы были получены практический опыт применения методологий TDD (Test-Driven Development) и BDD (Behavior-Driven Development) в процессе разработки программного обеспечения.