|  |
| --- |
| Изображение выглядит как зарисовка, рисунок, графическая вставка, символ  Автоматически созданное описание |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** |
| **по дисциплине** |
| **«Тестирование и верификация программного обеспечения»** |

|  |  |
| --- | --- |
| Выполнили студенты группы ИКБО-04-22 | Егоров Л.А. |
| Принял ассистент | Петрова А.А. |

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

Москва 2024

# 1 РАЗРАБОТКА ПЛАНА ТЕСТИРОВАНИЯ

План тестирования был разработан для игры «Крестики-нолики», согласно варианту №07 Практической работы №3.

## Идентификатор тестового плана

Идентификатор тестового плана: TП-ИКН-1.0

Расшифровка шаблона идентификатора:

* ТП - префикс, указывающий на тестовый план;
* ИКН - название тестируемого приложения – «Игра Крестики-Нолики»;
* 1.0 - версия тестового плана, совпадающая с версией продукта.

Уровень плана: данный тестовый план относится к уровню модульного тестирования, так как фокусируется на функциональности отдельных модулей приложения.

Автор тестового плана: Егоров Леонид

Контактная информация: egor.leonidov@yandex.ru

История изменений:

* 1.0: Создан начальный тест-план на версию продукта 1.0, который

включает функциональные и модульные тесты для компонентов

приложения.

## 1.2 Ссылки на используемые документы

* План проекта;
* Спецификация требований к ПО;
* Пользовательское руководство и описание игры "Крестики-нолики";
* Стандарт тестирования: IEEE Std 829-2019 - Standard for Software and System Test Documentation.

## 1.3 Введение

Целью данного плана является определение подхода к тестированию консольного игры "Крестики-нолики". План охватывает тестирование функциональности приложения, в том числе проверку основных сценариев. План описывает процесс тестирования, который будет следовать установленным стандартам для проверки качества и функциональности приложения.

## 1.4 Тестируемые элементы

Функции игры "Крестики-нолики", которые подлежат тестированию:

* функция **change\_state** изменяет состояние игры в зависимости от пользовательского ввода и текущего состояния игры;
* функция **print\_board** выводит в консоль текущее игровое поле с расставленными крестиками и ноликами;
* функция **handle\_game\_input** заполняет нужную ячейку на игровом поле;
* функция **check\_game\_end** проверяет, не завершилась ли игра после выполнения хода.

## 1.5 Проблемы риска тестирования ПП

К ключевым проблемам риска относятся:

* сложность обеспечения полной конфигурации тестовой среды: необходимо проверять, чтобы на системах, запускающих тесты, был установлен интерпретатор языка Python версии не ниже 3.0.0;
* недостаточная документация или сложные модули: плохо описанные компоненты или модификации сложных модулей могут вызывать непредвиденные ошибки, что повышает риск появления новых дефектов при дальнейших изменениях;
* сложности при масштабировании: могут возникнуть трудности при расширении игрового поля до больших размеров, чем 5 на 5.

## 1.6 Особенности или свойства, подлежащие тестированию

Цель тестирования – проверить, что приложение отвечает ожиданиям пользователей и соответствует требованиям, заложенным в бизнес-логику.

1. Интерфейс приложения

* Описание: проверка удобства и интуитивности интерфейса, корректности работы элементов управления, логичности навигации по разделам.

1. Функциональные возможности

* Описание: проверка базовых функций приложения, таких как отображение игрового поля, определение окончания игры и победителя.

1. Обработка ошибок пользователя и системы

* Описание: проверка корректной работы при вводе пользователем некорректных данных, устойчивость приложения к возможным сбоям и непредвиденным ситуациям.

1. Стабильность при обновлениях и изменениях

* Описание: проверка корректности работы всех функций после обновлений или внесения изменений в код.

## 1.7 Особенности (свойства), не подлежащие тестированию

* корректность отображаемых предложений для пользовательского ввода, поскольку это не влияет на функциональную составляющую игры;
* тестирование производительности не проводится из-за того, что программа рассчитана на запуск на локальной машине и поддерживает только режим игры с двумя игроками;

## 1.8 Подход

### 1.8.1 Общая стратегия тестирования

Тестирование игры "Крестики-нолики" будет проводиться в несколько этапов:

* **Юнит-тестирование** для проверки работы отдельных функций.
* **Функциональное тестирование** с целью проверки работы ключевых функций с точки зрения пользователя.
* **Ручное тестирование пользовательского интерфейса** для проверки удобства ввода, правильности отображения игрового поля и определения победителя в игре.

### 1.8.2 Инструменты

Для юнит-тестирования могут быть использованы стандартные библиотеки тестирования, такие как unittest (Python), а также behave для выполнения BDD-тестов. Специальное обучение для использования инструментов тестирования не требуется.

### 1.8.3 Сбор метрик

Метрики будут включать:

* количество пройденных тест-кейсов;
* количество выявленных дефектов на каждом уровне тестирования;
* серьезность и приоритет дефектов;
* уровень покрытия кода (если применимо).

### 1.8.4 Управление конфигурациями

Специальное тестирование на различных конфигурациях не требуется.

### 1.8.5 Особые требования

Приложение не требует тестирования производительности или надежности.

## 1.9 Критерии смоук-тестирования

Смоук-тестирование будет считаться успешным, если:

* приложение запускается и отображает приветственное сообщение;
* пользователь может ввести позицию на поле, и соответствующая фигура отобразится на поле в введённой позиции;
* приложение корректно выводит поле в зависимости от заданных размеров (3\*3 или 5\*5).

## 1.10 Критерии прохождения тестов

Тестирование будет считаться успешно завершенным, если выполнены следующие критерии:

* Все тест-кейсы пройдены успешно.
* Обнаруженные дефекты низкой или средней серьезности исправлены или задокументированы.
* Отсутствие критических ошибок, которые мешают нормальной работе приложения.
* Минимум 95% покрытия функциональных тестов для ключевых функций программы.

## 1.11 Критерии приостановки и возобновления работ

Тестирование будет *приостановлено* в случае возникновения следующих условий:

* Обнаружен критический дефект, препятствующий дальнейшему тестированию (например, программа не запускается или не обрабатывает пользовательский ввод).
* Выявлены многочисленные ошибки в базовой логике игры, в том числе, некорректное отображение поля после каждого хода игроков или неверное определение условия завершения игры или победителя.

Тестирование будет *возобновлено* после устранения выявленных критических дефектов, которые препятствовали прогрессу.

## 1.12 Тестовая документация

* План тестирования;
* Отчет о дефектах;
* Отчет о выполнении тестов;
* Отчет о корректирующих действиях;

## 1.13 Основные задачи тестирования

Проверка корректности логики программы: правильный ход игры, корректная обработка пользовательского ввода и сохранение занятых позиций на поле.

## 1.14 Необходимый персонал и обучение

**Тестировщик** для проведения функционального, юнит и регрессионного тестирования. Обучение тестировщика работе с приложением, объяснение основных принципов логики приложения.

**Разработчик** для исправления найденных дефектов.

## 1.15 Требования среды

Приложение должно корректно работать в операционной системе Windows и UNIX-подобных системах.

Специальные аппаратные и программные компоненты не требуются, так как приложение разрабатывается в виде консольного приложения.

На системе должен быть установлен интерпретатор языка Python версии не ниже 3.0.0, дополнительных зависимостей не требуется, поскольку код использует только стандартную библиотеку.

## 1.16 Распределение ответственности

Тестировщик — выполняет тест-кейсы, анализирует результаты, составляет отчеты об ошибках и завершении тестирования.

Разработчик — исправляет выявленные дефекты, дорабатывает функционал по результатам тестирования.

## 1.17 График работ (календарный план)

В Таблице 1 представлен график работ (календарный план) по тестированию ПО.

*Таблица 1 – График работ*

|  |  |  |
| --- | --- | --- |
| **Этап тестирования** | **Зависимость** | **Срок выполнения** |
| Подготовка тестовой документации | Разработка ПО | 5 календарных дней |
| Проведение функциональных тестов | Сдача сборки | 7 календарных дней |
| Регрессионное тестирование | Внесение изменений | 3 календарных дня |
| Подготовка отчета о тестировании | Завершение тестов | 2 календарных дня |

## 1.18 Риски и непредвиденные обстоятельства

***Изменение требований*** в процессе разработки может потребовать пересмотра тест-кейсов и планов тестирования. В таком случае календарный план тестирования будет скорректирован, а команда проинформирована о возможных переносах.

## 1.19 Утверждение плана тестирования

Процесс утверждения плана тестирования включает согласование с менеджером проекта и разработчиком.

## 1.20 Глоссарий

**Тест-кейс** — описание конкретного теста, который проводится для проверки определенной функции или компонента.

**Дефект** — ошибка, обнаруженная в процессе тестирования, которая требует исправления.

**Регрессионное тестирование** — проверка приложения после внесения изменений для подтверждения того, что оно работает корректно.

# 2 АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ WEB-ПРИЛОЖЕНИЯ

В качестве веб-ресурса для проведения разработки скриптов и проведения тестирования с помощью библиотеки Selenium языка Python выбран kinopoisk.ru. Для самого тестирования использована стандартная библиотека unittest.

Проведено тестирование жанровой подборки на сайте. Если пользователь хочет выбрать конкретный жанр, то сайт должен в первых строчках выдать ему фильмы, соответствующие выбранному жанру. Выполнение тестирования представлено на Рисунках 2.1 – 2.2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт

Автоматически созданное описание

Рисунок 2.1 – Тестирование отображения страницы с жанрами

Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт

Автоматически созданное описание

Рисунок 2.2 – Тестирование отображения фильмов в жанре «Мультфильм»

Далее проведён тест поиска фильмов – при вводе в поисковую строку названия фильма пользователю на первой строке должен предлагаться искомый фильм, а при вводе несуществующего названия ничего предлагаться не должно. Выполнение тестирования представлено на Рисунках 2.3 – 2.4.

Изображение выглядит как текст, снимок экрана, Веб-сайт, программное обеспечение

Автоматически созданное описание

Рисунок 2.3 – Тестирование поиска существующего фильма

Изображение выглядит как текст, человек, снимок экрана, Человеческое лицо

Автоматически созданное описание

Рисунок 2.4 – Тестирование поиска несуществующего фильма

Последним проведено тестирование страницы с индексами сериалов – выводимый процент должен совпадать с рассчитанным отношением баллов индекса сериала к индексу общего интереса. Выполнение тестирования представлено на Рисунке 2.5.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.5 – Тестирование отображения индексов сериалов

Далее приведены результаты успешного выполнения представленных тестов на Рисунках 2.6 – 2.7.

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Рисунок 2.6 – Результат тестирования жанровой подборки

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 2.7 – Результат тестирования индексов сериалов

Программный код всех тестов представлен в Листинге А.1.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были освоены основные принципы разработки плана тестирования программного обеспечения. Полученные знания были применены на практике. Также проведено автоматизированное тестирование web-приложения kinopoisk.ru с помощью инструмента Selenium.

# Приложения

Приложение А – Программный код юнит-тестов

### Приложение А

Программный код юнит-тестов

Листинг А.1 – Код юнит-тестов на языке Python

import time

import unittest

from selenium import webdriver

from selenium.webdriver.common.by import By

plural\_words = {

    'комедии': 'комедия',

    'мультфильмы': 'мультфильм',

    'ужасы': 'ужасы',

    'фантастика': 'фантастика',

    'триллеры': 'триллер',

    'боевики': 'боевик',

    'мелодрамы': 'мелодрама',

    'детективы': 'детектив',

    'приключения': 'приключения',

    'фэнтези': 'фэнтези',

    'военные': 'военный',

    'семейные': 'семейный',

    'аниме': 'аниме',

    'исторические': 'история',

    'драмы': 'драма',

    'документальные': 'документальный',

    'детские': 'детский',

    'криминал': 'криминал',

    'биографии': 'биография',

    'вестерны': 'вестерн',

    'фильмы-нуар': 'фильм-нуар',

    'спортивные': 'спорт',

    'реальное тв': 'реальное тв',

    'короткометражки': 'короткометражка',

    'музыкальные': 'музыка',

    'мюзиклы': 'мюзикл',

    'ток-шоу': 'ток-шоу',

    'игры': 'игра',

}

real\_films = [

    'американский психопат', 'во все тяжкие', 'чужой', 'легенда', 'крик',

    'аватар', 'риддик', 'паранормальное явление', 'острые козырьки',

    'и гаснет свет...', 'ходячие мертвецы', 'гравити фолз', 'фиксики'

]

false\_films = [

    'ропвыроапвыроапроы', 'рту мирэа', 'сиаод', ' ', '...', 'видеокарта'

]

class TestKinopoisk(unittest.TestCase):

    def setUp(self) -> None:

        self.browser = webdriver.Firefox()

    def test\_check\_genres(self):

        url = 'https://www.kinopoisk.ru/lists/categories/movies/8/'

        self.browser.get(url)

        time.sleep(10)

        genres = self.browser.find\_elements(By.CSS\_SELECTOR,

Продолжение Листинга А.1

                                            '.styles\_content\_\_2mO6X a')

        genre\_urls = [genre.get\_attribute('href') for genre in genres]

        genre\_names = [

            genre.find\_element(By.CLASS\_NAME,

                               'styles\_name\_\_G\_1mq').text.lower()

            for genre in genres

        ]

        for genre\_name, genre\_url in zip(genre\_names, genre\_urls):

            print(f"Проверка жанра: {genre\_name}")

            self.browser.get(genre\_url)

            time.sleep(1)

            films = self.browser.find\_elements(By.CLASS\_NAME,

                                               'styles\_root\_\_ti07r')

            for film in films:

                film\_desc = film.find\_element(

                    By.CLASS\_NAME,

                    'desktop-list-main-info\_truncatedText\_\_IMQRP').text.lower(

                    )

                self.assertIn(

                    plural\_words[genre\_name], film\_desc,

                    f"Жанр '{plural\_words[genre\_name]}' не найден в описании.")

            time.sleep(1)

    def test\_film\_search(self):

        url = 'https://www.kinopoisk.ru/'

        self.browser.get(url)

        time.sleep(10)

        search\_field = self.browser.find\_element(By.NAME, 'kp\_query')

        search\_field.click()

        for film\_name in real\_films:

            search\_field.send\_keys(film\_name)

            time.sleep(1)

            suggested\_film\_name = self.browser.find\_element(

                By.CLASS\_NAME, 'styles\_mainLink\_\_A4Xkh').text

            self.assertIn(film\_name, suggested\_film\_name.lower())

            search\_field.clear()

        for film\_name in false\_films:

            search\_field.send\_keys(film\_name)

            time.sleep(1)

            self.assertEqual(

                self.browser.find\_element(By.CLASS\_NAME,

                                          'styles\_emptySuggest\_\_XEkB0').text,

                'По вашему запросу ничего не найдено')

            search\_field.clear()

    def test\_serial\_index(self):

        url = 'https://www.kinopoisk.ru/special/index/#/?dateFrom=2024-10-21&dateTo=2024-10-27'

        self.browser.get(url)

        time.sleep(10)

        main\_index = int(

            self.browser.find\_element(By.CLASS\_NAME,

                                      'MainHeader\_value\_\_2mIDd').text)

        rows = self.browser.find\_elements(By.CLASS\_NAME,

                                          'Table\_link\_\_Fz2Jp')[:100]

        for i, row in enumerate(rows):

            spans = row.find\_elements(By.TAG\_NAME, 'span')

            index = int(spans[2].text)

            percent = float(spans[3].text.removesuffix('%').replace(',',

Окончание Листинга А.1

'.'))

            self.assertAlmostEqual(round(index / main\_index \* 100, 2), percent,

                                   1)

            print(f'{i + 1} итерация\n Указанное значение: {percent}, реальное значение: {round(index / main\_index \* 100, 2)}')

            time.sleep(0.1)

    def tearDown(self) -> None:

        self.browser.close()

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

    unittest.main()