МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

КОНСУЛЬТАНТ
Unity Team Lead
_____К.А. Терентьев
«___»_____2023 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____Д.В. Топольский
«___»_____2023 г.

Разработка программного модуля для функционального тестирования программ на платформе Unity

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.215 ПЗ ВКР

Руководитель работы,
к.пед.н., доцент каф. ЭВМ
_____Ю.Г. Плаксина
«___»_____2023 г.

Автор работы,
студент группы КЭ-406
_____М.А. Клюшин
«___»_____2023 г.

Нормоконтролер,
ст. преподаватель каф. ЭВМ
_____С.В. Сяськов
«___»_____2023 г.

Челябинск-2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____Д.В. Топольский
«___»_____2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Клюшину Михаилу Алексеевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка программного модуля для функционального тестирования программ на платформе Unity» утверждена приказом по университету от 25.04.2023 г. № 753-13/12.

2. **Срок сдачи студентом законченной работы:** 01 июня 2023 г.

3. **Исходные данные к работе:**
   1. Программный модуль для функционального тестирования.
   2. Требования к структуре и функционированию системы.
      Четыре функциональные подсистемы:
      - подсистема поиска и редактирования;
      - подсистема формирования сценариев;
      - подсистема хранения сценариев;
      - подсистема проверки сценариев.
   3. Разрабатываемый программный модуль должен быть кроссплатформенным.

4. **Перечень подлежащих разработке вопросов:**
   Выпускная квалификационная работа (ВКР) должна содержать разработку следующих вопросов:
   1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.
   2. Проектирование программной архитектуры.

3. Разработка программного решения.
4. Проведение тестирования программного решения.

5. **Дата выдачи задания:** ___ _____ 202__ г.

Руководитель работы _____/ Ю.Г. Плаксина
Студент _____/ М.А. Клюшин

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы	01.03.2023	
Проектирование программной архитектуры	03.04.2023	
Разработка программного решения	10.05.2023	
Проведение тестирования программного решения	19.05.2023	
Компоновка текста работы и сдача на нормоконтроль	24.05.2023	
Подготовка презентации и доклада	30.05.2023	

Руководитель работы _____/ Ю.Г. Плаксина

Студент _____/ М.А. Клюшин

Аннотация

М.А. Клюшин. Разработка программного модуля для функционального тестирования программ на платформе Unity. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2023, 64 с.,11 ил., библиогр. список – 28 наим.

В рамках выпускной квалификационной работы проводится проектирование и разработка программного модуля для функционального тестирования программ. Проведен анализ доступных инструментов для функционального тестирования программ на платформе Unity. Выявлены преимущества и недостатки существующих решений. Определены ключевые функции программного модуля для организации тестирования программ. Составлен список требований к программному модулю. Цель выпускной квалификационной работ – разработать инструмент для функционального тестирования программ, позволяющий создание тестов без использования средств языков программирования и обеспечивающий автоматическую проверку тестов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Темой выпускной квалификационной работы является разработка эффективного инструмента, реализующего функционал поиска объектов и компонентов, вызова методов и получения значений из найденных результатов на сцене в платформе разработки Unity, использующий синтаксис XPath.

Unity на данный момент является одним из самых популярных кроссплатформенных решений для создания интерактивного контента. Согласно официальной статистике от разработчиков за 2021 год, более 50% игр для мобильных устройств, ПК и консолей были созданы с помощью Unity [1].

Благодаря модульной архитектуре Unity появляется возможность создания собственных инструментов разработки (программных модулей), которые в дальнейшем, по мере необходимости, можно внедрить в любой проект, разрабатываемый на данной платформе [2].

Во время разработки программных продуктов остро встает вопрос качества, заставляющий постоянно отслеживать состояние работоспособности продукта. Во время тестирования программного решения воспроизводятся различные сценарии работы продукта, с целью выявления ошибок. Часто возникают ситуации, когда при исправлении одной ошибки необходимо проверить и работоспособность уже протестированного сценария, ведь возможно, эти сценарии связаны. Чтобы тестировщик тратил меньше времени на поиск ошибок, необходимо покрывать проверенные сценарии автоматизированными тестами. На данный момент в среде разработки Unity нет инструмента позволяющего при написании теста быстро находить необходимый вам элемент. В связи с этим время тестировщика тратится не на выявление ошибок, а на поиск необходимого ему объекта или компонента. Все усложняется тем, что в крупном проекте таких элементов может быть тысячи.

Актуальность темы дипломной работы подтверждает заинтересованность в разработке данного инструмента компанией ООО «Айси Студио», которая выступает в лице заказчика. Основным видом деятельности ООО «Айси Студио» является разработка программного обеспечения. Одним из важных направлений является программного обеспечения на платформе Unity.

В качестве синтаксиса для поиска необходимого компонента был выбран язык запросов XPath [3]. Данный язык широко зарекомендовал себя в WEB-разработке, использующийся для навигации по элементам и атрибутам XML-документа. Отличительной особенностью которого является простота освоения.

Целью выпускной квалификационной работы является разработать инструмент для функционального тестирования программ, позволяющий создание тестов без использования средств языков программирования и обеспечивающий автоматическую проверку тестов.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.
2. Проектирование программной архитектуры.
3. Разработка программного решения.
4. Проведение тестирования программного решения.

1 АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ ПО ТЕМАТИКЕ РАБОТЫ

Предметной областью данной выпускной квалификационной работы является разработка и тестирование программного продукта в среде разработки Unity.

Объектом исследования является интеграция функционала XPath в платформу разработки Unity.

Данное программное решение, позволит сократить время, затрачиваемое на тестирование продукта приблизительно в два раза. Это предоставит возможность оперативно отслеживать состояние продукта и вовремя вносить критические изменения и исправления.

На данный момент не существует аналогов данного решения на платформе Unity, так же есть заинтересованная компания в данном инструменте, что говорит об актуальности, приведенной выше проблемы.

Согласно техническому заданию, данный инструмент должен быть выделен в отдельный репозиторий Git в виде и поставляться в Unity в виде NPM пакета, предполагающий использование в CI/CD pipeline.

Задача CI/CD – сократить время, необходимое для доставки программного обеспечения пользователям, без ущерба для качества. Чтобы этого добиться, необходимо регулярно проверять наличие изменений, тщательно их тестировать и быстро обрабатывать полученную обратную связь, чтобы внедрять изменения как можно чаще [4].

Говоря о CI/CD pipeline, подразумевается ряд этапов, через которые проходит код: от выхода с компьютера разработчика через тестирование вплоть до доставки пользователям.

Стратегическая задача CI/CD – поставить этот процесс на регулярную основу, как правило – несколько раз в день. Поэтому необходимо максимально автоматизировать процесс, чтобы при завершении каждого этапа автоматически запускался следующий или выдавалось уведомление об ошибке. Автоматизация не только ускоряет процесс в целом, а следовательно, и отдельные циклы обратной связи, но и обеспечивает надежное последовательное выполнение каждого этапа.

Пакетом в Node.js называется один или несколько JavaScript-файлов, представляющих собой какую-то библиотеку или инструмент.

NPM (Node Package Manager) – это стандартный менеджер пакетов, автоматически устанавливающийся вместе с Node.js. Он используется для скачивания пакетов из облачного сервера npm, либо для загрузки пакетов на эти сервера [5].

Репозиторий Git или репозиторий – это папка, в которой Git отслеживает изменения. На компьютере может быть любое количество репозиториев, каждое из которых хранится в собственной папке. Каждый репозиторий Git в системе является независимым, поэтому изменения, сохраненные в одном репозитории Git, не влияют на содержимое другого [6].

1.1 Обзор аналогов

В данном аналитическом обзоре рассмотрены аналоги программного решения как для среды разработки Unity, так и для других платформ разработки. В качестве аналогов разрабатываемого программного модуля были рассмотрены следующие решения: AltTest, AutoPlay, Unit Test.

1.1.1 AltTester

AltTester – это инструмент автоматизации тестирования на основе пользовательского интерфейса с открытым исходным кодом, который помогает находить объекты в вашем проекте Unity и взаимодействовать с ними с помощью тестов, написанных на C#, Python или Java [7].

Основной функционал инструмента AltTester при работе с иерархией объектов Unity [8]:

- получение компонентов, сборок, методов, полей и свойств объекта без доступа к исходному коду;
- получение селекторов и проверка их перед запуском тестов;
- взаимодействие с игрой на рабочем столе AltTester с помощью клавиатуры, мыши, сенсорного экрана и джойстика;
- загрузка любой сцены или уровня;

- контроль скорости игры для отладки и разработки тестов.

Рабочий интерфейс программы AltTester представлен на рисунке 1.

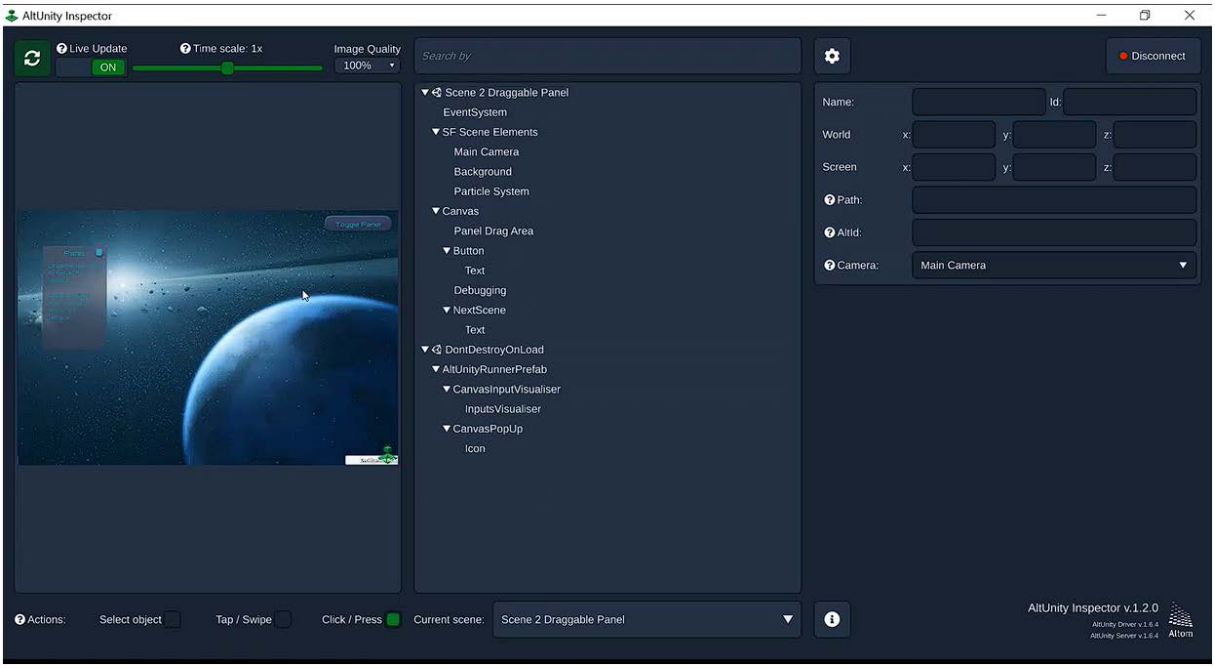

Рисунок 1 – Интерфейс инструмента AltTester

Данный инструмент позволяет быстро находить необходимый объект или компонент и интегрировать его в скрипт теста. Основная проблема в том, что для написания тестов необходимо писать код, что имеет ряд недостатков:

- необходим высокий уровень знаний программирования от специалиста по тестирования;
- повышается риск составления неверного теста;
- нет возможности оперировать группой объектов.

1.1.2 AutoPlay

AutoPlay – это бесплатный инструмент для тестирования автоматизации мобильных игр с открытым исходным кодом, который помогает автоматизировать приложения как для Android, так и для iOS. Помимо работы с Unity, так же может быть интегрирован в Unreal Engine [9].

Рабочий интерфейс программы AutoPlay представлен на рисунке 2.

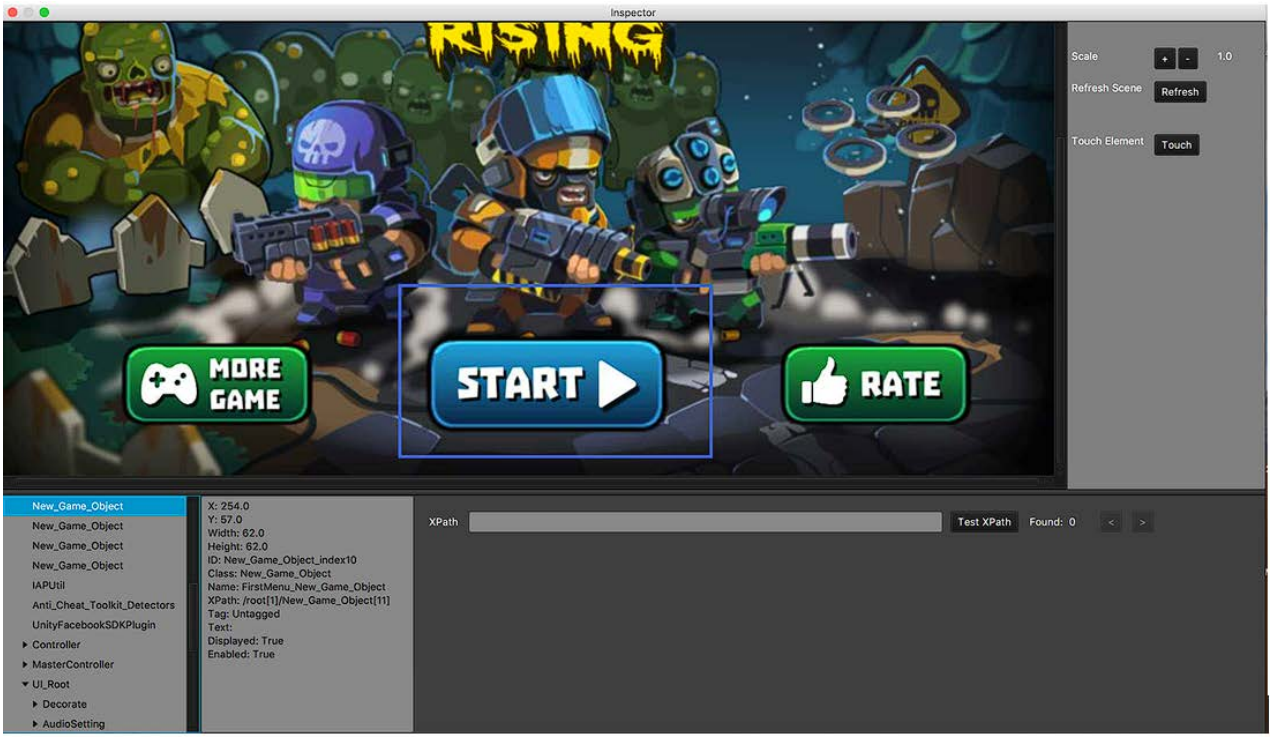

Рисунок 2 – Интерфейс инструмента AutoPlay

Данный инструмент предоставляет собственную реализацию иерархии проекта, позволяющий напрямую взаимодействовать с объектом либо воспользоваться поисковой строкой, использующей синтаксис XPath [10].

Недостатки данного инструмента:

- не полная документация, есть инструкция по установке и подключению инструмента, но нет описания функциональных возможностей;
- для составления тестов необходимо взаимодействовать с кодом;
- для запуска теста необходимо устройство Android или iOS.

1.1.3 Unit Test

Unit Test – предназначен для тестирования отдельной части кода «юнита». Состав «юнита» может варьироваться, но важно, что юнит-тест должен тестировать ровно один «элемент» за раз [11].

Юнит-тесты необходимо создавать для проверки того, что небольшой логический фрагмент кода в конкретном сценарии выполняется именно так, как вы ожидаете. Ниже приведен пример юнит-теста.

Написан метод (рисунок 3), позволяющий пользователю вводить имя. Метод написан так, что в имени не допускаются цифры, а само имя может состоять только

из десяти или менее символов. Метод перехватывает нажатие каждой клавиши и добавляет соответствующий символ в поле name.

```
public string name = ""
public void UpdateNameWithCharacter(char: character)
{
    // 1
    if (!Char.IsLetter(char))
    {
        return;
    }

    // 2
    if (name.Length >= 10)
    {
        return;
    }

    // 3
    name += character;
}
```

Рисунок 3 – Пример Unit Test

Разбор кода метода:

- если символ не является буквой, то код выполняет предварительный выход из функции и не добавляет символ в строку;
- если длина имени составляет десять или более символов, то код не позволяет пользователю добавить ещё один символ;
- если эти две проверки пройдены, то код добавляет в конец имени символ.

Этот юнит-тест можно протестировать, потому что он представляет собой «модуль» выполняемой работы. Юнит-тесты принудительно выполняют логику метода.

В целом юнит-тестирование можно считать хорошим способом, позволяющим проверить работоспособность и качество кода, но все же есть несколько моментов, которые могут привести к проблемам. Одним из них является то, что юнит-тест –

это обычно всего лишь одна функция. Поэтому, если вы реализуете несколько функций, то вам потребуется написать несколько юнит-тестов, каждый из которых будет проверять отдельную функцию, и это может стать довольно трудоемким и затратным делом.

Для сравнения аналогов был выделен следующий ряд критериев:

1. Аналог является программой или программным модулем.
2. Аналог тестирует функционал объектов или компонентов на сцене.
3. Аналог выполняет автоматизированную проверку тестов.
4. Аналог позволяет формирование тестов без использования программного кода.

Результат сравнение аналогов по ряду критериев, приводится в таблице 1.

Таблица 1 – Сравнение аналогов

Критерии	AltTest	AutoPlay	Unit Test
Программа или программный модуль для тестирования	+	+	–
Тестирует функционал объектов или компонентов	+	+	+/–
Автоматизированная проверка тестов	+	–	+
Формирование тестов без использования программного кода	–	–	–

1.2 Определение требований

1.2.1 Функциональные требования

Разработка будет разделена на этапы [12].

Каждый этап характеризуется введением дополнительных команд в формате языка XPath, расширяющих функционал программного модуля.

Этап 1.

Выражения пути к объекту или компоненту представлены в таблице 2.

Таблица 2 – Выражения пути

Выражение	Описание
имя узла	Выбирает все узлы с именем "nodename"
/	Выбирает из корневого узла
//	Выбирает узлы в документе из текущего узла, которые соответствуют выделенному, независимо от того, где они находятся
@	Выбирает атрибуты (компоненты)

Поддерживаемые компоненты.

У каждого компонента есть свойство "value", связанное с ним. Значение возвращается в соответствующих результатах поиска и будет использоваться при поиске по предикатам. Значение XPathObject должно быть установлено на первый подключенный компонент (собственный или дочерний).

Поддерживаемые компоненты представлены в таблице 3.

Таблица 3 – Поддерживаемые компоненты

Компонент	Значение
Button / TMP Button	Ярлык на кнопке
Slider	Значение ползунка
Text / TMP Text	Содержание текста

Продолжение таблицы 3

Компонент	Значение
Input Field / TMP Input Field	Ввод текста
Dropdown / TMP Dropdown	Выделенный текст
Toggle	Переключатель установлен
Image / Raw Image	Название спрайта / текстуры
Любой компонент, который реализует IXPathAttribute интерфейс	IXPathAttribute.value

Примеры работы команд представлены в таблице 4.

Таблица 4 – Пример работы команд

Выражение пути	Результат
bookstore	Выбирает все узлы с именем "bookstore".
/bookstore	Выбирает корневой элемент bookstore. Примечание: Если путь начинается с косой черты (/), он всегда представляет абсолютный путь к элементу!
bookstore/book	Выбирает все элементы book, которые являются дочерними элементами bookstore
//book	Выбирает все элементы book независимо от того, где они находятся в документе
bookstore//book	Выбирает все элементы book, которые являются потомками элемента bookstore, независимо от того, где они находятся под элементом bookstore
//@lang	Выбирает все атрибуты (компоненты) с именем lang независимо от того, где они находятся
@lang	Выбирает все атрибуты (компоненты) с именем lang из корня сцены

Этап 2.

Предикаты – условия фильтрации внутри квадратных скобок. Необходимо убедиться, что значение атрибута является правильным. Работа фильтрации представлена в таблице 5.

Таблица 5 – Работа фильтрации

Выражение пути	Результат
/bookstore/book[1]	Выбирает первый элемент book, который является дочерним элементом элемента bookstore.
/bookstore/book[last()]	Выбирает последний элемент book, который является дочерним элементом элемента bookstore
/bookstore/book[last()-1]	Выбирает предпоследний элемент book, который является дочерним элементом элемента bookstore
/bookstore/book[position()<3]	Выбирает первые два элемента book, которые являются дочерними элементами элемента bookstore
//title[@lang]	Выбирает все элементы title, у которых есть атрибут (компонент) с именем lang
//title[@lang='en']	Выбирает все элементы title, которые имеют атрибут "lang" (компонент) со значением "en"
/bookstore/book[price>35.00]	Выбирает все элементы book элемента bookstore, которые имеют элемент price со значением, превышающим 35,00. В Unity это означает, что у "книги" есть дочерний элемент "цена" с числовым значением (ползунок, ввод, текст и т. Д.), превышающим 35
/bookstore/book[price>35.00]/title	Выбирает все элементы заголовка элементов book элемента bookstore, которые имеют элемент price со значением, превышающим 35,00

Этап 3.

Знаки выбора неизвестных узлов представлены в таблице 6.

Таблица 6 – Знаки выбора неизвестных узлов

Подстановочный знак	Описание
*	Соответствует любому элементу node
@*	Соответствует любому узлу атрибута
node()	Соответствует любому узлу любого типа

Примеры выбора узлов представлены в таблице 7

Таблица 7 – Примеры выбора узлов

Выражение пути	Результат
/bookstore/*	Выбирает все узлы дочернего элемента элемента bookstore
//*	Выбирает все элементы в документе
//title[@*]	Выбирает все элементы заголовка, которые имеют хотя бы один атрибут любого вида

Выбор нескольких путей представлен в таблице 8.

Таблица 8 – Примеры выбора путей

Выражение пути	Результат
//book/title	//book/price
//title	//price
/bookstore/book/title	//price

Модели для Unity:

- компонент XPath:
    - имя строки;
    - строковое значение.
- XPathObject:
    - имя строки;

- компоненты XPathComponent[];
- XPathObject[] дочерние элементы;
- родительский объект XPathObject.

Примеры поисковых запросов приведены в листинге А1, приложения А.

Формат сериализации объекта XPathObject в JSON для сохранения в файл представлен на рисунке 4.

```
{
  "name": "name",  // component name
  "components": [ // all components attached to object except Transfrom & RectTransfrom + Value (if present)
    {
      "name": "Button"
    },
    {
      "name": "MyComponent"
    },
    {
      "name": "Text",
      "value": "Text label"
    }
  ],
  "children": [ // names of direct children
    "label"
  ],
  "parent": "foo" // name of parent or ""
}
```

Рисунок 4 – Сериализация XPathObject в JSON

XPathResult:

- объекты XPathObject[];
- дочерние элементы(индекс) – возвращает XPathResult со всеми дочерними элементами;
- Parent() – возвращает XPathResult со всеми родителями;
- Get(строка xpath) – выполняет запрос ко всем объектам XPathResult.

Пример сцены представлен на рисунке 5.

- foo
  - button1 [Button]
    - label [Text, Value="Button 1"]
  - button2 [Button]
    - label [Text, Value="Button 2"]
- bar
  - label1 [Text, Value="Bar"]
  - label25 [Text, Value="Description"]
- panel
  - bg [Image, Value="bg_sprite_name"]
  - label [Text, Value="Input your name"]
  - Input [Input, Value="Some name"]

Рисунок 5 – Пример сцены

1.2.1.1 Требования к структуре и функционированию системы

Разрабатываемая система должна получать объекты и компоненты, которые хранятся в иерархии проекта Unity.

Выделяется четыре функциональные подсистемы:

- подсистема поиска и редактирования, которая предоставляет интерфейс взаимодействия разработчика с программой с возможностью поиска и редактирования релевантных объектов и компонентов;
- подсистема формирования сценариев, которая предоставляет интерфейс формирования новых сценариев;
- подсистема хранения сценариев, которая получает новые сценарии от тестировщика и хранит их в структурированном виде, для отображения в пользовательском интерфейсе и последующей проверки сценария;
- подсистема проверки сценариев, которая получает сценарии из хранилища и проверяет правильность их выполнения.

1.2.1.2 Требования к функциям, выполняемым системой

Разрабатываемая программа должна предоставлять следующий набор функций:

1. Предоставлять доступ к объектам и компонентам в иерархии сцены Unity.

2. Предоставлять пользователю интерфейс поиска объектов и компонентов.
3. Предоставлять пользователю интерфейс создания сценариев тестирования.
4. Предоставлять пользователю интерфейс проверки сценариев тестирования.

1.2.2 Нефункциональные требования

1.2.2.1 Общие требования

1. Реализация плагина в виде NPM (Node Package Manager) пакета для Unity.
2. Предусмотреть архитектурную возможность расширения функционала.
3. Программный модуль интегрируется в платформу Unity в виде отдельного окна инспектора, позволяющего взаимодействовать с его функционалом.
4. В качестве синтаксиса поисковых запросов использовать язык XPath.

1.2.2.2 Требования к клиентскому программному обеспечению

Разрабатываемый система поддерживает следующие операционные системы:

- Windows 10 version 1909 (build 18363) или новее (только x64);
- Mac OS: Mojave 10.14+ (Intel editor), Big Sur 11.0 (Apple silicon Editor);
- Linux: Ubuntu 20.04 и Ubuntu 18.04.

1.2.2.3 Требования к техническому обеспечению

Конфигурация компьютера:

- процессор: минимум 4-ядерный с частотой выше 2.5 ГГц. Например, Intel Core i3 или Amd Ryzen 3;
- видеокарта: любая с поддержкой DX10 и выше и памятью от 2 Гб;
- оперативная память: 8 Гб и выше;
- 12 ГБ свободного места на жестком диске.

Выводы по разделу один

В результате выполнения анализа предметной области были проведены обзор аналогов разрабатываемого программного модуля и определение функциональных и нефункциональных требований.

Как видно из результатов сравнения рассмотренных аналогов разрабатываемого программного модуля, представленных в таблице 1, ни один из рассмотренных аналогов не соответствует всем критериям, из этого формулируется проблема, что на данный момент не существует программного решения, которое полностью соответствует выбранным критериям.

Исходя из выявленной проблемы была сформулирована следующая цель разработки программного модуля – разработать инструмент для функционального тестирования программного обеспечения, позволяющий создание тестов без использования средств языков программирования и обеспечивающий автоматическую проверку тестов.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ АРХИТЕКТУРЫ

В данной главе приводится описание работы над проектированием программного модуля.

Проектирование – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект – целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для последующей реализации [13].

Для наиболее наглядного представления архитектуры программного модуля, было решено построить концептуальные модели программного модуля и его подсистем.

Концептуальная модель – это абстрактная модель, определяющая структуру моделируемой системы, свойства ее элементов и причинно-следственные связи, присущие системе и существенные для достижения цели моделирования [14].

Концептуальная модель программного модуля, представленную на рисунке 6.

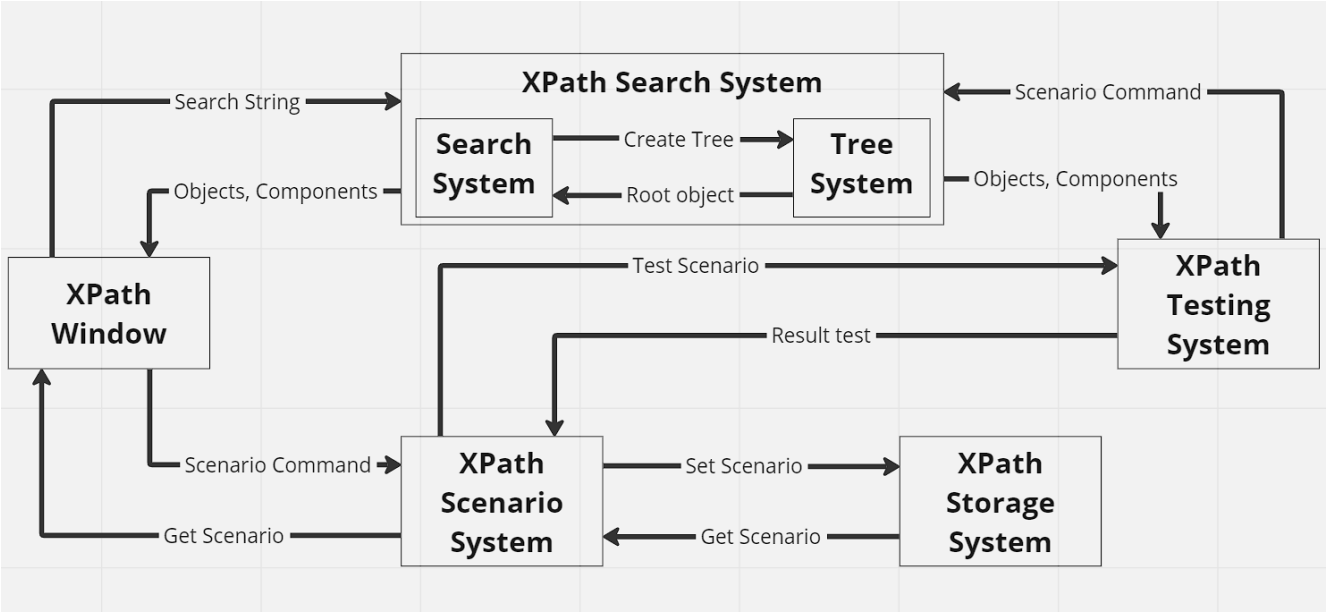

Рисунок 6 – Концептуальная модель программного модуля

Концептуальная модель программного модуля состоит из следующих частей:

- подсистема поиска объектов и компонентов;
- подсистема формирования сценариев;
- подсистема хранения сценариев;

- подсистема проверки сценариев.

Данный набор подсистем позволит реализовать весь основной функционал разрабатываемого программного модуля.

2.1 Подсистема поиска объектов и компонентов

Подсистема поиска состоит из двух систем:

- система построения дерева иерархии;
- система поиска.

2.1.1 Система построения дерева иерархии

Для проведения операции поиска необходимо подготовить объекты и компоненты, среди которых будет проводиться данная операция. В качестве исходных данных используется иерархия объектов и компонентов в сцене Unity [15].

Необходимо получить доступ к иерархии объектов и компонентов в сцене Unity для возможности взаимодействия и получения хранящихся данных подсистемой поиска.

2.1.2 Система поиска

Система поиска осуществляет поиск объектов и компонентов, удовлетворяющих поисковому запросу, а также предоставляет возможность вызова методов у найденных объектов и компонентов.

В качестве входных данных для системы поиска используется строка в формате языка запросов XPath.

Выходными данными является список объектов и компонентов удовлетворяющие запросу.

Для подсистемы поиска была построена концептуальная модель, представленная на рисунке 7.

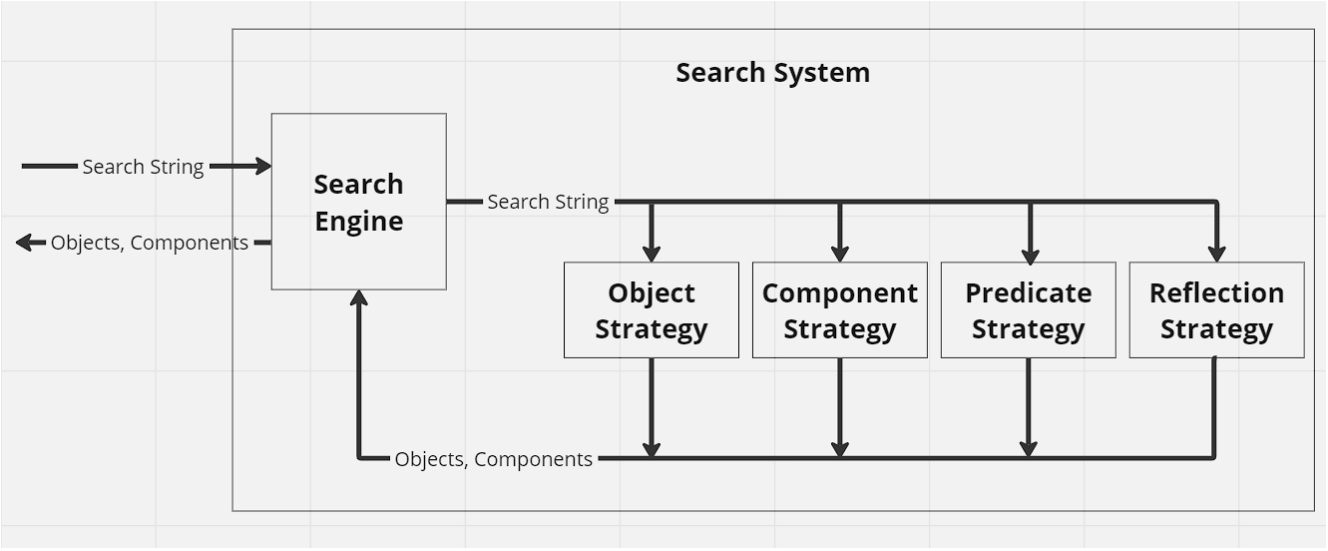

Рисунок 7 – Концептуальная модель системы поиска

Концептуальная модель системы поиска включает в себя сервис «Search Engine», предоставляющий доступ к функционалу поиска.

2.2 Подсистема формирования сценариев

Подсистема формирования сценариев представляет программный интерфейс для формирования тестовых сценариев.

Основной функционал подсистемы формирования сценариев:

- добавление запроса в формате XPath в сценарий;
- редактирование добавленных запросов;
- удаление запроса из сценария.

Приведенный выше функционал позволяет пользователю настраивать сценарий под собственные задачи.

Для подсистемы поиска была построена концептуальная модель, представленная на рисунке 8.

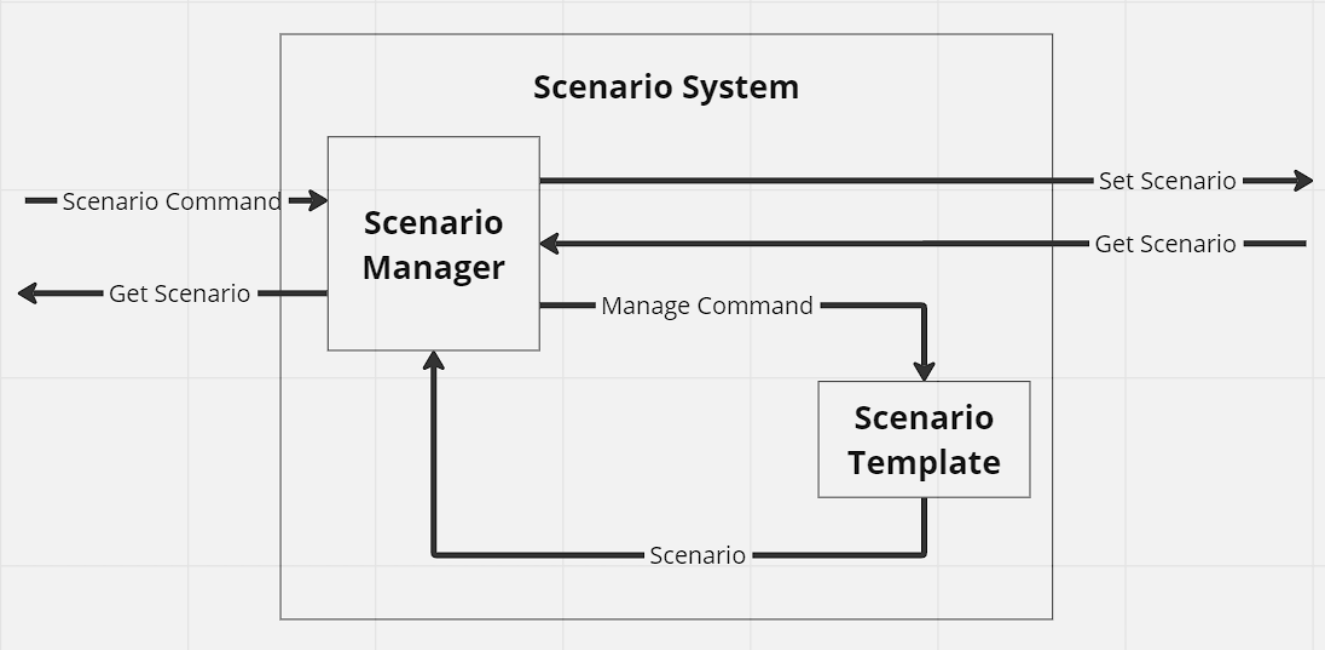

Рисунок 8 – Концептуальная модель подсистемы формирования сценариев

Концептуальная модель подсистемы формирования сценариев предоставляет доступ к шаблону сценария, с которым пользователь может взаимодействовать для построения функционального теста.

2.3 Подсистема хранения сценариев

Подсистема хранения сценариев осуществляет хранение сформированных сценариев тестирования.

Основной функционал подсистемы хранения сценариев:

- сохранение сформированного сценария;
- загрузка сохраненных сценариев.

Для реализации функционала подсистемы использовать формат текстовых данных JSON.

2.4 Подсистема проверки сценариев

Подсистема проверки сценариев осуществляет проверку корректности выполнения сценария тестирования.

Для определения корректности выполнения сценария теста необходимо выполнить запрос в подсистеме поиска.

Корректность проверки, проводимой для сценария, который не храниться в подсистеме хранения осуществляется по результату, полученному от подсистемы поиска.

Для проверки сценария сохраненного в подсистеме хранения необходимо провести сравнение между полученным результатом от подсистемы поиска и результатом сценария, сохраненным в подсистеме хранения сценариев.

3 РАЗРАБОТКА ПРОГРАММНОГО РЕШЕНИЯ

В качестве платформы для реализации программного модуля использовалась среда разработки Unity 2021.3.18f1 [16]. Язык программирования, используемый для реализации C#. Исходный код реализации программного модуля представлен в листинге Б1, приложения Б.

3.1 Реализация подсистемы поиска объектов и компонентов

3.1.1 Реализация системы построения дерева иерархии

В качестве исходных данных для системы построения дерева иерархии являются объекты и компоненты на сцене иерархии Unity. Таким образом мы получаем, что дерево строится на основе объектов и компонентов.

В иерархии Unity родительскими объектами являются наследники класса Object (объекты) [17], их дочерними объектами являются объекты класса Component (компоненты) [18].

Для увеличения скорости получения результата при поиске, было решено строить дерево на основе объектов, а компоненты получать динамически при обращении к ним.

Для построения дерева иерархии был разработан следующий набор классов:

- программный интерфейс INode – контейнер для хранения объектов классов ObjectNode и ComponentNode, является узлом дерева иерархии;
- программный класс ObjectNode – контейнер для хранения объектов класса Object, наследует интерфейс INode;
- программный класс ComponentNode – контейнер для хранения объектов класса Component, наследует интерфейс INode.

Программные свойства интерфейса INode:

- public string Name { get; };
- public string Path { get; };
- public Object Entity { get; };
- public INode PrevNode { get; };

- public List<INode> NextNodeList { get; }.

Свойство Entity интерфейса INode используется для хранения объектов классов ObjectNode и ComponentNode.

Построение дерева иерархии начинается с получения экземпляра сцены Unity и получении у него корневых объектов [19]. Затем создается корневой объект класса ObjectNode с незаполненными свойствами, и в свойство NextNodeList помещаются корневые объекты сцены Unity. Дальнейшее получение объектов из иерархии сцены Unity проводится рекурсивно.

Для обхода дерева иерархии Unity используется поиск в глубину. Поиск в глубину – обход дерева проходит вниз до самого дальнего узла прежде, чем идти к следующему родственному узлу. В бинарных деревьях обход дерева начиная с корня является операцией обработки вершины рекурсивно.

3.1.2 Реализация системы поиска

Входными данными для реализации поиска объектов и компонентов является текстовый запрос в формате языка XPath. Выходными данными является экземпляры объектов и компонентов.

Для реализации прямого вызова функций объектов или компонентов для дальнейшего тестирования было решено использовать встроенную возможность языка C#, а именно Reflection. Функционал Reflection позволяет получить доступ к свойствам, методам и событиям, что позволяет тестировать функционал объектов или компонентов [20].

Язык запросов XPath используется для запросов к элементам XML-документа, в связи с этим, возникла необходимость адаптировать синтаксис языка под запросы к иерархии сцены Unity для получения доступа к объектам и компонентам. Иерархия сцены в Unity представляет собой древовидную иерархическую структуру. Для построения дерева иерархии используется алгоритм обхода дерева в глубину. Следовательно, необходимо составить команды позволяющие производить прямой и рекурсивный поиск в дереве иерархии.

Команды прямого поиска:

- / – поиск следующего объекта в иерархии;
- /@ – поиск следующего компонента в иерархии;

Команды рекурсивного поиска:

- // – поиск всех объектов в иерархии;
- //@ – поиск всех компонентов в иерархии;
- имя объекта – поиск всех объектов в иерархии с указанным именем;
- @имя компонента – поиск всех компонентов в иерархии с указанным именем;
- * – любое имя объекта или компонента;

Для сужения результатов поиска были разработаны команды предикаты, позволяющие установить условия для результатов поиска. Ключевой парой символов обозначающие предикат являются [] (квадратные скобки). Предикат указывается внутри квадратных скобок.

Команды предикаты поиска:

- last() – поиск последнего элемента в результате поиска;
- position() – позволяет обратиться к позиции элемента в результате поиска;
- номер элемента – получение элемента по его позиции в результате поиска;
- <, >, >=, <=, ==, != – операторы сравнения;
- –, + – математические операторы.

Для вызова функционала Reflection был введен ключевой символ . (точка), позволяющий обращаться к методам объектов и компонентов.

Для выполнения определения входной команды было решено использовать регулярные выражения [20].

Регуля́рные выраже́ния – формальный язык, гибкий и эффективный способ обработки текста, используемый для поиска и осуществления манипуляций с подстроками в тексте [21].

Опираясь на разработанные команды и требования к программному модулю, в результате была разработана диаграмма классов подсистемы поиска, представленная на рисунке 9.

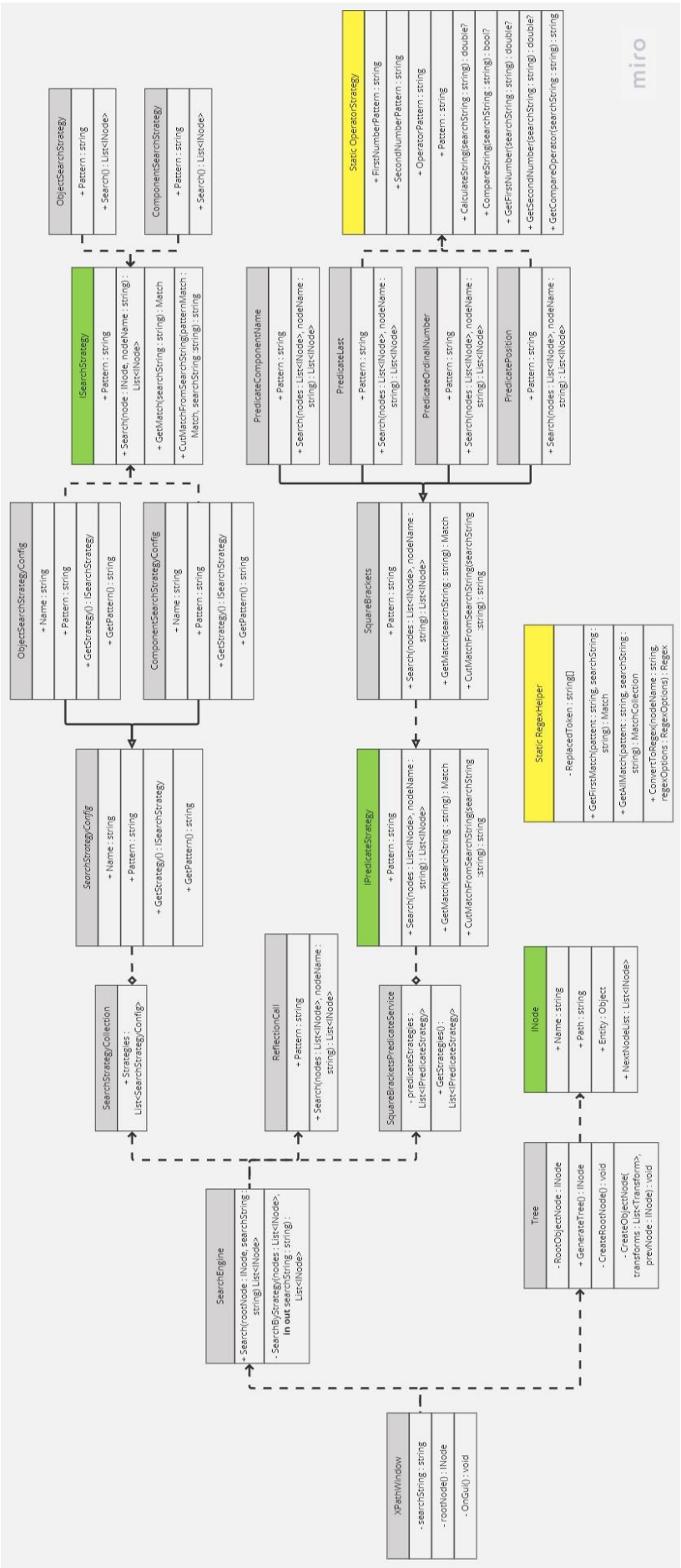

Рисунок 9 – Диаграмма классов подсистемы поиска

3.2 Реализация подсистемы формирования сценариев

Подсистема формирования сценариев в качестве входных данных получает список строковых запросов в формате XPath. Для временного хранения данных о запросах был добавлен список TestList.

Добавление запросов осуществляется через графический интерфейс пользователя. При нажатии на кнопку Add Command строковый запрос добавляется в список TestList.

Для редактирования строки запроса используется текстовое поле в графическом интерфейсе пользователя, содержащее текст строки запроса.

Чтобы удалить строку запроса из сценария теста необходимо выбрать в графическом интерфейсе текстовое поле, соответствующее строке запроса и нажать кнопку Remove Command.

3.3 Реализация подсистемы хранения сценариев

Подсистема хранения сценариев реализована с использованием встроенной в Unity библиотеки JSONSerializeModule [22].

Модуль JsonSerialize предоставляет класс JsonUtility [23], который позволяет сериализовать объекты Unity в формат JSON и десериализовать объекты JSON в объекты Unity.

Сериализация (в JSON) – процесс перевода объекта в строку для дальнейшей передачи или сохранения. Обратной к операции сериализации является операция десериализации – создание объекта из строки [24].

Для сериализации объектов TestList используется функция JsonUtility.ToJson, для десериализации используется функция JsonUtility.FromJson.

3.4 Реализация подсистемы проверки сценариев

Подсистема проверки сценариев проверяет корректность выполнения сценария тестирования.

При отсутствии сценария тестирования в подсистеме хранения проверка осуществляется последовательным выполнением каждого запроса входящего в

сценарий в подсистеме поиска объектов и компонентов, если имеется хотя бы один объект или компонент в результате, то запрос считается корректным.

Когда сценарий тестирования загружен из подсистемы хранения сценариев, производится проверка результата каждого запроса с корректным результатом, имеющимся в подсистеме хранения. Если несоответствия отсутствуют, то запрос считается корректным.

При корректном результате, в консоль выводится сообщение об успешном выполнении сценария тестирования.

В случае отсутствия результата, в консоль выводится сообщение об ошибке и номер запроса в сценарии тестирования вызвавший ее.

3.5 Реализация графического пользовательского интерфейса

Интерфейс программного модуля представляет собой пользовательское окно редактора Unity (EditorWindow) [25], реализованное с помощью встроенной сборки UnityEditor предоставляющая доступ к API (Application Programming Interface) [26] интерфейсам редактора Unity.

Пользовательское окно, представлено на рисунке 10, состоит из 3 основных разделов:

- поиск (Search);
- результат поиска (Result List);
- управление сценарием тестирования (Test).

Раздел поиска состоит из следующих элементов интерфейса:

- Поисковая строка.
- Кнопка, запускающая функцию поиска (Search).
- Кнопка добавления запроса в сценарий тестирования (Add Command).

Раздел результат поиска состоит из списка объектов и компонентов результата поиска.

Раздел управление сценарием тестирования состоит из следующих элементов интерфейса:

- список запросов сценария тестирования (Test);
- кнопка удаления запроса из сценарий тестирования (Remove Command);
- кнопка проверки выполнения теста (Check Test);
- кнопка сохранить сценарий тестирования (Save Test);
- кнопка открытия сценария тестирования их подсистемы хранения (Open Test).

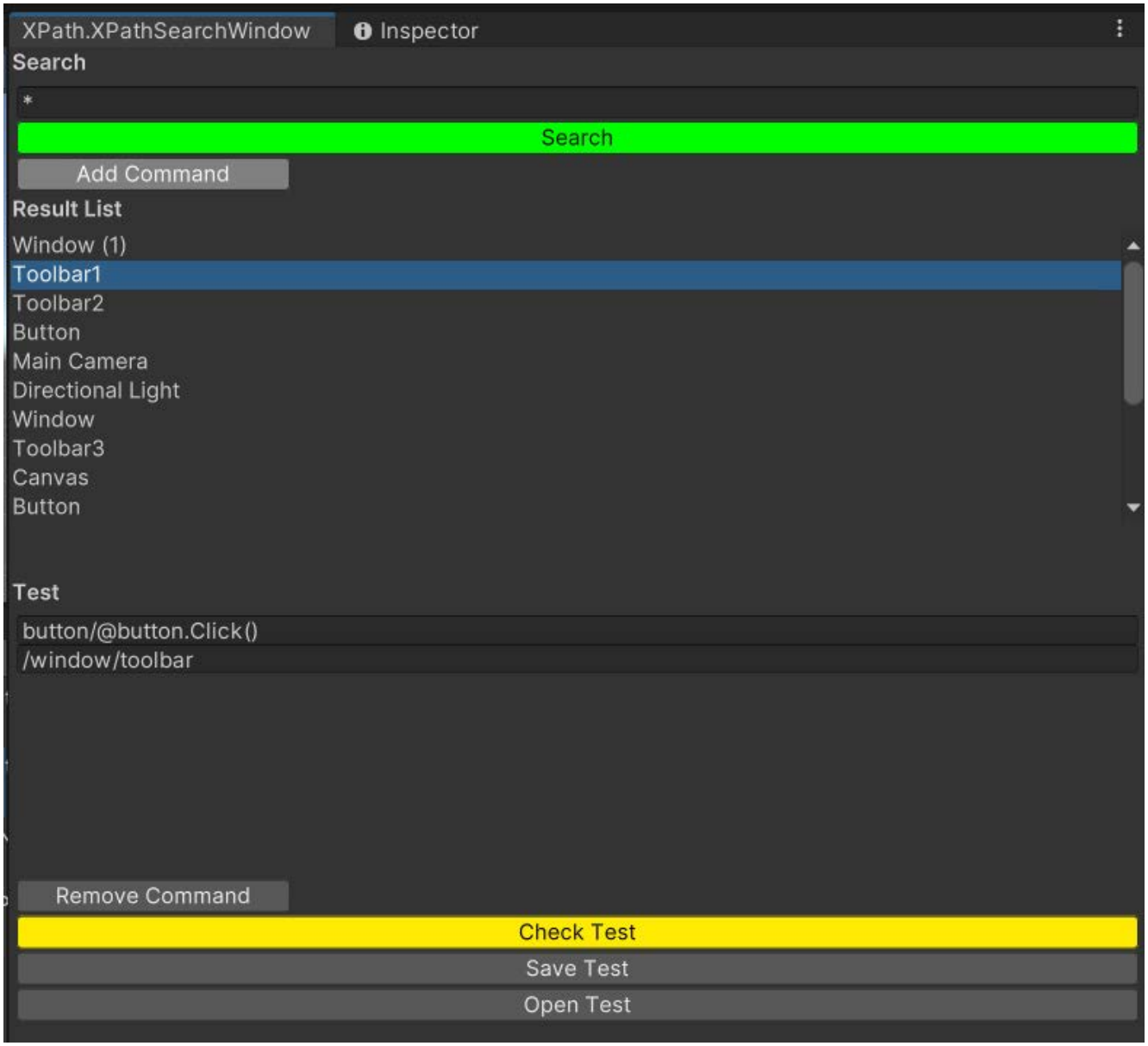

Рисунок 10 – Окно графического пользовательского интерфейса

4 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ ПРОГРАММНОГО РЕШЕНИЯ

4.1 Методология тестирования

Автоматизированное тестирование – это метод тестирования программного обеспечения, при котором выполнение теста, а именно запуск, инициализация, выполнение, анализ и выдача результата выполняется с использованием специальных программных средств, которые, в свою очередь необходимы для выполнения набора тестовых примеров [27]. Программное обеспечение для автоматизации тестирования также может вводить тестовые данные в тестовую среду, сравнивать ожидаемые и фактические результаты и создавать подробные отчеты о тестах.

Для тестирования создается конечный набор тестов, тесты выбираются в соответствии с обычными действиями, выполняемыми в прикладной области, и обеспечивают проверку соответствия ожидаемому поведению системы.

4.2 Проведение процедуры тестирования

Для проведения автоматизированного тестирования был создан набор текстовых запросов, имитирующий запросы пользователя к разработанному программному модулю.

Данные запросы создавались на основе разработанных команд синтаксиса XPath, указанных в пункте 2.1. Функциональные требования.

На рисунке 11 представлен графический пользовательский интерфейс запуска автоматизированного тестирования и результат выполнения тестов.

Исходный код тестов программного модуля представлен в листинге Б2, приложения Б.

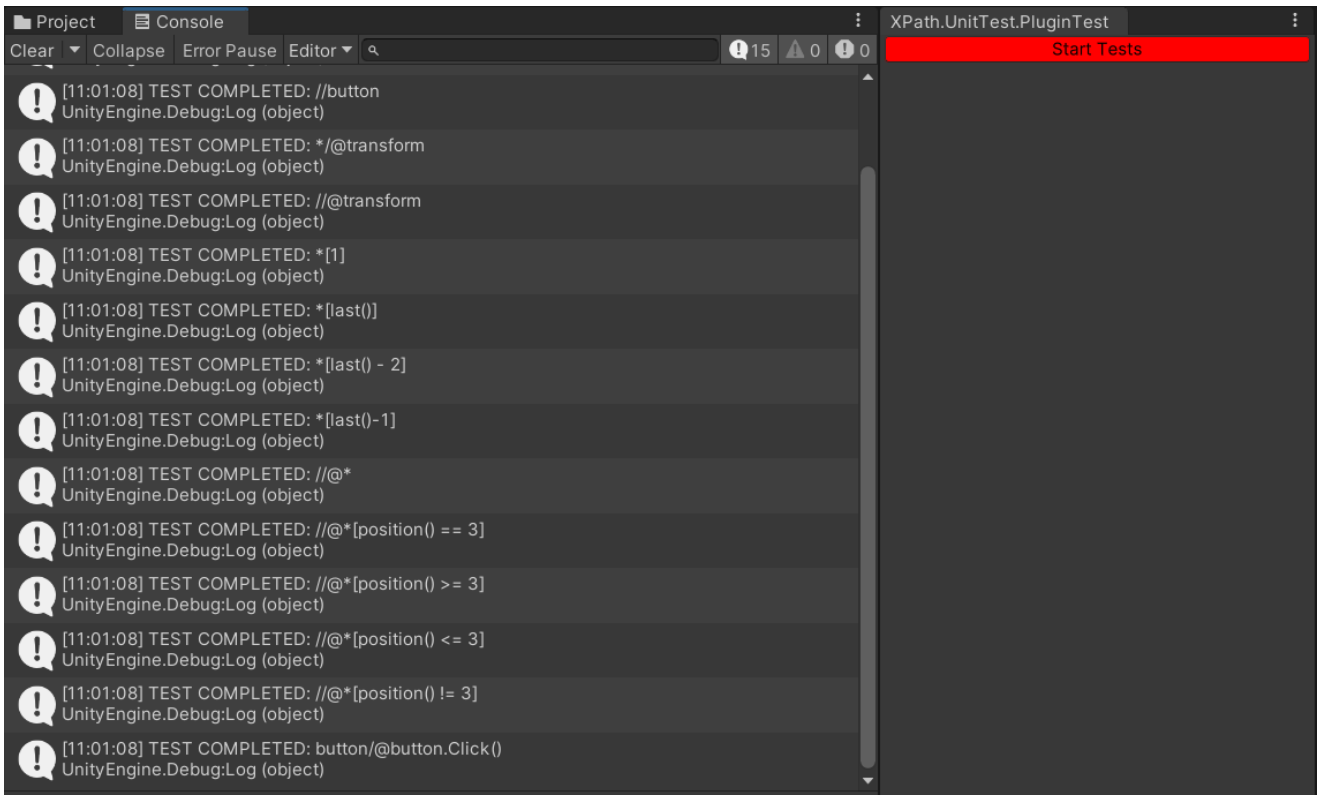

Рисунок 11 – Результаты автоматизированного тестирования

В результате проведенного тестирования была проверена работа функционала программного модуля, из результате тестирования все тесты пройдены успешно.

ЗАКЛЮЧЕНИЕ

В рамках представленной выпускной квалификационной работы был спроектирован и реализован программный модуль для функционального тестирования программ на платформе Unity.

В процессе выполнения была сформулирована следующая цель – разработать инструмент для функционального тестирования программ, позволяющий создание тестов без использования средств языков программирования и обеспечивающий автоматическую проверку тестов.

Для достижения поставленной цели были выполнены следующие задачи:

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.
2. Проектирование программной архитектуры.
3. Разработка программного решения.
4. Проведение тестирования программного решения.

В рамках выполнения поставленных задач было выполнен анализ предметной области, в результате которого были сформированы ключевые критерии и рассмотрены ближайшие аналоги разрабатываемого программного модуля.

Выполнено проектирование программного модуля. В результате которого, были построены диаграммы концептуальной модели, описывающие структуру программного модуля.

В процессе реализации программного модуля была построена диаграмма классов подсистемы поиска и реализован функционал на языке программирования C#.

Выполнено тестирование программного модуля. В процессе которого, был разработан ряд тестов для проверки функционала программного модуля, использующих методологию автоматизированного тестирования. В результате проведенной проверки, все тесты завершились корректно.

Текущая реализация позволяет организовать автоматизированное тестирование графического интерфейса без использования программного кода.

Архитектура программного модуля позволяет расширение функционала, в дальнейшей работе над проектом планируется расширить возможность программного модуля тестировать не только графический интерфейс, но и логику поведения объектов и компонентов моделируя поведение пользователя через использование команд устройств ввода.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Статистика Unity [Электронный ресурс]. – URL: https://unity.com/our-company (Дата обращения: 11.11.2022).

2 Unity Documentation [Электронный ресурс]. – URL: https://docs.unity.com (Дата обращения: 11.11.2022).

3 XPath Syntax [Электронный ресурс]. – URL: https://www.w3schools.com/xml/xpath_syntax.asp (Дата обращения: 11.11.2022).

4 Основные сведения о CI/CD pipeline [Электронный ресурс]. – URL: https://www.jetbrains.com (Дата обращения: 11.11.2022).

5 Официальный сайт Node.js [Электронный ресурс]. – URL: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm (Дата обращения: 12.11.2022).

6 Официальный сайт Git [Электронный ресурс]. – URL: https://git-scm.com (Дата обращения: 12.11.2022).

7 Официальный сайт AltTester [Электронный ресурс]. – URL: https://altom.com/testing-tools/alttester/ (Дата обращения: 12.11.2022).

8 AltTester SDK Documentation [Электронный ресурс]. – URL: https://altom.com/alttester/docs/sdk/home.html (Дата обращения: 12.11.2022).

9 Официальный сайт AutoPlay [Электронный ресурс]. – URL: https://github.com/AutoplayAutomation/AutoPlay (Дата обращения: 13.11.2022).

10 AutoPlay Documentation [Электронный ресурс]. – URL: https://github.com/AutoplayAutomation/AutoPlay/wiki (Дата обращения: 13.11.2022).

11 Osherove R. The art of Unit Testing with Examples in .NET/R. Osherove – New York: By Manning Publications Co, 2009. – 324 p.

12 Технические требования заказчика [Электронный ресурс]. – URL: https://gist.github.com/dmitry-icvr/6ea3450295178f8f87c43ff72f4cba6f (Дата обращения: 13.11.2022).

13 Термин проектирование [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Проектирование (Дата обращения: 03.04.2023).

14 ГОСТ 33244-2015 Информационные технологии [Электронный ресурс]. – URL: https://docs.cntd.ru/document/1200030195 (Дата обращения: 04.04.2023).

15 Сцена в Unity [Электронный ресурс]. – URL: https://docs.unity3d.com/2022.2/Documentation/Manual/CreatingScenes.html (Дата обращения: 04.04.2023).

16 Unity 2021.3.18f1 [Электронный ресурс]. – URL: https://unity.com/releases/editor/whats-new/2021.3.18 (Дата обращения: 04.04.2023).

17 Unity Scripting API: Object [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/Object.html (Дата обращения: 04.04.2023).

18 Unity Scripting API: Component [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/Component.html (Дата обращения: 04.04.2023).

19 Unity Scripting API: Scene.GetRootGameObjects [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/SceneManagement.Scene.GetRootGameObjects.html (Дата обращения: 04.04.2023).

20 Reflection в .NET [Электронный ресурс]. – URL: https://learn.microsoft.com/ru-ru/dotnet/framework/reflection-and-codedom/reflection (Дата обращения: 05.04.2023).

21 Регулярные выражения .NET [Электронный ресурс]. – URL: https://learn.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expressions (Дата обращения: 05.04.2023).

22 Unity Scripting API: UnityEngine.JSONSerializeModule [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/UnityEngine.JSONSerializeModule.html (Дата обращения: 05.04.2023).

23 Unity Scripting API: JsonUtility [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/JsonUtility.html (Дата обращения: 05.04.2023).

24 Сериализация и десериализация JSON [Электронный ресурс]. – URL: https://learn.microsoft.com/ru-ru/dotnet/standard/serialization/system-text-json/overview (Дата обращения: 05.04.2023).

25 Unity Scripting API: EditorWindow [Электронный ресурс]. – URL: https://docs.unity3d.com/ScriptReference/EditorWindow.html (Дата обращения: 05.04.2023).

26 Термин API [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/API (Дата обращения: 03.04.2023).

27 ГОСТ Р 56921-2016. Системная и программная инженерия [Электронный ресурс]. – URL: https://docs.cntd.ru/document/1200134997 (Дата обращения: 04.04.2023).

28 Автоматизированное тестирование [Электронный ресурс]. – URL: https://logrocon.ru/news/automation_testing (Дата обращения: 04.04.2023).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Примеры поисковых запросов

Листинг А1. Примеры поисковых запросов.

```
XPath.Get("button1") // Возвращает объект с именем button1
      [
        {
          "name": "button1",
          "components": [
            {
              "name": "Button"
            }
          ],
          "children": [
            "label"
          ],
          "parent": "foo"
        }
      ]

XPath.Get("button1").Parent() // Возвращает всех родителей результата
[
        {
          "name": "foo",
          "components": [],
          "children": [
            "button1", "button2"
          ],
          "parent": ""
        }
      ]

XPath.Get("button*").Children() // Возвращает всех прямых потомков результата
[
  {
    "name": "label",
          "components": [
            {
              "name": "Text",
              "value": "Button 1"
            }
          ],
          "children": [],
          "parent": "button1"
        },
        {
          "name": "label",
          "components": [
            {
              "name": "Text",
              "value": "Button 2"
            }
          ],
          "children": [],
          "parent": "button2"
        }
      ]
```

```
XPath.Get("/label") // Поиск с корня (или предыдущего результата), в данном случае ничего не находит: []

XPath.Get("//label") // Поиск с любого места

[
  {
    "name": "label",
    "components": [
      {
        "name": "Text",
        "value": "Button 1"
      }
    ],
    "children": [],
    "parent": "button1"
  },
  {
    "name": "label",
    "components": [
      {
        "name": "Text",
        "value": "Button 2"
      }
    ],
    "children": [],
    "parent": "button2"
  },
  {
    "name": "label",
    "components": [
      {
        "name": "Text",
        "value": "Input your name"
      }
    ],
    "children": [],
    "parent": "panel"
  }
]


XPath.Get("//button2/label") // Поиск надписи у кнопки

XPath.Get("//button2").Get("/label") // Поиск надписи у кнопки

XPath.Get("/foo/button2/label") // Поиск надписи у кнопки

[
  {
    "name": "label",
    "components": [
      {
        "name": "Text",
        "value": "Button 2"
      }
    ],
```

```
      "children": [],
      "parent": "button2"
    },
  ]
```

```
XPath.Get("//@button") // Поиск всех объектов с компонентом button
```

```
  [
    {
      "name": "button1",
      "components": [
        {
          "name": "Button"
        }
      ],
      "children": [
        "label"
      ],
      "parent": "foo"
    },
    {
      "name": "button2",
      "components": [
        {
          "name": "Button"
        }
      ],
      "children": [
        "label"
      ],
      "parent": "foo"
    }
  ]
```

ПРИЛОЖЕНИЕ Б

Исходный код программного модуля

Листинг Б1. Исходный код основных классов программного модуля.

```
public interface INode
{
   public string Name { get; }
   public string Path { get; } //TODO: for Debug

   public Object Entity { get; }

   public INode PrevNode { get; }
   public List<INode> NextNodeList { get; }
}

public class ObjectNode : INode
{
   public string Name { get; }
   public string Path { get; }
   public Object Entity { get; }

   public INode PrevNode { get; }
   public List<INode> NextNodeList { get; } = new List<INode>();

   public ObjectNode(string name, string path, Object entity, INode prevNode)
   {
       Name = name;
       Path = path;
       Entity = entity;
       PrevNode = prevNode;
   }
}

public class ComponentNode : INode
{
   public string Name { get; }
   public string Path { get; }
   public Object Entity { get; }
   public INode PrevNode { get; }

   public List<INode> NextNodeList { get; } = new List<INode>();

   public ComponentNode(string name, string path, Object entity, INode
  prevNode)
   {
       Name = name;
       Path = path;
       Entity = entity;
       PrevNode = prevNode;
   }
}

public class Tree
{
   private INode RootObjectNode { get; set; }

   public INode GenerateTree()
   {
       CreateRootNode();
       return RootObjectNode;
   }
```

```
    private void CreateRootNode()
    {
        var activeScene = SceneManager.GetActiveScene();
        var rootEntity = activeScene.GetRootGameObjects();

        List<Transform> rootNextTransforms = new();

        // Objects
        foreach (var element in rootEntity)
        {
            rootNextTransforms.Add(element.transform);
        }

        RootObjectNode = new ObjectNode("", "", null, null);

        CreateObjectNode(rootNextTransforms, RootObjectNode);
    }


    private static void CreateObjectNode(List<Transform> transforms, INode
    prevNode)
    {
        foreach (var element in transforms)
        {
            var gameObject = element.gameObject;
            var name = gameObject.name;
            var path = prevNode.Path + "/" + name;
            var node = new ObjectNode(name, path, gameObject, prevNode);

            //CreateComponentNode(element, node);

            prevNode.NextNodeList.Add(node);
            List<Transform> nextTransforms = new();

            foreach (Transform child in element.transform)
            {
                nextTransforms.Add(child);
            }

            CreateObjectNode(nextTransforms, node);
        }
    }

    private static void CreateComponentNode(Component objectTransform, INode
    prevNode)
    {
        var components = objectTransform.gameObject.GetComponents<Component>();
        foreach (var element in components)
        {
            var name = element.GetType().Name;
            var path = prevNode.Path + "/@" + name;
            var node = new ComponentNode(name, path, element, prevNode);

            prevNode.NextNodeList.Add(node);
        }
    }
}

public class SearchEngine
{
    public List<INode> Search(INode rootNode, string searchString)
```

```
{
    var searchResultCollection = new List<INode>(){rootNode};
    do
    {
        searchResultCollection = SearchByStrategies(searchResultCollection,
ref searchString);

    } while ((searchResultCollection.Count != 0) && (searchString.Length !=
0));

    return searchResultCollection;
}


 private List<INode> SearchByStrategies(List<INode> nodes, ref string
searchString)
{
    var resultNodes = new List<INode>();

    //Global search objects and components
    var strategyConfigCollection =
Resources.Load<SearchStrategyCollection>("Strategies");

    foreach (var strategyConfig in strategyConfigCollection.Strategies)
    {
        var strategy = strategyConfig.GetStrategy();
        var patternMatch = strategy.GetMatch(searchString);

        if (patternMatch == null) continue;

        foreach (var node in nodes)
        {
            var tmpResult = strategy.Search(node, patternMatch.Value);

            resultNodes.AddRange(tmpResult);
        }

        searchString = strategy.CutMatchFromSearchString(patternMatch,
searchString);
        return resultNodes;
    }

    // SquareBracketsPredicate search
    var squareBracketsService = new SquareBracketsPredicateService();
    var squareBracketsStrategies = squareBracketsService.GetStrategies();

    foreach (var predicateStrategy in squareBracketsStrategies)
    {
        var patternMatch = predicateStrategy.GetMatch(searchString);

        if (patternMatch == null) continue;

        var tmpResult = predicateStrategy.Search(nodes, searchString);

        if (tmpResult.Count != 0)
        {
            searchString =
predicateStrategy.CutMatchFromSearchString(searchString);
            resultNodes = tmpResult;
            return resultNodes;
        }
    }
```

```
        var reflectionCall = new DotCall();
        var myresult = reflectionCall.Search(nodes, searchString);

        if (myresult.Count > 0)
        {
            searchString = "";
            return myresult;
        }

        return new List<INode>();
    }
}

public interface ISearchStrategy
{
    public string Pattern { get; }
    public List<INode> Search(INode node, string nodeName);

    public Match GetMatch(string searchString)
    {
        return RegexHelper.GetFirstMatch(Pattern, searchString);
    }

    public string CutMatchFromSearchString(Match patternMatch, string
   searchString)
    {
        var lengthValidString = patternMatch.Index + patternMatch.Length;
        var substring = searchString.Remove(0, lengthValidString);

        return substring;
    }
}


public class ObjectSearchStrategy : ISearchStrategy
{
    private readonly bool _findRecursively;
    private readonly string _patternFormat = @"(?<=^{0})[^\@\[\.][\w\-
   \(\)\*]*";

    private readonly List<INode> _result = new();

    public string Pattern { get; }

    public ObjectSearchStrategy(string pattern, bool findRecursively)
    {
        Pattern = string.Format(_patternFormat, pattern);
        _findRecursively = findRecursively;
    }

    public List<INode> Search(INode node, string nodeName)
    {
        if (nodeName == "*")
        {
            nodeName = "";
        }

        if (nodeName == null)
        {
            _result.Clear();
            return _result;
        }
```

```
        else
        {
            if (_findRecursively)
            {
                _result.Clear();
                RecursiveSearch(node, nodeName);
                return _result;
            }
            else
            {
                return DirectSearch(node, nodeName);
            }
        }
    }

    private List<INode> DirectSearch(INode node, string nodeName)
    {
        var result = new List<INode>();

        var regex = RegexHelper.ConvertToRegex(nodeName,
   RegexOptions.IgnoreCase);

        foreach (var element in node.NextNodeList)
        {
            var matches = regex.Matches(element.Name);

            if (matches.Count > 0)
            {
                result.Add(element);
            }
        }

        return result;
    }

    private void RecursiveSearch(INode node, string nodeName)
    {
        var regex = RegexHelper.ConvertToRegex(nodeName,
   RegexOptions.IgnoreCase);

        foreach (var element in node.NextNodeList)
        {
            var matches = regex.Matches(element.Name);

            if (matches.Count > 0)
            {
                _result.Add(element);
            }

            RecursiveSearch(element, nodeName);
        }
    }
}

public class ComponentSearchStrategy : ISearchStrategy
{
    private readonly bool _findRecursively;
    private readonly List<INode> _result = new();
    private readonly string _patternFormat = @"(?<=^{0})[^\@\[\.][\w\-\*]*";
    public string Pattern { get; }

    public ComponentSearchStrategy(string pattern, bool findRecursively)
```

```
{
    Pattern = string.Format(_patternFormat, pattern);
    _findRecursively = findRecursively;
}


public List<INode> Search(INode node, string componentNodeName)
{
    if (componentNodeName == "*")
    {
        componentNodeName = "";
    }

    if (componentNodeName == null)
    {
        _result.Clear();
        return _result;
    }
    else
    {
        if (_findRecursively)
        {
            _result.Clear();
            RecursiveSearch(node, componentNodeName);
            return _result;
        }
        else
        {
            _result.Clear();
            SearchComponentNode(node, componentNodeName);
            return _result;
        }
    }
}

private void RecursiveSearch(INode node, string componentNodeName)
{
    SearchComponentNode(node, componentNodeName);

    foreach (var element in node.NextNodeList)
    {
        RecursiveSearch(element, componentNodeName);
    }
}

private void SearchComponentNode(INode node, string componentNodeName)
{
    if (node.Entity == null) return;

    var regex = new Regex(componentNodeName, RegexOptions.IgnoreCase);

    var gameObject = node.Entity.GameObject();
    var components = gameObject.GetComponents<Component>();

    foreach (var element in components)
    {
        var name = element.GetType().Name;
        var path = node.Path + "/@" + name;
        var matches = regex.Matches(name);

        if (matches.Count > 0)
        {
```

```
                var componentNode = new ComponentNode(name, path, element,
  node);
                _result.Add(componentNode);
            }
        }
    }
}

public abstract class SearchStrategyConfig : ScriptableObject
{
    public string Name { get; }
    public string Pattern { get; }

    public abstract ISearchStrategy GetStrategy();
    public abstract string GetPattern();
}

public class ObjectSearchStrategyConfig : SearchStrategyConfig
{
    public string Name;
    public string Pattern;
    public bool FindRecursively;

    public override ISearchStrategy GetStrategy()
    {
        return new ObjectSearchStrategy(Pattern, FindRecursively);
    }

    public override string GetPattern()
    {
        var strategy = new ObjectSearchStrategy(Pattern, FindRecursively);
        return strategy.Pattern;
    }
}

public class ComponentSearchStrategyConfig : SearchStrategyConfig
{
    public string Name;
    public string Pattern;
    public bool FindRecursively;

    public override ISearchStrategy GetStrategy()
    {
        return new ComponentSearchStrategy(Pattern, FindRecursively);
    }

    public override string GetPattern()
    {
        var strategy = new ComponentSearchStrategy(Pattern, FindRecursively);
        return strategy.Pattern;
    }
}

public interface IPredicateStrategy
{
    public string Pattern { get; }

    public List<INode> Search(List<INode> nodes, string searchString);
    public Match GetMatch(string searchString);
    public string CutMatchFromSearchString(string searchString);
}
```

```
public class SquareBrackets : IPredicateStrategy
{
   private const string PatternConst = @"(?<=^\[)(.*?)(?=\])";
   public virtual string Pattern { get; } = @"(?<=^\[)({0})(?=\])";

   public virtual List<INode> Search(List<INode> nodes, string searchString)
   {
       throw new System.NotImplementedException();
   }

   public virtual Match GetMatch(string searchString)
   {
       return RegexHelper.GetFirstMatch(PatternConst, searchString);
   }

   public virtual string CutMatchFromSearchString(string searchString)
   {
       var patternMatch = GetMatch(searchString);
       var lengthValidString = patternMatch.Index + patternMatch.Length + 1;
       var substring = searchString.Remove(0, lengthValidString);

       return substring;
   }
}

public class PredicateComponentName : SquareBrackets
   {
       public override string Pattern { get; } = "@";

       public override List<INode> Search(List<INode> nodes, string
  searchString)
       {
           ISearchStrategy componentStrategy = new
  ComponentSearchStrategy(Pattern, false);

           var resultNodes = new List<INode>();
           var squareBracketsMatch = base.GetMatch(searchString);
           var patternMatch =
  componentStrategy.GetMatch(squareBracketsMatch.Value);

           if (patternMatch == null) return new List<INode>();

           foreach (var node in nodes)
           {
               var foundComponent = componentStrategy.Search(node,
  patternMatch.Value);

               if (foundComponent.Count > 0) resultNodes.Add(node);
           }

           return resultNodes.Count > 0 ? resultNodes : new List<INode>();
       }
   }
}

public class PredicateLast : SquareBrackets     // last()-number strategy
{
   public override string Pattern { get; } = @"^last\(\)";

   public override List<INode> Search(List<INode> nodes, string searchString)
   {
       var squareBracketsMatch = base.GetMatch(searchString);
```

```
        var patternMatch = RegexHelper.GetFirstMatch(Pattern,
    squareBracketsMatch.Value);

        if (patternMatch == null) return new List<INode>();

        var lastIndex = nodes.Count - 1; // lastElement = nodes[count - 1]

        searchString = squareBracketsMatch.Value;
        searchString = searchString.Replace(patternMatch.Value,
    lastIndex.ToString());

        var tmpIndex = OperatorStrategy.CalculateString(searchString) ??
    lastIndex;

        var searchIndex = Convert.ToInt32(tmpIndex);

        if ((searchIndex < 0) || (searchIndex >= nodes.Count)) return new
    List<INode>();

        return new List<INode>() { nodes[searchIndex] };
    }
}

public class PredicateOrdinalNumber : SquareBrackets
{
    public override string Pattern { get; } = @"^[0-9]+$";

    public override List<INode> Search(List<INode> nodes, string searchString)
    {
        var squareBracketsMatch = base.GetMatch(searchString);
        var patternMatch = RegexHelper.GetFirstMatch(Pattern,
    squareBracketsMatch.Value);

        if (patternMatch == null) return new List<INode>();

        var index = Convert.ToInt32(patternMatch.Value) - 1; // 1 = nodes[0]

        if ((index < 0) || (index >= nodes.Count)) return new List<INode>();

        return new List<INode>() { nodes[index] };
    }
}

public class PredicatePosition : SquareBrackets
{
    public override string Pattern { get; } = @"^position\(\)" + @" {0,1}"
                                                               +
    OperatorStrategy.OperatorPattern
                                                               + @" {0,1}"
                                                               +
    OperatorStrategy.SecondNumberPattern;
    public override List<INode> Search(List<INode> nodes, string searchString)
    {
        var squareBracketsMatch = base.GetMatch(searchString);
        var patternMatch = RegexHelper.GetFirstMatch(Pattern,
    squareBracketsMatch.Value);

        if (patternMatch == null) return new List<INode>();

        searchString = squareBracketsMatch.Value;

        var logicOperator = OperatorStrategy.GetCompareOperator(searchString);
```

```
        var secondNumber = OperatorStrategy.GetSecondNumber(searchString);
        var positionIndex = Convert.ToInt32(secondNumber);

        if ((positionIndex < 0) || (positionIndex > nodes.Count)) return new
    List<INode>(); //test

        switch (logicOperator)
        {
            case ">":
                nodes.RemoveRange(0, positionIndex);
                return nodes;
            case ">=":
                nodes.RemoveRange(0, positionIndex - 1);
                return nodes;
            case "<":
                nodes.RemoveRange(positionIndex - 1, nodes.Count -
    (positionIndex - 1));
                return nodes;
            case "<=":
                nodes.RemoveRange(positionIndex, nodes.Count - (positionIndex));
                return nodes;
            case "==":
                return new List<INode>(){ nodes[positionIndex - 1] };
            case "!=":
                nodes.RemoveAt(positionIndex - 1);
                return nodes;
            default:
                return new List<INode>();
        }
    }
}

public class SquareBracketsPredicateService
{
    private readonly List<IPredicateStrategy> _predicateStrategies = new
    List<IPredicateStrategy>()
    {
        new PredicateOrdinalNumber(),
        new PredicateComponentName(),
        new PredicateLast(),
        new PredicatePosition()
    };

    public List<IPredicateStrategy> GetStrategies()
    {
        return _predicateStrategies;
    }
}


       public class JsonTestData
       {
          public JsonTestData() { }

          private readonly string _fileName =
    "Assets/Resources/Tests/FirstTest.json";

          public void SaveTest(TestFile testFile)
          {
              string jsonString = JsonUtility.ToJson(testFile);
              File.WriteAllText(_fileName, jsonString);
          }
```

```
        public TestFile LoadTest()
        {
            string jsonString = File.ReadAllText(_fileName);
            var testFile = JsonUtility.FromJson<TestFile>(jsonString);
            return testFile;
        }
    }

    [Serializable]
    public class TestFile
    {
        public string FileName;
        public List<string> Content;
}

public static class OperatorStrategy
{
   public const string FirstNumberPattern = @"^[0-9]*";
   public const string SecondNumberPattern = @"[0-9]+$";
   public const string OperatorPattern = @"[\+\-\/\*\!\<\>\=]{1,2}";
   public const string Pattern = FirstNumberPattern + @" {0,1}" +
  OperatorPattern + @" {0,1}" + SecondNumberPattern;

   public static double? CalculateString(string searchString)
   {
       var expressionMatch = GetMatch(searchString);
       if (expressionMatch == null) return null;

       var result = Convert.ToDouble(new
  DataTable().Compute(expressionMatch.Value, ""));

       return result;
   }

   public static bool? CompareString(string searchString)
   {
       var expressionMatch = GetMatch(searchString);
       if (expressionMatch == null) return null;

       var firstNumberMatch = RegexHelper.GetFirstMatch(FirstNumberPattern,
  expressionMatch.Value);
       if (firstNumberMatch == null) return null;

       var secondNumberMatch = RegexHelper.GetFirstMatch(SecondNumberPattern,
  expressionMatch.Value);
       if (secondNumberMatch == null) return null;

       var operatorMatch = RegexHelper.GetFirstMatch(OperatorPattern,
  expressionMatch.Value);
       if (operatorMatch == null) return null;

       var firstNumber = Convert.ToDouble(firstNumberMatch.Value);
       var secondNumber = Convert.ToDouble(secondNumberMatch.Value);
       var operatorValue = operatorMatch.Value;

       var result = Operator(operatorValue, firstNumber, secondNumber);

       return result;
   }

   public static double? GetFirstNumber(string searchString)
   {
```

```
        var firstNumberMatch = RegexHelper.GetFirstMatch(FirstNumberPattern,
    searchString);
        if (firstNumberMatch == null) return null;

        return Convert.ToDouble(firstNumberMatch.Value);
    }

    public static double? GetSecondNumber(string searchString)
    {
        var secondNumberMatch = RegexHelper.GetFirstMatch(SecondNumberPattern,
    searchString);
        if (secondNumberMatch == null) return null;

        return Convert.ToDouble(secondNumberMatch.Value);
    }

    public static string GetCompareOperator(string searchString)
    {
        var operatorMatch = RegexHelper.GetFirstMatch(OperatorPattern,
    searchString);
        if (operatorMatch == null) return null;

        return operatorMatch.Value;
    }

    private static Match GetMatch(string searchString)
    {
        var logicOperator = RegexHelper.GetFirstMatch(Pattern, searchString);
        return logicOperator;
    }

    private static bool Operator(string logic, double x, double y)
    {
        return logic switch
        {
            ">" => x > y,
            "<" => x < y,
            "==" => x.Equals(y),
            "!=" => !x.Equals(y),
            ">=" => x >= y,
            "<=" => x <= y,
            _ => throw new Exception("Invalid compare operator!")
        };
    }
}

public static class RegexHelper
{
    private static readonly string[] ReplacedToken = new[]
    {
        "(",
        ")"
    };

    public static Match GetFirstMatch(string pattern, string searchString)
    {
        var regex = new Regex(pattern, RegexOptions.IgnoreCase);
        var matches = regex.Matches(searchString);

        return matches.Count == 0 ? null : matches.First();
    }
```

```
    public static MatchCollection GetAllMatch(string pattern, string
   searchString)
    {
        var regex = new Regex(pattern, RegexOptions.IgnoreCase);
        var matches = regex.Matches(searchString);

        return matches.Count == 0 ? null : matches;
    }

    public static Regex ConvertToRegex(string nodeName, RegexOptions
   regexOptions)
    {
        foreach (var token in ReplacedToken)
        {
            nodeName = nodeName.Replace(token, "\\" + token);
        }

        return new Regex(nodeName, regexOptions);
    }
}

public class DotCall
{
    private const string CallNamePattern = @"(?<=^\.)[\w]*";
    public string Pattern { get; } = @"\.+[\w]*\(\)+$";

    public List<INode> Search(List<INode> nodes, string searchString)
    {
        var patternMatch = RegexHelper.GetFirstMatch(Pattern, searchString);

        if (patternMatch == null) return new List<INode>();

        var callNamePattern = RegexHelper.GetFirstMatch(CallNamePattern,
   patternMatch.Value);

        //GetPublicMethodAndProperty(nodes);
        InvokeMethod(nodes, callNamePattern.Value);

        return nodes;
    }

    private void InvokeEvent(List<INode> nodes, string methodName)
    {
        foreach (var node in nodes)
        {
            var myType = node.Entity.GetType();
            var events = myType.GetEvents();
            var member = myType.GetMembers();
            var methods = myType.GetMethods();
            var method0 = myType.GetMember("get_onClick");
            var method1 = myType.GetMethod(methodName);
            EventInfo eventInfo= myType.GetEvent(methodName);


            var method = eventInfo.GetRaiseMethod();
            method?.Invoke(node.Entity, parameters: null);

            Debug.Log("");
        }
    }

    private void InvokeMethod(List<INode> nodes, string methodName)
```

```
    {
        foreach (var node in nodes)
        {
            var myType = node.Entity.GetType();
            var events = myType.GetEvents();
            var member = myType.GetMembers();
            var method = myType.GetMethod(methodName);
            Button b = node.Entity.GetComponent<Button>();
            b.onClick.Invoke();


            //c.HandleEvent();
            /*method?.Invoke(node.Entity, parameters: null);
            Debug.Log("");*/

        }
    }

    private List<MethodInfo> GetPublicMethodAndProperty(List<INode> nodes)
    {
        var methods = new List<MethodInfo>();

        foreach (var node in nodes)
        {
            var myType = node.Entity.GetType();
            methods.AddRange(myType.GetMethods());
        }

        if (methods.Count == 0) return new List<MethodInfo>();

        var uniqueMethods = methods.Distinct().ToList();

        return uniqueMethods;
    }
}
```

Листинг Б2. Исходный код графического интерфейса.

```
public class XPathSearchWindow : EditorWindow
{
    private List<INode> _resultSearch = new ();
    private List<string> _commandsTest = new ();

    private const float MenuLabelHeight = 20f;
    private const float ListViewItemHeight = 16f;
    private const float MenuListViewHeight = 160f;

    [MenuItem("Window/XPathSearchWindow")]
    public static void OpenDemoManual()
    {
        GetWindow<XPathSearchWindow>().Show();
    }

    public void Update()
    {
        Repaint();
    }

    public void CreateGUI()
    {
        var spaceLabel = new Label();
```

```
    //SEARCH BOX
    var searchBox = new Box();
    var searchLabel = new Label("Search");
    searchLabel.style.unityFontStyleAndWeight = new
StyleEnum<FontStyle>(FontStyle.Bold);
    searchLabel.style.height = MenuLabelHeight;

    var searchField = new TextField();
    var searchButton = new Button
    {
        text = "Search",
        style =
        {
            color = Color.black,
            backgroundColor = Color.green
        }
    };

    var addCommandButton = new Button
    {
        text = "Add Command",
        style =
        {
            width = 150f,
            color = Color.white,
            backgroundColor = Color.gray
        }
    };

    searchBox.Add(searchLabel);
    searchBox.Add(searchField);
    searchBox.Add(searchButton);
    searchBox.Add(addCommandButton);
    rootVisualElement.Add(searchBox);

    //RESULT SEARCH BOX
    var resultSearchBox = new Box
    {
        style =
        {
            height = 16f+ MenuLabelHeight + MenuListViewHeight
        }
    };


    var resultSearchLabel = new Label("Result List")
    {
        style =
        {
            unityFontStyleAndWeight = new
StyleEnum<FontStyle>(FontStyle.Bold),
            height = MenuLabelHeight
        }
    };

    var listView = CreateResultList(_resultSearch);

    resultSearchBox.Add(resultSearchLabel);
    resultSearchBox.Add(listView);
    rootVisualElement.Add(resultSearchBox);
```

```
    searchButton.clicked += () =>
    {
        var tree = new Tree();
        var rootNode = tree.GenerateTree();

        var searchEngine = new SearchEngine();
        _resultSearch = searchEngine.Search(rootNode, searchField.value);

        resultSearchBox.Remove(listView);

        listView = CreateResultList(_resultSearch);

        resultSearchBox.Add(listView);
    };

    //TEST BOX
    var testBox = new Box
    {
        style =
        {
            height = MenuLabelHeight + MenuListViewHeight
        }
    };

    var testLabel = new Label("Test")
    {
        style =
        {
            unityFontStyleAndWeight = new
StyleEnum<FontStyle>(FontStyle.Bold),
            height = MenuLabelHeight
        }
    };

    var testView = new ListView(_commandsTest, ListViewItemHeight);
    testView.style.height = MenuListViewHeight;
    testView.selectionType = SelectionType.Multiple;
    testView.makeItem = () => new TextField();
    testView.bindItem = (item, index) => { ((TextField)item).value =
_commandsTest[index]; };

    testBox.Add(spaceLabel);
    testBox.Add(testLabel);
    testBox.Add(testView);
    rootVisualElement.Add(testBox);

    addCommandButton.clicked += () =>
    {
        _commandsTest.Add(searchField.value);
        testView.RefreshItems();
    };

    //TEST MANAGER BOX
    var testManagerBox = new Box
    {
        style =
        {
            height = MenuLabelHeight + MenuListViewHeight
        }
    };
    var checkTestButton = new Button()
    {
```

```
            text = "Check Test",
            style =
            {
                color = Color.black,
                backgroundColor = Color.yellow
            }
        };

        checkTestButton.clicked += () =>
        {
            if (_commandsTest.Count == 0)
            {
                Debug.Log("TEST LOG: Missing commands in the test!");
                return;
            }

            var tree = new Tree();
            var rootNode = tree.GenerateTree();

            for (var i = 0; i < _commandsTest.Count; i++)
            {
                var searchEngine = new SearchEngine();
                _resultSearch = searchEngine.Search(rootNode, _commandsTest[i]);

                if (_resultSearch.Count != 0) continue;

                Debug.Log("TEST LOG: Error occurred while executing the command
" + (i + 1) + " [" + _commandsTest[i] + "]");
                return;
            }
            Debug.Log("TEST LOG: Successful completion of the test!");
        };

        var removeCommandButton = new Button()
        {
            text = "Remove Command",
            style =
            {
                width = 150f
            }
        };
        removeCommandButton.clicked += () =>
        {
            var selectedItems = testView.selectedIndex;
            _commandsTest.RemoveAt(selectedItems);
            testView.RefreshItems();
        };

        var saveTestButton = new Button() { text = "Save Test" };
        saveTestButton.clicked += () =>
        {
            var testData = new JsonTestData();
            var testFile = new TestFile();

            if (_commandsTest.Count > 0)
            {
                testFile.FileName = "Test1";
                testFile.Content = _commandsTest;
                testData.SaveTest(testFile);
                Debug.Log("DATA TEST: Test saved!");
            }
            else
```

```
        {
            Debug.Log("DATA TEST: Test command is empty!");
        }

    };


    var openTestButton = new Button() { text = "Open Test" };

    testManagerBox.Add(removeCommandButton);
    testManagerBox.Add(checkTestButton);
    testManagerBox.Add(saveTestButton);
    testManagerBox.Add(openTestButton);
    rootVisualElement.Add(testManagerBox);
}

private void OnInspectorUpdate()
{
    Repaint();
}

private ListView CreateResultList(List<INode> resultSearch)
{
    var listView = new ListView(resultSearch, ListViewItemHeight)
    {
        selectionType = SelectionType.Multiple,
        makeItem = () => new Label(),
        bindItem = (item, index) => { ((Label)item).text =
resultSearch[index].Name; },
        style =
        {
            height = MenuListViewHeight
        }
    };

    listView.onItemsChosen += items =>
    {
        var firstSelectedItem = items.First();
        var typeItem = firstSelectedItem.GetType().Name;

        if (typeItem == nameof(ObjectNode))
        {
            var element = (ObjectNode) firstSelectedItem;
            EditorGUIUtility.PingObject(element.Entity);
        }
        else if (typeItem == nameof(ComponentNode))
        {
            var element = (ComponentNode) firstSelectedItem;
            EditorGUIUtility.PingObject(element.PrevNode.Entity);
        }
    };

    listView.onSelectionChange += items =>
    {
        var firstSelectedItem = items.First();
        var typeItem = firstSelectedItem.GetType().Name;

        if (typeItem == nameof(ObjectNode))
        {
            var element = (ObjectNode) firstSelectedItem;
            EditorGUIUtility.PingObject(element.Entity);
        }
```

```
            else if (typeItem == nameof(ComponentNode))
            {
                var element = (ComponentNode) firstSelectedItem;
                EditorGUIUtility.PingObject(element.PrevNode.Entity);
            }
        };

        return listView;
    }
}
```

Листинг Б2. Исходный код тестирования программного модуля.

```
public class PluginTest : EditorWindow
{
    [MenuItem("Window/XPath Plugin Tests")]
    public static void OpenDemoManual()
    {
        GetWindow<PluginTest>().Show();
    }

    private List<string> _searchRequestList = new List<string>()
    {
        "window",
        "/window",
        "//button",
        "*/@transform",
        "//@transform",
        "*[1]",
        "*[last()]",
        "*[last() - 2]",
        "*[last()-1]",
        "//@*",
        "//@*[position() == 3]",
        "//@*[position() >= 3]",
        "//@*[position() <= 3]",
        "//@*[position() != 3]",
        "button/@button.Click()"
    };

    public void CreateGUI()
    {
        var startTestsButton = new Button
        {
            text = "Start Tests",
            style =
            {
                color = Color.black,
                backgroundColor = Color.red
            }
        };
        rootVisualElement.Add(startTestsButton);

        startTestsButton.clicked += () =>
        {
            var tree = new Tree();
            var rootNode = tree.GenerateTree();

            var searchEngine = new SearchEngine();
            foreach (var request in _searchRequestList)
```

```
            {
                var resultSearch = searchEngine.Search(rootNode, request);

                if (resultSearch.Count > 0)
                {
                    Debug.Log("TEST COMPLETED: " + request);
                }
                else
                {
                    Debug.Log("TEST ERROR: " + request);
                }
            }
        };
    }
}
```