Казанский (Приволжский) Федеральный Университет
Высшая Школа Информационных Технологий и
Интеллектуальных Систем

Выпускная квалификационная работа

РАЗРАБОТКА ИНСТРУМЕНТА АВТОМАТИЗАЦИИ ДЕЙСТВИЙ ДЛЯ ИГРОВОГО ДВИЖКА UNITY

Выполнил: студент гр. 11-605
О.А. Бедрин

Казань 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка проблемы

Цель и задачи

Дополнительные задачи

Дополнительное требование при реализации данного проекта – возможность сериализовывать, экспортировать и импортировать записанные данные для последующего использования в построении сценария для тренажеров в виртуальной реальности [**?**]. Такая форма записи действий пользователя в виртуальной среде может быть использована для генерации сценария обучающего тренажера [**?**], что поможет избежать рутинной работы по формированию треков обучения [**?**].

Связанные работы

Задача автоматизации функционального тестирования сегодня является довольно тривиальной, так как во многих популярных платформах, таких как Web, iOS, Android, она уже решена. Уже существуют разные инструменты для приложений с различными возможностями и характеристиками, такие как Selenium, Appium, Robotium и UI Automator. Формы решения данной задачи существуют разные, например, для Web используется WebDriver для записи и имитации ввода [**?**], для iOS и Android используется система RPC вызовов [**?**], если же нет возможности вмешаться в поток вывода для облегчения записи, то используется компьютерное зрение для распознования как и с чем взаимодействует пользователь [**?**].

У каждой из формы есть свои плюсы и минусы, однако их всех объединяет одна модель поведения: запись действий пользователя и их проигрывание таким образом, чтобы субъект тестирования не различал реальный ли ввод или же записанный. Особняком здесь стоит вопрос об автоматизации тестирования приложений на игровом движке *Unity*. Каждая из вышеупомянутых форм решения задачи в этом случае проигрывает в эффективности и степенью удобства интеграции, так как зачастую иг-

ры насыщены графикой и, в общем случае, не все игры вообще имеют графический интерфейс. В этой связи предлагается новая форма решения задачи, основанная на внутренней логике игрового движка *Unity*, а именно на возможности декорирования методов взятия ввода, обусловленной внутренней архитектурой системы ввода.

Объект и предмет разработки

Модель уровня определения требований

Модель уровня анализа требований

Модель уровня реализации требований

1 Проектирование

1.0.1 Класс совместимых приложений

Описанный в данной работе набор ассетов Automated Test Framework (ATF) предназначен для автоматизации действий с целью функционального тестирования. Его можно интегрировать с любым *Unity*-проектом, система ввода которого построена вокруг класса из стандартной библиотеки *Unity* по взаимодействию с вводом Input. Под это описание подходит большая часть *Unity*-приложений и в этом выражена универсальность предложенного решения. Ранее среди свободных для использования ассетов не было представлено решений для автоматизации ни success-тестов, ни тестирования главного потока сценария использования (use-case), ни других подходов для функционального тестирования приложений, кроме как через механизмы стандартного пакета unit-тестирования *Unity*. Однако как бы ни был хорош подход использования инструментария unit-тестирования, для того чтобы покрыть все возможные сценарии действий внутри проекта, пришлось бы либо писать свой модуль тестов под каждый из аспектов, либо же на его основе реализовывать сложную универсальную систему.

1.0.2 Определение теоретической базы

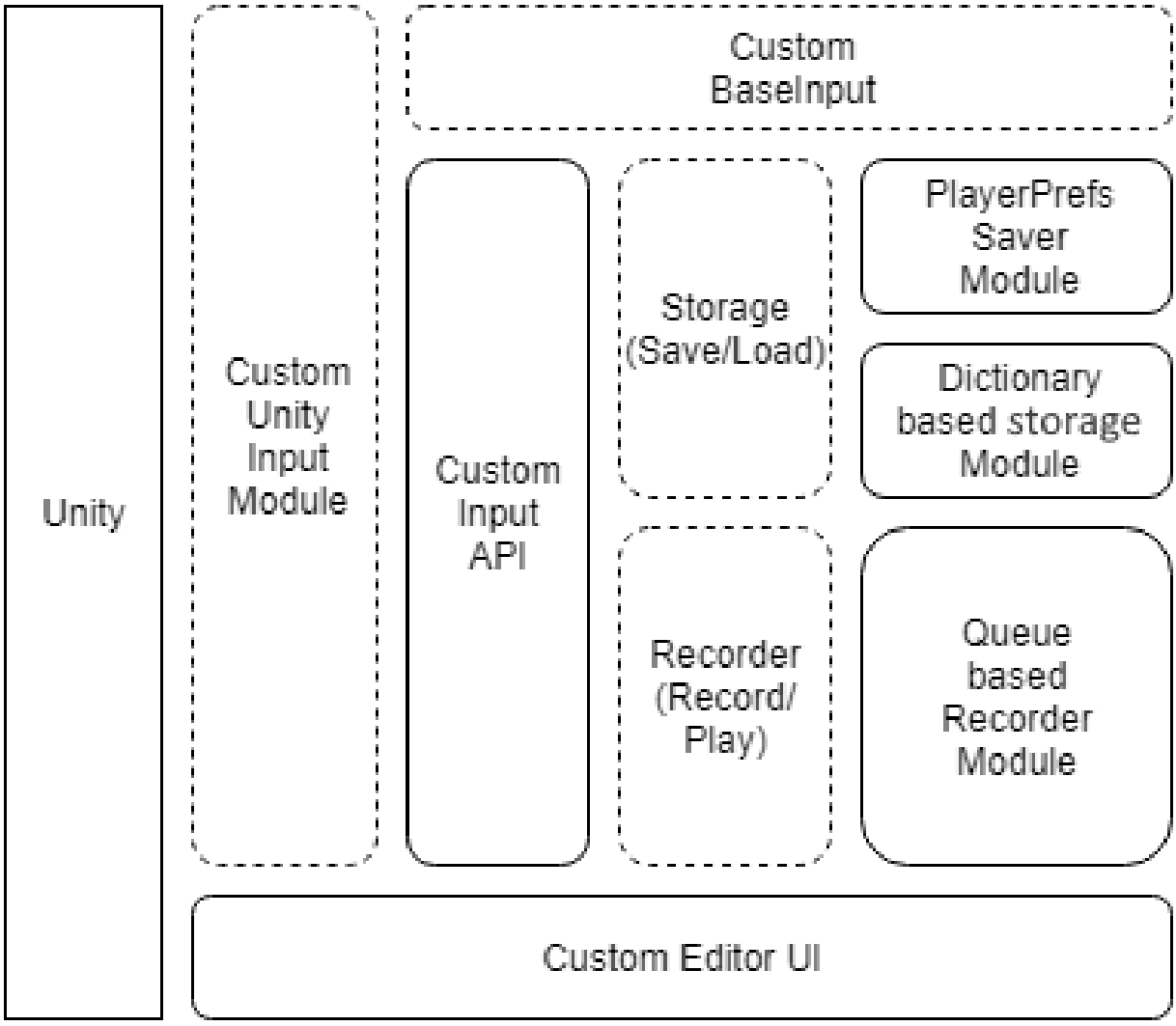

Рисунок 1.1 — Диаграмма платформы решения

На этапе проектирования была составлена диаграмма платформы (см. рис. 1.1), каждый блок которой – это изолированная группа методов API, решающая свои задачи:

— *Unity* – непосредственно сам игровой движок;

— *Custom Unity Input Module* – абстракция, объединяющая управление вводом;

— *Custom Input API* – собственно API, который вызывает нативные методы по запросу ввода;

— *Custom BaseInput* – сущность, которяа является реализацией объекта обработки потока данных через мост (Bridge), объединяя статичные методы по перехвату/симуляции ввода и обернутые события (Events);

— *Storage* – абстракция, отвечающая за функционал хранения и манипуляции записанных действий;

— *Recorder* – абстракция, отвечающая за запись действий;

— *Custom Editor UI* – система пользовательских окон для управления всеми процессами;

— *PlayerPrefs Save/Load Module* – система реализации абстракции модуля по сохранению/загрузке записанных действий на базе стандартного класса PlayerPrefs;

— *Dictionary based Module* – реализация абстракции хранилища записанных действий, основанная на структуре данных “Словарь”;

— *Queue based Recorder Module* – реализация абстракции, отвечающей за запись действий, основанная на структуре данных “Очередь”.

Для выполнения поставленных задач было разработано решение, являющееся, по своей сути, модифицированным адаптером, который перехватывает и симулирует ввод. Для его эффективной реализации стало органичным использовать несколько архитектурных паттернов:

— *Interceptor* – шаблон для перехвата и подмены входных данных с периферийных устройств [**?**];

— *Broker* – шаблон для интеграции и взаимодействия с встроенной системой управления входными данными *Unity* [**?**];

— *PAC (Presentation–abstraction–control)* – шаблон для организации взаимодействия зависимых систем [**?**].

Для подмены стандартного класса Input был создан перехватчик ATFInput (см. рис. 1.2), который наследуется от стандартного класса BaseInput для использования встроенной пользовательской системы управления вводом. ATF – это сокращение от английского Automated Test Framework, которое переводиться как: фреймворк автоматизированного тестирования. Так как класс Input содержит в себе только статичные методы, декорация их для перехвата внутри ATFInput позволила совместить и классический перехват ввода и облегчить в дальнейшем перехват событий ввода. Иначе говоря, было проведено совмещение перехватчика для

класса Input и класса BaseInput для взаимодействия с событиями ввода внутри *Unity*.

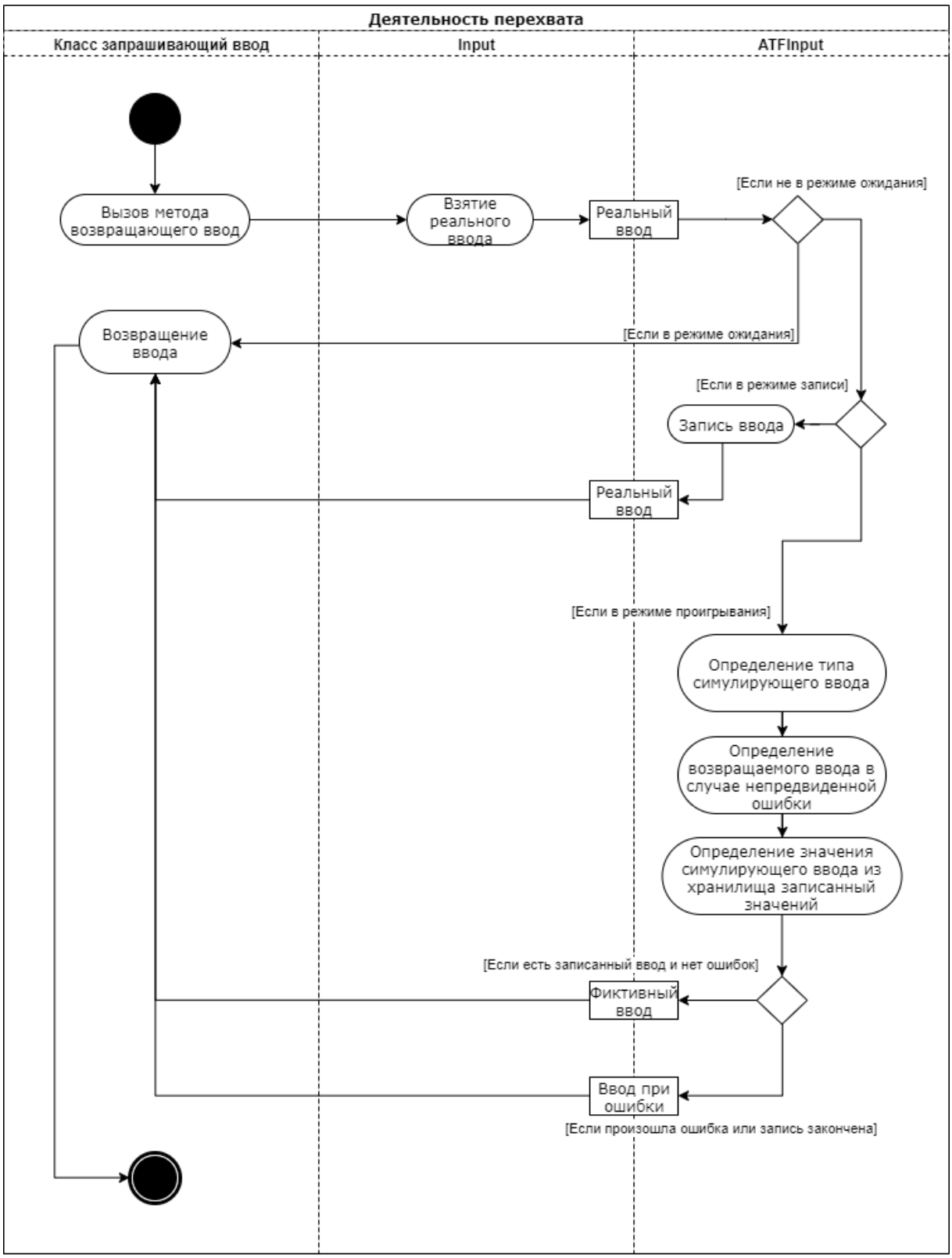

Рисунок 1.2 – Диаграмма деятельности ATFInput

Для интерпретации (проигрывания) первоначально использовался паттерн Interpreter с терминалами ожидания внутри Coroutine. Однако после попытки тестовой реализации данного шаблона был произведен отказ от этого шаблона в пользу метода записи, основанного на структуре данных “Очередь”, так как для правильной работы терминалов ожидания необходимо было просчитывать заранее действия пользователя, что затруднительно.

2 Ход разработки

2.1 Реализация вспомогательных систем

2.1.1 Реализация синглтона для объектов сцены

2.1.2 Реализация DI контейнера для объектов сцены

2.2 Создание базовой системы интеграции

2.2.1 Реализация инициализатора

2.2.2 Реализация адаптера для класса Input

2.2.3 Реализация расширенного модуля ввода

2.3 Реализация основных систем

2.3.1 Реализация системы записи и проигрывания

2.3.2 Реализация системы хранилища действий

2.3.3 Реализация системы сохранения и загрузки хранилища действий

2.4 Реализация окон управления

2.4.1 Создание окна управления записью и проигрыванием

2.4.2 Создание окна управления хранилища действий

3 Результаты

ЗАКЛЮЧЕНИЕ

Текст заключения

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Беркутов, А.М.* Системы комплекстной электромагнитотерапии / А.М. Беркутов. — М.: Бином, 2000.

2. *И.Е. Золотухина, В.С. Улащик*. Основы импульсной магнитотерапии / В.С. Улащик И.Е. Золотухина. — Витебск: Витебская областная типография, 2008.

3. *Улащик, В.С.* Физиотерапия. Универсальная медицинская энциклопедия / В.С. Улащик. — Минск: Книжный дом, 2008.

4. *С.А. Гуляр, Ю.П. Лиманский*. Постоянные магнитные поля и их применение в медицине / Ю.П. Лиманский С.А. Гуляр. — Киев: Ин-т физиол. им. А.А. Богомольца НАН Украины, 2006.

Приложение ПРИЛОЖЕНИЕ

Приложение А Конкретная система