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

В жизненном цикле разработки каждого программного обеспечения (ПО) присутствует этап приемо-сдаточных работ. Перед тем как передавать ответственному за этот этап уставному органу проекта собранное ПО необходимо провести один из завершающих этапов разработки, а именно регрессионное тестирование. Алгоритм проведения данного вида тестирования технически довольно прост: создаются сценарии пользовательского поведения в виде диаграммы вариантов использования, далее выбирается формальный исполнитель, который шаг за шагом выполняет действия описанные в главных, а затем альтернативных потоках вариантов использования.

Вопрос выбора формального исполнителя здесь стоит довольно остро, так как это напрямую влияет на стоимость разработки. Для разных групп используемых технологий разработки данная проблема решается по-разному. Например, для популярных платформ таких как PC, Web, iOS, Android уже существуют разные автоматические формальные исполнители для приложений с различными возможностями и характеристиками, для однократной записи и множественного проигрывания действий потоков. Ими, например, являются Selenium, Appium, Robotium и UI Automator. Формы решения данного вопроса выбора исполнителя существуют разные, например, для Web используется WebDriver для записи и имитации ввода [1], для iOS и Android используется система RPC вызовов [2], если же нет возможности вмешаться в поток ввода для облегчения записи, то используется компьютерное зрение для распознования как и с чем взаимодействует пользователь [3]. Метод зачастую довольной эффективный для графических систем с конечным набором однотипных элементов (напр. класс систем Windows, Icons, Menus, Pointers (WIMP) [4]), однако гораздо менее точный при большом разнообразии динамических графических элементов. Также, вышеописанные исполнители для данных платформ ограничены устройствами ввода, а именно клавиатурой и мышкой.

Особняком здесь стоит задача о выборе формального исполнителя для тестирования игровых приложений на популярной платформе для разработки приложений визуализации и игр *Unity* [5]. Каждая из вышеупомянутых форм решения вопроса в этом случае проигрывает в эффективности и степени удобства интеграции инструмента записи и проигрывания действий, так как зачастую игры насыщены динамическими графическими элементами, а некоторые игры вообще не имеют графический интерфейс. Также, в отличии от других платформ, *Unity* - не ограничена устройствами ввода. Поэтому часто формальным исполнителем здесь становится группа специалистов или других людей которые играют роль тестировщиков. Им, частично или полностью, выдаётся информация о действиях внутри потоков и они их исполняют, составляя в заключении необходимые отчёты. В данном случае стоимость разработки также повышается в виду создания и поддержки системы контроля работы тестировщиков.

Альтернативный подход к регрессионному тестированию на платформе *Unity* является автоматизированное модульное тестирование. Однако при использовании данного подхода необходимо для каждого основного потока писать комплекс из юнит-тестов (алгоритмов тестирования описанных в коде ПО и исполняемых в контексте с ним средствами выбранной платформы). Время написания подобного комплекса примерно равно объёму времени этапа основной разработки при этом появляется необходимость поддержки в дополнении к основной кодовой базе ещё и юнит-тестов со всеми вытекающими издержками.

В этой связи предлагается форма решения задачи регрессионного тестирования для игровых приложений платформы *Unity* в виде импортируемой группы ассетов (исходных файлов кода и предзаготовленных графических и модульных инструментов), основанная на вышеописанных подходах для платформ PC, Web и др., и на возможности декорирования методов взятия ввода, обусловленной внутренней архитектурой платформы. Это позволяет избежать регулярных потерь времени при человеческом тестировании засчёт однократной записи ввода не только с клавиатуры и компьютерной мыши, но и любого другого периферийного

устройства, и дальнейшего проигрыша записанного ввода по необходимости регрессионного тестирования.

Задачи разработки

В настоящей выпускной квалификационной работе рассматриваются задачи связанные напрямую с исследованием распространённости описанной выше проблематики, а также задачи цикла разработки данных ассетов на платформе *Unity*.

В первой главе рассматриваются исследование среднего количества использований внутреннего инструмента взятия ввода платформы *Unity* среди самых значимых и популярных проектов на данной платформе. Происходит анализ класса совместимых приложений с описанными в данной работе ассетами, а также определение теоретической базы приёмов программной инженерии, которые были использованы в рамках данной работы.

Во второй главе описан процесс разработки описываемой группы ассетов, состоящий из трёх подпроцессов: реализация вспомогательных систем для управления зависимостями, создание базовой системы интеграции ассетов и реализация основных систем, позволяющих осуществить запись и проигрывание, хранение, загрузку, экспорт и импорт действий.

Дополнительная задача при разработке данной группы ассетов описанная в конце второй главы – возможность сериализовывать, экспортировать и импортировать записанные данные для последующего использования. Например для построения сценария для тренажеров в виртуальной реальности [6] или для генерации сценария обучающего тренажера [7], что поможет избежать рутинной работы по формированию треков обучения [8].

Третья глава посвящена циклу разработки окон управления группой ассетов после интеграции в сторонний проект, а также разработке и развёртыванию онлайн-документации, в которой описывается процесс использования окон управления, а также принципы дополнения и расши-

рения исходной кодовой базы описываемых ассетов для всех желающих сторонних разработчиков.

Объект и предмет разработки

Объектом разработки является процесс автоматизации регрессионного тестирования для приложений с богатой или отсутствующей динамической графикой и пользовательским интерфейсом на платформе *Unity*. В качестве предмета разработки в данной работе описывется группа ассетов предназначенных для автоматизации действий пользователя.

Иными словами, результатом представлен набор дополнительных окон и утилит для редактора *Unity* для записи и проигрывания сценариев, автоматической интеграции группы ассетов в готовую кодовую базу стороннего проекта, а также для сохранения и загрузки записанных сценариев представляющих из себя входные данные со всех периферийных устройств, что были активны во время работы приложения, в которое группа ассетов была импортирована.

Также дополнением к результату является документация к пользованию разработанными ассетами, а также их расширению для нужд сторонних разработчиков.

1 Проектирование

1.1 Исследование распространённости использования класса Input

Имея перед собой цель создания ассетов способных автоматизировать действия разрабатываемого приложения, было принято решение о том, что в качестве основного способа запрашивания ввода необходимо было взять класс Input из стандартной библиотеки платформы *Unity* как наиболее лёгкий для использования.

Критерием лёгкости использования способов запрашивания ввода здесь является минимальное количество необходимых единиц компиляции (ЕК) и количество использованных строк кода (ИСК).

Таблица 1.1 — Количество ЕК и ИСК на способ запрашивания ввода

Способ запрашивания ввода	Количество	
	ЕК	ИСК
Класс Input	10	1
Интерфейсы маркеры	10	1
Input System	10	1

Утверждение: Input - популярен, т.е Input - распространён, распространённость = мат ожидание ген совокупности больше 0 ген совокупность: множество чисел означающих количество использований класса Input в кодовой базе проекта который либо самый релевантный по запросу (проект на Unity), имеет больше всего звёзд, имеет больше всего форков, самый живой - причиной именно такого выбора фильтров поиска является тот факт, что по-сути в выбранных проектах используются самые распространённые и удобные практики, т.е по 1 ому такому проекту можно судить, что глядя на него другие разрабы тоже поступают также, ведь выбранный проект весьма жив и успешен. источник данных GitHub

Выбран интервальный критерий бутстрапа с параметрами: уровень значимости 0.1, количество повторных выборок 99509 - равный количе-

ству всех найденных проектов на юнити на гитхабе, длина выборки 3970 = максимальное количество данных которых удалось извлечь

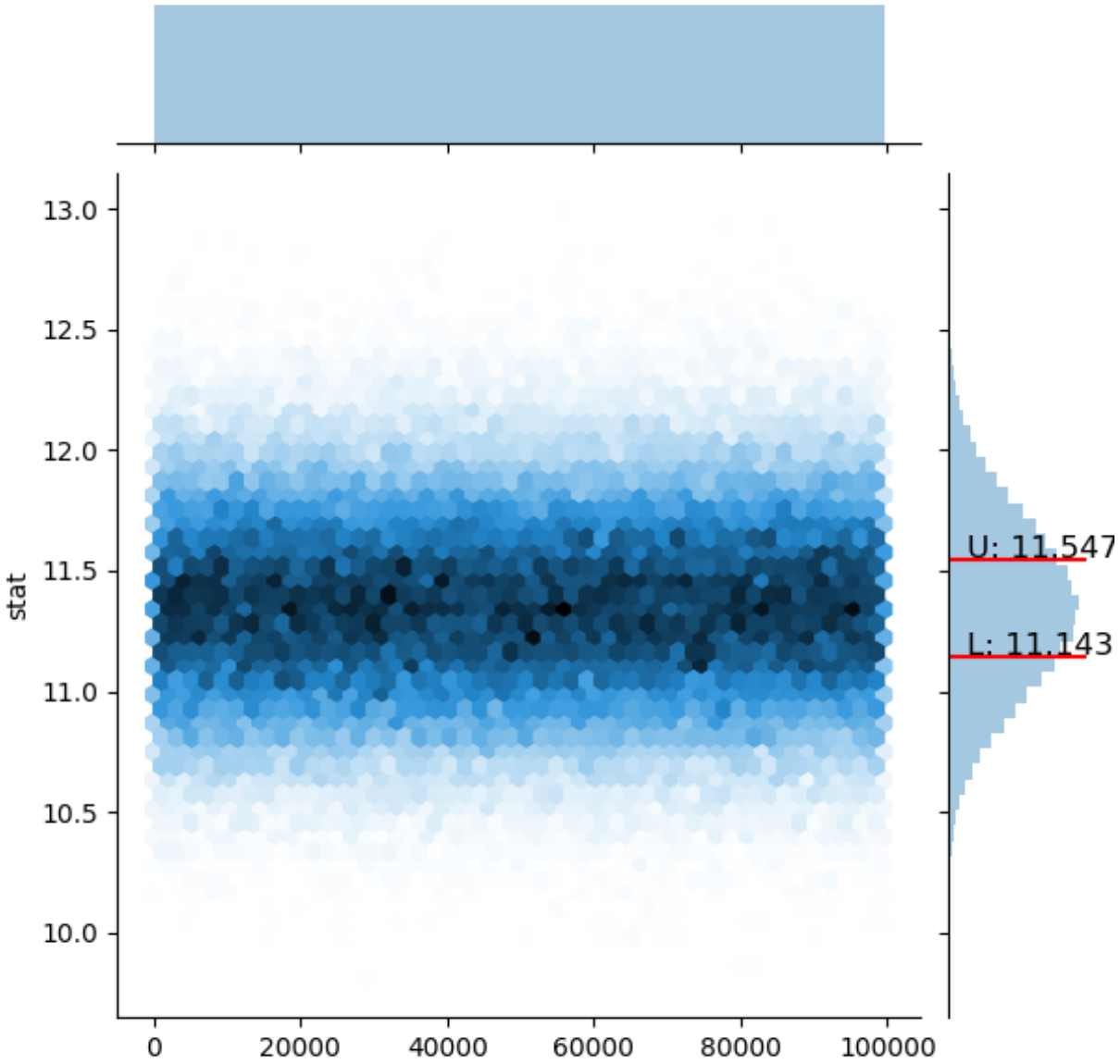

Рисунок 1.1 — Диаграмма рассеивания бутстраповской интервальной оценки

1.2 Класс совместимых приложений

Разработанная в рамках данной работы группа ассетов имеет название Automated Test Framework (ATF) и может быть интегрирована 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 выраже-

на универсальность предложенного решения. Ранее среди свободных для использования ассетов не было представлено решений для автоматизации ни success-тестов, ни тестирования главного потока сценария использования, 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.

1.3 Определение теоретической базы

— создание расширяемой базы кода, которая бы отвечала всем требованиям принципов SOLID [9], DRY [10] и принципа бритвы Оккама [11],

— компоновка решения в множество ассетов для легкой переносимости результирующего продукта,

— а также обеспечение интеграции в код уже написанных продуктов.

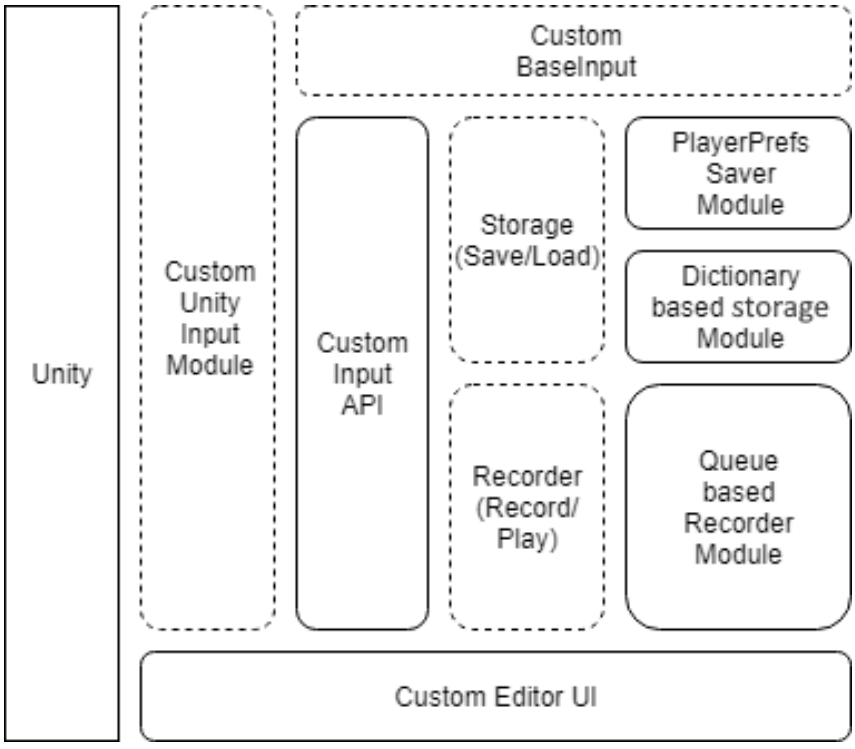

Рисунок 1.2 — Диаграмма платформы решения

На этапе проектирования была составлена диаграмма платформы (см. рис. 1.2), 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 [14];

— *Broker* – шаблон для интеграции и взаимодействия с встроенной системой управления входными данными *Unity* [15];

— *PAC (Presentation–abstraction–control)* – шаблон для организации взаимодействия зависимых систем [16].

Для подмены стандартного класса Input был создан перехватчик ATFInput (см. рис. 1.3), 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*.

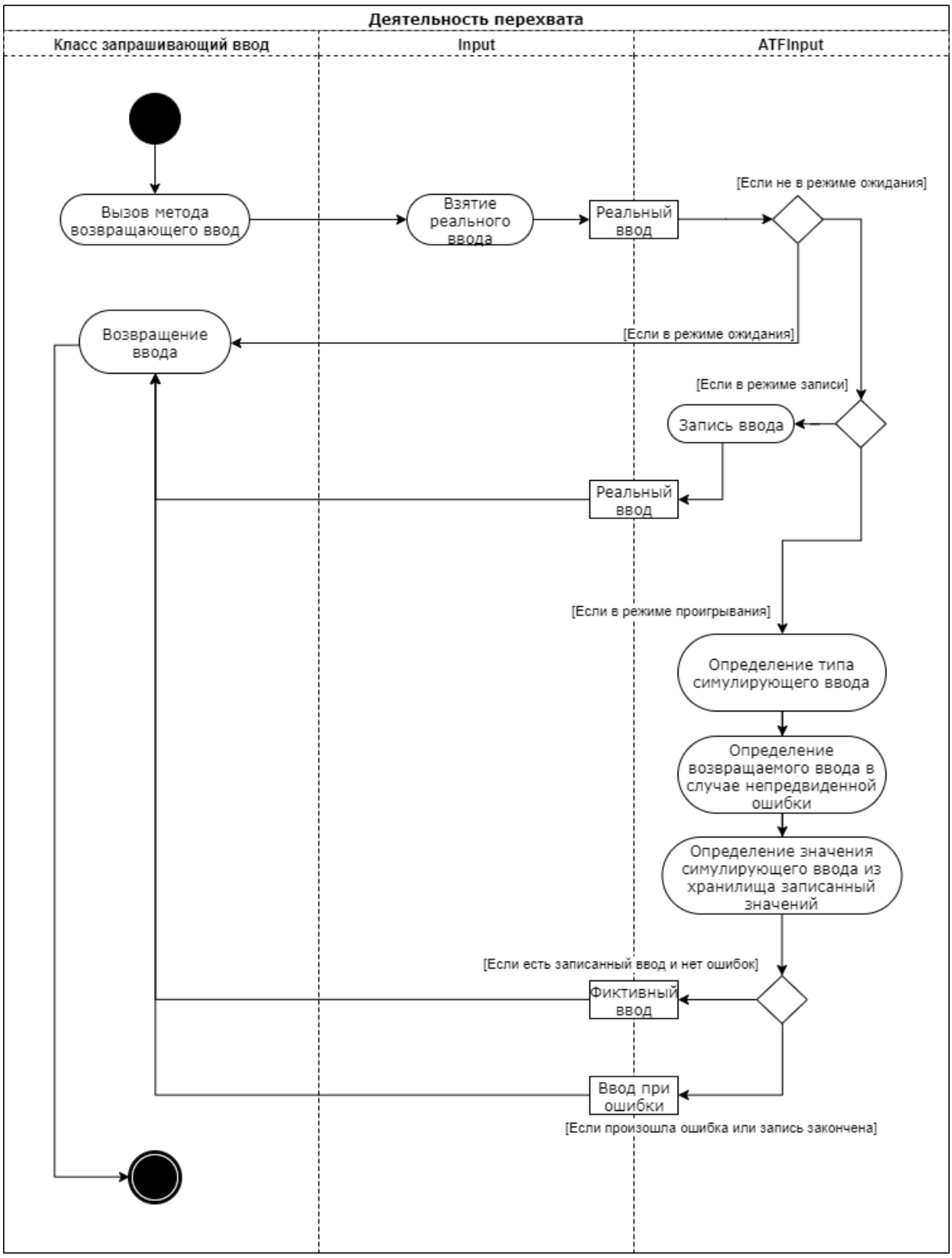

Рисунок 1.3 — Диаграмма деятельности ATFInput

Для интерпретации (проигрывания) первоначально использовался паттерн Interpreter с терминалами ожидания внутри Coroutine. Однако после попытки тестовой реализации данного шаблона был произведен от-

каз 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 Разработка внутренних систем

2.1 Реализация вспомогательных систем

Все вспомогательные системы спроектированы по паттерну “Одиночка” (Singleton) для *Unity*, что означает, что на сцене и в ссылках на экземпляры класса компонента будет находиться только одна инициализированная на сцене сущность.

2.1.1 Реализация синглтона для объектов сцены

Подробное описание основного функционала данного шаблона представлено в листинге А.3. Если описывать в двух словах реализацию этого довольно простого паттерна, то поведение наследников данного класса сводится к тому, что если при вызове поля Instance на сцене не окажется уже установленного объекта, то он создается и конфигурируется автоматически.

2.1.2 Реализация DI контейнера для объектов сцены

В качестве основного функционала контейнера представлен метод, который осуществляет проверку на участие в инъекции зависимостей, поиск подходящей зависимости и непосредственно инъекцию. Все три этапа работают по известному механизму описанному в стандарте JSR-299 [17], а именно: с помощью атрибутов, которые выполняют роль аннотаций, помечаются классы, участвующие в инъекции и поля, в которые нужно её проводить, далее контейнер на этапе запуска проекта проводит все инъекции. Бизнес-логика данного метода подробно представлена в листинге А.4.

Для того чтобы провести инъекцию в целом продукте, также был реализован механизм, который вызывает вышеупомянутый метод на каждом найденном по названию пространства имен типе. Инъекция происходит в двух случаях: объекта из сцены в объект на сцене и объекта из памяти в объект на сцене.

Данный принцип сюръекции нужен только для того, чтобы предупреждать появления ошибок при использовании или дополнении данного

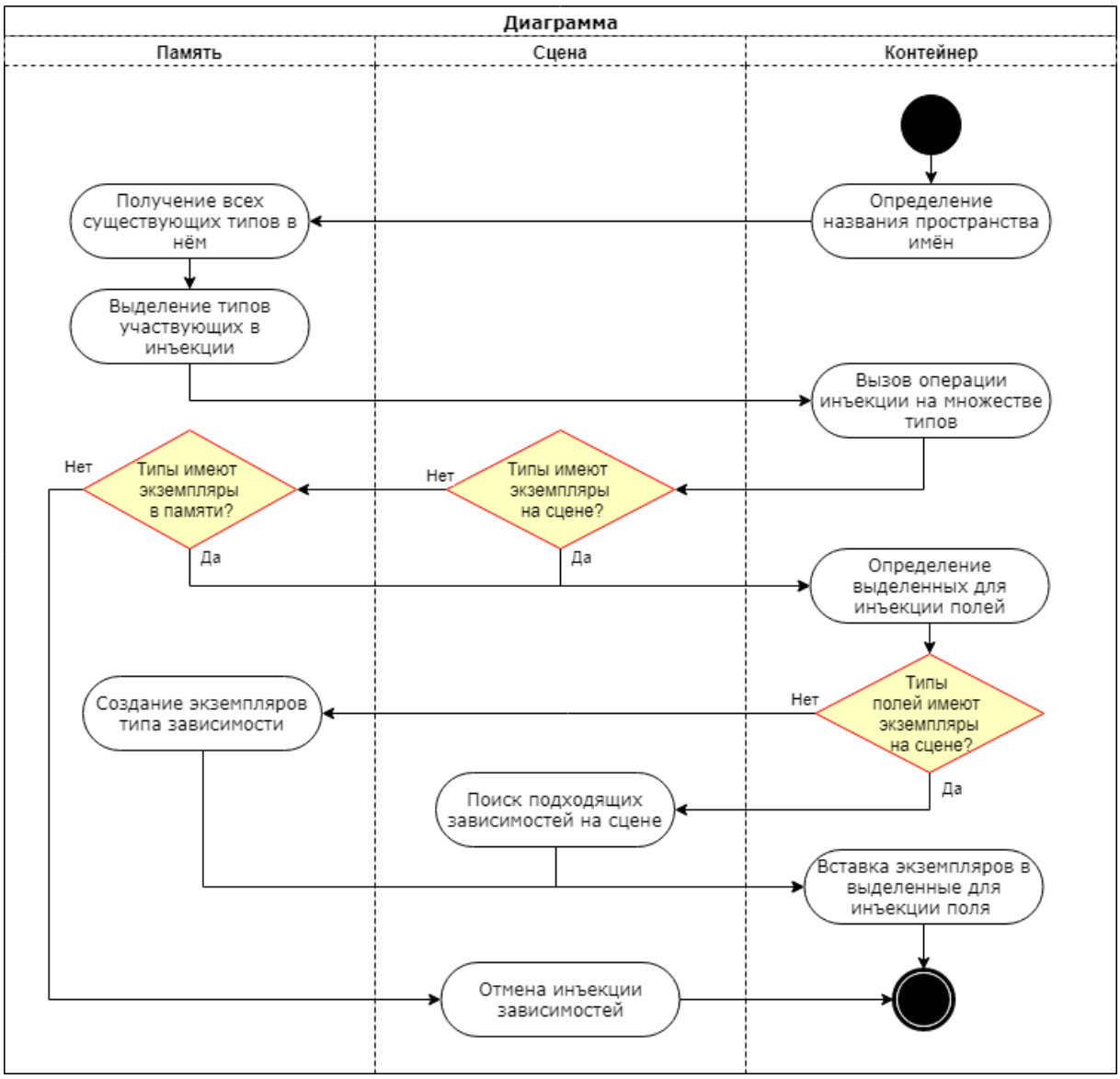

Рисунок 2.1 — Диаграмма деятельности работы DI контейнера для *Unity*

контейнера, когда объект, куда происходит инъекция, не создан на сцене или не инициализирован в памяти. Работа контейнера кратко описана в диаграмме деятельности на рис. 2.1.

2.2 Создание базовой системы интеграции

2.2.1 Реализация базового инициализатора

Для легкой интеграции данного продукта был реализован префаб пустого экземпляра класса GameObject, композиция которого содержит в себе единственный компонент (см. листинг А.5) – его работа иллюстрируется алгоритмом 1.

Механизм работы заключается в следующем: каждая из систем наследуется от интерфейса, который используется в зависимых классах и куда происходит инъекция. Сделано это было с требованием последнего принципа SOLID. Данный интерфейс также наследуется от другого интерфейса IATFInitializable, который содержит в себе единственный метод, отвечающий за инициализацию всех параметров системы после её создания на сцене путем вызова поля Instance. Далее происходит инициализация DI контейнера, настройка его на инъекцию в пространство имён ATF и вызов самой инъекции.

2.2.2 Реализация расширенного инициализатора

Описание атрибутного инициализатора

2.2.3 Реализация адаптера для класса Input

Далее представлен алгоритм 2, составляющий бизнес-процесс класса адаптера, в котором и осуществляется перехват и вычисление относительно текущего состояния систем управления записью и хранилища (см. листинг А.6). Кратко описывая принцип работы, можно выделить три ключевых метода: $Intercept, GetCurrentFakeInput, RealOrFakeInputOrRecord$. Они вызываются стеком, начиная с первого метода.

Первым делом осуществляется перехват (в данном примере) входных данных Input.anyKeyDown, затем происходит вычисление записанного результата из STORAGE с помощью метода $GetCurrentFakeInput$, независимо от того, записывается ли процесс или проигрывается. Далее происходит решение, что выдавать на выход функции anyKeyDown в зависимости от того, в каком сейчас состоянии проигрыватель. Если происходит проигрывание, то выдается результат, который был сохранен в хранилище, если же происходит запись, то выдается результат Input.anyKeyDown и он же записывается в хранилище, а если ничего не происходит, то просто выдается результат, как если бы прослойки для записи вообще не существовало.

2.2.4 Реализация расширенного модуля ввода

Этот пользовательский модуль (см. листинг А.8) был спроектирован с целью дальнейшего расширения функционала по перехвату на внутреннюю систему обмена сообщениями внутри *Unity*. Можно заметить, где конкретно реализован шаблон проектирования Bridge – внутри StandaloneInputModule уже созданы события и описания поведения для работы с сообщениями ввода с периферийных устройств. Именно эти события и будут в будущем подвержены перехвату.

2.3 Реализация основных систем

2.3.1 Реализация системы записи и проигрывания

За основу системы записи и проигрывания был взят интерфейс (см. листинг А.7), регламентирующий поведение, схожее с недетерминированным автоматом, структура которого напоминает полный граф. Дело в том, что данная система подразумевает, что может происходить запись только одной структуры, которая составляет из себя набор различных вводов, в одну единицу вызова.

То есть, для того чтобы начать запись, необходимо первым делом использовать статичные методы ATFInput, во-вторых вызвать метод SetCurrentRecordingName и далее уже вызвать один из методов, изменяющих состояние проигрывателя, в данном конкретном случае – StartRecord. Реализация интерфейса основана на главном свойстве структуры данных “Очередь” – когда происходит запись, то пара FakeInput и объект значения записывается в очередь которая располагается внутри системы хранилища. Там же, при проигрывании выполняется выгрузка данной очереди в отдельную переменную и при каждом вызове перехватываемого метода, просто происходит выдача того, что было первым в этой очереди и так, раз за разом, пока вся очередь не опустошится.

2.3.2 Реализация системы хранилища действий

За основу поведения системы хранилища действий был взят интерфейс IATFActionStorage (см. листинг А.2), регламентирующий поведение книжной полки (статичного хранилища данных).

Принцип работы довольно прост, реализация данного хранилища базируется на вложенной структуре данных “Словарь” или иногда его называют хеш-таблицей. Внутри системы находятся поля следующих типов:

— `Dictionary<string, Dictionary<FakeInput, Dictionary<object, Queue<Action>>>>;`

— `Dictionary<FakeInput, Dictionary<object, Queue<Action>>>.`

Первый тип – это основное хранилище, туда происходит загрузка свежезаписанных сценариев, а также из жесткой памяти, например, из реестра. Второй тип – это та самая переменная, в которую копируется очередь определенного типа ввода и которая опустошается на протяжении каждого перехватываемого вызова проверки ввода. Другие же методы отвечают за операции управления над самим хранилищем.

2.3.3 Решение оптимизационной задачи системы хранилища действий

2.3.4 Реализация системы сохранения и загрузки хранилища действий

Система сохранения и загрузки является PAC компонентом, она используется напрямую только хранилищем записанных действий. Интерфейс IATFActionStorageSaver (см. листинг А.1) отвечает за реализацию системы сохранения и загрузки хранилища в потенциально любой носитель информации.

Реализация интерфейса (см. листинг А.1) основана на работе с реестром через встроенный в стандартную библиотеку *Unity* класс PlayerPrefs, обычно отвечающий за сохранение небольшой по количеству

информации. Для сохранения сложных объектов эта реализация использует сериализацию в JSON объект всего хранилища либо же только одной записи, в зависимости от метода, который вызывается. Поведение реализации похоже на поведение недетерминированного автомата, по уже вышеупомянутым причинам.

2.3.5 Реализация системы интеграции в готовую кодовую базу

2.3.6 Реализация системы упаковки данных хранилища

3 Разработка окон управления

При создании окон управления не были использованы ни синглтон заготовки, ни написанный DI контейнер ввиду особенностей распределения времени выполнения у *Unity* – пространство имен, взаимодействующее с классами, которые регламентируют внешний вид и поведение пользовательских окон, изолировано от других пространств, что делает невозможным использование DI и других вспомогательных структур.

3.1 Создание окна управления записью и проигрыванием

Данное окно (см. рис. 3.1) состоит из трёх секций: указание текущей реализации проигрывателя, состояние проигрывателя и элементы управления проигрыванием и записью.

Рисунок 3.1 — Editor UI для проигрывателя

В процесс управления записью входит:

— просмотр текущего состояния проигрывателя по чекбоксам состояния на рис.3.1;

— определение имени записи, которую нужно проиграть или под которое нужно записать поток действий;

— непосредственно сами кнопки управления записью – Start, Stop, Play, Continue;

— аналогичные кнопки управления проигрыванием – Start, Stop, Play, Continue.

Для того чтобы проиграть выбранную запись, необходимо сначала вписать нужное имя в поле *Name of the recording* или же просто щёлкнуть правой кнопкой мыши по записи в активной зоне окна управления хранилищем.

3.2 Создание окна управления хранилища действий

Окно управления хранилищем (см. рис. 3.2) также состоит из трёх секций – информация о реализации, секции управления сохранением и загрузкой, а также зоны иллюстрации текущего состояния хранилища.

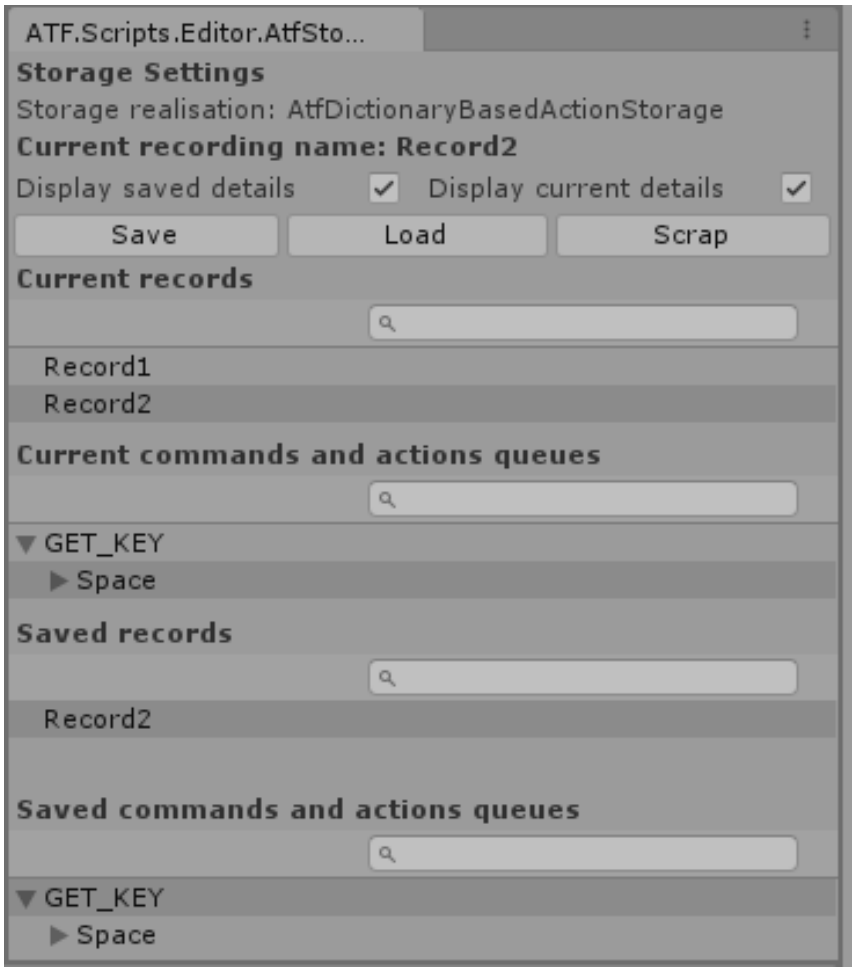

Рисунок 3.2 — Editor UI для хранилища действий

В процесс управления хранилищем входит:

— просмотр текущей записи хранилища;

— определение имени записи, которую нужно загрузить или сохранить;

— непосредственно сами кнопки управления сохранением – Save, Load, Scrap (удалить);

— загрузка в активную зону записей из реестра (Окна: Current records, Current commands and actions queues).

Зоны иллюстрации хранилища делятся на активную и пассивную. Активная зона хранит в себе те записи, которые находятся сейчас непосредственно в оперативной памяти, пассивная же зона хранит в себе записи, которые были сохранены на внешнем накопителе или реестре.

3.3 Создание окна управления интеграцией

3.4 Создание онлайн-документации

ЗАКЛЮЧЕНИЕ

Текст заключения

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Vila E., Novakova G., Todorova D. Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats // Proceedings of the International Conference on Advances in Image Processing (ICAIP 2017). 2017. P. 144-150.

2. Sinaga A. M., Adi Wibowo P., Silalahi A., Yolanda N. Performance of Automation Testing Tools for Android Applications // 10th International Conference on Information Technology and Electrical Engineering (ICITEE). 2018. P. 534-539.

3. Mozgovoy M., Pyshkin E. Unity Application Testing Automation with Appium and Image Recognition // In: Itsykson V., Scedrov A., Zakharov V. (eds) Tools and Methods of Program Analysis. TMPA 2017. Communications in Computer and Information Science. 2017. №779. P. 139–150.

4. Andries van Dam Post-WIMP user interfaces // Commun. ACM. 1997. №2. P. 63–67.

5. Лучший игровой движок по версии пользователей хабра // Хабр URL: https://habr.com/ru/post/307952/ (дата обращения: Апрель 11, 2020).

6. Кугуракова В.В МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ТРЕНАЖЕРОВ С ПОГРУЖЕНИЕМ В ИММЕРСИВНЫЕ ВИРТУАЛЬНЫЕ СРЕДЫ: дис. ... канд. тех. наук: 05.13.11. Казань, 2019. 187 с.

7. Kugurakova V.V. Automated approach to creating multi-user simulators in virtual reality // CEUR Workshop Proceedings. 2018. №2260. P. 313-320.

8. Kugurakova V.V., Abramov V.D., Abramsky M.M., Monks N., Maslaviev A. Visual editor of scenarios for virtual laboratories // DeSE 2017, Developments in the design of electronic systems.. Paris: Conference Publication Services (CPS), 2017. P. 242-247.

9. Мартин Р.С., Ньюкирк Д.В., Косс Р.С. Быстрая (гибкая) разработка программ на Java и C++: принципы, шаблоны, практика. М.: Издательский дом “Вильямс”, 2001. 752 с.

10. Haoyu W., Haili Z. Basic Design Principles in Software Engineering // Fourth International Conference on Computational and Information Sciences. Chongqing: IEEE Computer Society, 2012. P. 1251-1254.

11. Blumer A., Ehrenfeucht A., Hasler D., Warmuth M.K. Occam's Razor // Information Processing Letters 24. 1987. №6. P. 377-380.

12. High-performance multithreaded stack of unity information-oriented technologies // unity.com URL: https://unity.com/ru/dots (дата обращения: Июль 19, 2019).

13. Unity Foundation – Introduction to ECS // unity3d.com URL: https://unity3d.com/ru/learn/tutorials/topics/scripting/introduction-ecs (дата обращения: Июль 19, 2019).

14. Schmidt D., Stal M., Rohnert H., Buschmann F. Pattern-Oriented Software Architecture // Patterns for Concurrent and Networked Objects. 2001. №2. P. 109-140.

15. Architectural pattern "Broker" // cs.uno.edu URL: http://cs.uno.edu/ jaime/Courses/4350/broker.ppt (дата обращения: Июль 19, 2019).

16. Coutaz J. PAC: AN OBJECT ORIENTED MODEL FOR IMPLEMENTING USER INTERFACES // ACM SIGCHI Bulletin. 1987. №19. P. 37-41.

17. JSR 299: Contexts and Dependency Injection for the JavaTM EE platform // jcp.org URL: https://jcp.org/en/jsr/detail?id=299 (дата обращения: Июль 19, 2019).

18. Брюс П., Брюс Э. Практическая статистика для специалистов Data Science. СПб.: БХВ-Петербург, 2018. 304 с.

Приложение А Листинги

Листинг А.1 — Интерфейс модуля загрузчика хранилища записей

```
public interface IATFActionStorageSaver : IATFInitializable
{
  void SaveRecord();
  void LoadRecord();
  void ScrapRecord();

  IEnumerable GetActions();
  void SetActions(IEnumerable actionEnumerable);
  List<TreeViewItem> GetSavedNames();
  List<TreeViewItem> GetSavedRecordDetails(string recordName);
}
```

Листинг А.2 — Интерфейс модуля хранилища записей

```
public interface IATFActionStorage : IATFInitializable
{
  object GetPartOfRecord(FakeInput kind, object
     fakeInputParameter);
  void Enqueue(string recordName, FakeInput kind, object
     fakeInputParameter, AtfAction atfAction);
  AtfAction Dequeue(string recordName, FakeInput kind, object
     fakeInputParameter);
  AtfAction Peek(string recordName, FakeInput kind, object
     fakeInputParameter);
  bool PrepareToPlayRecord(string recordName);
  void ClearPlayStorage();
  void SaveStorage();
  void LoadStorage();
  void ScrapSavedStorage();
  List<TreeViewItem> GetSavedRecordNames();
  List<TreeViewItem> GetCurrentRecordNames();
  List<TreeViewItem> GetCurrentActions(string recordName);
  List<TreeViewItem> GetSavedActions(string recordName);
}
```

Листинг А.3 — Бизнес-логика MonoSingleton<T>

```
public class MonoSingleton<T> : MonoBehaviour where T :
   MonoBehaviour
{
  private static T s_Instance;
  private static bool s_IsDestroyed;
  public static T Instance
  {
  get
  {
  if (s_IsDestroyed)
  return null;
  if (s_Instance == null)
  {
  s_Instance = FindObjectOfType(typeof(T)) as T;

  if (s_Instance == null)
  {
  var gameObject = new GameObject(typeof(T).Name);
  DontDestroyOnLoad(gameObject);
  s_Instance = gameObject.AddComponent(typeof(T)) as T;
  }
}
return s_Instance;
...
}
```

Листинг А.4 — Метод инъекции зависимостей в экземпляр класса типа

```
public static void InjectType(Type t)
{
if (!ContainsAnyAttributeOfType(t.GetCustomAttributes(false),
   typeof(InjectableAttribute))) return;
foreach (var fi in t.GetFields())
{
...
if (temp.ComponentType == null && isScenePathEmpty)
   temp.ComponentType = fi.FieldType;
...
if (!isScenePathEmpty)
{
```

```
var hierarchyAndComponent =
   GetHierarchyPathAndComponentName(temp.ScenePath);
if (!hierarchyAndComponent.Valid)
{
... continue;
}
gameObjectContainingObjectToInject =
   GameObject.Find(hierarchyAndComponent.Result[0]);
if (!gameObjectContainingObjectToInject)
{
... continue;
}
objectToInject = gameObjectContainingObjectToInject
.GetComponent(hierarchyAndComponent.Result[1]);
}
else
{
objectToInject = FindObjectOfType(temp.ComponentType);
if (objectToInject)
{
gameObjectContainingObjectToInject =
   GameObject.Find(objectToInject.name);
}
if (!gameObjectContainingObjectToInject && !temp.LookInScene)
{
objectToInject = Activator.CreateInstance(temp.ComponentType)
   as UnityEngine.Object;
if (!objectToInject)
{
... continue;
} ...
}
}
if (!gameObjectContainingObjectToInject && objectToInject ==
   null && temp.LookInScene)
{
... continue;
}
...
```

```
dynamic typeToWhichInjected =
   FindObjectOfType(t.GetTypeInfo());
if (typeToWhichInjected == null)
{
... continue;
}
fi.SetValue(typeToWhichInjected, objectToInject);
...
}
```

Листинг А.5 — Бизнес-логика базовой системы интеграции

```
public class ATFInitializer : MonoBehaviour
{
private void Awake()
{
IATFInitializable[] ALL_SYSTEMS = {
ATFQueueBasedRecorder.Instance,
ATFDictionaryBasedActionStorage.Instance,
ATFPlayerPrefsBasedActionStorageSaver.Instance
};
foreach (IATFInitializable i in ALL_SYSTEMS)
{
i.Initialize();
}
DependencyInjector.Instance.Initialize("ATF");
DependencyInjector.Instance.Inject();
}
}
```

Листинг А.6 — Бизнес-логика системы симуляции и перехвата ввода

```
[Injectable]
public class ATFInput : BaseInput
{
[Inject(typeof(ATFQueueBasedRecorder))]
public static readonly IATFRecorder RECORDER;
[Inject(typeof(ATFDictionaryBasedActionStorage))]
public static readonly IATFActionStorage STORAGE;
...
```

```
private static object RealOrFakeInputOrRecord(object
   realInput, object fakeInput, object fip, FakeInput kind)
{ ... }
private static object GetCurrentFakeInput(FakeInput inputKind,
   object fip)
{
return STORAGE.GetPartOfRecord(inputKind, fip);
}
...
private static T Intercept<T>(object realInput, FakeInput
   fakeInputKind, T defaultValue, object fakeInputParameter =
   null)
{
return IfExceptionReturnDefault(
() => (T) RealOrFakeInputOrRecord(realInput,
   GetCurrentFakeInput(fakeInputKind, fakeInputParameter),
   fakeInputParameter, fakeInputKind),
defaultValue
);
}
public static bool anyKeyDown => Intercept(Input.anyKeyDown,
   FakeInput.ANY_KEY_DOWN, false);
...
}
```

Листинг А.7 — Интерфейс модуля записи

```
public interface IATFRecorder : IATFInitializable
{
bool IsRecording();
bool IsPlaying();
bool IsRecordingPaused();
bool IsPlayPaused();
void PlayRecord();
void PausePlay();
void ContinuePlay();
void StopPlay();
void StartRecord();
void PauseRecord();
void ContinueRecord();
```

```
void StopRecord();
void SetRecording(bool value);
void SetPlaying(bool value);
void SetRecordingPaused(bool value);
void SetPlayPaused(bool value);
void Record(FakeInput kind, object input, object
   fakeInputParameter);
}
```

Листинг А.8 — Пользовательский модуль ввода для ATFInput

```
namespace ATF
{
public class ATFStandaloneInputManager : StandaloneInputModule
{
protected override void Start()
{
base.Start();
m_InputOverride = gameObject.AddComponent<ATFInput>();
DependencyInjector
.InjectType(m_InputOverride.GetType());
}
...
}
```

Algorithm 1 Работа инициализатора решения

$allSystems \leftarrow IAtfInitializable[3]$
$allSystems[0] \leftarrow Recorder.Instance$
$allSystems[1] \leftarrow Storage.Instance$
$allSystems[2] \leftarrow Saver.Instance$
for all system in allSystems **do**
    system.Initialize()
end for
$DependencyInjector.InjectIntoNamespace('ATF')$

Algorithm 2 Перехват и симуляция для Input.anyKeyDown

```
procedure ATFINPUT.ANYKEYDOWN
    return Intercept(Input.anyKeyDown,
    GetCurrentFakeInput(),
    false)
end procedure
procedure INTERCEPT(RealInput, FakeInput, DefaultValue)
    if FakeInput is acquired then
        return RealOrFakeInputOrRecord(RealInput, FakeInput)
    else
        return DefaultValue
    end if
end procedure
procedure REALORFAKEINPUTORRECORD(RealInput, FakeInput)
    if is Recording then
        Recorder.Record(RealInput)
        return RealInput
    else if is Playing then
        return FakeInput
    else
        return RealInput
    end if
end procedure
```