

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	4
1 Проектирование	8
1.1 Выбор основного способа запрашивания ввода	8
1.2 Класс совместимых приложений	12
1.3 Определение теоретической базы	13
2 Внутренние модули и системы группы ассетов	17
2.1 Вспомогательные системы	17
2.1.1 Spider-программа для сбора данных с GitHub.com	17
2.1.2 Синглтон для объектов сцены	19
2.1.3 DI контейнер для объектов сцены	20
2.2 Модули инициализации и перехвата ввода	20
2.2.1 Инициализатор сцены	21
2.2.2 Адаптер для класса Input	21
2.2.3 Расширенный модуль ввода	22
2.3 Основные системы группы ассетов	23
2.3.1 Система записи и проигрывания	23
2.3.2 Система хранилища действий	23
2.3.3 Решение проблемы избыточности хранилища	24
2.3.4 Система сохранения и загрузки хранилища	25
2.3.5 Система упаковки данных хранилища	26
2.3.6 Система интеграции в готовую кодовую базу	26
3 Человеко-машинный интерфейс	28
3.1 Окно управления записью и проигрыванием	28
3.2 Окно управления хранилища действий	30
3.3 Окно управления интеграцией	31
3.4 Онлайн-документация и способ доступа	33
4 Практическое применение	37
4.1 Портинг на Unity 2017.3.1f1	37
4.2 Интеграция в симулятор VRTK	38
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43
А Листинги	47

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, 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 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 распознавания того, как и с чем взаимодействует пользователь [3]. Метод зачастую довольно 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. В дополнение 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 и в эффективности, и в степени удобства интеграции инструмента записи и проигрывания действий, так как зачастую игры насыщены динамическими графическими элементами, а некоторые игры вообще не имеют графический интерфейс. В отличие 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 ввиду 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.

Цель и задачи разработки

Целью данной работы является реализация и применение формы решения описанной выше проблемы регрессионного тестирования для игровых приложений платформы *Unity* в виде импортируемой группы ассетов (исходных файлов кода и пред заготовленных графических и модульных инструментов).

Реализуемое решение основано на вышеописанных подходах для платформ PC, Web и др., и на возможности декорирования методов взятия ввода, обусловленной внутренней архитектурой платформы. Две главных

особенностей данного решения – это высокая точность распознавания и перехвата ввода и возможность избежать регулярных потерь времени при человеческом тестировании за счёт 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.

В настоящей выпускной квалификационной работе рассматриваются задачи связанные напрямую с исследованием применения инструментария взятия ввода, 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 статистических исследований и 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. Здесь же описываются принципы дополнения и расширения исходной кодовой базы описываемых ассетов для всех желающих сторонних разработчиков.

Четвертая и заключительная глава рассматривает опыт портирования и применения разработанной группы ассетов внутри проекта виртуальной био-технологической лаборатории.

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 описывается 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 собой входные данные со всех периферийных устройств, что были активны во время работы приложения, в которое группа ассетов была импортирована.

Дополнением к результату является документация к пользованию разработанными ассетами, их расширению для нужд сторонних разработчиков, а также видеофайл, демонстрирующий работу с симулятором VR устройства в проекте DML BioLab [9], где происходит запись и проигрывание первого пункта первого сценария.

1 Проектирование

Процесс проектирования в данной работе представлен из двух частей: исследование аспектов проблематики и составление базы приёмов программной инженерии, которые будут использоваться в процессе разработки.

1.1 Выбор основного способа запрашивания ввода

Имея перед собой цель создания ассетов способных автоматизировать действия пользователя внутри разрабатываемого приложения, необходимо было выбрать самый лёгкий способ запрашивания ввода, который будет перехватываться и записываться.

Метрикой лёгкости способов запрашивания ввода здесь будет являться длина двумерного вектора из количества необходимых для использования единиц компиляции (ЕК) и количества использованных строк кода (ИСК). Далее была сформулирована нулевая гипотеза: самый лёгкий способ запрашивания ввода имеет минимальную длину вектора данной проверочной статистики и является классом `Input`.

Согласно официальному мануалу посвящённому системам запрашивания ввода [10] на платформе *Unity* существует два модуля подобных систем: `Input Manager` и `Input System`. Первая система включает в себя два способа: класс взятия универсального ввода `Input` и интерфейсы-маркеры для взаимодействия только с указателем (мышкой, пальцем и т.д.). Вторая система, по сути своей, является альтернативой первой, она поставляется отдельно, требует установки дополнительных зависимостей и содержит единственный способ взятия ввода путём создания специальных профилей, где описывается ассоциация определённого объекта в памяти с потоком данных из определённого периферийного устройства.

Для того чтобы определить количество единиц компиляции, выражение на языке программирования `CSharp` (основном языке платформы *Unity*) переписывается с использованием классов пространства имён `System.Reflection` стандартной библиотеки `CSharp`. Количество экземпля-

ров классов задействованных в переписанной версии выражения и будет являться количеством единиц компиляции.

Для подсчёта ЕК и ИСК в интерфейсе-маркере использовался пример из Unity Scripting API [11], а для Input System использовался пример из Unity Manual [12].

Таблица 1.1 — Таблица векторов данных и их длин

Способ запрашивания ввода	Количество		Метрика
	ЕК	ИСК	Длина вектора
Класс Input	3	1	$\sqrt{3^2 + 1} = \sqrt{10}$
Интерфейс-маркер	6	4	$\sqrt{6^2 + 4^2} = \sqrt{44}$
Input System	7	6	$\sqrt{7^2 + 6^2} = \sqrt{85}$

Опираясь на данные приведённые в таблице 1.1 можно утвердить нулевую гипотезу. Однако, всего вышесказанного недостаточно, чтобы сделать окончательный выбор и остановиться на классе Input. Данный вывод лишь указывает на необходимое условие критерия выбора способа запрашивания ввода. Теперь же нужно предоставить достаточное условие.

Достаточным условием выбора класса Input в качестве основы для перехвата и записи является то, что этот инструмент популярен и распространён среди других Unity-проектов. Теперь формируем следующую нулевую гипотезу: математическое ожидание количества использований класса Input в кодовой базе любого проекта на *Unity* больше нуля.

Другими словами, в среднем по всем значимым проектам количество использований класса Input будет больше нуля. Это будет указывать не только на популярность использования данного способа, но и на распространённость, ведь создавая новый продукт, каждый разработчик старается опираться на лучший опыт других значимых проектов и это выражается в виде существования библиотек для ПО, фреймворков и др.

Генеральной совокупностью в рамках данной работы принято считать все Unity-проекты с максимальным количеством использования

CSharp (т.е. доля языка CSharp в проекте доминирует), с максимальным количеством форков (проектов, в основе которых лежит выбранный проект), со стабильным обновлением и с максимальной поддержкой своих сообществ.

Далее была произведена выборка проектов с помощью реализованной в рамках данной работы spider-программы описанной в начале второй главы. Структура выборки наглядно иллюстрируется в таблице 1.2. Размер выборки составляет 3970 проектов из 99509 возможных по запросу “language:csharp unity” на сайте GitHub.com. Ввиду технических ограничений данного сайта, существует возможность исследовать только столько проектов, сколько составляет размер описанной выборки.

Таблица 1.2 — Структура выборки с количеством использований класса Input

Название проекта	Извлечённые данные	
	Количество исп.	Номер
github-for-unity/Unity	1	0
Unity-Technologies/UnityCsReference	0	1
XINCGer/Unity3DTraining	62	2
...
KuraiAndras/MainThreadDispatcher.Unity	0	3969

Итак, данные собраны. Для наглядной оценки математического ожидания количества использований, был выбран бутстраповский доверительный интервал [13] с уровнем доверия 90%. Алгоритм просчёта границ интервала довольно прост:

- Определяем размер выборки n и некоторое большое число R ;
- далее извлекаем случайную выборку размера n с возвратом из данных (повторная выборка);
- записать целевую статистику для повторной выборки;
- повторить предыдущие шаги R раз;

- для x -процентного доверительного интервала отсечь $[(1 - [x/100])/2] \%$ от R результатов слева и справа;
- точками отсечения являются конечные точки x -процентного бутстраповского доверительного интервала.

Для данной работы, параметру R было присвоено значение 99509, как если бы техническое ограничение источника данных на длину выборки отсутствовало, а параметрам n и x присвоены значения 3970 и 90 соответственно. Результаты подсчёта интервальной оценки иллюстрируются на рис. 1.1, где ось абсцисс – номер повторной выборки, а ось ординат – выборочное математическое ожидание.

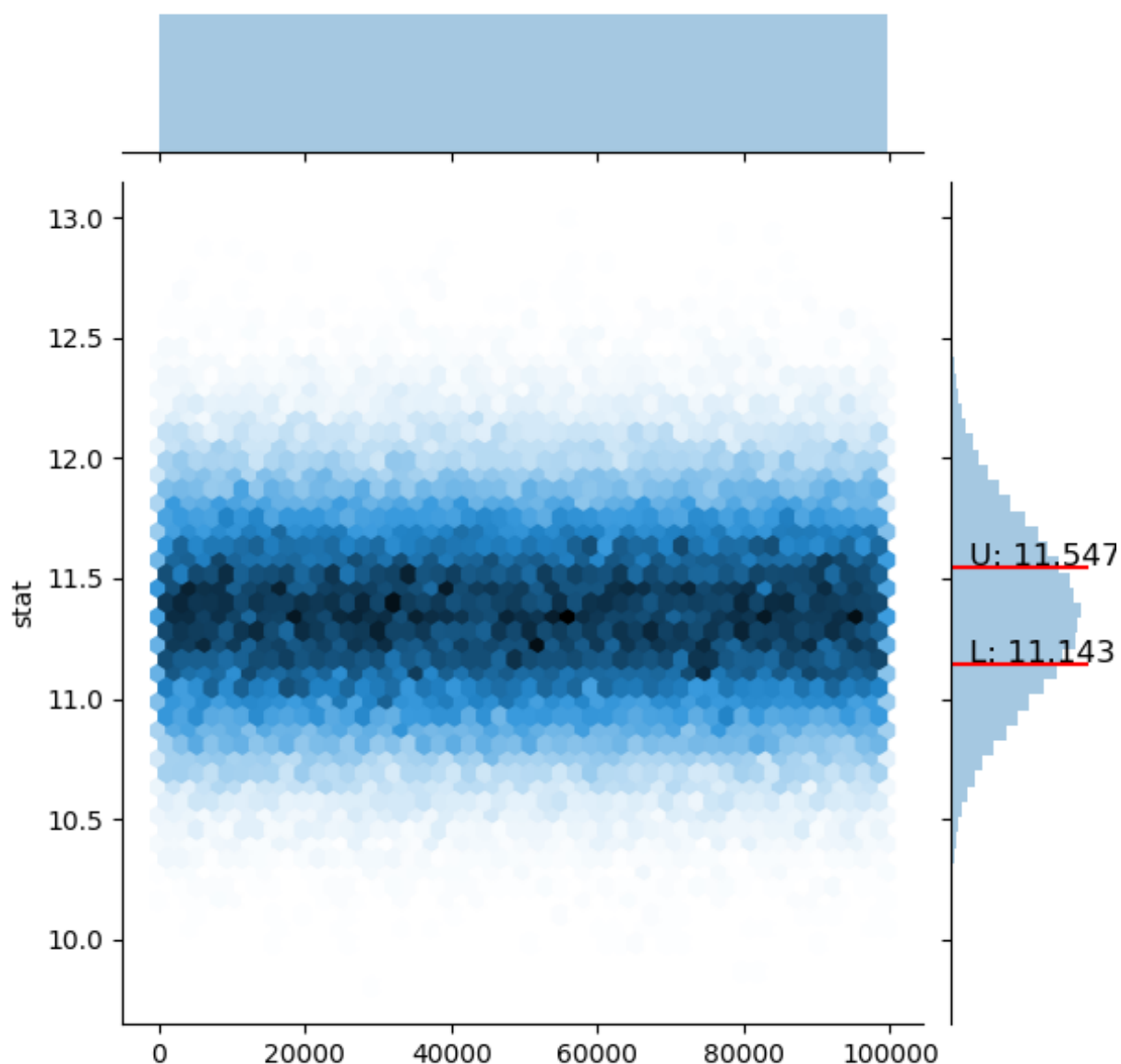


Рисунок 1.1 — Диаграмма рассеивания бутстраповской интервальной оценки

Основываясь на данных полученных в результате подсчёта описанного выше доверительного интервала, можно с уверенностью 90% утвердить нулевую гипотезу, так как полученные границы находятся выше нуля и равны 11.143 и 11.547 соответственно.

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 запрашиванию ввода Input. Под это описание подходит большая часть *Unity*-приложений, как было обосновано в предыдущем разделе.

1.3 Определение теоретической базы

В основе проектирования сложных ассетов всегда необходимо учитывать фундаментальные принципы построения ПО с использованием объектно-ориентированного подхода. Этими принципами в рамках описываемого процесса разработки являются пять правил SOLID [14], принцип ухода от повторений единиц компиляции и интерпретации DRY [15], а также принцип лезвия Оккама, который позволяет избежать избытка разрабатываемых сущностей в виде классов. Дополнительными особенностями данной кодовой базы можно назвать автоматическую систему переноса и упаковки группы ассетов, характерную для *Unity*-платформы (“*.unitypackage*” пакетирование), а также возможность автоматической интеграции в исходный код любого *Unity*-проекта, где будет использован предмет данной разработки.

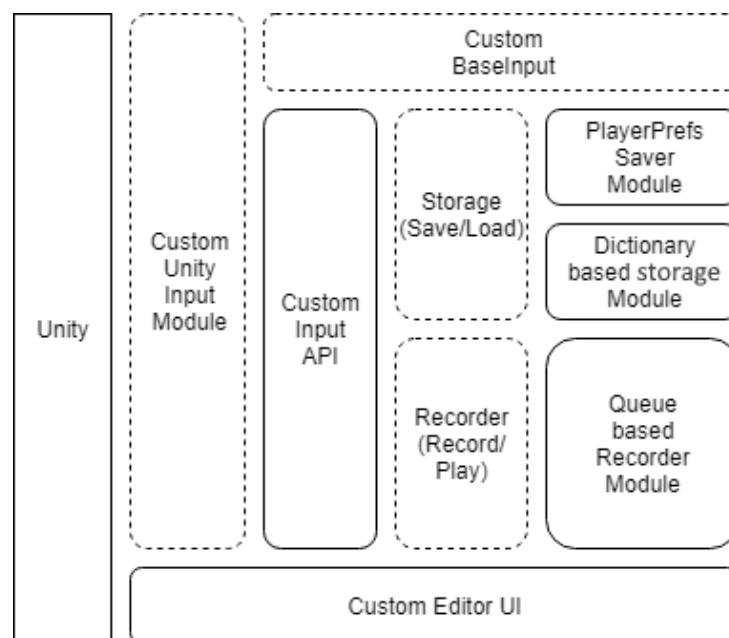


Рисунок 1.2 — Диаграмма платформы решения

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, которая является реализацией объекта обработки потока данных через мост (Bridge), объединяя статические 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 [19];
- *Broker* – шаблон для интеграции и взаимодействия с встроенной системой управления входными данными *Unity* [20];

— *PAC (Presentation–abstraction–control)* – шаблон для организации взаимодействия зависимых систем [21].

Для подмены стандартного класса `Input` был создан перехватчик класс `ATFInput` (см. рис. 1.3). Данный класс наследуется от класса `MonoSingleton<ATFInput>`, который является специальной реализацией паттерна `Singleton` для объектов на сцене. Также перехватчик имплементирует интерфейс `IBaseInput`, который объединяет в себе интерфейсы `Input` класса и стандартного класса `BaseInput`. Данное объединение необходимо для того, чтобы обеспечить полноту копирования интерфейса класса `Input`, одновременно сохраняя совместимость с системой распределения события `Unity`.

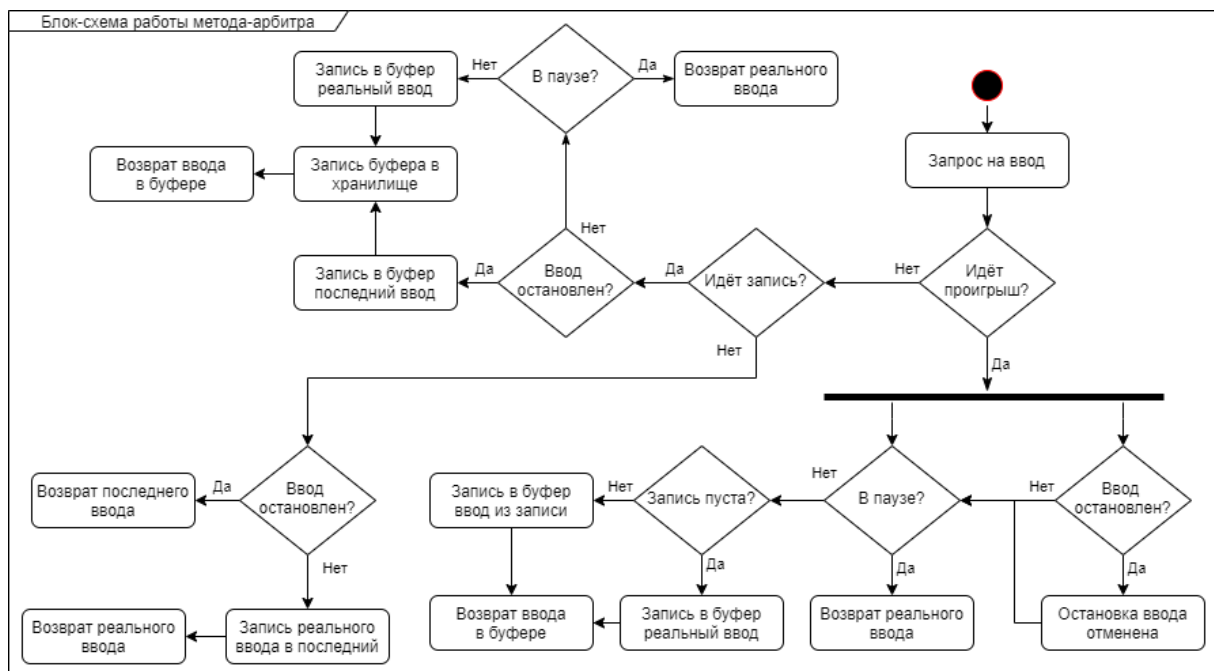


Рисунок 1.3 — Блок-схема деятельности метода-арбитра `ATFInput`

Стоит заметить, что под “Запросом на ввод” имеется ввиду вызов любого метода класса `ATFInput`, который повторяет сигнатуру метода класса `Input`. Некоторые из методов класса `ATFInput` также доступны по объектному доступу. Именно это оставляет совместимость `ATFInput` со стандартным компонентом `Event System`, потому как данный компонент обращается к методам перехватчика по объектному доступу.

Для интерпретации (проигрывания) первоначально использовался паттерн `Interpreter` с терминалами ожидания внутри `Coroutine`. Однако по-

сле попытки тестовой реализации данного шаблона был произведен отказ в пользу метода записи, основанного на структуре данных “Очередь”, так как для правильной работы терминалов ожидания необходимо было просчитывать заранее действия пользователя, что значительно усложнило бы задачу.

2 Внутренние модули и системы группы ассетов

Процесс разработки внутренних модулей и систем в данной главе представлен в виде краткого описания главных аспектов модуля или системы, которое дополнено приложением с подробной иллюстрацией бизнес-процесса.

2.1 Вспомогательные системы

Все вспомогательные системы, кроме spider-программы описанной ниже, 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 Spider-программа для сбора данных с GitHub.com

Для сбора данных и проведения эксперимента по вычислению бутстраповского доверительного интервала, в рамках данной работы, была разработана программа, позволяющая перемещаться по страницам сайта GitHub.com, с возможностью сбора необходимых данных. Данная программа в своём имени имеет префикс *spider*, так как она подобно пауку может ползать во всемирной паутине улавливая данные.

Инструментарий для проектирования spider-программы

В качестве основы для spider-программы использовался фреймворк *Scrapy* [22]. Данный фреймворк автоматизирует решение задач о том, какими именно инструментами и каким образом происходит получение веб-страниц. На вход его механизма по сбору интернет-ресурсов задаётся массив из необходимых URL адресов, а также функция по обработке данных полученных по ним веб-страниц. Его использование позволило сконцентрироваться на алгоритме обработки данных, что дало избежать написания сквозного кода (усложняющего чтение кода).

Алгоритм поведения spider-программы

На вход алгоритма по обработки данных (см. приложение А.1) подается один из адресов поиска по сайту с настроенным фильтром: самый релевантный запросу, самый популярный по рейтингу звёзд, с самым большим количеством форков и самые стабильно обновляющийся проект, где средний разброс времени между коммитами у проекта невелик. Далее происходит вычленение по страницам полные названия репозиторий, которые используются для загрузки всех доступных файлов исходного кода репозитория по очереди. Выгруженные файлы исходного кода затем фильтруются по ключевому слову *Input*, что превращает поток файлов в поток сниппетов кода (блоков кода) с использованием слова *Input* в той или иной команде или функции. На выходе данный поток ещё более фильтруется, но уже по регулярному выражению, которое выявляет любые использования класса *Input*. Затем вычисляется размер результирующего потока, который впоследствии попадает в выборку рядом с полным именем просмотренного репозитория.

Формат выходных данных spider-программы

В качестве результата работы этапа сборки данных, в корневую папку spider-программы сохраняется “.csv” файл со структурой идентичной таблице 1.2.

Утилиты spider-программы для проведения расчётов

В дополнении к ядру spider-программы были реализованы вспомогательные утилиты для сбора данных в один .csv файл (см. приложение А.2) и для проведения бутстрапа и расчётов (см. приложение А.3). Стоит заметить, что проведение бутстрапа и расчётов задача вычислительно ёмкая, поэтому предложенный алгоритм автоматически использует все доступные ядра на процессоре, чтобы провести вычисления параллельно.

В итоге, на выходе работы spider-программы создаётся диаграмма рассеяния с шестиугольным фильтром указывающая на нижнюю и верхнюю границы доверительного интервала среднего (см. рис. 1.1).

2.1.2 Синглтон для объектов сцены

Подробное описание основного функционала данного шаблона представлено в приложении А.6. 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.

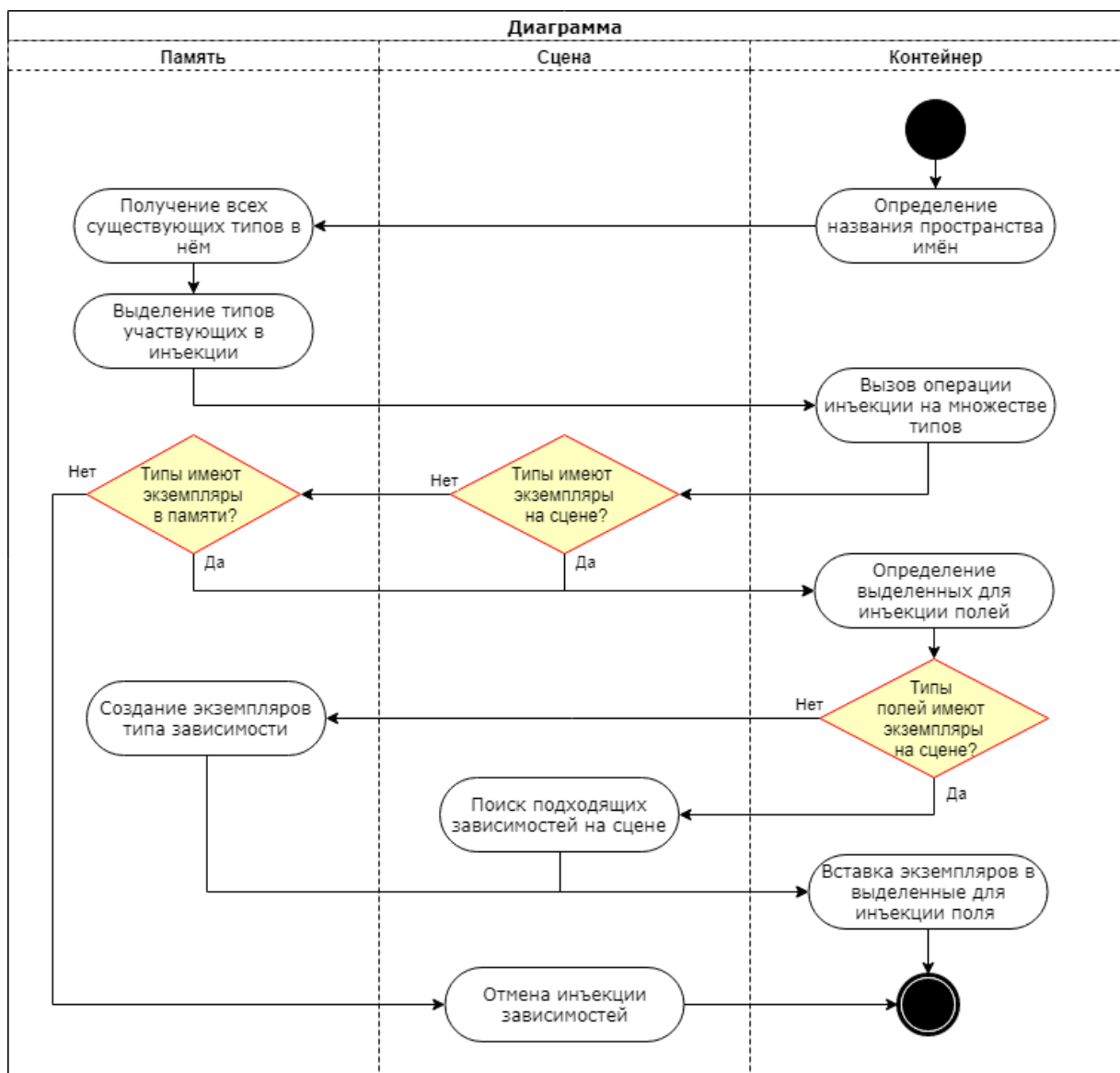


Рисунок 2.1 — Диаграмма деятельности работы DI контейнера для *Unity*

2.1.3 DI контейнер 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 [23], 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 приложении А.7.

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 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 Модули инициализации и перехвата ввода

Для того чтобы сократить количество лишних действий после импортирования данной группы ассетов в сторонний проект, был реализован модуль инициализации, главная цель которого заключалась в автоматической настройке основных систем при переходе в Play Mode.

Модуль перехвата является ядром разрабатываемой группы ассетов, в основе которого лежит тот факт, что если полностью скопировать сигнатуры статических методов стандартного класса Input, то для дальнейшей интеграции и использования необходимо будет лишь использовать имя другого класса, вместо имени Input. Все же основные аспекты стандартного взятия ввода останутся нетронутыми.

2.2.1 Инициализатор сцены

Для легкой интеграции данного продукта был реализован префаб пустого экземпляра класса `GameObject`, композиция которого содержит в себе единственный компонент (см. приложение А.8) – 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.

Чтобы избежать дополнительного вмешательства в исходный код реализации базового инициализатора при добавлении новых систем, был создан механизм составления очереди на вызов поля `Instance` и инициализацию путём маркировки новых систем атрибутом `AtfSystem`. Другими словами, чтобы добавить новую систему при этом не вмешиваясь в код базового инициализатора, необходимо чтобы добавляемая система типа `T` наследовалась от класса `MonoSingleton<T>` и при этом её класс был бы помечен соответствующим атрибутом.

2.2.2 Адаптер 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. приложение А.9).

Кратко описывая принцип работы, можно выделить три ключевых метода:

- а) *Intercept* – метод, отвечающий на реакцию в запросе на ввод.
- б) *GetCurrentFakeInput* – метод, возвращающий значение ввода из хранилища действий.
- в) *RealOrFakeInputOrRecord* – метод, отвечающий за распределение операций записи, возврата последнего ввода и возврата реального или записанного ввода

Исполнение каждой из описанных функций во время запроса на ввод организовано в стек вызовов, где на первой позиции находится (в), на второй (б) и на последней (а).

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.

2.2.3 Расширенный модуль ввода

Этот пользовательский модуль (см. приложение А.11) 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 Основные системы группы ассетов

Далее под словом “система” будет пониматься класс типа T, который наследуется от класса `MonoSingleton<T>` и обладает однородным интерфейсом с другими “системами”.

2.3.1 Система записи и проигрывания

За основу системы записи и проигрывания был взят интерфейс (см. приложение A.10), 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 Система хранилища действий

За основу поведения системы хранилища действий был взят интерфейс `IATFActionStorage` (см. приложение A.5), 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, AtfActionRleQueue>>>;  
– Dictionary<FakeInput, Dictionary<object,  
AtfActionRleQueue>>.
```

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 опустошается на протяжении каждого перехватываемого вызова проверки ввода. Другие же методы отвечают за операции управления над самим хранилищем.

Стоит отдельно упомянуть возможности данной системы по импорту и экспорту первого типа. Мотивация к реализации данного функционала заключается в перспективе переиспользования данных хранилища не только в целях основной задачи. Например, данные первого типа могут послужить ядром механизма по тестированию и генерации сценариев пользовательского поведения, где шаги описываются не последовательностью ввода, а последовательностью состояний объектов на сцене. Данный подход описания широко используется в виртуальных тренажерах с редакторами сценария [25].

2.3.3 Решение проблемы избыточности хранилища

При использовании в основе хранилища стандартной структуры данных “Очередь” для хранения записываемых действий было замечено, что при каждом использовании функции записи очередного действия, создаётся новый объект параметра ввода и записывается в очередь. Такой подход приводит к чрезвычайно большим очередям наполненным одинаковыми действиями. Использование класса Input возможно только в рамках главного цикла компонента, итерации которого вызываются шестьдесят раз в секунду. Предположим, что мы включили запись действий игрока в игру, но по условиям определённым игрой необходимо будет прождать какое-то количество секунд. И каждую секунду в очередь будут записываться шестьдесят одинаковых значений ввода, что будет разду-

вать до невероятных размеров очередь, а такой её размер приведёт к аварийной остановки редактора Unity.

В качестве решения данной проблемы, к стандартной структуре данных был применён метод сжатия без потерь RLE [26]. При каждом новом вводе, если в начале очереди стоит точной такой же ввод, то записывается лишь количество его повторений. Если же в начале очереди другой ввод, то происходит простая постановка в очередь. Результат данного процесса проиллюстрирован на рис. 2.2.

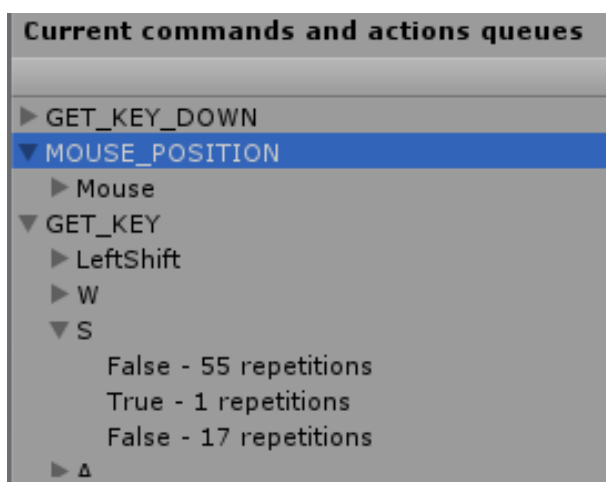


Рисунок 2.2 — Пример работы RLE-сжатия очереди записываемых действий

2.3.4 Система сохранения и загрузки хранилища

Система сохранения и загрузки является РАС компонентом, она используется напрямую только хранилищем записанных действий. Интерфейс `IATFActionStorageSaver` (см. приложение А.4) отвечает за реализацию системы сохранения и загрузки хранилища в потенциально любой носитель информации.

Реализация интерфейса (см. приложение А.4) 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 реали-

зации похоже на поведение недетерминированного автомата, по уже вышеупомянутым причинам.

2.3.5 Система упаковки данных хранилища

Для обеспечения гибкости использования алгоритмов сериализации и сжатия хранилища данных была реализована система упаковки реализующая интерфейс IAtfPacker (см. приложение А.13). Данная система позволяет не вмешиваясь в методы системы сохранения и загрузки менять данные для удобства сериализации и потенциальной оптимизации.

В текущей своей реализации система упаковки использует жадный алгоритм перевода данных в сериализуемую структуру классов, что призвано упростить чтение исходного кода системы сохранения и загрузки хранилища действий.

2.3.6 Система интеграции в готовую кодовую базу

Дополнительный функционал системы интеграции позволяет автоматизировать процессы интеграции ATF в сторонний проект путём поиска и замены текста исходного кода всей базы или конкретных выбранных скриптов (файлов исходного кода компонентов Unity) с помощью регулярного выражения, которое определяет использование класса Input в исходном коде. Система интеграции реализует интерфейс IAtfIntegrator (см. приложение А.12).

Данная система может функционировать в двух режимах:

- Автоматический – система сканирует всю кодовую базу стороннего проекта, находя файлы исходного кода с расширением “.cs” и по очереди, с помощью жадного перебора, выполняет замену имени класса Input на ATFInput.Instance, при этом не создавая новых файлов с интегрированным перехватчиком.

- Полуавтоматический – система наполняется путями к файлам исходного кода вручную с помощью вспомогательного окна, а затем задействует подстановку, как и в случае с автоматическим режимом, но можно выбрать: создать дополнительный файл с суффиксом ATF, где будет уже

внедрён перехватчик, или же произвести замену без создания дополнительного файла.

3 Человеко-машинный интерфейс

В рамках данной работы человеко-машинный интерфейс представлен в виде пользовательских окон управления в Unity Editor, а также в виде онлайн-ресурса посвящённого обучению и модификации описываемой работы.

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 Окно 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.

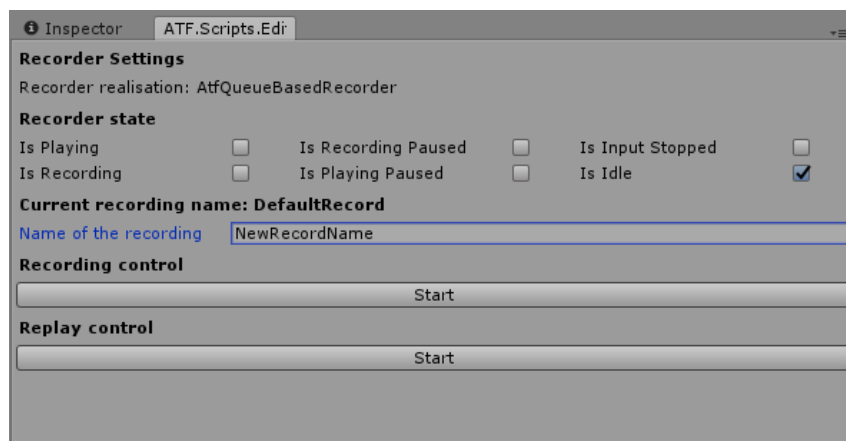


Рисунок 3.1 — Editor UI для проигрывателя

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.

Также, была создана функция остановки ввода. Для этого необходимо нажать комбинацию клавиш “Ctrl+I”. Мотивация к созданию функционала по остановке ввода, была в необходимости, не используя мышку, в полной мере пользоваться ATF. Существует множество приложений визуализации и игр разработанных на платформе *Unity*, которые используют мышку во время исполнения. Без возможности останавливать ввод, было бы невозможно использовать любые окна управления ATF, оставляя при этом неизменным состояние приложения.

В дополнение к возможностям управления без мышки были добавлены новые комбинации клавиш (см. рис. 3.2).

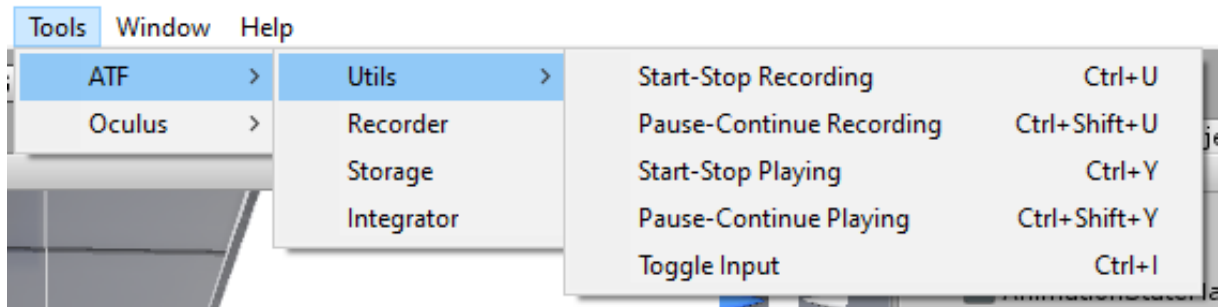


Рисунок 3.2 — Комбинации клавиш управления окном записи

Данные комбинации работают также как переключатели света. Другими словами первое нажатие активирует функцию, второе – выключает её.

- “Ctrl+U” – Включение или выключение записи.
- “Ctrl+Shift+U” – Пауза или продолжение записи.
- “Ctrl+Y” – Включение или выключение проигрывания.
- “Ctrl+Shift+Y” – Пауза или продолжение проигрывания.

3.2 Окно управления хранилища действий

Окно управления хранилищем (см. рис. 3.3) также состоит из трёх секций – информация о реализации, секции управления сохранением и загрузкой, а также зоны иллюстрации текущего состояния хранилища.

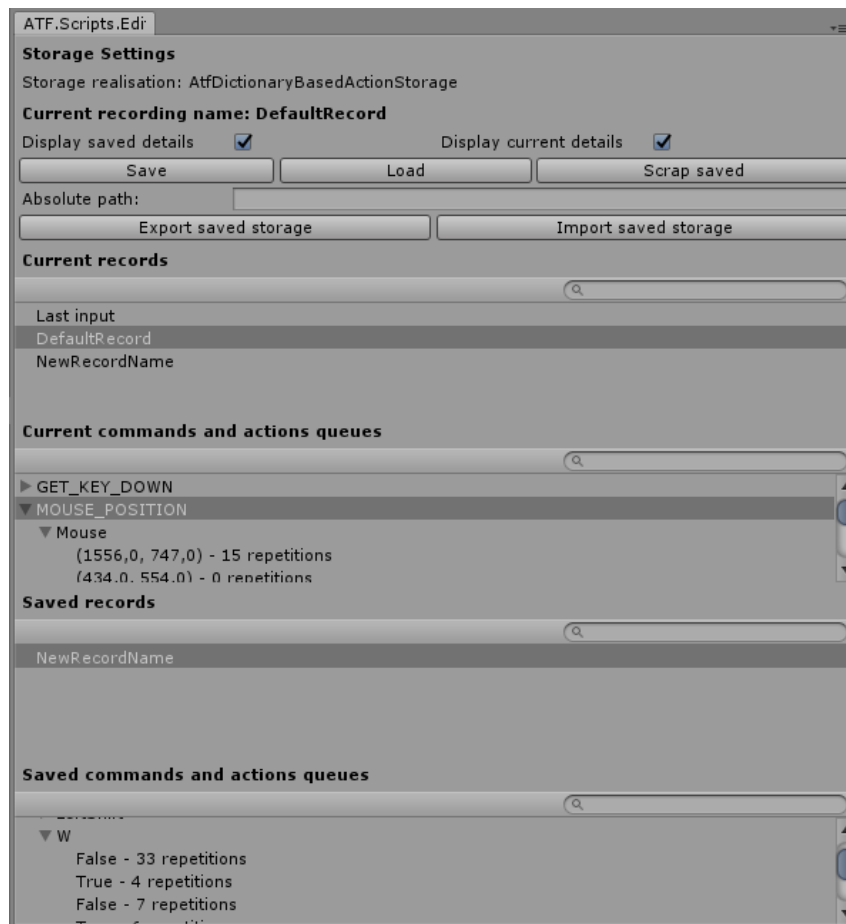


Рисунок 3.3 — Editor UI для хранилища действий

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);
- экспорт и импорт записей в пассивной зоне на внешний накопитель.

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.

Чтобы экспортировать всё, что находится в пассивной зоне, необходимо вписать абсолютный путь к файлу с расширением “.json”, где и будет располагаться сохранённое хранилище. Далее необходимо нажать на кнопку *Export saved storage*.

Для выполнения импорта хранилища, нужно проделать те же шаги, но путь указать к уже существующему файлу. Далее следует нажать на кнопку *Import saved storage*.

3.3 Окно управления интеграцией

Окно управления интеграцией состоит из трёх секций и работает в двух режимах интеграции и в двух режимах выбора путей. Первая секция – информация об используемой реализации, вторая – секция добавления и удаления путей к файлам исходного кода и третья – секция с отображением выбранных файлов и кнопками управления.

Находясь в первом режиме выбора путей (см. рис. 3.4), пользователь вводит пути к файлам исходного кода, в которые следует интегрировать ATF, вручную. Вводимый путь является относительным и его корень лежит в папке Assets, в директории проекта.

Второй режим выбора путей (см. рис. 3.5) позволяет упростить и ускорить сбор путей к файлам исходного кода. Для этого нужно отключить чекбокс *Manual path choosing*. После его отключения, файлы исходного кода можно выделять мышкой во вкладке Project в Unity Editor. Все выбранные файлы автоматически попадают в очередь на интеграцию.

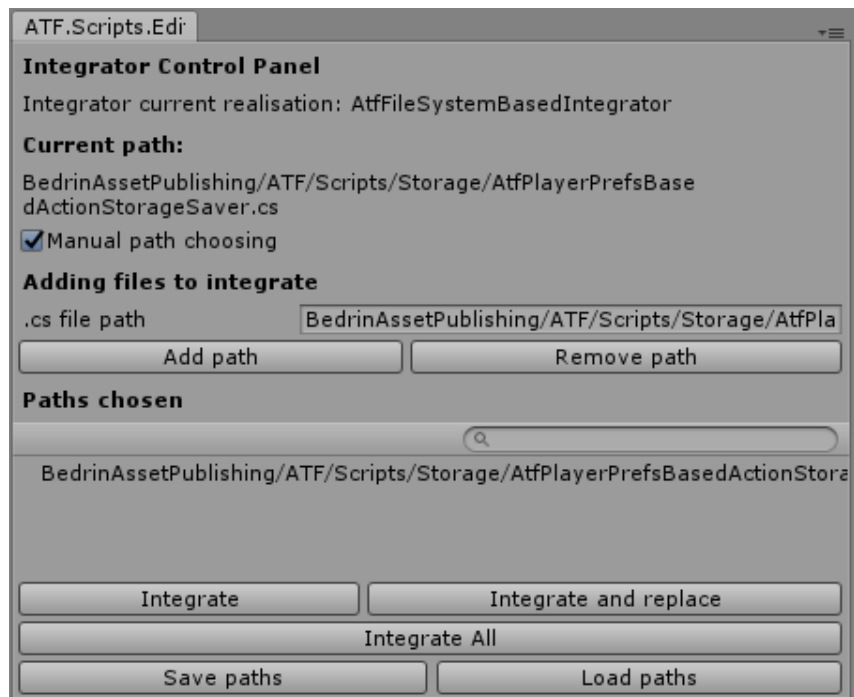


Рисунок 3.4 — Editor UI для управления интеграцией в ручном режиме

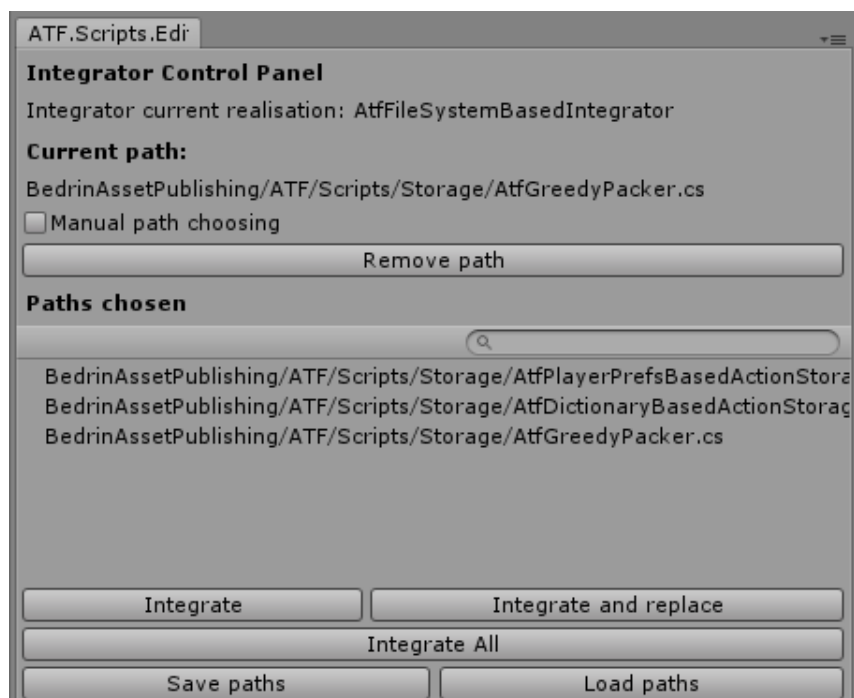


Рисунок 3.5 — Editor UI для управления интеграцией в автоматическом режиме

Удаление путей в обоих режимах происходит одинаково: выделение пути в списке и нажатие на кнопку *Remove path*. Стоит заметить, что, если не убрать полностью выделение во вкладке *Project*, курсор в виде текущего выбранного пути, отображаемого в секции *Current path*, будет

обращён на последний выделенный путь, из-за чего при невнимательном управлении можно случайно удалить не те пути, что были выбраны в списке.

Так как система интеграции поддерживает два режима работы: автоматический и полуавтоматический, при проектировании данного окна было решено выделить отдельные кнопки для обоих режимов:

— Кнопки *Integrate* и *Integrate and replace* отвечают за полуавтоматический режим и воздействуют на представленные в третьей секции конкретные файлы исходного кода.

— Кнопка *Integrate All* отвечает за использование автоматического режима.

Кнопки *Save paths* и *Load paths* отвечают за сохранение и загрузку представленных в третьей секции файлов исходного кода. Для облегчения работы с файлами, вместо конкретных файлов во второй секции было решено использовать пути файлов.

3.4 Онлайн-документация и способ доступа

Для лучшего понимания функционала разработанной группы ассетов была создана онлайн-документация [24]. Данный ресурс содержит в себе учебный материал не только для обучения пользованием ассетами, но и информацию о каждом интерфейсе основной системы и учебный материал, где рассказывается о том, как можно модифицировать разработанные ассеты, а также создавать новые.

В качестве платформы для создания онлайн-документации было решено использовать ReadTheDocs.org (RTD) с генератором документации Sphinx (см. рис. 3.6). Критерием к выбору платформы и генератора служила возможность бесплатного хостинга ресурса с адекватным на мнение автора именем хоста, лёгкость языка шаблонизатора у генератора, а также высокий уровень репутации ресурса.

Процесс развёртывания данной онлайн-документации на платформе RTD заключается всего в двух шагах: инициализация в сервисе хо-

стинга github-репозитория с исходным кодом статей на языке ReST и запрос на автоматическую сборку и развёртывание онлайн-документации.

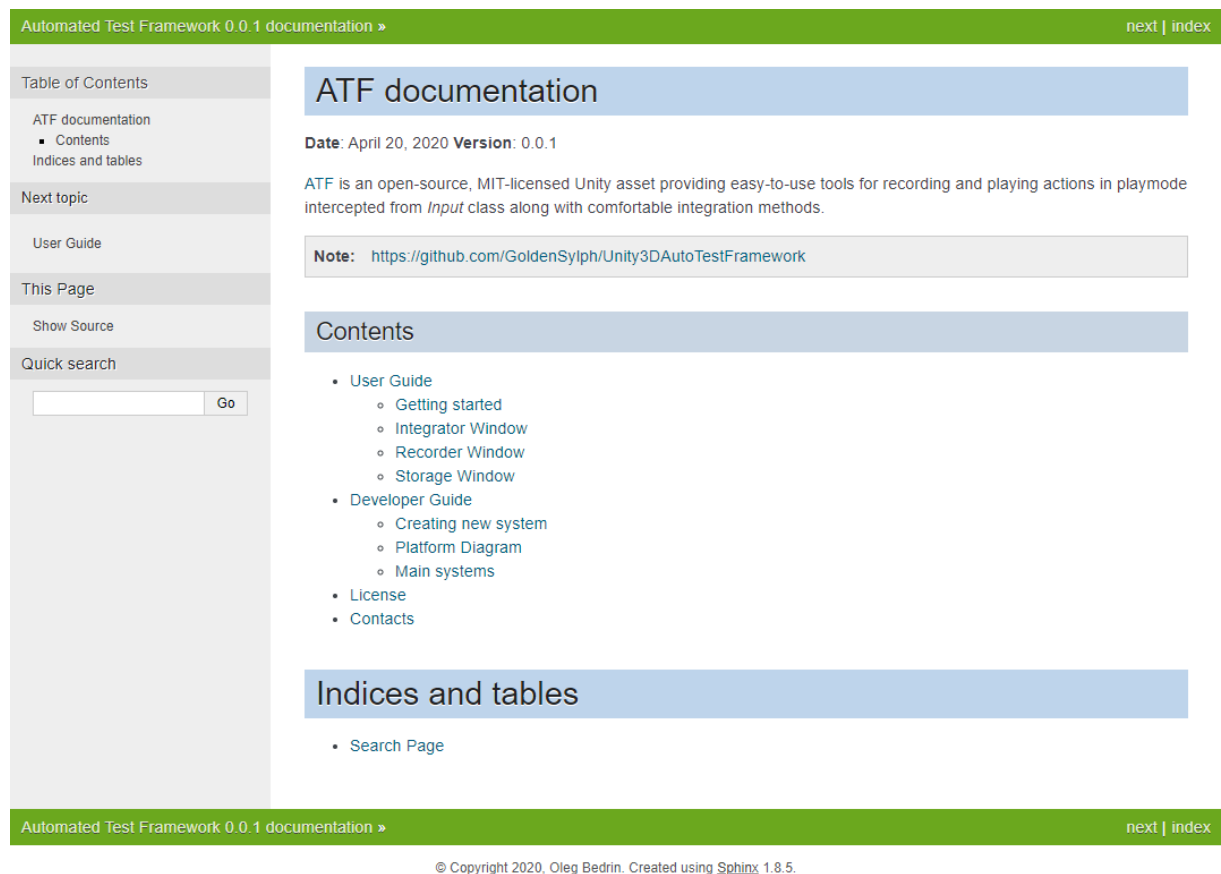


Рисунок 3.6 — Главная страничка онлайн-документации

Это было необходимо, чтобы не отпугнуть потенциального контрибьютора от изучения документации. Возможности контрибьютинга (совместной разработки с публикой) обусловлены тем, что лицензия результатов данной разработки создана на основе MIT.

В дополнение к созданной онлайн-документации была добавлена возможность быстрого поиска по статьям представленным в ней. В качестве языка шаблонов был использован reStructuredText – облегчённый язык разметки схожий по синтаксису с языком Markdown, однако позволяющий автоматически генерировать файлы PDF, HTML, ODT, LaTeX, а также формат презентаций S5.

Так как разработанная группа ассетов распространяется по лицензии MIT, доступ к исходному коду открыт и доступен на сайте [GitHub.com](https://github.com) [27].

В руководство для пользователей (разработчиков проекта, в который ATF был интегрирован) входит множество инструкций по использованию ATF (см. рис. 3.7).

Первая инструкция посвящена интеграции ATF в новый пустой проект. Там иллюстрируется способ доступа к свежему пакету с ATF, а также объясняется как подготовить сцену и где найти управляющие окна. Вторая инструкция демонстрирует интеграцию в существующий проект и аспекты использования статического или объектного доступа к перехватчику. Третья и четвертая инструкции иллюстрирует работу всех режимов перехватчика и окна записи соответственно. Здесь объясняется то, как можно использовать механизм остановки ввода, не нарушая состояния приложения во время исполнения. Пятая инструкция посвящена функционалу манипулирования хранилищем действий, описывая его зоны для просмотра содержания записи и функции его импорта и экспорта. Заключительная шестая инструкция объясняет как пользоваться комбинациями клавиш для управления записью.

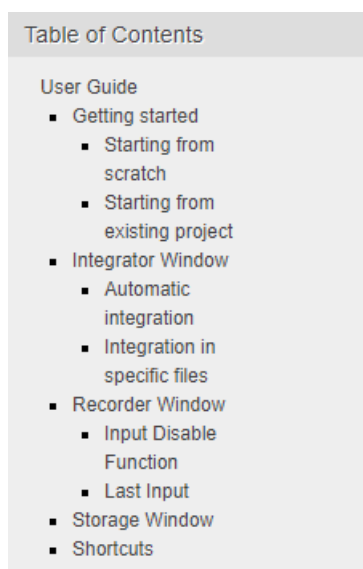


Table of Contents	
User Guide	
■	Getting started
■	Starting from scratch
■	Starting from existing project
■	Integrator Window
■	Automatic integration
■	Integration in specific files
■	Recorder Window
■	Input Disable Function
■	Last Input
■	Storage Window
■	Shortcuts

Рисунок 3.7 — Содержание руководства для пользователей

В руководство для контрибьюторов (см. рис. 3.8) входят обзорные статьи посвященные принципам построения кодовой базы ATF.

Первая статья посвящена созданию новой системы, которая бы автоматически инициализировалась на сцене, сразу же после инъекции зависимостей. Вторая статья посвящена обзору платформы ATF и тому, как организовано взаимодействие со стандартными системами *Unity*. Заключительная статья содержит описания всех интерфейсов основных систем ATF, включая комментарии по хранению данных и модификации.

Table of Contents
Developer Guide
▪ Creating new system
▪ Initializer
▪ DI Container
▪ Platform Diagram
▪ Main systems
▪ Recorder System
▪ Action Storage System
▪ Packer System
▪ Action Storage Saver System
▪ Integrator System

Рисунок 3.8 — Содержание руководства для контрибьюторов

4 Практическое применение

В качестве доказательства применимости разрабатываемой, в рамках данной работы, группы ассетов была проведена интеграция в проект виртуальной био-технологической лаборатории DML BioLab. Было осуществлено портирование кодовой базы ATF с версии движка Unity 2019.3.6f1 на версию Unity 2017.3.1f1.

Был создан тест, успешное прохождение которого и завершало бы доказательство. Критерием успешности прохождения данного теста являлся задокументированный в виде видеофайла факт успешной записи и прохождения первого шага в первом сценарии DML BioLab.

Первый шаг в выбранном сценарии составляет следующую группу действий: Взятие баночки PBS+20Tween (буферного раствора) и заполнение её содержимым ванночки для набора, с помощью диспенсеров (многоканальных или одноканальных капиллярных заборных устройств для жидкостей).

4.1 Портирование на Unity 2017.3.1f1

Первая стадия портирования состоит в подготовке проекта DML BioLab к импортированию, а именно создаётся резервная копия проекта, которая отделяется от системы контроля версий Perforce.

Так как исходный код ATF написан с использованием .NET Framework 4.6 и версией языка CSharp 7.0, а исходный код проекта виртуальной лаборатории DML BioLab написан на .NET Framework 3.5 Stable и версией языка 6.0, необходимо было перевести все тела методов и параметров перехватчика с тел лямбда-выражений, на обыкновенные блоки (см. пример в приложении А.14).

В то же время, перевод с версии .NET фреймворка 3.5 на 4.6 не составил труда, и был осуществлён путём изменения соответствующей конфигурации (см. рисунок 4.1) Player в Unity Editor.

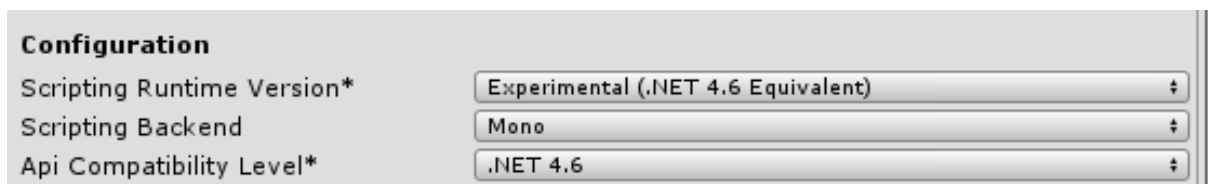


Рисунок 4.1 — Конфигурация среды исполнения Player

4.2 Интеграция в симулятор VRTK

Проект виртуальной био-технологической лаборатории, в качестве менеджера VR устройств (контроллеров и шлема), использует фреймворк VRTK [28]. В процессе разработки приложения с использованием данного фреймворка, можно обойтись и без настоящего VR устройства, используя встроенный его симулятор, который автоматически заменяется на реальное VR устройство при его доступности. Однако стоит заметить, что если не использовать симулятор, то в качестве метода взятия ввода используется система интерфейсов-маркеров, что не покрывается разработанным перехватчиком.

Данный симулятор использует в качестве устройств ввода клавиатуру и мышь, а в качестве метода взятия ввода – комбинацию из `Input` и `BaseInput` классов. Описанная комбинация позволяет выполнить автоматическую интеграцию группы ассетов ATF.

Далее выясняется, что некоторые из методов класса `Input` по взятию ввода не присутствуют в классе `BaseInput` в статичной или в объектной доступности. Так как класс `Input` является `sealed` (недоступен для наследования) и в языке CSharp невозможно множественное наследование классов, а для полноценной работы перехватчика необходимо было использовать методы одновременно из классов `Input`, `BaseInput` и `MonoSingleton<ATFInput>`, было принято решение использовать структурный паттерн `Simulated Multiple Inheritance (SMI)` (см. рисунок 4.2).

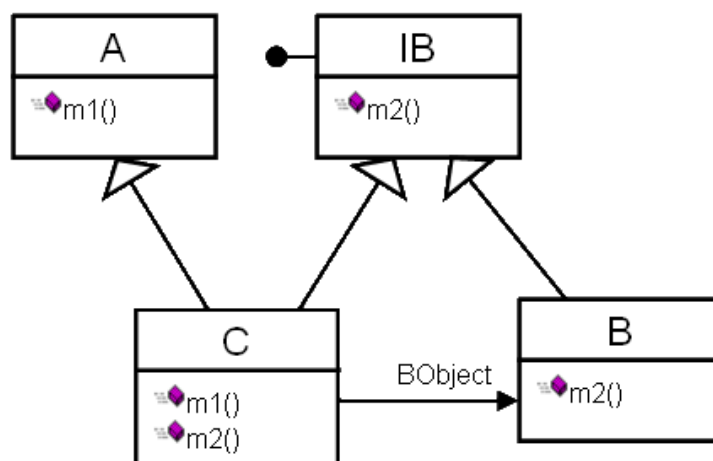


Рисунок 4.2 — Диаграмма классов для паттерна SMI

В качестве А класса на рисунке 4.2 используется класс `MonoSingleton<AtfInput>`, в качестве IB взят `IBaseInput` интерфейс, который объединяет интерфейсы `BaseInput` класса и `Input` класса, а С класс является классом-арбитром `AtfInput`. Экземпляр класса `BaseInput` загружается в память на этапе инъекции зависимостей. Данное нововведение позволяет обращаться ко всем, включая объектные методы, по параметру `AtfInput.Instance`, в то же время все статичные методы остались по-умолчанию доступны по ссылке на класс `AtfInput`.

Необходимость создания статического и объектного доступа к методам перехватчика была продиктована именно вышеописанной проблемой.

В результате интеграции был успешно проведён тест, результаты которого находятся в открытом доступе на сайте [YouTube.com](https://www.youtube.com) [29].

ЗАКЛЮЧЕНИЕ

Согласно цели и задачам поставленным в начале данной выпускной квалификационной работы были получены результаты, которые позволяют не только разрешить указанную проблему формального исполнителя для регрессионного тестирования, но и в дополнение применить эффективную [30] модель дальнейшего развития проекта в качестве open-source ПО.

В процессе исследования популярности выбранного способа запрашивания ввода на платформе *Unity* были созданы несколько версий spider-программ, однако предпочтения были отданы одной, которая в своём ядре разделяет процесс сбора данных с помощью параллельных вычислений. Также, в качестве приёма доказательства поставленной гипотезы была адаптирована под логическую функцию тождественности цепочка статистических выводов, что позволило успешно завершить исследование и приступить к проектированию.

Краткий анализ класса совместимых приложений позволил выделить основные приёмы разработки, которые были использованы для аналогов с других платформ. Были определены конкретные шаблоны проектирования и стратегические направления в развитии архитектуры группы ассетов имеющие фундаментальные основания в объектно-ориентированном программировании.

Реализация системы управления зависимостями и автоматический инициализации перехватчика ввода позволило очистить исходный код основных систем от сквозного кода благодаря использованию приёмов программной рефлексии (обращению к единицам компиляции как к классам и объектам) и возможностям языка программирования CSharp создавать атрибуты классов (аннотативные сущности несущие вспомогательную информацию).

Главным выводом по факту создания основных систем группы ассетов, является то, что в большинстве своём исходный код подчиняется основным принципам проектирования SOLID характерным для enterprise-проектов в объектно-ориентированном программировании. Это позволяет производить масштабирование функционала системы используя всего

два действия: создание .cs файла и аннотирование специальными атрибутами декларацию и зависимости класса добавляемой системы. Далее, при переходе Unity Editor в Play Mode разработанная группа ассетов сама настроит новый функционал.

Благодаря выбранной архитектуре и построению зависимостей над абстрактными сущностями (интерфейсами, абстрактными классами и т.д.), все записи действий могут быть экспортированы и импортированы согласно двум основным системам: системе упаковки и системе сохранения и загрузки хранилища действий. Каждая из этих систем использует сигнатуры методов интерфейсов, поэтому создание и модификация этих систем не приведёт к лавинному эффекту с правками кода.

Реализация пользовательских окон управления данной группой ассетов для Unity Editor позволило быстро взаимодействовать с перехватом и записью действий без создания дополнительной кодовой базы, как если бы результаты данной работы были представлены лишь в виде Application Programmable Interface (API).

Говоря о полноте решений поставленных задач, конечно нельзя не заметить, что хоть и класс Input является самым распространённым среди других способов взятия ввода, огромная часть приложений использует и другие способы, а также их комбинации, в зависимости от целевой платформы сборки. Из этого следует, что дальнейший вектор развития данной работы указывает на охват большего количества способов взятия ввода.

В дополнение, развитие платформы *Unity* тоже не стоит на месте, способ Input System описанный в начале второй главы активно развивается и постепенно станет доминировать при переходе с объектно-ориентированного подхода к построению Unity-приложений к ориентированному на данные. Данный прогноз исходит из того, что если сейчас построить Unity-приложение с помощью уже существующих ориентированных на данные приёмов, то использование других способов запрашивания ввода станет технически невозможным.

Результат данной работы рекомендуется использовать в проектах с длинным временем прохождения и с линейным или частично линейным

потоком действий. Также, благодаря активному развитию, следует осторожно использовать данную группу ассетов, так как ни один разработчик не застрахован от совершения алгоритмических ошибок.

Стоит отметить, что результаты данной работы были опубликованы на официальном магазине платформы *Unity Asset Store* [31]. Также был сделан доклад на конференции *Software Engineering Conference Russia (SECR 2019)* [32] в городе Санкт-Петербург, с последующей подачей заявки на публикацию в журнал “Программная инженерия” [33].

Исходный код работы размещён по ссылке:
<https://github.com/GoldenSylph/Unity3DAutoTestFramework>

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. Abramov V.D., Kugurakova V.V., Rizvanov A.A., Abramskiy M.M., Manakhov N.R., Evstafiev M.E., Ivanov D.S. Virtual Biotechnological Lab Development // BioNanoScience. 2017. №7. P. 363-365.

10. Unity – Manual: Input // Unity Manual URL: <https://docs.unity3d.com/Manual/Input.html> (дата обращения: Апрель 12, 2020).
11. Unity – Scripting API: IPointerDownHandler // Unity Scripting API URL: <https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EventSystems.IPointerDownHandler.html> (дата обращения: Апрель 12, 2020).
12. Quick start guide // Unity Input System Manual URL: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/QuickStartGuide.html> (дата обращения: Апрель 12, 2020).
13. Брюс П., Брюс Э. Практическая статистика для специалистов Data Science. СПб.: БХВ-Петербург, 2018. 304 с.
14. 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.
15. Naoyu 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.
16. Blumer A., Ehrenfeucht A., Hasler D., Warmuth M.K. Occam’s Razor // Information Processing Letters 24. 1987. №6. P. 377-380.
17. High-performance multithreaded stack of unity information-oriented technologies // unity.com URL: <https://unity.com/ru/dots> (дата обращения: Июль 19, 2019).
18. Unity Foundation – Introduction to ECS // unity3d.com URL: <https://unity3d.com/ru/learn/tutorials/topics/scripting/introduction-ecs> (дата обращения: Июль 19, 2019).
19. Schmidt D., Stal M., Rohnert H., Buschmann F. Pattern-Oriented Software Architecture // Patterns for Concurrent and Networked Objects. 2001. №2. P. 109-140.
20. Architectural pattern “Broker” // cs.uno.edu URL: <http://cs.uno.edu/jaime/Courses/4350/broker.ppt> (дата обращения: Июль 19, 2019).

21. Coutaz J. PAC: An object oriented model for implementing user interfaces // ACM SIGCHI Bulletin. 1987. №19. P. 37-41.
22. Scrapy – A Fast and Powerful Scraping and Web Crawling Framework // scrapy.org URL: <https://scrapy.org/> (дата обращения: Апрель 17, 2020).
23. JSR 299: Contexts and Dependency Injection for the Java™ EE platform // jcp.org URL: <https://jcp.org/en/jsr/detail?id=299> (дата обращения: Июль 19, 2019).
24. ATF documentation // unity3dautotestframework.readthedocs.io URL: <https://unity3dautotestframework.readthedocs.io/en/latest/?badge=latest> (дата обращения: Апрель 17, 2020).
25. Kugurakova V. V., Abramov V. D., Abramskiy M. M., Manakhov N., Maslaviev A. Visual editor of scenarios for virtual laboratories // 10th International Conference on Developments in eSystems Engineering (DESE 2017). Paris: IEEE, 2017. P. 242-247.
26. Смит С. Форматы сжатия данных // Электронные компоненты. 2009. №8. С. 83-87.
27. GoldenSylph / Unity3DAutoTestFramework // github.com URL: <https://github.com/GoldenSylph/Unity3DAutoTestFramework> (дата обращения: Май 22, 2020).
28. VRTK - Virtual Reality Toolkit // vrtoolkit URL: <https://vrtoolkit.readme.io/> (дата обращения: Май 18, 2020).
29. Automated Test Framework Demo with DML BioLab // YouTube URL: <https://youtu.be/YOUdXHIUIW4> (дата обращения: Май 21, 2020).
30. Levine S.S., Prietula M.J. Open Collaboration for Innovation: Principles and Performance // Organization Science. 2013. №5. P. 1287-1571.
31. Automated Test Framework // Asset Store URL: <https://assetstore.unity.com/packages/tools/utilities/automated-test-framework-167509> (дата обращения: Май 29, 2020).
32. Система автоматизации функционального тестирования для приложений на игровом движке Unity - Доклад SECR 2019 СПб // SECR 2019 URL: <https://2019.secrus.org/program/submitted-presentations/>

unity-engine-application-functional-testing-automation-based-on-native-tools/
(дата обращения: Май 29, 2020).

33. Авторам издательства "Новые технологии" // novtex.ru URL:
<http://www.novtex.ru/autor.htm> (дата обращения: Май 29, 2020).

ПРИЛОЖЕНИЕ А Листинги

Листинг А.1 — Исходный код класса обработки веб-страниц проектов с сайта GitHub.com

```
# -*- coding: utf-8 -*-
import scrapy
import requests
import json
import base64
import re
import time
import pandas as pd
class GithubSpiderSpider(scrapy.Spider):

    name = 'github-spider'
    main_data = pd.DataFrame(columns=['repo_name',
        'method_invocations'])
    file_name = 'main_data_most_forks.csv'
    config = False
    recently_updated_url =
    'https://github.com/search?o=desc&p={ }
        &q=unity+language%3AC%23&s=updated&type=Repositories'
    most_stars_url =
    'https://github.com/search?o=desc&p={ }
        &q=unity+language%3AC%23&s=stars&type=Repositories'
    most_forks_url =
    'https://github.com/search?o=desc&p={ }
        &q=unity+language%3AC%23&s=forks&type=Repositories'
    best_match_url =
    'https://github.com/search?p={ }
        &q=language%3Acsharp+unity&type=Repositories'
```

```

def start_requests(self):
    urls = [self.most_forks_url.format(i + 1) for i
            in range(101)]
    for url in urls:
        yield scrapy.Request(url=url,
                             callback=self.parse)

def auth_data(self):
    if self.config:
        return (self.config['login'],
                self.config['password'])
    with open('github_config.json', 'r',
              encoding='utf-8') as github_file:
        self.config = json.load(github_file)
    return (self.config['login'],
            self.config['password'])

def get(self, url):
    result = requests.get(url,
                          auth=self.auth_data())
    if result.status_code == 403:
        header_name = 'X-RateLimit-Reset'
        if header_name in result.headers:
            reset_time =
                float(result.headers[header_name])
            sleep_time = reset_time - time.time()
            if sleep_time < 0:
                self.logger.info('Sleep time is
                                negative, adjusting to 1...')
                sleep_time = 1
            self.logger.info('Exceeded rate limit.
                             Waiting for: {}'.format(sleep_time))
            time.sleep(sleep_time)

```

```

        result = requests.get(url, auth=auth_data)
    else:
        return False
    self.logger.info('{} status:
        {}'.format(result.headers,
        result.status_code))
    return result

def get_json(self, response):
    try:
        return response.json()
    except:
        self.logger.info(traceback.format_exc())
        return False

def closed(self, reason):
    self.logger.info('Done. Specified reason:
        {}'.format(str(reason)))

def parse(self, response):
    repo_names = [r[1:] for r in
        response.xpath('//div[@class="f4
        text-normal"]/a/@href').extract()]
    for repo_name in repo_names:
        self.logger.info('Working with repo:
            {}'.format(repo_name))
        code_search_url =
            'https://api.github.com/search/code?
            q=Input+in:file+language:csharp+repo:{}'
            .format(repo_name)
        code_response = self.get(code_search_url)
        if not code_response:

```



```

        self.logger.info('something wrong with
                           header, continue...')
        continue
code_data = self.get_json(code_response)
if not code_data:
    continue
method_invocations_count = 0
code_urls = [code_item['url'] for code_item in
              code_data['items']]
for code_url in code_urls:
    self.logger.info('Source url:
                      {}'.format(code_url))
    source_details_response = self.get(code_url)
    if not source_details_response:
        self.logger.info('something wrong with
                           header in source details response,
                           continue...')
        continue
    source_details_data =
        self.get_json(source_details_response)
    if not source_details_data:
        continue
    if 'message' in source_details_data:
        self.logger.info(source_details_data['message'])
        continue
    try:
        source =
            base64.b64decode(source_details_data['content']
                             .encode('utf-8')).decode('utf-8')
    except:
        self.logger.info(traceback.format_exc())
        continue
method_invocations_count_in_source =

```

```

        len(re.findall(r'[(\[(\s,=\+\\-\*/\?&|]Input\.',
            source))
    method_invocations_count +=
        method_invocations_count_in_source
    new_data = pd.DataFrame({'repo_name':
        repo_name, 'method_invocations':
        method_invocations_count}, index=[0])
    self.main_data =
        self.main_data.append(new_data,
            ignore_index=True)
    self.logger.info('Data
        gathered:\n{}'.format(new_data))
    self.main_data.to_csv(self.file_name)

```

Листинг A.2 — Утилита для сбора собранных spider-программой данных
воедино

```

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", context="talk")
whole_data = pd.DataFrame(columns=['repo_name',
    'method_invocations', 'filter'])
metadata = [('main_data_best_match.csv',
    'best_match'),
    ('main_data_most_stars.csv', 'most_stars'),
    ('main_data_most_forks.csv', 'most_forks'),
    ('main_data_recently_updated.csv',
    'recently_updated')]
for data in metadata:
    new_data = pd.read_csv(data[0], index_col=0)
    new_data['filter'] = data[1]
    whole_data = whole_data.append(new_data,
        ignore_index=True)

```

```

whole_data['filter'] = whole_data['filter'] +
    whole_data.duplicated(subset='repo_name').apply(str)
print(whole_data)
whole_data.to_csv('main_data.csv')
sns.scatterplot(x=whole_data.index,
    y='method_invocations', data=whole_data,
    hue='filter')
plt.show()
plt.clf()

```

Листинг A.3 — Утилита для проведения бутстрапа и параллельных расчётов границ интервала среднего

```

import math as m
import pandas as pd
import multiprocessing as mp
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib

def stat(x):
    return round(x.mean(), 3)

if __name__ == '__main__':

    whole_data = pd.read_csv('main_data.csv')

    R = 99509
    n = len(whole_data)
    confidence = 90
    specific_data = whole_data['method_invocations']

    process_count = mp.cpu_count()

```

```

chunk_size = 1000
print('Processes: {}, chunksize:
      {}'.format(process_count, chunk_size))

with mp.Pool(process_count) as pool:
    boot_data =
    pd.Series(pool.map(stat,
        [specific_data.sample(n, replace=True) for i
         in range(R)],
        chunksize=chunk_size))

boot_data_len = len(boot_data)
cut_percent = m.ceil((1 - confidence / 100) / 2)
cut = round(boot_data_len * cut_percent / 100)
middle_cuts = [cut, boot_data_len - cut]
tail_cut = boot_data_len - cut
print('boot_data_len: {}, cut_percent: {}, cut:
      {}, middle_cuts: {}, tail_cut:
      {}'.format(boot_data_len, cut_percent, cut,
        middle_cuts, tail_cut))
head = boot_data[:cut]
middle = boot_data[middle_cuts[0]:middle_cuts[1]]
tail = boot_data[tail_cut:]
main_data = pd.DataFrame(columns=['stat', 'kind'])
main_data =
    main_data.append(pd.DataFrame({'stat': head,
    'kind': 'head'}), ignore_index=True)
main_data =
    main_data.append(pd.DataFrame({'stat': middle,
    'kind': 'interval'}), ignore_index=True)
main_data =
    main_data.append(pd.DataFrame({'stat': tail,
    'kind': 'tail'}), ignore_index=True)

```

```

sns.jointplot(x=main_data.index,
              y=main_data['stat'], kind='hex')
plt.axhline(y=main_data.at[cut, 'stat'],
            color='r')
plt.axhline(y=main_data.at[tail_cut, 'stat'],
            color='r')
plt.text(cut, main_data.at[cut, 'stat'] + 0.01,
         'L: {}'.format(main_data.at[cut, 'stat']),
         fontsize=12)
plt.text(cut, main_data.at[tail_cut, 'stat'] +
         0.01, 'U: {}'.format(main_data.at[tail_cut,
         'stat']), fontsize=12)
plot_backend = matplotlib.get_backend()
mng = plt.get_current_fig_manager()
if plot_backend == 'TkAgg':
    mng.resize(*mng.window.maxsize())
elif plot_backend == 'wxAgg':
    mng.frame.Maximize(True)
elif plot_backend == 'Qt4Agg':
    mng.window.showMaximized()
plt.savefig('experiment.png')
plt.show()

```

Листинг А.4 — 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 A.5 — 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 A.6 — 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.7 — 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.8 — 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 A.9 — Бизнес-логика системы симуляции и перехвата ввода

```

[Injectable]
[Serializable]
[AtfSystem]
public class ATFInput : MonoSingleton<T>, IBaseInput
{
    [Inject(typeof(ATFQueueBasedRecorder))]
    public static readonly IATFRecorder RECORDER;
    [Inject(typeof(ATFDictionaryBasedActionStorage))]
    public static readonly IATFActionStorage STORAGE;
    [Inject(typeof(BaseInput))]
    public static readonly BaseInput BASE_INPUT;
    ...
    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>(T realInput,
    FakeInput fakeInputKind, object
    fakeInputParameter = null)
{
    if (fakeInputParameter == null)
    {
        fakeInputParameter = "EMPTY FAKE INPUT
            PARAMETER";
    }
    dynamic result =
        RealOrFakeInputOrRecord(realInput,
            fakeInputParameter, fakeInputKind);
    if (result is string)
    {
        Debug.Log(result);
    }
    return result;
}
public static bool anyKeyDown =>
    Intercept(Input.anyKeyDown,
        FakeInput.ANY_KEY_DOWN, false);
...
}

```

Листинг А.10 — Интерфейс модуля записи

```

using ATF.Scripts.Helper;

namespace ATF.Scripts.Recorder
{

```

```

public interface IAtfRecorder :
    IAtfGetSetRecordName
{
    bool IsRecording();
    bool IsPlaying();
    bool IsRecordingPaused();
    bool IsPlayPaused();
    bool IsInputStopped();
    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 SetInputStopped(bool value);
    void Record(FakeInput kind, object input,
        object fakeInputParameter);
    object GetLastInput(FakeInput kind, object
        fakeInputParameter);
    void SetLastInput(FakeInput kind, object
        realInput, object fakeInputParameter);
}
}

```

Листинг A.11 — Пользовательский модуль ввода для ATFInput

```

namespace ATF
{

```

```

public class ATFStandaloneInputManager :
    StandaloneInputModule
{
    protected override void Start()
    {
        base.Start();
        m_InputOverride = AtfInput.BASE_INPUT;
        DependencyInjector
            .InjectType(m_InputOverride.GetType());
    }
    ...
}

```

Листинг A.12 — Интерфейс модуля интеграции в готовую кодовую базу

```

using System.Collections.Generic;
using ATF.Scripts.Helper;

namespace ATF.Scripts.Integration.Interfaces
{
    public interface IAtfIntegrator :
        IAtfGetSetRecordName
    {
        void SetUris(IEnumerable<string> filePaths);
        void Integrate();
        void IntegrateAndReplace();
        void IntegrateAll();
        void SaveUris();
        IEnumerable<string> LoadUris();
    }
}

```

Листинг A.13 — Интерфейс модуля упаковки хранилища данных

```

using System.Collections.Generic;
using ATF.Scripts.Storage.Utills;
using UnityEngine;
namespace ATF.Scripts.Storage.Interfaces
{
    public interface IAtfPacker
    {
        List<Record> Pack(Dictionary<string,
            Dictionary<FakeInput, Dictionary<object,
            AtfActionRleQueue>>> input);
        Dictionary<string, Dictionary<FakeInput,
            Dictionary<object, AtfActionRleQueue>>>
            Unpack(Slot slot);
        string ValidatePacked(List<Record> packed);
    }
}

```

Листинг A.14 — Пример изменения тела метода и параметра класса Input
в классе-арбитре AtfInput

```

//CSharp 7.0
public static bool simulateMouseWithTouches
{
    get => Intercept(Input.simulateMouseWithTouches,
        FakeInput.SIMULATE_MOUSE_WITH_TOUCHES,
        "Simulate mouse with touches");
    set => Input.simulateMouseWithTouches = value;
}

```

```

//CSharp 6.0
public static bool simulateMouseWithTouches
{
    get { return
        Intercept(Input.simulateMouseWithTouches,

```

```

        FakeInput.SIMULATE_MOUSE_WITH_TOUCHES,
        "Simulate mouse with touches"); }
    set { Input.simulateMouseWithTouches = value; }
}

```

...

```

//CSharp 7.0
public static bool GetMouseButton(int button) =>
    Intercept(Input.GetMouseButton(button),
        FakeInput.GET_MOUSE_BUTTON, button);

```

```

//CSharp 6.0
public static bool GetMouseButton(int button)
{
    return Intercept(Input.GetMouseButton(button),
        FakeInput.GET_MOUSE_BUTTON, button);
}

```

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())
end procedure

procedure INTERCEPT(RealInput, FakeInput)
    dynamic result := RealOrFakeInputOrRecord(RealInput, FakeInput)
    if result is string then
        throw new Exception($"Deserialization failed: {result}")
    end if
    return result
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
```
