**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 6](#_Toc515928073)

[ВВЕДЕНИЕ 7](#_Toc515928074)

[1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc515928075)

[1.1 Анализ предметной области 9](#_Toc515928076)

[1.2 Автоматизация создания игровых сценариев 9](#_Toc515928077)

[1.3 Автоматизация создания уровней и работы с ресурсами 10](#_Toc515928078)

[1.4 Автоматизация процесса создания анимации 11](#_Toc515928079)

[1.5 Автоматизация работа с локализацией 13](#_Toc515928080)

[1.6 Постановка задачи 13](#_Toc515928081)

[2 ОБЗОР ТЕХНОЛОГИЙ 15](#_Toc515928082)

[2.1 Unity3D 15](#_Toc515928083)

[2.2 Платформа Mono 20](#_Toc515928084)

[2.3 Платформа .Net Framework 23](#_Toc515928085)

[3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ 27](#_Toc515928086)

[3.1 Общая архитектура 27](#_Toc515928087)

[3.2 Характеристика пользовательских ресурсов 28](#_Toc515928088)

[3.3 Инструмент автоматизации создания анимации 40](#_Toc515928089)

[3.4 Архитектура редактора уровней 42](#_Toc515928090)

[3.5 Архитектура редактора игровых сценариев 47](#_Toc515928091)

[3.6 Этапы разработки 50](#_Toc515928092)

[4 ТЕХНИКО–ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА 52](#_Toc515928093)

[4.1 Краткая характеристика проекта 52](#_Toc515928094)

[4.2 Расчет затрат на разработку программного обеспечения 53](#_Toc515928095)

[4.3 Расчет стоимостной оценки результата 62](#_Toc515928096)

[4.4 Выводы по технико–экономическому обоснованию 66](#_Toc515928097)

[ЗАКЛЮЧЕНИЕ 68](#_Toc515928098)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 69](#_Toc515928099)

[ПРИЛОЖЕНИЕ А 70](#_Toc515928100)

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

В настоящей пояснительной записке применяются следующие определения и сокращения.

**Атлас** – одна текстура, содержащая в себе несколько изображений.

**Виджет** – примитив графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия.

**Локализация** – адаптация программного продукта к языковым и культурным особенностям некоторых регионов.

**Префаб** – шаблон объекта в виде набора компонентов и их настроек.

**Сериализация** – это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.

**Спрайт** – графический объект в компьютерной графике.

**Скриптовый язык** – язык сценариев – высокоуровневый язык сценариев, кратких описаний действий, выполняемых системой. Сценарий – программа, имеющая дело с готовыми программными компонентами.

**AOT** – Ahead-of-time Compilation - это вид транслятора, который использует метод компиляции перед исполнением.

**API** – Application Programming Interface – интерфейс программирования приложений.

**CIL** – Common Intermediate Language – низкоуровневый, язык программирования, реализующий спецификацию CLI.

**CLI** -Common Language Infrastructure – спецификация, описывающая исполняемый код и среду исполнения.

**CLR** – Common Language Runtime – среда исполнения языка CIL.

**JIT** – Just-in-time Compilation - компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

**JSON** – JavaScript Object Notation – текстовый формат обмена данными, основанный на языке JavaScript.

**LоС** – Lines of Code – количество строк кода.

# ВВЕДЕНИЕ

В последнее время индустрия разработки игр претерпела крупные изменения. Компании, занимающиеся разработкой игр, перешли от внутренних разработок, игровых движков, к унифицированным инструментам, предоставляемым сторонними компаниями. Наиболее популярные актуальные игровые движки: Unity3d, Unreal Engine, CryEngine, имеют бесплатные версии, доступные для большинства современных платформ, поставляются в виде отдельного приложения, объединяющего в рамках одного интерфейса целый набор инструментов для разработки.

Другим важным изменением для индустрии стал стремительный рост рынка игр, во многом связанный с появлением новых игровых платформ: социальные сети, мобильные устройства, чаты, новые поколения игровых приставок [1]. Несмотря на быстрый рост пользовательской базы, рынок игровых приложений стал еще более конкурентным, так как значительно выросло и число выпускаемых проектов, а значит, очень важно оперативно реагировать на изменения рынка и быстро выпускать проекты. Поэтому задача автоматизации различных этапов работы остается актуальной.

Современные игровые движки используют компонентную архитектуру, что позволило со временем создать внутри себя еще один рынок. Рынок небольших компонентов – средств автоматизации различных задач. Unity3d и Unreal, имеют встроенный магазин компонентов, включая различные графические ресурсы, системы пользовательского интерфейса, системы работы с ресурсами и другие узкоспециализированные средства. Однако этот рынок недостаточно зрелый, что влечет за собой некоторые трудности при работе с этими компонентами в вопросах поддержки, более тонкой настройки, либо доступа к исходному коду. Поэтому многие средства автоматизации и небольшие компоненты целесообразно разрабатывать внутри компании. Они легко поддерживаются внутренней командой разработчиков инструментов, являются тонко специализированными под конкретные нужды проекта, команды или компании, они могут быть в любой момент времени изменены и доработаны, так как являются частью системы. Также могут быть использованы как отдельные части для других проектов, быть отдельным продуктом на рынке средств автоматизации, а также отличным маркетинговым инструментом, позволяющий сообщить о компании, ее продуктах и разработках.

Так компания ООО Халфбас, столкнулась с необходимостью оптимизировать работу некоторых своих сотрудников, и пришла к тому, чтобы создать набор инструментов, который бы позволил ускорить работу над текущим проектом, игрой Basement. Это и стало целью данного дипломного проекта.

В рамках данного дипломного проекта, были решены задачи частичной автоматизации процесса создания игровых сценариев, уровней, анимации, работы с ресурсами и локализацией.

# 1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Анализ предметной области

Компания Халфбас для разработки использует игровой движок Unity3d, поэтому при поиске аналогичных систем необходимо было искать решения, соответствующие целевой платформе. Соответственно, основным местом поиска аналогов был магазин компонентов Unity Asset Store [2]. Однако среди всех компонентов, тех, что полностью воспроизводили весь набор необходимых функций, найдено не было. Однако некоторые из них, отвечающие за ту или иную небольшую часть системы, были найдены.

## 1.2 Автоматизация создания игровых сценариев

Так, для автоматизации процесса создания игровых сценариев было решено интерпретировать какой-то скриптовой язык внутри игры. Bolt – компонент позволяющий создавать визуально скрипты, с помощью системы графов. При первом взгляде отлично подходил для решения задачи. Ниже приведен пример работы системы (см. рис. 1.1).

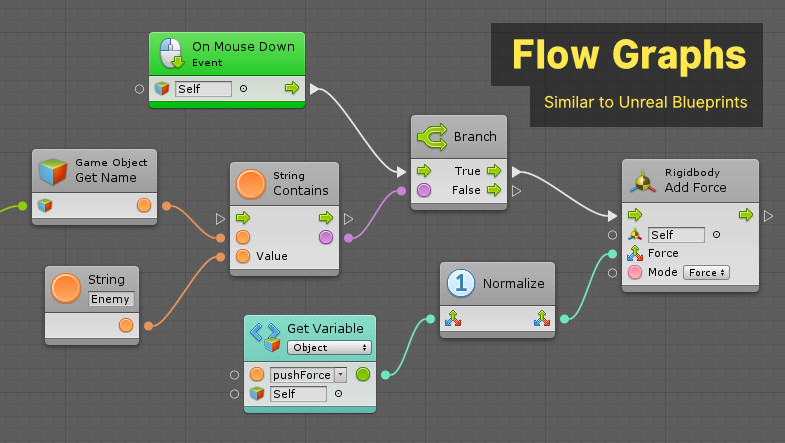


Рисунок 1.1 – Пример работы компонента Bolt

Данный компонент позволяет создавать простые игровые сценарии, однако для компании этот и аналогичные варианты не являлись приемлемыми, так как важным требованием было возможность создавать сценарии без наличия установленного движка Unity3d. Предполагалось, что игроки смогут сами создавать игровые сценарии во встроенном редакторе, и делиться ими с другими игроками с помощью встроенного в игру сервиса.

По той же причине не подошли аналоги: Playmaker, NodeCanvas, Behavior Designer, uNode.

## 1.3 Автоматизация создания уровней и работы с ресурсами

Так как игровые уровни являются частью игры и плотно завязаны на код игровой логики, найти такое специфическое решение было проблематично. Стоит отметить, что игра представляет собой менеджмент-стратегию, где каждый уровень представляет собой линейную улицу с различными зданиями, состоящими из комнат и коридоров различной конфигурации. Помимо комнат, в зданиях есть работники, которыми управляет игрок, а также транспорт. Для визуального разнообразия здания украшаются различным декором, который свободно ставится, без привязки к сетке. Помимо визуальной составляющей, уровень хранит в себе часть настроек игрового процесса: количество противников, набор предметов, зданий, видов работников, и прочее. Так же было необходимо, чтобы уровень хранился в виде отдельного файла. Внутри компании принят стандарт: все файлы конфигураций, локализаций, диалогов хранятся в формате json. Поэтому дополнительно, было необходима разработка системы работы с пользовательскими ресурсами. Такими ресурсами для игрового редактора стали уровни, здания, персонажи, транспорт, комнаты и их компоненты, настройки фонов игрового уровня, файлы сценариев, настройки эффектов персонажей, файлы изображений, которые используются в качестве декора или для создания анимации, файлы диалогов, локализаций, настроек игровых товаров. Исходя из требований, все эти ресурсы должны быть доступны для редактирования вне игрового движка Unity3d, чтобы в дальнейшем, игроки могли обмениваться локально созданными уровнями, персонажами, сценариями, декором.

В компании считают, что таким образом можно сохранить интерес к проекту у основной аудитории на длительный срок: позволить игрокам участвовать в процессе создания игры.

Пример того, как выглядит уровень внутри игры (см. рис. 1.2).



Рисунок 1.2 – Игровой уровень

## 1.4 Автоматизация процесса создания анимации

Проект является двумерной игрой, использования трехмерных моделей не предусмотрено; Процесс анимации этих двух видов графики значительно отличается друг от друга, притом некоторые приемы, перешли из одной области в другую. Так, например, возможно создание анимации с использованием костей (Упрощенный граф, к вершинам которого привязываются вершины трехмерных объектов). Данный прием использовался в текущем проекте, и это значительно облегчало создание анимации, так как для такого способа существует огромное количество профессиональных программ: Autodesk 3dsMax, Autodesk Maya, Blender, Cinema 4d, XSI и прочие. Однако, для текущего проекта очень важна была визуальная составляющая, пиксельная графика, как в старых играх. Поэтому процесс создания анимации был специфичным для такой стилистики. Пришлось полностью отказаться от анимации с использованием костей. Хотя существует отличное решение этой задачи даже для двухмерных игр, позволяющий анимировать части тела персонажей с помощью кривых матрицы трансформаций объекта, Spine[3], (см. рис. 1.3).

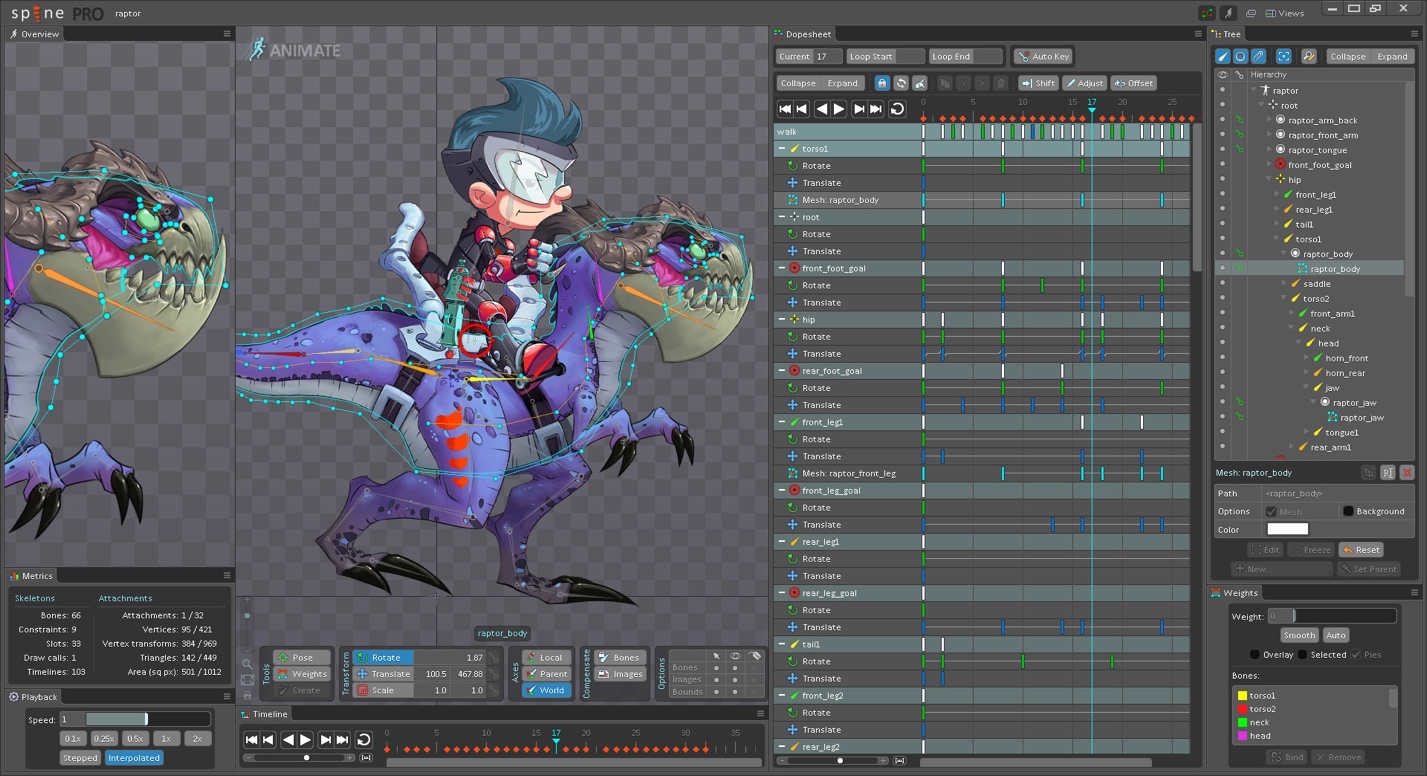


Рисунок 1.3 – Пример работы приложения Spine

В итоге было решено осуществить переход на старую систему анимации наложением. Когда растровое изображение переключается с одного на другое без плавных переходов, (см. рис. 1.4).

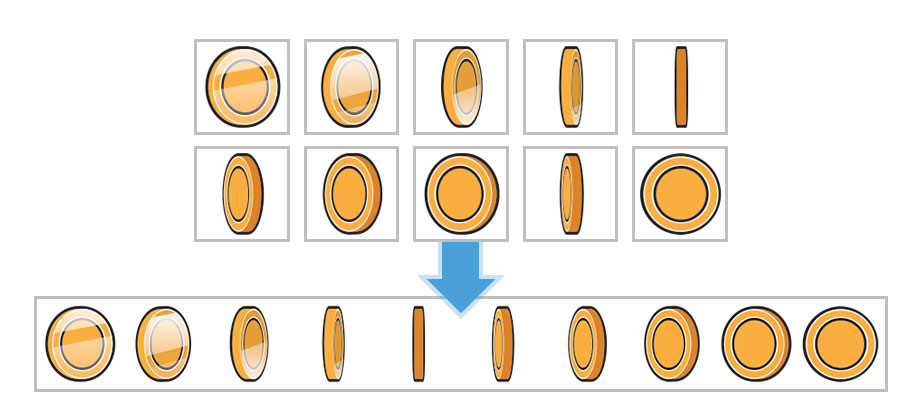


Рисунок 1.4 – Пример файла анимации наложением

Процесс автоматизации создания анимации с таким подходом заключается в том, чтобы автоматически разрезать атлас с изображениями на составляющие анимацию части. Пронумеровать и назвать их, автоматически создать файл анимации, который содержит информацию о ключевых кадрах, и изменениях изображений в определенные моменты времени.

Инструментов в Unity Asset Store, которые позволяли это делать не оказалось. Минимально необходимый функционал представлял компонент Animation Processor [4], но в данном случае, оказалось целесообразно разрабатывать собственный компонент

## 1.5 Автоматизация работа с локализацией

Также было необходимо автоматизировать процесс работы с текстовой информацией внутри игры. Логично было хранить файлы локализации вне игрового движка, чтобы дать возможность редактировать ее переводчикам. К тому же, компания хотела давать игрокам возможность переводить игру самостоятельно на свой язык, просто редактируя таблицу с различными строками текста, который используется в игре.

Для решения данной задачи был найден готовый компонент L2 Localization [5], который позволял хранить данные локализации в таблицах Google Sheets, умел выставлять различные стартовые настройки в зависимости от целевой платформы, подгружать актуальную версию локализации во время работы приложения, имел поддержку различных шрифтов, способов написаний, замены графических и аудиофайлов для конкретной локализации. Однако лицензионные соглашения, обязывающее покупать копию на каждое рабочее место, а также закрытый исходный код решения и его громоздкость подтолкнули к написанию собственного решения внутри компании.

## 1.6 Постановка задачи

Объектом исследования являются процессы создания игрового контента. Предмет исследования – конкретные процессы связанные с текущим игровым проектом ООО Халфбас.

Основными целями исследования являются:

* изучение предметной области;
* установка процесса взаимодействия с художниками, игровыми дизайнерами и дизайнерами уровней;
* проектирование и реализация пакета программ автоматизации;

Главной целью дипломного проекта являются проектирование и реализация пакета программ для автоматизации работ, связанных с разработкой игрового контента. Разработанные компоненты должны удовлетворять различным требованиям, исходящим от членов команды разработки: игровых дизайнеров, дизайнеров уровней, специалистов по контролю качества. Данный пакет программ также должен быть гибким, переносимым между проектами студии, легко изменяемым или обновляемым в любой момент времени для удовлетворения потребностей сотрудников компании. Компоненты должны быть интегрированы в редактор основного проекта компании, работать как отдельное приложение, поставляться вместе с игрой, которая является кроссплатформенным приложением.

# 2 ОБЗОР ТЕХНОЛОГИЙ

## 2.1 Unity3D

Unity3D – игровой движок, или же просто набор инструментов, объединенных одним графическим интерфейсом, разрабатываемый компанией Unity Technologies [6]. Это кроссплатформенный движок, с его помощью можно создавать приложения для большого числа платформ: персональные компьютеры под управлением Windows, Linux, macOS, мобильные устройства с операционными системами Android, Tizen, iOS, веб-браузер с поддержкой HTML5, телевизоры и телевизионные приставки под управлением Android TV, tvOS, Tizen, игровые консоли Microsoft Xbox, Sony PlayStation, Nintendo Switch и другие [7]. На рисунке 2.1 представлен интерфейс Unity3D.



Рисунок 2.1 – Интерфейс игрового движка Unity3D

Важно отметить что в интерфейс движка встроен плеер - окно, которое позволяет мгновенно запускать приложение, внутри редактора. При отсутствии зависимостей с внешними библиотеками, редактор умеет перезагружать библиотеку с скриптами прямо во время выполнения программы. Это значительно увеличивает скорость работы, позволяет мгновенно видеть результат внесенных изменений.

Для работы с ресурсами у Unity3d предусмотрен файловой менеджер, с возможностью выбора доступных для работы файлов и редактирования специфичной для этого типа метаинформации, включающий различные форматы изображений, 3D моделей, звуков, видео, скриптов. Файловый менеджер так же позволяет создавать специализированные файлы, имеющие непосредственное отношение к игровому движку: файлы анимации, шаблоны игровых объектов(префабы), игровые сцены, шаблоны файлов шейдеров и скриптов, материалы, физические материалы, настройки света, спрайты и прочее, вплоть до пользовательских типов, которые можно создавать с использованием отдельного API, для программирования расширений движка. Стоит также отметить, что Unity никак не модифицирует содержимое файлов, вся метаинформация хранится в файлах с таким же названием, но с расширением .meta.

Другой важной составляющей является редактор графа игровой сцены. Исторически сложилось и оказалось удобным представлять игровую сцену в виде дерева/графа игровых объектов. Внешне редактор дерева сцены (scene graph) выглядит похожим на файловый менеджер, в котором есть иерархия вложенных папок, где каждая папка, по сути, игровой объект. На рисунке 2.2 представлен интерфейс редактирования иерархии сцены.

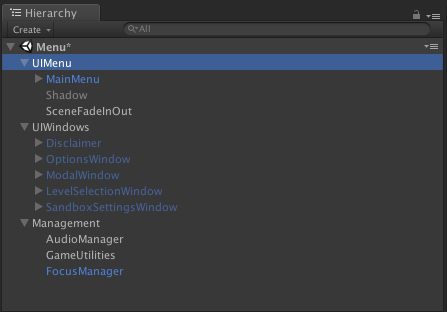


Рисунок 2.2 – Интерфейс редактирования иерархии сцены

В иерархии сцены можно создавать различные объекты, пустые объекты, стандартные 3D и 2D объекты, вроде куба, сферы, спрайта, системы частиц, источники света, звука, камеры и компоненты пользовательского интерфейса. Возможно создание пользовательских объектов на сцене.

То, как Unity 3D использует компонентную архитектуру можно понять при исследовании объектов сцены. Каждый объект сцены, как и ресурс в файловом менеджере, показывает все доступные настройки в окне инспектора. Внешний вид окна инспектора представлен на рисунке 2.3.

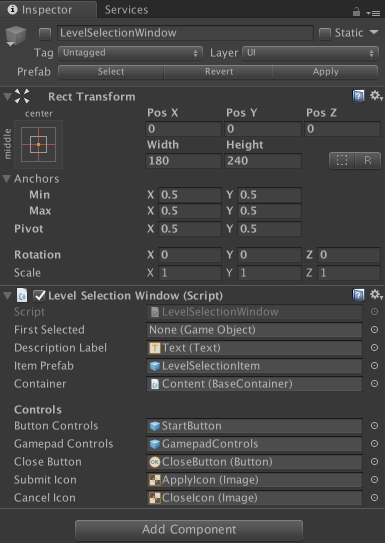


Рисунок 2.3 – Интерфейс инспектора

Каждый игровой объект в сцене имеет один стандартный компонент трансформаций. Изменение его параметров напрямую влияет на положение объекта в мировых координатах. Также есть возможность добавить любое количество компонентов включая пользовательские и стандартные.

Стандартные компоненты, чаще всего отвечают за довольно важные части игрового движка, вроде физического поведение объектов или отображение 3D-моделей. Эти компоненты написаны с использованием языка C++ для максимального быстродействия, пользовательские же компоненты можно создавать с использованием языков C# или UnityScript (особым диалектом JavaScript, специально созданным для создания игровых сценариев). Также доступна возможность использовать библиотечные функции из подключенных .dll библиотек. Средой исполнения пользовательских скриптов является mono (кроссплатформенный, програмно совместимый аналог CLR от Microsoft). Более подробно о платформе mono написано в разделе 2.2.

Unity3D обладает 4мя разными версиями [8]: Personal(Бесплатная), Plus, Pro, Enterprise. Данные версии отличаются по ряду возможностей, включая работу с шейдерами, системой частиц, света, сторонними сервисами для работы с проектами, аналитикой, автоматической сборкой версий, поддержки работы команды и прочие. Также бесплатная версия ограниченно поддерживает основные мобильные платформы. Важно отметить так же, что Unity3D позволяет создавать версии приложения для VR/AR систем: Hololens, Oculus Rift, ARKit, ARCore, Vuforia и т.д. Новых платформ в веб: WebGL, WebAssembly. И набора платформ, специально созданных во взаимодействии с разными компаниями-владельцами интернет-магазинов игр, например: Facebook Gameroom, Xioami Store.

При использовании Unity3D были выявлены следующие положительные стороны, выгодно отличающие этот движок от других; Сочетание широкого инструментария внутри единого приложения, с расширяемым и программируемым интерфейсом. Возможность расширения функционала движка с использованием API редактора. Огромное количество целевых платформ. Высокий уровень качества поддерживаемой графики, включая PBR, трассировку лучей, богатый стандартный набор пост эффектов, сложную систему частиц, поддержку шейдеров и современных графических API: Metal, Vulkan, DirectX 12, OpenGL ES 3.0. Плотная интеграция с физическими движками PhysX и Box2D. Невысокая цена. Возможность получить техническую поддержку от компании-разработчика вплоть до открытия частей исходного кода. Все это определило выбор многих компаний-разработчиков игр в пользу этого набора инструментов.

Unity3D представляет доступ не только к инструментарию, но и к смежным сервисам, которые позволяют управлять запуском проекта, его монетизацией, аналитикой, поддержки нескольких игроков, покупок и прочие.

Unity Ads – сервис для монетизации приложения, с помощью вывода различно рекламы на дисплей. Он позволяет также отслеживать, как работает реклама, выбирать рекламодателей и точно настраивать целевую аудиторию. Является источником прибыли для многих компаний, разрабатывающих игры по модели free2play.

Unity Analytics – сервис для быстрого доступа к данным статистики приложения. Отображает множество параметров, позволяющие отследить поведение пользователей. Также предоставляет инструмент для сбора специальных данных.

Unity Certification – сервис сертификации разработчика на Unity3D. Сертификационный экзамен представляет собой тест из 100 вопросов из 16 различных сфер разработки.

Unity Cloud Build – сервис для доступа к актуальным версиям разрабатываемого приложения, при использовании подхода CI (Continuous Integration). Сервис автоматический собирает конфигурирует и тестирует приложение, собирает статистику по каждому артефакту. Это позволяет сфокусироваться на разработке, и не тратить время на сборку приложения. Unity Cloud Build имеет прямой доступ к репозиторию и может осуществлять сборку проекта при каждом изменении, гарантируя наличие постоянно работающей версии продукта.

Unity Collaborate – облачный сервис для совместной работы над проектами.

Unity Everplay – сервис для записи и загрузки видео игрового процесса.

Unity IAP – сервис для отслеживания внутриигровых покупок в приложении. Использует единый API, позволяет подключить поддержку магазинов App Store, Google Play, Windows Marketplace, Xiaomi Store, Tizen Store и прочие. Всего несколько строк кода и можно полностью настроить, и оптимизировать игровую экономику.

Unity Multiplayer – простой способ создавать прототипы сетевых игр в масштабе реального времени. Их легко развертывать и настраивать. Серверы Unity обеспечивают возможность найти игрокам друг друга, сервис (Matchmaking) Низкоуровневые API дают доступ к ядру среды, чтобы оптимизировать работу в рамках целевого сетевого окружения. Unity Relay – управляет сетевым трафиком, чтобы обеспечить качество сеансов независимо от местоположения игроков.

Unity Performance Reporting – автоматически собирает ошибки на разных устройствах и платформах. Позволяет определить проблемы, ошибки, исключительные ситуации и быстро на них реагировать. Сервис в автоматическом режиме собирает информацию о всех ошибках при работе приложений на конечных устройствах и агрегирует их на серверах.

## 2.2 Платформа Mono

Платформа Mono изначально разрабатывалась, как неофициальный перенос платформы .net от Microsoft, под платформы, которые не поддерживались в тот момент: macOS, Linux, Unix и прочие. Реализация mono полностью соответствовала ECMA стандарту по языку C# и платформе CLR. Впоследствии, с появлением большого числа различных аппаратных платформ, не являющимися частью экосистемы Microsoft, mono стал быстро набирать популярность, так как в тот момент позволял с использованием встроенного набора библиотек Xamarin легко переносить приложение с одной платформы на другую. Это было одно из немногих рабочих кроссплатформенных решений во времена, когда создавался движок Unity3D.

Платформа Mono поддерживалась компаниями Novell, Xamarin, а теперь и Microsoft. Возросшая популярность платформы с открытым исходным кодом mono, вынудила Microsoft открыть исходный код среды исполнения и объединить платформы .net и mono. платформа включает в себя:

Mono Runtime – среда исполнения, которая полностью реализует стандарт ECMA по CLI (Common Language Infrastructure). Среда исполнения представляет JIT и AOT компиляторы, загрузчик библиотек, сборщик мусора, и систему потоков.

Net Framework Class Library – библиотеку стандартных функций, для разработки приложений. Полностью совместимый с реализацией от Microsoft.

Mono Class Library – библиотека дополнительных функций для работы приложений на других платформах, например классы для работы с Gtk+, Zip files, LDAP, OpenGL, Cairo, POSIX, и прочие.

C# Compiler – компилятор языка C#, с поддержкой версий языка C# 1.0, 2.0, 3.0, 4.0, 5.0 и 6.0 (ECMA).

Особенности платформы:

Поддержка большого числа платформ: Linux, macOS, BSD, Windows на всех архитектурах, включая x64, x86, ARM, s390, PowerPC.

Поддержка большого числа языков: C#, VB 8, Java, Python, Ruby, Eiffel, F#, Oxygen и другие.

Бинарная совместимость по стандарту CLI ECMA.

Поддержка многих API от Microsoft: ASP.NET, ADO.NET, Silverlight, Windows.Forms, без перекомпиляции.

Открытый исходный код, бесплатно по лицензии MIT.

Преимущества использования этой платформы заключаются в том, что она очень популярна, поддерживает языки высокого уровня, обладает богатой стандартной библиотекой, является кроссплатформенным решением, поддерживает CLR. Как в случае Unity3D, платформа может использоваться в качестве скриптового инструмента, встроенного в приложение.

Скриптование приложений, встраивание среды исполнения – это техника, при которой приложение можно сделать максимально расширяемым, без переписывания его первоначального кода.

В прошлом, программы полностью были написаны на одном языке программирования под конкретную платформу. Разработчикам приходилось искать баланс между высокой производительностью и необходимостью использовать низкоуровневые языки или медленным исполнением, но с использованием языка высокого уровня. Например так выглядит приложение написанное на C/C++, (см. рис. 2.4) и c использованием скриптового движка, (см. рис. 2.5).

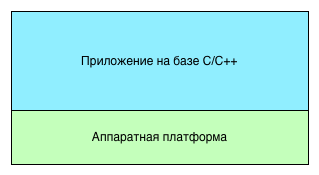


Рисунок 2.4 – Архитектура приложения на языке С/С++

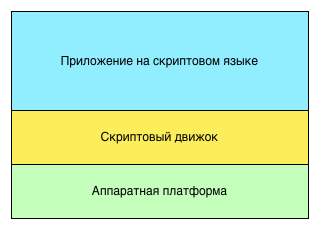


Рисунок 2.5 – Архитектура при использовании скриптового языка

Выбор варианта использования довольно сложный, так как имеется множество нюансов использования. Высокоуровневые языки более продуктивны, и разработчики могут больше сделать в одной строке кода. Однако за это приходится платить свою цену, программы будут работать не так быстро, либо могут требовать больше памяти для работы. В зависимости от проекта приходиться делать выбор между этими вариантами. Однако разработчики придумали скриптование, как механизм, балансирующий эти два мира. Движок приложения пишется с использованием языка C/C++, обычно поддерживается и разрабатывается наиболее опытными разработчиками компании. В это же время части диалогов пользовательского интерфейса, взаимодействия и другие не требовательные к скорости работы части пишутся на высокоуровневом языке скриптования [9]. Результат выглядит так (см. рис. 2.6).

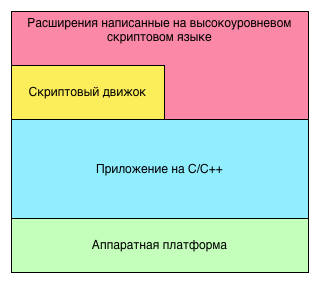


Рисунок 2.6 – Пример сочетания скриптовых и низкоуровневых языков в одном приложении

Исторически, разработчики обращались к огромному количеству разных технологий, для создания скриптов, от очень простых скриптов пакетной обработки, к их эволюции в виде языка Tcl, к специализированным внутренним разработкам (LSL – скриптовый язык игры Second Life [10]) до более обобщенных и популярных языков как Lua, Perl, Python.

Чаще всего, внутренние специализированные языки представляют собой скорее быстрые решения. Авторы этих языков не являются экспертами ни в разработки языков программирования, ни в разработке компиляторов, отчего страдают их разработки. Такие языки обычно медленные, мало документированы, содержат много ошибок и тонкостей. Поэтому многие разработчики стараются найти какие-то более зрелые реализации, если это возможно. Расширение возможностей приложения скриптами имеет множество преимуществ. У вас будет среда, позволяющая быстро создавать прототипы в целевом домене без необходимости переписывать основную часть программы. Также скриптовые платформы предоставляют дополнительный уровень безопасности, отделяя системную реализацию, которая требует знания деталей многих низкоуровневых процессов. Разделяя ответственность разработчиков языка и разработчиков скриптов, можно снизить сложность разрабатываемой системы, позволить разработчикам сфокусироваться на большой картине, нежели на деталях.

Разработчики игр являются наиболее частыми пользователями технологии скриптовых движков. Они разрабатывают чувствительный к производительности функционал на C и ассемблере, чтобы использовать преимущества графических ускорителей, физических библиотек, аппаратного обеспечения, многопроцессорности и многопоточности, чем оптимизируют использование ресурсов.

## 2.3 Платформа .Net Framework

Платформа .NET Framework – это встроенный компонент Windows, который поддерживает выполнение приложений и веб–служб. Основными компонентами .NET Framework являются общеязыковая среда выполнения (CLR) и библиотека классов .NET Framework, включающая ADO.NET, ASP.NET, Windows Forms и Windows Presentation Foundation. .NET Framework предоставляет среду исполнения управляемого кода, ускоряет разработки и развертывания, а также создает возможности интеграции со многими языками программирования.

Основой платформы .NET Framework является среда CLR. Среда выполнения является агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность разработанного приложения. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом.

Другой основной компонент платформы .NET Framework, библиотека классов. Она представляет полную объектно–ориентированную коллекцию типов, которые применяются для разработки приложений.

Платформа .NET Framework может запускаться неуправляемыми компонентами, которые загружают среду CLR в собственные процессы и запускают выполнение управляемого кода, создавая таким образом программную среду, позволяющую использовать средства как управляемого, так и неуправляемого выполнения.

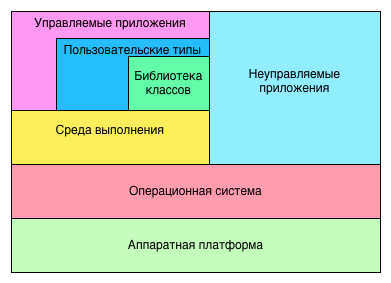


Рисунок 2.7 – Взаимосвязь среды выполнения CLR с операционной системой

Среда CLR управляет памятью, выполнением потоков, выполнением кода, проверкой безопасности кода, компиляцией и другими системными службами. Эти средства являются внутренними для управляемого кода, который выполняется в среде CLR.

Для обеспечения безопасности компонентам присваиваются разные степени доверия, зависящие от ряда факторов: их происхождение (интернет, сеть предприятия или локальный компьютер пользователя), может или не может выполнять операции доступа к файлам, операции доступа к реестру или другие важные функции, даже если он используется лишь в одном активном приложении. Строго разграничивается доступ к различным ресурсам приложениями.

Среда также обеспечивает надежность кода, реализуя инфраструктуру строгой типизации и проверки кода, которую называют системой общих типов (CTS). Система общих типов обеспечивает описание всего управляемого кода. Все языковые компиляторы создают управляемый код, удовлетворяющий системе общих типов. Значит, управляемый код обеспечит правильность типов и строгую типизацию. CLR автоматически управляет размещением объектов и ссылками на объекты в памяти, освобождая их, когда они больше не используются. Автоматическое управление памятью исключает наиболее часто возникающие ошибки приложений вроде утечки памяти или недействительные ссылки на память.

Среда также поддерживает предыдущие наработки в области программного обеспечения. Взаимодействие управляемого и неуправляемого кодов позволяет разработчикам использовать необходимые существующие компоненты COM и библиотеки DLL.

Механизм компиляции по требованию (JIT) позволяет всему управляемому коду выполняться на машинном языке компьютера. Между тем диспетчер памяти устраняет возможность фрагментации памяти и увеличивает объем адресуемой памяти для дополнительного повышения производительности.

Библиотека классов платформы .NET Framework представляет собой коллекцию типов, которые тесно интегрируются со средой CLR. Библиотека классов является объектно-ориентированной, предоставляя типы, от которых пользователь может наследовать собственные типы, что упрощает работу с платформой .NET Framework, и уменьшает время на изучение новых средств.

В классах коллекций .NET Framework реализуется набор интерфейсов, которые можно использовать для разработки пользовательских классов коллекций. Пользовательские классы коллекций будут без затруднений объединяться с любыми другими классами платформы.

Типы .NET Framework позволяют решать типовые задачи программирования, включая работу со строками, сбор данных, подключения к базам данных и доступ к файлам, работа с сетью, многопоточностью и прочими системными функциями. В отличие от многих стандартных библиотек других языков, например D, Go, C++, .NET Framework также предоставляет специализированные типы для работы с экосистемой Microsoft, включающий в себя сервера IIS, веб-приложения ASP.NET, веб-службы, WCF, WPF и прочие продукты компании.

Важной идеей, реализованной в платформе .NET Framework, является совместимость программных частей, написанных на разных языках. Службы, приложения, библиотеки способны обращаться к методам друг друга через систему CLR. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Большинство CLI–языков компилируются в Common Intermediate Language (CIL), промежуточный язык, который может быть транслирован непосредственно в машинный код при помощи среды выполнения Common Language Runtime (CLR).

# 3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

## 3.1 Общая архитектура

Проект состоит из двух отдельных приложений, суть работы которых значительно отличается. Данный подход обеспечивает группировку функциональности в пределах одного модуля. Приложения слабо связаны между собой, и осуществляют обмен данными через потоковые ресурсы (Streaming Assets). Первое приложение это, собственно, сама игра. Поддерживается и разрабатывается отдельно от второй части, но все равно требует взаимодействия между членами команды. Второе приложение – внутриигровой редактор, который позволяет создавать контент для игры. Взаимодействие между приложениями осуществляется через набор сущностей(Entity). Физически они располагаются в отдельной папке Streaming Assets, что позволяет загружать их из внешних источников (Unity3D, при сборке проекта игнорирует содержимое папки Streaming Assets, не генерирует никакой метаинформации и не использует алгоритмы сжатия данных на этих файлах. Просто кладет его в папку рядом с исполняемым файлом игры.). Все это позволяет реализовать важную функцию: дать игрокам возможность самим создавать внутриигровой контент и делиться им друг с другом. Это значительно удешевляет производство контента, ведь фанатам игры не нужно платить зарплату, они буду делать уровни сами для себя. Это значительно повышает реиграбельность проекта, ведь с удачной игровой механикой можно будет исследовать много уровней. Так же, проекты, которые поддерживают данный функционал, особенно в связке с сервисами интернет-магазина Steam, дают им конкурентное преимущество в поисковой выдаче магазина. Прямо сейчас, когда ежедневно выходят десятки игр, поддержка платформы и улучшение видимости проекта – напрямую конвертируется в деньги. Решение этой задачи было приоритетом для компании и сильно повлияло на ход разработки проекта.

Ввиду того, что редактором будут пользоваться не только сотрудники компании, но и сторонние люди, часто, далекие от технических особенностей реализации, было очень важно создать понятный, удобный и простой инструмент. Это также повлияло на глубину и степень проработки некоторых компонентов, особенно, редактора сценариев.

Приложения разделены лишь по игровым сценам, находятся внутри одного репозитория, одного Unity3D-проекта. Это сделано для удобства тестирования сценариев, прямо из редактора можно перейти в игровой режим и протестировать изменения в конкретном уровне. Для обеспечения дополнительного слоя, для работы с пользовательскими ресурсами, необходимо было создать отдельный класс управления ресурсами и его представлением в коде, в виде локальной базы данных, который бы работал параллельно со встроенной базой ассетов Unity3D, которая программно доступна лишь из редактора (сборка UnityEngine.dll). Ниже представлена архитектура работы с пользовательскими и медиа-ресурсами (см. рис. 3.1).



Рисунок 3.1 – Архитектура работы с ресурсами в проекте Basement

## 3.2 Характеристика пользовательских ресурсов

Список внешних(пользовательских) ресурсов включает в себя фоновые изображения и изображения декора комнат и зданий в формате .png, которые автоматически загружаются в базу данных именованных спрайтов (NamedSprites), собираются в атласы, и кодом устанавливаются при загрузке каждого уровня.

Настройки зданий, представляют собой json-файлы сериализованных объектов класса BuildingSettings. Они хранят данные о здании, включая расположение комнат, название, настройки ИИ, данные расположения комнат, их настройки, данные расположения персонажей и их настройки, данные о положении декора зданий, уникальный ключ. Схема класса (см. рис. 3.2).

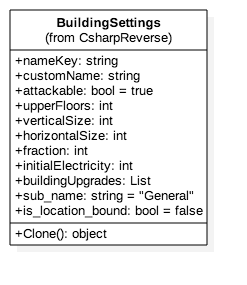


Рисунок 3.2 –Схема класса BuildingSettings

Следующий пользовательский ресурс включает настройки ячеек. Они содержат информацию о типе ячейки и настройки графики (см. рис. 3.3).

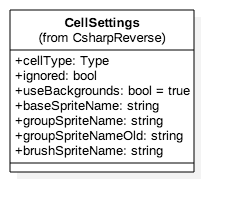


Рисунок 3.3 – Схема класса CellSettings

Настройки эффектов персонажей, хранятся в файлах CharacterEffects.cs. Хранят настройки эффектов. Они могут быть частью каких-то предметов, или сторонними эффектами применения оружия. Они разделяют настройки UpgradePropertyChange с другими классами. Схема класса (см. рис. 3.4).

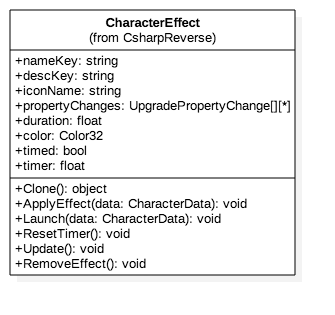


Рисунок 3.4 – Схема класса CharacterEffect

Довольно большая иерархия классов представляет различные виды персонажей. Эти настройки копируются в файлы настроек здания в момент, когда персонажи расставляются в редакторе внутри здания. Существует 4 базовых компонента персонажей: Character, Fighter, Employee, Junkie, Citizen. Данные этих компонентов представлены в классах CharacterData, EmployeeData, JunkieData. Данные и поведение были намеренно разделены. Компонент инкапсулирует в себе данные базовых характеристик: силы атаки, здоровье персонажа, скорость перемещения, выносливость, зарплата, настройки внешнего вида, уровень прокачки, список текущих предметов, эффектов, выпадающих предметов, биография. Все эти характеристики доступны для редактирования снаружи в текстовом файле, через редактор в контекстных меню и используются внутри игры, и в сохранениях, которые, по сути, являются файлами уровней. Ниже представлена схема классов персонажей, (см. рис. 3.5).

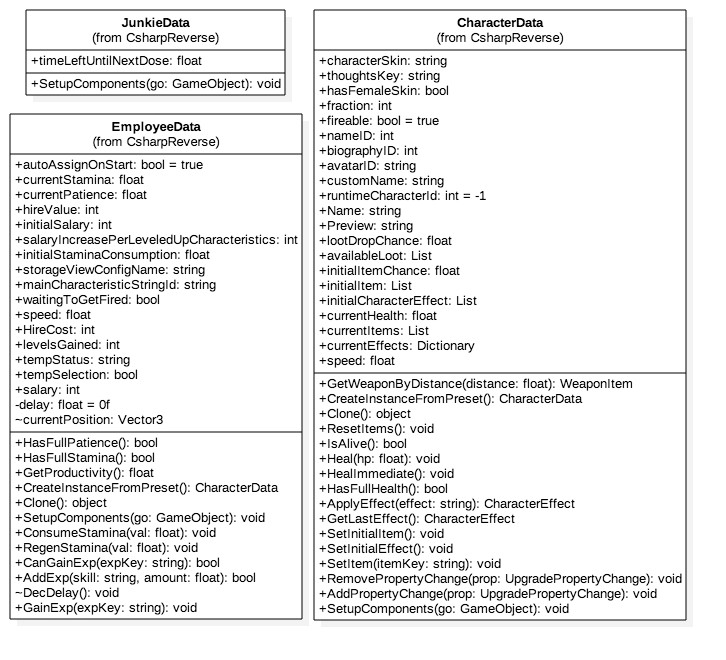


Рисунок 3.5 – Схема классов JunkieData, EmployeeData, CharacterData

Отдельными файлами лежат диалоги в папке Dialogues. В зависимости от локализации, диалоги попадают в папки с каким-то конкретным языком (Ru/En/De и другие). Сами диалоги написаны по спецификации стороннего языка для создания диалогов, который называется Ink. Это простой скриптовый язык, который позволяет составлять диалоги. Он был использован в большом количестве сторонних проектов, например: 80days, Sorcery!, Heaven's Vault и другие [11]. Этот инструмент предоставляется бесплатно с открытым исходным кодом, вместе с отдельным приложением-редактором, которое позволяет сразу тестировать диалог. Так как инструмент предоставляется с открытым исходным кодом, он был немного изменен: добавлена поддержка переменных, настройки изображений участников диалога, настройки фонов.

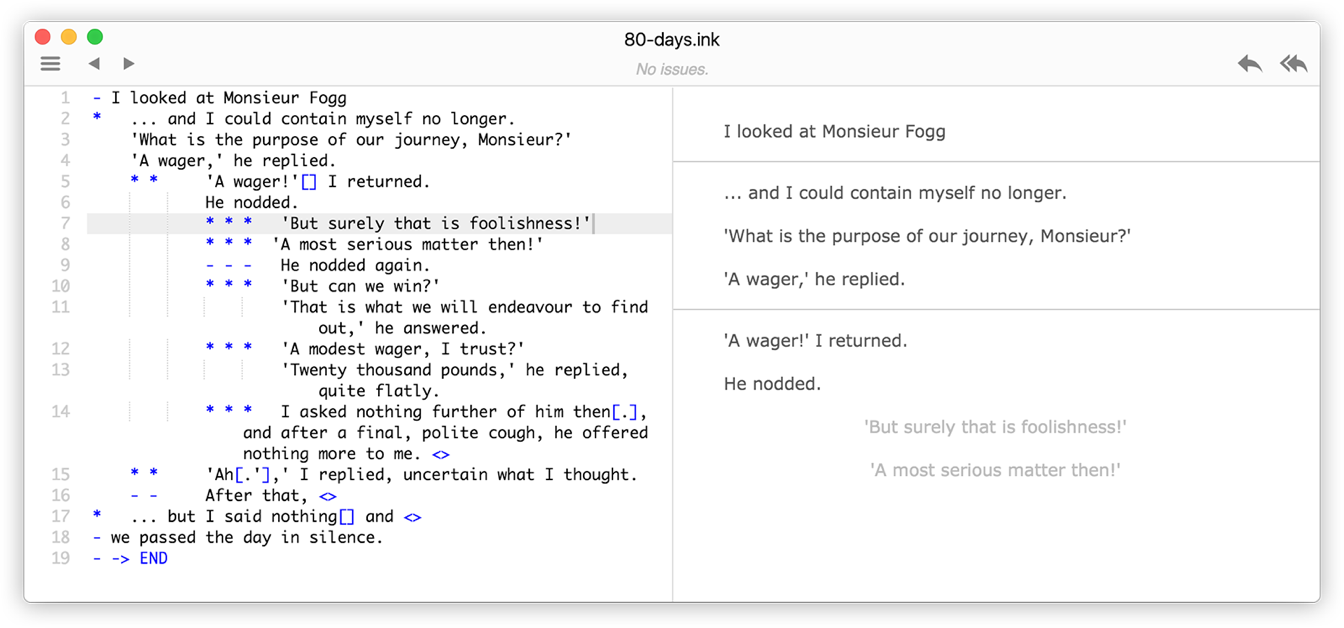


Рисунок 3.6 – Окно редактора Inky и фрагмент диалога на языке ink

Цепочки событий (EventChain) – это ресурсы которые представляют небольшие скрипты-сценарии. Они представляют собой цепочки вложенных друг в друга объектов событий (EventObject). Все функциональные элементы напрямую наследуются от EventObject. По сути, эта цепочка строится на структуре данных дерева. Единственное неудобство в работе с этой структурой состоит в том, что необходимо обрабатывать эту структуру рекурсивным алгоритмом. Особенно это касается алгоритмов переноса узлов дерева в разные места, вставка новых узлов, удаление с сохранением первой или самой глубокой дочерней цепочки. Реализация всех этих алгоритмов была необходимой частью создания редактора сценариев. Стоит отметит тот факт, что EventChain тоже дочерний класс EventObject. Это позволяет хранить отдельные цепочки и переиспользовать их как объекты в других цепочках. Это так же легко позволяет создавать циклы. У каждого игрового уровня есть одноименная цепочка с постфиксом EventChain, которая является основным сценарием уровня, и является точкой входа, для исполнения сценариев в игре. При запуске уровня сразу же запускается одноименная цепочка. В игре также предусмотрен механизм запуска случайных цепочек событий, что позволяет разнообразить игровой процесс, позволяет игроку сыграть дополнительные, небольшие сторонние сценарии внутри игры, что повышает реиграбельность. Отмечу, что реиграбельность – это довольно важный параметр и метрика игры, которая показывает отношение того, сколько фактического времени средний игрок проведет в игре к просчитанному числу игрового времени. Как выглядит окно редактора цепочек событий (см. рис. 3.7).

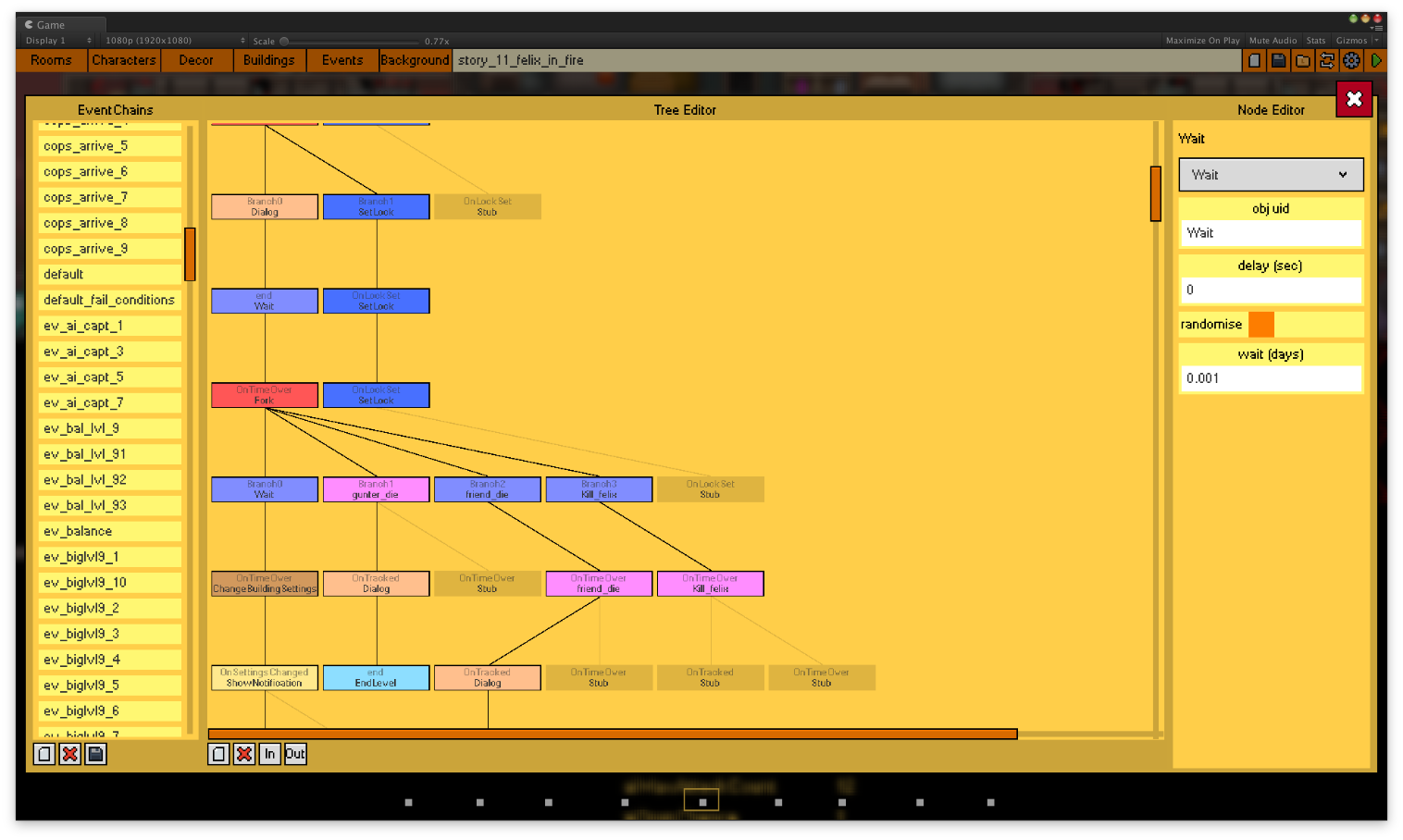


Рисунок 3.7 – Окно редактора цепочек событий

Предметы персонажей, являются важной частью уровней и имеют довольно широкий набор характеристик. Существует несколько видов предметов: Consumables (Потребляемые, расходуемые, одноразовые), включает книги, разные зелья. Weapons(Оружие) и просто Items(Предметы), которые улучшают характеристики игровых персонажей. Каждый из них представлен отдельным классом, унаследованным от CharacterItem, (см рис. 3.8).

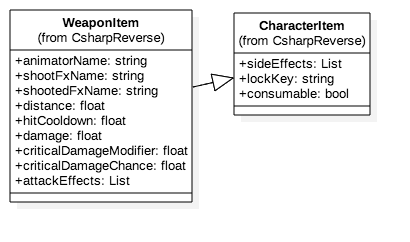


Рисунок 3.8 – Схема классов WeaponItem и CharacterItem

Файлы локализации доступны из папки Localization. Они тоже не пакуются вместе с исполняемым файлом игры, а подгружается отдельно, с выбором локализации. Это обеспечивает меньший размер установочного файла, и дает возможность поддерживать актуальность перевода, автоматическим обновлением. Как отмечалось в части 1. Для редактирования файлов локализации используется обычная таблица Google Sheets. На рисунке 3.9 представлен фрагмент файла локализации, который потом, с помощью небольшого скрипта на языке JavaScript, в виде дополнения к Google Sheets создает json-файл со строками и ключами локализации. Все текстовые поля, которые отображаются внутри игры содержат лишь ключ текста. Сами данные по ключу загружаются в момент появления этого поля на экране. Например, по ключу ui\_status\_resting, на экране, в зависимости от локализации, отобразится Resting, отдыхает или Erholt sich.

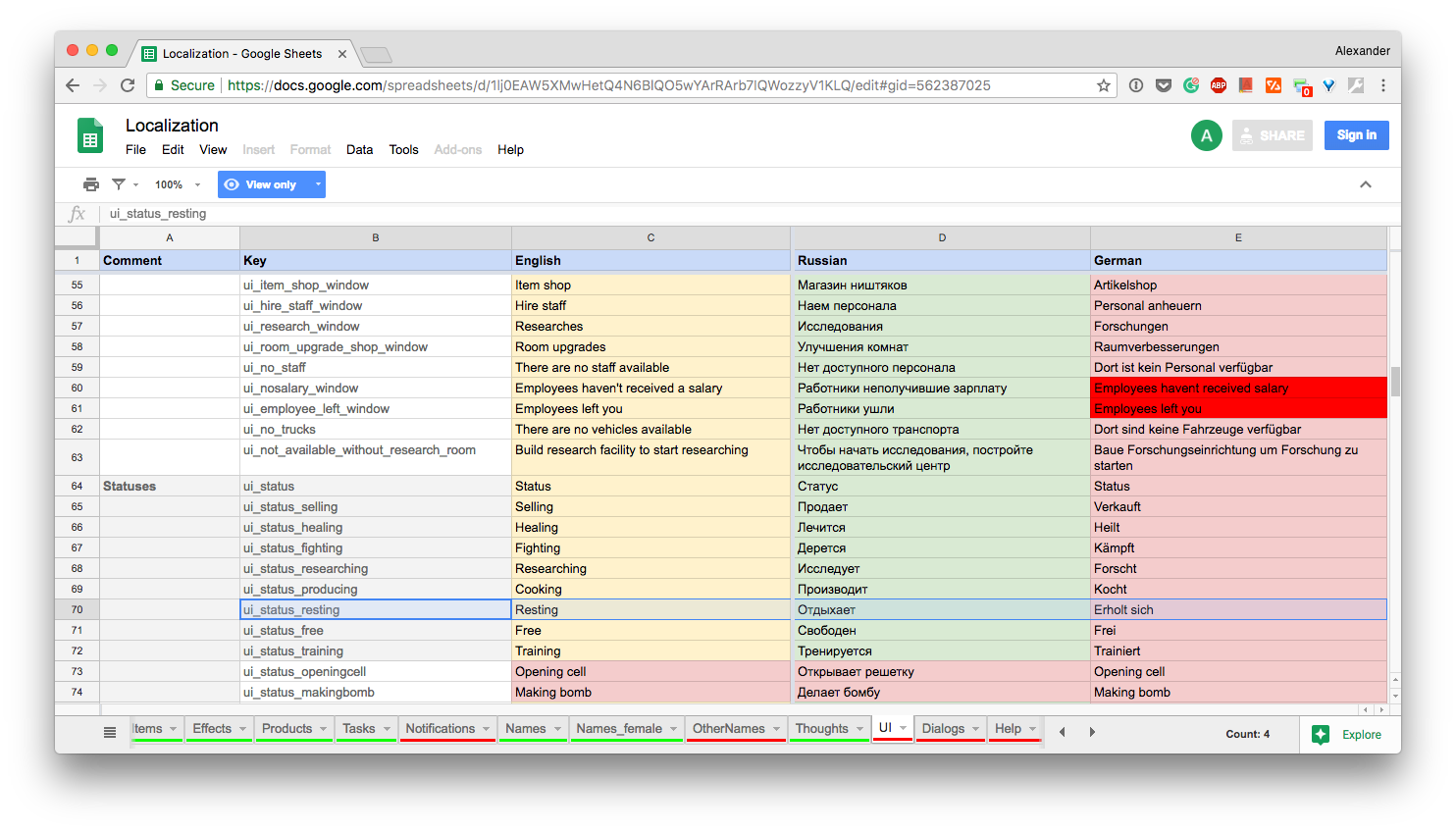


Рисунок 3.9 – Фрагмент файла локализации Google Sheet

Следующий ресурс – является шаблоном настроек фонов для уровней. Так как этот конфигурационный файл является довольно сложным в настройке, было принято решение отделить его от настроек уровня в виде отдельной структуры, которая может быть частью нескольких разных уровней. Так и появился класс LocationBackgroundSettings; это сильно упростило работу со структурой уровня, и заодно позволило оптимизировать размер файла уровня. Структура класса представлена на рисунке 3.10.

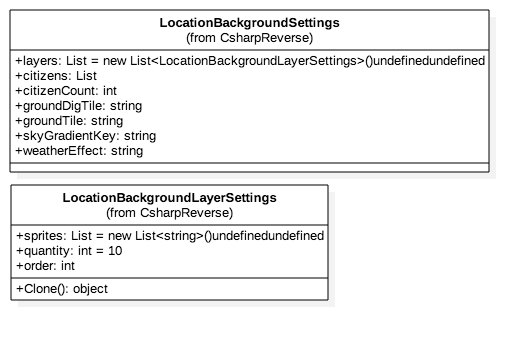


Рисунок 3.10 – Диаграмма классов для настройки фонов уровня

Собственно файлы самого уровня, хранятся в папке LocationSettings. Каждый уровень находится в одноименной подпапке. Вместе с настройками уровня, там находятся копии зданий и их настроек в дочерней папке Buildings, а также основная цепочка сценария, с постфиксом \_MainQuest. Здания автоматически именуются по их положению на прямой слева направо. Ниже представлена структура папки с настройками уровня (см. рис. 3.11).

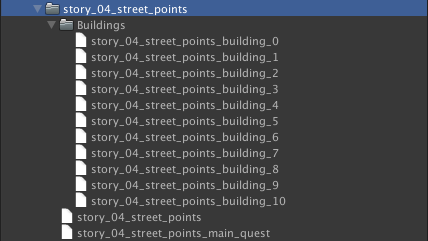


Рисунок 3.11 – Структура папки с настройками уровня

Основные данные все же хранятся в настройках зданий. Настройки локации содержат, в основном, ссылки на разные вспомогательные структуры, списки зданий, настройки фонов, настройки врагов и т.д. Нижу представлена схема класса LocationSettings, (см. рис. 3.12).

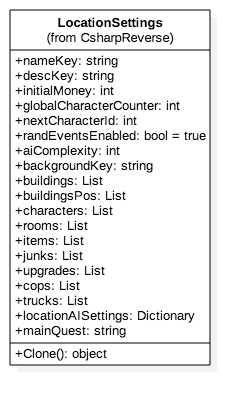


Рисунок 3.12 – Схема класса LocationSettings

Стоит отметить, что сохранение игры происходи тоже в файл с сериализованной копией LocationSettings, включающий в себя расстановку всех зданий, персонажей, декора, и всех игровых параметров. Поэтому процесс загрузки уровня, созданного в редакторе или просто сохраненного состояния игры идентичный. Это позволяет переиспользовать ресурсы, и код, что делает это решение оптимальным для работы системы уровней. К тому же, предполагается, что для того, чтобы запустить новый уровень на другом устройстве – достаточно скопировать папку с названием уровня.

Далее идут настройки игровых товаров. Они являются частью игровой экономики, могут быть использованы персонажами, меняют свои свойства и характеристики с применением различных исследований. На рисунке 3.13 изображена схема класс ProductData.

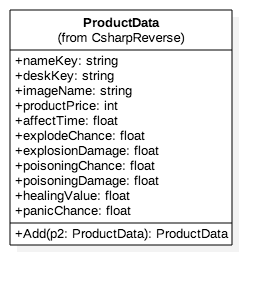


Рисунок 3.13 – Схема класса ProductData

Настройки комнат – это еще одна широкая иерархия классов. Все они унаследованы от класса RoomData, и имеют множество разных дополнительных параметров для каждого уникального типа комнат. Внутри иерархии есть еще один специализированный класс, имеющий наследников, BaseStorageRoomData, который имеет настройки отображения комнат-складов товаров и денег. Приведенные на рисунке 3.14 схемы типов, покрывают все разновидности комнат, которые используются в игре. Эти классы являются лишь структурами данных, они не имеют никакого поведения внутри, кроме метода с настройкой компонентов поведения, они представлены отдельными одноименными классами и не используются в редакторе. Задача редактора – дать возможность изменять эти данные в удобной форме, эти параметры редактируются в контекстных окнах, либо прямо в .json-файлах, которые лежат на диске. Однако изменения в редакторе производить приоритетнее, ибо там есть проверка ввода и генерация правильных .json-файлов.

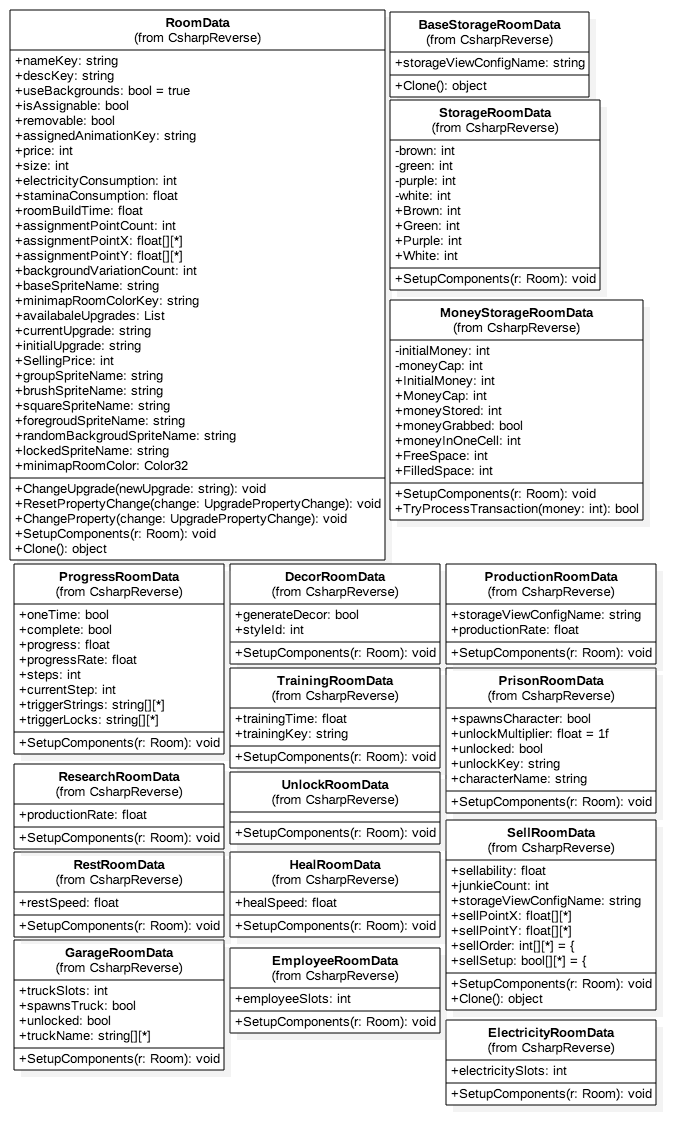


Рисунок 3.14 – Набор классов с данными различных типов комнат

Компоненты комнат находятся отдельно, так как часть из них переиспользуются разными комнатами. Компоненты комнат изменяют различные параметры при помощи строковых литералов. При добавлении компонента, каждая комната отдельно разбирается с тем, как изменить свои параметры. Они используют свойство UpgradePropertyChange аналогично с CharacterEffect, CharacterItem и прочими. Процесс применения и удаления изменений выглядит следующим образом: в методе AddPropertyChange, происходит изменение внутренних характеристик исходя из правил, описанных в объекте UpgradePropertyChange. В зависимости от типа, может произойти замена параметра, умножение параметра на число или сложение параметра с числом. Все числа находятся на вещественной прямой. Ниже представлена схема классов связанной с компонентами комнат (см. рис. 3.15).

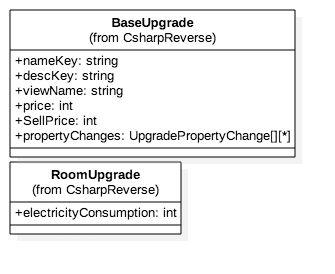


Рисунок 3.15 – Схема классов описывающих компоненты комнат

Последний внешний ресурс – это настройки транспорта. Описываются классами Truck и DeliveryTruck. В игре существует 2 типа грузовиков, для перевозки товаров и персонажей. Они содержат в себе данные о скорости, загрузке грузовиков, их визуальные настройки. Грузовики мало меняются, и в редакторе нет контекстных меню для настройки их характеристик, однако их модификация возможно напрямую в тексте json-файлов. Ниже можно ознакомиться со структурой классов (см. рис. 3.16).

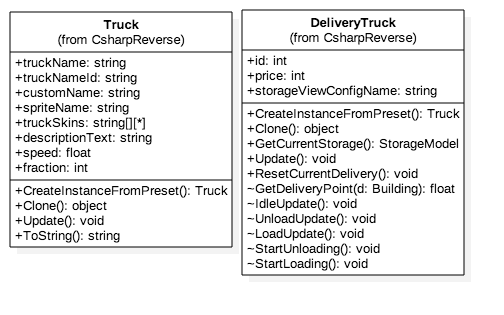


Рисунок 3.16 – Схема классов, описывающих настройки транспорта

## 3.3 Инструмент автоматизации создания анимации

Этот инструмент целесообразно расписать отдельно, так как он мало связан с пользовательскими ресурсами, но является частью пакета программ. Как упоминалось в части 1, для создания анимации использовался способ анимирования наложением. Этот способ является трудоемким для художников, поэтому было необходимо разработать механизм интеграции такого рода медиа-ресурсов в игру. Многие объекты в игре разделяют какие-то типы анимации, которые они проигрывают. Так, например, персонажи перемещаются, атакую, отдыхают, работают в комнатах; транспорт ожидает и перемещается; оружие стреляет, перезаряжается; комнаты ожидают работников, работают и так далее. Было решено представить анимации, относящиеся к какому-то классу в виде двумерной таблицы, в которой строки выступают в качестве экземпляра анимационного клипа, а столбцы – номера кадров. Обычно достаточно 4-10 кадров для того, чтобы анимация выглядела удовлетворительно.

Внутри игры был создан специальный компонент, который находится в той же иерархии, что и файлы подготовленных изображений с сеткой.

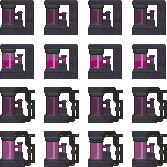


Рисунок 3.17 – Пример подготовленного для анимации атласа

В Unity3D предусмотрена возможность создавать пользовательские скрипты, исполняемые во время импорта ресурсов в проект. Так был создан специальный компонент, который, при импорте нового файла, искал ближайший конфигурационный файл со специальным названием \_ImportSettings, который представляет собой json-файл с настройками анимационного атласа. Скрипт, при импорте файла выполняет настройку параметров импорта: тип фильтрации, тип представления в видеопамяти, использование альфа-слоя, генерации дополнительных текстур и другой метаинформации, затем создает специальный файлы с кадрами анимации из этого атласа(координаты и размеры прямоугольника, обрамляющего кадр), именует их особым образом и, с помощью программного интерфейса редактора Unity3D, создает файлы анимационных клипов, именует их особым образом, помещает в специальную папку, из которой эти анимации будут автоматически добавлены в ранее созданные ресурсы, содержащие описания конечных автоматов, управляющих состояниями анимированного объекта. Этот ресурс, называется AnimationController, скрипт импорта автоматически подставляет конкретные названия анимации по постфиксам, указанных в шаблоне контролера. Использование файлов настройки прямо в иерархии проекта позволяет быстро и удобно автоматизировать создание анимации, без потери контроля. К сожалению, этот компонент не удалось встроить в отдельное приложение редактора, так как для создания файла анимационного клипа, необходимо использование библиотеки UnityEngine.dll, что не позволило вынести анимационные файлы в виде отдельного внешнего ресурсы. Это значит, что для добавления новой анимации, будет необходимо осуществлять сборку ресурсов заново.

Пример содержимого файла настроек импорта приведен в приложении.

## 3.4 Архитектура редактора уровней

Редактор уровней, представляет собой отдельную сцену, а по сути, и приложение, которое может работать автономно от игры. В основе лежит координатная плоскость с двумерной сеткой, для привязки курсора. По нулевой координате оси Y проходит граница. В игре, на этом нулевом уровни ставятся здания (вход в здание), ездят машины, ходят по улице персонажи.

По оси X откладываются точки, в которых будут находиться здания, другие объекты на карте не располагаются, все привязано к зданиям, кроме погодных эффектов и фоновых изображений.

Фоновые изображения, погодные эффекты, градиент тумана и текстуры земли, границ и ячеек доступных для выкапывания настраивается в отдельно редакторе фонов. Он доступен из основного окна по кнопке с надписью Background. Как было указано в разделе 3.2, настройки фоновых изображений хранятся в виде отдельных файлов LocationBackgroundSettings. Как видно на рисунке 3.18, слева можно загрузить какой-то готовый шаблон фонов.



Рисунок 3.18 – Окно редактора фоновых изображений

Помимо настроек фоновых изображений, уровень также содержит настройки игры: начальные ресурсы, начальные настройки врагов, доступные комнаты, предметы, персонажи, и прочее. Окно редактора представлено на рисунке 3.19.

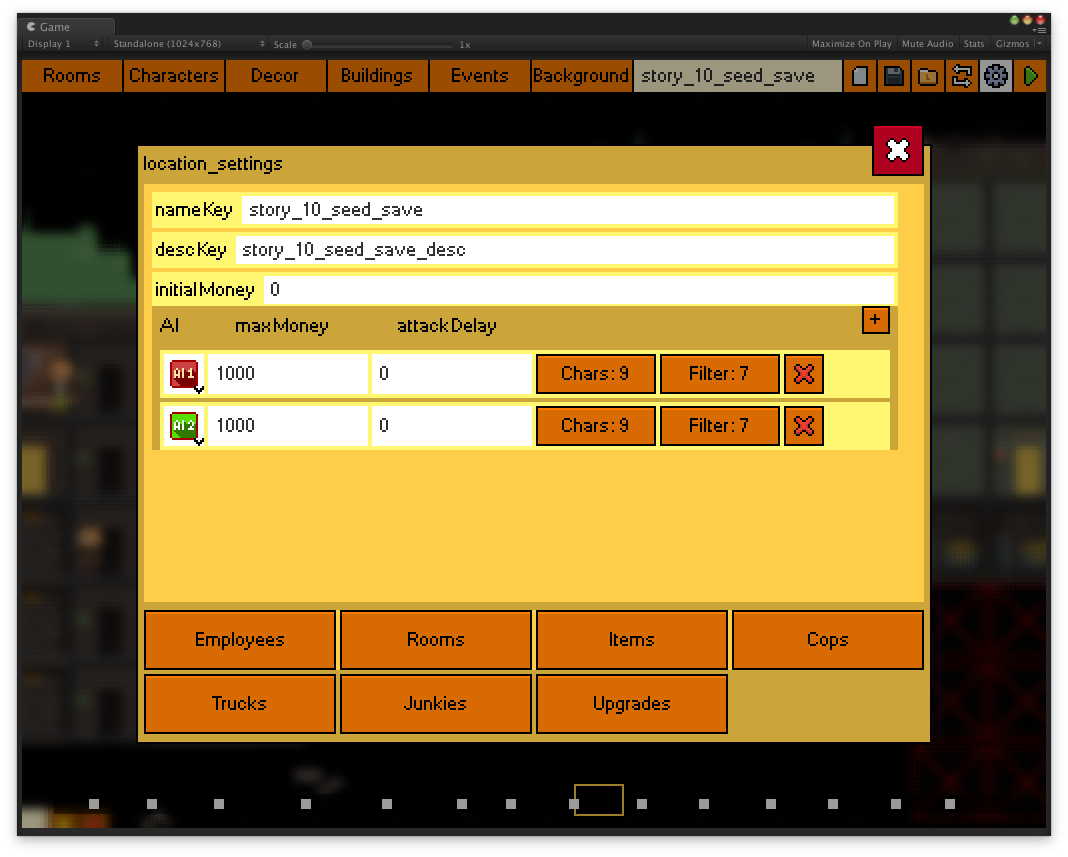


Рисунок 3.19 – Окно редактора настроек уровня

В верхней части окна редактора представлено несколько вкладок. Первые 4 из которых, являются вкладками с набором кистей(шаблонов) объектов, которые расположены в разных подпапках Streaming Assets.

Следующим по уровню иерархии, является объект здание; Готовые шаблоны зданий можно ставить из выпадающего меню Buildings. Под каждым зданием находится список параметров: начальное электричество, кому это здание принадлежит, и настройки ИИ, в случае если здание находится в его владении. Здания можно удалять, перемещать, и сохранять в виде шаблонов в подпапках Streaming Assets. Для того чтобы поставит здание, необходимо кликнуть на кисть шаблон в меню, а затем на точку в координатной плоскости, где будет расположено здание. Как выглядит здание, выставленное в редакторе, со списком параметров, доступных для настройки, можно увидеть на рисунке 3.20.



Рисунок 3.20 – Внешний вид редактора здания

Кнопки со знаками + и – позволяют изменять размер сетки для комнат здания. Сетка определяется заранее и имеет прямоугольное расположение, так как информация о комнатах расположена в линейном списке. Это намного проще, нежели создание графа, со связями комнат друг с другом.

В третьей вкладке находятся кисти с элементами декора, с помощью которых происходит украшение зданий: крыши, трубы, стенки, пристройки, деревья и прочее. Все они рисуются в том же слое, что и комнаты и являются частью настроек здания. Они представлены аналогичным списком с подкатегориями, как и во вкладке.

Во второй вкладке находятся персонажи. Они сгруппированы по коалициям, к которым они принадлежат: нейтральные, персонажи игрока, персонажи различных вражеских фракций. В контекстном меню справа можно настроить параметры шаблона, а также конкретного экземпляра в сцене. Обращаю внимание, что это различные сущности. Информация о персонажах или комнатах, размещенных в здании, пишется в файл, описывающий здание. Шаблоны же при этом, не изменяются. Это позволяет создавать различные вариации персонажей прямо в зданиях, например, назначив им какой-то предмет или немного изменив их характеристики. Увидеть контекстное меню можно на рисунке 3.21.



Рисунок 3.21 – Контекстное меню выделенного персонажа

Контекстное меню используются для демонстрации параметров во многих частях игры и редактора, включая окна описания событий, комнат, персонажей, всех интерактивных элементов игры и окон покупки. Контекстное меню представляет собой растягивающийся список со списком стандартных виджетов: текст, кнопка, поле для редактирования, выпадающий список, выпадающее окно с дополнительным списком параметров. Построив такую библиотеку компонентов, что значительно облегчило работу по созданию пользовательского интерфейса. Если бы не ограничение Unity3D по использованию механизмов рефлексии .net на мобильных платформах, было бы можно обобщить процесс создания контекстных окон просто из структур данных. К сожалению, AOT-компилятор Mono накладывает строгие ограничения на использование механизмов рефлексии.

Все персонажи, расставленные внутри комнат здания, записываются в линейный список внутри объекта, представляющего настройки здания. Комнаты, выбор шаблонов которых производится в первой вкладке (см. рис. 3.22).



Рисунок 3.22 – Пример выбора комнат для расстановки в здании

Из дополнительных удобных функций в редакторе уровней, важно отметить наличие мини-карты с расстановкой зданий на уровне, возможность очистить изменения всех экземпляров шаблонов, расставленных на карте, автоматическое сохранение, быстрый переход в игровой режим, для тестирования уровня, привязка к сетке различного масштаба, управление масштабом игровой камеры.

Для редактирования декора, так же был создан инструмент, позволяющий расставлять его по слоям, представляющий собой обычный линейный список, с возможностью сортировки элементов в нем (см. рис. 3.23).

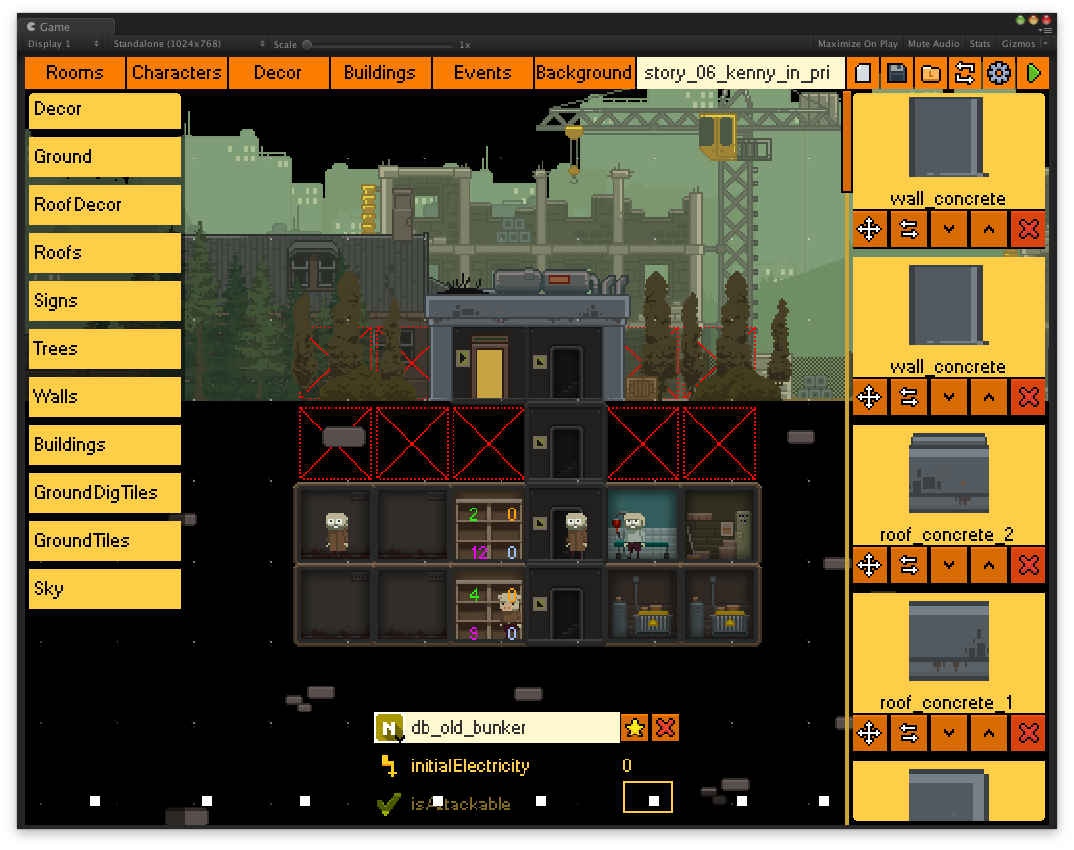


Рисунок 3.23 – Изображение редактора слоев декора зданий

## 3.5 Архитектура редактора игровых сценариев

Редактор игровых сценариев я выделил в отдельный раздел, потому что он занимает значительную часть редактора, и потребовал внесения большого количества изменений в структуру самой игры.

Основная идея о цепочке объектов, как в древовидных структурах, была описана выше. Хотелось бы раскрыть детали, касательно процесса исполнения сценариев. В игре существует объект-одиночка, который инициализируется с запуском уровня. EventManager автоматически загружает основную цепочку событий. У каждого объекта цепочки событий есть свой жизненный цикл.

При запуске уровня EventManager, запускает события с корневого узла, вызывая метод OnPreEnter. Этот метод является виртуальным, и позволяет настроить объект до момента его использования. Базовый класс регистрирует текущий исполняемый объект в списке активных объектов EventManager, чтобы тот, в случае необходимости сохранить состояние игры, мог записать исполняемый в текущий момент объект, и при загрузке восстановить поток событий с этого момента.

После успешной регистрации, каждый объект может либо пребывать в ожидании запуска, или начать исполнения, вызвав виртуальный метод OnEnter. В этом методе происходят полезные действия того или иного объекта цепочки событий. Например, создается рейд в здание, меняются характеристики персонажей, создаются новые персонажи и так далее.

Внутри каждого класс должен быть гарантирован вызов следующего метода жизненного цикла OnExit. Этот виртуальный метод обеспечивает освобождение всех ресурсов, задействованных в текущем объекте, удаляет себя из списка активных объектов EventManager, и передает управление следующему элементу, либо элементам цепочки. Так как выходов может быть несколько, можно легко организовывать условные переходы, разветвления, и циклы.

Иногда для корректной работы, нужно удалить соседнюю цепочку объектов. Это делается с помощью особого объекта цепи событий EventObjectRemoveNode. Выполняется поиск по тэгу, имени или типу среди активных объектов цепочки и затем вызывается метод удаления OnRemove. Это обязательный для переопределения абстрактный метод всех объектов цепи событий.

Благодаря тому, что объект, описывающий цепочку событий, сам является наследником класса Event Object, возможно сохранять как поддерево, вставлять и переиспользовать в любой момент времени.

Окно редактора дерева событий состоит из трех панелей. Слева, панель выбора модифицируемой цепочки событий, и отдельный элемент, позволяющий переключиться в основную цепочку уровня. В центре большая панель, которая показывает все элементы дерева, связи, позволяет добавлять, вставлять, перемещать и удалять узлы. К сожалению, работать с древовидной визуальной структурой над очень сложными сценариями проблематично. Это требует много времени на поиск конкретного элемента цепочки, сложно выполнять отладку. Считаю, что лучше было бы сделать интеграцию какого-то простого скриптового языка вроде Lua. Однако осознание этого пришло значительно позже. В правой панели находится список стандартных компонентов, аналогичный контекстным меню, который позволяет изменять параметры и настраивать узлы дерева. С внешним видом редактора можно ознакомиться, взглянув на рисунок 3.24 ниже.

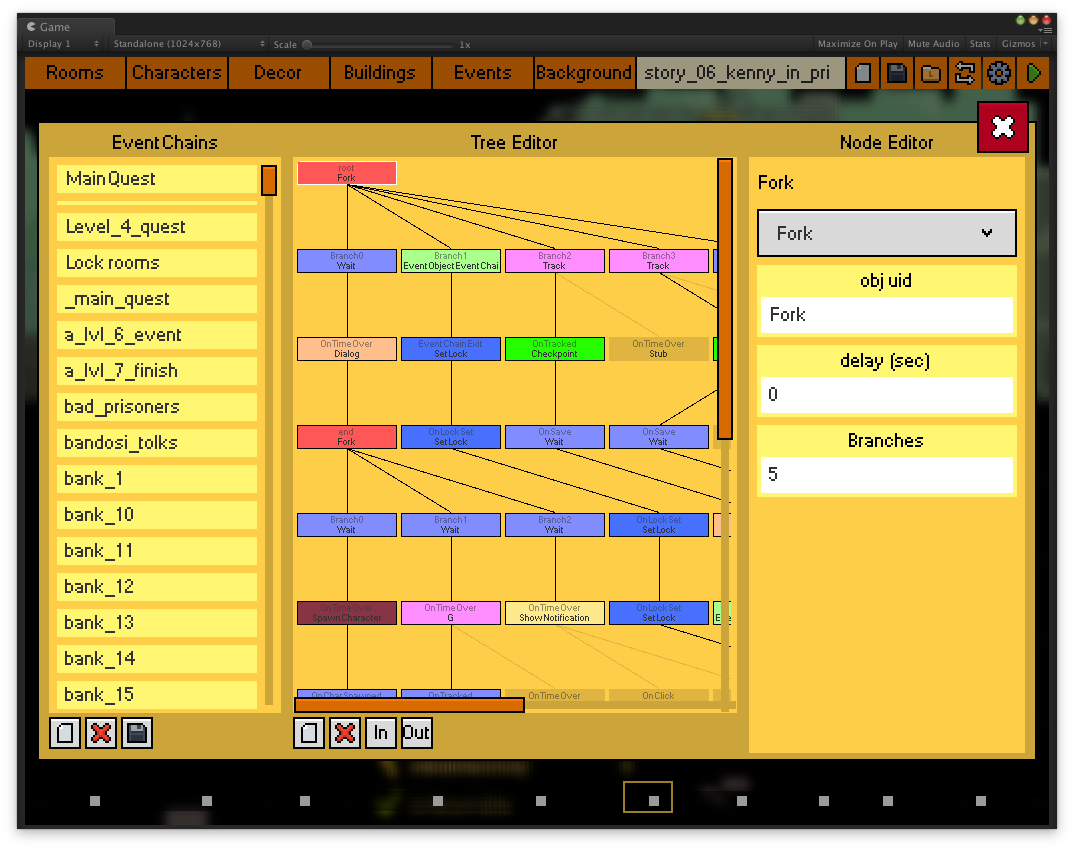


Рисунок 3.24 – Окно редактора дерева событий

Далее хотелось бы остановиться на описании различных команд и объектов, которые были созданы для разработки сценариев.

EventObject – базовый класс, обеспечивающий поддержку жизненного цикла элементов цепочки.

Stub – заглушка, на этом исполнение цепочки заканчивается.

Track – позволяет проверить значения любого параметра в словаре по ключу. Существует список используемых внутри игры параметров, которые изменяются в процессе игры. Доступ к значениям этих параметров и проверку можно осуществить используя этот объект.

Wait – позволяет осуществить ожидание в игровом масштабе времени, например 5 игровых дней.

Variable – позволяет установить значение какой-то переменной.

EndLevel – вызывает окно проигрыша или победы.

Notification – вызывает нотификацию в панели нотификаций в игре. Может иметь таймер, осуществлять переход в разные ветви в зависимости от действий игрока.

Dialog – запускает диалог с определенным ключом. Диалоги были описаны в разделе 3.2. В самом диалоге настраивается количество выходов из объекта.

ShowTip – вызывает подсказку.

Fork – запускает несколько параллельных цепочек.

EventChain – запускает другу цепочку событий по ее имени.

SpawnCharacter – создает персонажа в каком-то здании.

Raid – запускает рейд определенной фракции в случайное или конкретное здание, несколько зданий.

ChangeAISettings – меняет настройки ИИ в здании и уровне.

SetLock – блокирует какой-то элемент игры, например части интерфейса или определенный функционал, например возможно увольнять работников или удалять комнаты.

ChangeInventory – добавляет оружие и предметы в инвентарь игрока.

SpawnTruck – создает грузовик возле какого-то здания.

ChangeMoney – изменяет количество денег у игрока.

ChangeProductStorage – изменяет количество товара на складах игрока.

FireCharacter – удаляет персонажа с карты.

AddExp – добавляет опыта персонажу.

RemoveActiveNode – удаляет узел, было описано выше, в жизненном цикле EventObject, вызывает метод OnRemove.

RestartLevel – перезапускает уровень.

Instantiate – создает простой объект из списка именованных объектов.

LaunchRandomEvent – запускает случайное событие из списка.

SetRandomEvent – устанавливает доступные случайные события.

Check – проверяет выполнение задания игроком.

ChangeBuildingSettings – меняет настройки здания;

Checkpoint – вызывает сохранение игры.

CameraConrol – позволяет управлять камерой.

FacadeControl – изменяет фасад здания.

## 3.6 Этапы разработки

Разработка системы производилась в несколько этапов.

Сначала была изучена предметная область, включая сам проект и поставленные задачи.

Далее была разработана общая архитектура системы работы с внешними ресурсами. Параллельно была создана система автоматического импорта анимации, доработана архитектура игры, касаемо части работы с ресурсами.

Третий этап заключался в определении основных сущностей и модели данных системы. Была спроектирована структура данных и разработан уровень доступа к ней. Также были реализованы сервисы доступа к данным.

Четвертым этапом стала реализация редакторов для каждого вида внешних ресурсов, начиная с крупных частей: уровней, зданий, комнат, персонажей и далее.

На следующем этапе был спроектирован и разработан компонент создания и интерпретации игровых сценариев.

Заключительным этапом разработки стало всестороннее функциональное тестирование разработанной системы.

# 4 ТЕХНИКО–ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА

## 4.1 Краткая характеристика проекта

Цель дипломного проекта – проектирование и реализация специализированных компонентов игрового движка, инструментов, редакторов. Проектирование и реализация компонентов и приложений, позволяет сократить время, затрачиваемое на интеграцию графических, текстовых и аудио ресурсов в игровой проект, оптимизировать процесс разработки игровых сценариев, сцен и уровней.

Данное программное обеспечение будет использоваться игровыми дизайнерами и художниками для разработки игрового контента, а также разработчиками и специалистами по контролю качества для создания тестов, отладки и ускорения процесса поиска и устранения ошибок в различных игровых ситуациях. Является разработкой для собственных нужд компании. Распространение третьим лицам не предусмотрено, хотя при некоторой дополнительной работе – возможно.

Программное средство содержит несколько компонентов, которые связаны в рамках одного внутриигрового редактора. Включает редактор сценариев, систему для загрузки и подключения локализаций из внешних хранилищ, локальную базу ресурсов, редактор уровней, систему для автоматического создания анимации, конфигурирования игры.

Для разработки данной системы будут использоваться уже имеющиеся в компании технические средства и методы разработки. Программное средство обеспечит прирост прибыли за счет снижения себестоимости конечного продукта, а также экономии расходов на заработную плату в результате снижения трудоемкости операций по созданию игрового контента.

Программный продукт разработан и используется сотрудниками компании ООО Халфбас.

Расчеты выполнены на основе методического пособия [12].

В настоящем разделе рассмотрим вопросы, связанные с технико–экономическим обоснованием проекта. Разработка приложения относится к первой группе сложности. Имеет сложный интеллектуальный языковой интерфейс с пользователем, использует машинную графику. Обладает дополнительными характеристиками, влияющими на сложность. По степени новизны программный модуль относится к группе «В» с поправочным коэффициентом 0,7. Значение поправочного коэффициента, учитывающего использование стандартных модулей типовых программ 0,8.

## 4.2 Расчет затрат на разработку программного обеспечения

Стоимостная оценка ПО предполагает составление сметы затрат, которая в денежном выражении включает следующие статьи расходов:

* заработную плату: основную и дополнительную
* отчисления в фонд социальной защиты населения ;
* отчисления в Белгосстрах от несчастных случаев на производстве ;
* налоги от фонда оплаты труда ;
* материалы и комплектующие ;
* спецоборудование ;
* машинное время ;
* прочие затраты ;
* накладные расходы ;

Расчет основной заработной платы исполнителей, занятых разработкой, проведем на основе исходных данных, представленных в таблице 4.1.

Таблица 4.1 – Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Условные обозначения | Единицы | Норматив |
| Коэффициент новизны |  | ед. | 0,7 |
| Группа сложности |  | ед. | 1 |
| Дополнительный коэффициент сложности |  | ед. | 0,19 |
| Коэффициент, учитывающий степень использования при разработке ПО стандартных модулей |  | ед. | 0,8 |
| Годовой эффективный фонд времени |  | дней | 231 |
| Тарифная ставка 1–го разряда |  | руб. | 34 |
| Коэффициент премирования |  | ед. | 1,5 |
| Дополнительная заработная плата исполнителей |  | % | 20 |
| Ставка отчислений в фонд социальной защиты населения |  | % | 34 |
| Прочие затраты |  | % | 10 |
| Накладные расходы |  | % | 100 |
| Уровень рентабельности |  | % | 15 |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Условные обозначения | Единицы | Норматив |
| Уровень рентабельности |  | % | 15 |
| Ставка НДС (при отсутствии льгот) |  | % | 20 |
| Налог на прибыль при отсутствии льгот |  | % | 18 |
| Норма расходов материалов |  | руб. |  |
| Норма расхода машинного времени |  | маш./ч | 2 |
| Цена одного машино–часа |  | руб. | 10 |
| Норматив расходов на освоение ПО |  | % | 10 |
| Норматив расходов на сопровождение ПО |  | % | 20 |

Перечень и объем функций ПО определил на основе нормативных данных, приведенных в таблице 4.2. Программное обеспечение создавалось на языке программирования C#. Язык С# схож с Visual С++. Поэтому, при определении объема функций ПО, используем данные, характерные для определения строк исходного кода с использованием среды Visual С++ (Microsoft).

Таблица 4.2 – Перечень и объем функций

|  |  |  |  |
| --- | --- | --- | --- |
| Номер функции | Наименование (содержание) | Объем функции, LOC | |
| По каталогу | Уточненный |
| 101 | Организация ввода информации | 150 | 150 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 | 450 |
| 103 | Преобразование операторов входного языка и команды другого языка | 660 | 520 |
| 107 | Синтаксический и семантический анализ входного языка и генерация кодов команд | 5 400 | 3 000 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 3 200 | 4 000 |
| 201 | Генерация структуры базы данных | 4 300 | 400 |
| 204 | Обработка наборов и записей базы данных | 2 670 | 700 |
| 209 | Организация поиска и поиск в базе данных | 5 480 | 1 600 |

Продолжение таблицы 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| Номер функции | Наименование (содержание) | Объем функции, LOC | |
| По каталогу | Уточненный |
| 301 | Формирование последовательного файла | 290 | 200 |
| 305 | Обработка файлов | 720 | 900 |
| 402 | Генерация программ по описанию пользователей | 9 880 | 6 400 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 500 |
| 507 | Обеспечение интерфейса между компонентами | 970 | 400 |
| 707 | Графический вывод результатов | 480 | 980 |
|  | Итого | 35 060 | 20 200 |

Общий объём ПО () определяется исходя из количества и объёма реализуемых функций и рассчитываем по формуле:

Уточненный объем функций равен:

где – общий объем функций, строк исходного кода (LOC);

– уточненный объем функций, строк исходного кода (LOC);

– объём отдельной функций ПО;

– уточненный объём отдельной функций ПО;

– общее число функций.

В формулах (4.1) и (4.2) для расчёта общего объёма ПО используем данные, приведенные в таблице 4.2

(LOC),

(LOC).

На основании принятого к расчету объема и категории сложности определяем нормативную трудоемкость ПО:

ПО принято подразделять на три категории сложности. Категорию сложности ранее определил с руководителем экспертным путем. Нормативная трудоемкость служит основой для определения общей трудоемкости. Ее рассчитываем по формуле:

где – общая трудоемкость ПО, чел./дн.;

– нормативная трудоемкость ПО, чел./дн.;

– коэффициент, учитывающий повышение сложности ПО;

– поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

– коэффициент, учитывающий степень новизны ПО.

Коэффициент сложности рассчитаем по формуле:

где – коэффициент, соответствующий повышению сложности ПО за cчет конкретной характеристики;

n – количество учитываемых характеристик.

Вычисляем коэффициент , учитывающий повышение сложности ПО.

, так как ПО характеризуется функционированием в расширенной операционной среде (имеет связь с другим ПО);

, так как ПО характеризуется наличием интерактивного доступа;

, так как ПО характеризуется обеспечением хранения, ведения и поиском данных в сложных структурах;

, так как ПО имеет 2 характеристики по таблице характеристики категорий сложности ПО.

Подставим значения ,, и в формулу (4.4) и получим:

.

Поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей равен , так как реализуемые функции разработанной программы от 20 до 40 % охватываются стандартными модулями.

Коэффициент новизны , так как создаваемое программное обеспечение относится к ПО, являющимся развитием определенного параметрического ряда ПО, разрабатываемого для ранее освоенных типов конфигурации ПК и ОС (категория новизны B).

На основе вычисленных данных по формуле (4.3) рассчитаем общую трудоемкость:

Основной статьей расходов на создание ПО является заработная плата разработчиков (исполнителей) проекта. На основе общей трудоемкости определим плановое число разработчиков () и плановые сроки, необходимые для реализации проекта в целом (). При этом возможно решить следующие задачи:

– расчет числа исполнителей при заданных сроках разработки проекта;

– определение сроков разработки проекта при заданной численности исполнителей.

Численность исполнителей проекта рассчитаем по формуле:

где – общая трудоемкость проекта, чел./дней;

– срок разработки проекта, лет;

– эффективный фонд времени одного исполнителя, дней

Эффективный фонд рабочего времени рассчитаем по формуле:

где – количество дней в году (365 дней);

– количество праздничных дней в году (9 дней);

– количество выходных дней в году (103 день);

– количество дней отпуска (24 дня).

Таким образом, в расчете на 6 месяцев в проекте в среднем будет занято 3 разработчика. В соответствии со штатным расписанием на разработке будут заняты:

– руководитель проекта (16–й разряд; тарифный коэффициент – 3,72);

– один инженер–программист первой категории (14 разряд; тарифный коэффициент – 3.25);

– один инженер–программист второй категории (13 разряд; тарифный коэффициент – 3.04).

Месячную тарифную ставку каждого исполнителя () определяем путем умножения действующей месячной тарифной ставки 1–го разряда () на тарифный коэффициент (), соответствующий установленному тарифному разряду:

Месячная тарифная ставка первого разряда на данном предприятии принята равной 340 руб. Дневную тарифную ставку рассчитаем, разделив месячную тарифную ставку на установленную при 5–дневной недельной норме рабочего времени расчетную среднемесячную норму рабочего времени в днях – 22 дня:

где − дневная тарифная ставка, руб.;

− месячная тарифная ставка, руб*.*

Месячные и дневные тарифные ставки специалистов: руководителя проекта (, ), инженера–программиста 1–й кате­гории (, ), инженера–программиста 2–й категории (, ) определим ниже:

На основе результатов и исходных данных таблицы 4.1 рассчитаем сумму основной заработной платы () всех исполнителей по формуле:

где – количество исполнителей, занятых разработкой ПО;

– дневная тарифная ставка i–го исполнителя, руб.;

– эффективный фонд рабочего времени i–го исполнителя, дней;

– коэффициент премирования.

Дополнительную заработную плату () посчитаем по формуле:

где – норматив дополнительной заработной платы.

Отчисления в фонд социальной защиты населения () рассчитываем в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей, определенной по нормативу, в целом по организации:

где – норматив отчислений в фонд социальной защиты населения, %.

Отчисления в Белгосстрах от несчастных случаев на производстве () рассчитываем по формуле:

где – норматив отчислений в Белгосстрах от несчастных случаев на производстве, %.

Расходы по статье «Материалы» определяем на основании сметы затрат, разрабатываемой на ПО с учетом действующих нормативов. По статье «Материалы» отражаем расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки ПО. Нормы расходов материалов определяем в расчете на 100 строк исходного кода. Сумму затрат на расходные материалы рассчитываем по формуле:

где – норма расхода материалов в расчёте на 100 строк кода ПО;

– общий объем ПО (строк кода).

Во время разработки данного ПО технические и программные средства специального назначения использоваться не будут. Таким образом расходы по статье «Спецоборудование» не рассчитываем .

Расходы по статье “Машинное время” включают оплату машинного времени, необходимого для разработки и отладки ПО:

где – цена одного машино–часа;

– норматив расхода машинного времени на отладку 100 строк исходного кода, машино–часов.

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно–технической информации и специальной литературы. Рассчитываем расходы по статье «Прочие затраты» по формуле, в соответствии с нормативом, разрабатываемым в целом по научной организации, в процентах к основной заработной плате:

где – норматив прочих затрат в целом по организации, %.

Затраты по статье “Накладные расходы” рассчитываем по формуле:

где – норматив накладных расходов в целом по организации.

Общую сумму расходов по вышеизложенным статьям сметы рассчитываем по формуле:

Таблица 4.3 – Расчет себестоимости и прибыли ПО

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование статей | Условные обозначения | Руб. |
| 1 | Основная заработная плата исполнителей |  | 26 569 |
| 2 | Дополнительная заработная плата исполнителей |  | 5 313 |
| 3 | Отчисления в фонд социальной защиты населения |  | 10 840 |
| 4 | Отчисления в Белгосстрах от несчастных случаев на производстве |  | 191 |
| 5 | Материалы и комплектующие |  | 76 |
| 6 | Машинное время |  | 48 |
| 7 | Расходы на научные командировки |  | 2 657 |
| 8 | Прочие затраты |  | 2 657 |
| 9 | Накладные расходы |  | 26 569 |
| 10 | Полная себестоимость |  | 72 267 |

## 4.3 Расчет стоимостной оценки результата

Результатом использования данного программного продукта является прирост чистой прибыли. Прирост осуществляется за счет общей экономии всех видов ресурсов, включая расходы на заработную плату следующих специалистов: игровых дизайнеров, дизайнеров уровней, специалистов по контролю качества и тестировщиков. В результате использования данного пакета программ, значительно снижается трудоемкость операций по созданию внутриигрового контента. Данные для расчета экономии ресурсов в связи с применением ПО отображены в таблице 4.4.

Так как ПО разрабатывалось для внутреннего использования ООО Халфбас, то общие капитальные вложения соответствуют полной себестоимости данного по и составят 72 267 руб.

Таблица 4.4 – Исходные данные для расчета экономии ресурсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование показателей | Обозначения | Единицы измерения | Значения показателя | |
| В базовом варианте | В новом варианте |
| Капитальные вложения |  | руб. | - | 72 267 |
| Среднемесячная заработная плата одного специалиста |  | руб. | 800 | 800 |
| Среднемесячное число рабочих дней |  | день | 22 | 22 |
| Количество задач, решаемых за год |  | задача | 40 | 40 |
| Средняя трудоемкость работ в расчете на 1 задачу |  | Чел-час на задачу | 40 | 20 |
| Объем работ, выполняемый при решении одной задачи |  | обращений | 20 | 16 |
| Количество часов работы в день |  | часов | 8 | 8 |

Определим общую годовую экономию затрат при использовании пакета вспомогательных программ. Экономию затрат на заработную плату при использовании нового ПО в расчете на количество задач, выполняемых в год, рассчитываем по формуле:

где – экономия затрат на заработную плату при решении задач с использованием нового ПО в расчете на 1 задачу, руб.

– среднемесячная заработная плата, руб.;

– количество задач, решаемых в год в базовом варианте и при использовании нового ПО соответственно, задач;

– средняя трудоемкость работ в расчете на 1 задачу в базовом варианте и при использовании нового ПО соответственно, человеко-часов;

– объем выполняемых работ в расчете на 1 задачу в базовом варианте и при использовании нового ПО соответственно, обращений;

– количество часов работы в день, ч;

– среднемесячное количество рабочих дней.

Экономия затрат на заработную плату составляет:

Внедрение нового ПО позволит компании сэкономить на текущих затратах, и практически сразу получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении , которую определяем по формуле:

где – ставка налога на прибыль, 18%.

Чистая прибыль от экономии текущих затрат составляет:

Разработка данного ПО была начата в 2018 году, внедрение планируется во втором квартале 2018 года, поэтому компания может получить уже в 2018 году 50% прибыли:

Для расчета показателей экономической эффективности использования пакета программ, необходимо полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2018–й год) умножив результаты и затраты за каждый год на коэффициент дисконтирования , который рассчитываем по формуле:

где – норматив приведения разновременных затрат и результатов, 25%

– расчетный год, ;

– номер года, результаты и затраты которого приводятся к расчетному (2018 – 1, 2019 – 2, 2020 – 3, 2021 - 4).

Коэффициентам приведения по годам будут соответствовать следующие значения:

.

Все рассчитанные данные экономического эффекта сведем в таблицу 4.5.

Таблица 4.5 – Экономический эффект

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единица измерения | Годы | | | |
| 2018 | 2019 | 2020 | 2021 |
| Результаты: | | | | | |
| Прирост прибыли за счет экономии затрат | руб. | 35 781 | 71 563 | 71 563 | 71 563 |
| Коэффициент дисконтирования | ед. | 1 | 0,8 | 0,64 | 0,512 |
| Результат с учетом фактора времени | руб. | – | 57 250 | 45 800 | 36 640 |

Продолжение таблицы 4.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единица измерения | Годы | | | |
| 2018 | 2019 | 2020 | 2021 |
| Затраты (инвестиции): | | | | | |
| Инвестиции в разработку продукта | руб. | 72 267 | – | – | – |
| Инвестиции с учетом фактора времени | руб. | 72 267 | – | – | – |
| Экономический эффект: | | | | | |
| Чистый дисконтированный доход | руб. | -36 485 | 57 250 | 45 800 | 36 640 |
| Чистый дисконтированный доход с нарастающим итогом | руб. | -36 485 | 20 765 | 61 100 | 97 740 |

Из приведенной выше таблицы видно, что затраты на разрабатываемое программное обеспечение окупаются менее чем за два года.

В конечном итоге рассчитываем рентабельность инвестиций в разработку и внедрение пакета программ за расчетный период времени по формуле:

где – общие капитальные вложения компании в разработку, руб.

## 4.4 Выводы по технико–экономическому обоснованию

В технико–экономическом обосновании составлена смета затрат и выполнен расчет суммы расходов, связанных с разработкой компонентов. Расчет показывает, что данная разработка является экономически целесообразной, рентабельность инвестиций составляет 86%.

Чистый дисконтированный доход за 4 года эксплуатации составит 97 740 руб., что представляет собой положительный экономический эффект от создания нового программного средства.

Затраты на разработку программного продукта окупятся на второй год его использования.

Таким образом, разработка и применение данного пакета программ внутри компании является эффективной мерой, инвестиции целесообразно осуществлять. Положительный экономический эффект достигается за счет экономии затрат на заработную плату при использовании новой системы, сокращении начислений на заработную плату, экономии затрат на оплату машинного времени и материалы при использовании новой системы в рас­чете на выполненный объем работ.

# ЗАКЛЮЧЕНИЕ

Целью дипломного проекта была разработка пакета сервисных программ для оптимизации процесса создания игровых сценариев, уровней, анимации, работы с ресурсами и локализацией.

Для достижения поставленной цели исследования были предприняты следующие шаги:

- была изучена и проанализирована предметная область, включая сам игровой проект, аналогичные наработки сторонних разработчиков, а также процесс взаимодействия с художниками, игровыми дизайнерами и дизайнерами уровней.

- на основе полученных знаний была разработана общая архитектура системы, структура хранения данных и уровень доступа к ним;

- разработаны средства для отображения и модифицирования этих данных в виде визуального редактора.

спроектирован и разработан пакет сервисных инструментов.

Цели исследования достигнуты. Было разработано и протестировано несколько компонентов редактора, оптимизирующими работу. Эти компоненты были внедрены и успешно используются сотрудниками компании. В работе были рассмотрены особенности работы с движком Unity3D, средствами платформы .NET Framework и Mono.

В дальнейшем предполагается доработка системы за счет улучшения текущих компонентов и создания дополнительных. Так же пакет программ может быть выделен в отдельный продукт, который может продаваться на рынке небольших средств автоматизации.

Экономическая целесообразность разработанного приложения была описана в разделе технико–экономического обоснования. К моменту написания данной пояснительной записки, было создано 12 уровней, с общим игровым временем около 25 часов, разработано 269 игровых сценариев.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Newzoo.com [Электронный ресурс]. – Данные статистики. – Режим доступа: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/

2. Unity Asset Store [Электронный ресурс]. – Электронные данные. – Режим доступа: https://assetstore.unity.com/

3. Spine [Электронный ресурс]. – Электронные данные. – Режим доступа: http://esotericsoftware.com/

4. Animation Processor [Электронный ресурс]. – Электронные данные. – Режим доступа: https://assetstore.unity.com/packages/tools/animation/animation-processor-116688

5. I2 Localization [Электронный ресурс]. – Электронные данные. – Режим доступа: https://assetstore.unity.com/packages/tools/localization/i2

6. Unity Technologies [Электронный ресурс]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/Unity\_Technologies

7. Unity Platforms [Электронный ресурс]. – Электронные данные. – Режим доступа: https://unity3d.com/unity/features/multiplatform

8. Unity Store [Электронный ресурс]. – Электронные данные. – Режим доступа: https://store.unity.com/

9. Mono Scripting [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.mono-project.com/docs/advanced/embedding/scripting/

10. SecondLife LSL [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wiki.secondlife.com/wiki/LSL\_Portal

11. Inkl LSL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.inklestudios.com/ink/

12. Палицын В.А. Технико–экономическое обоснование дипломных проектов: Методическое пособие для студентов всех специальностей БГУИР. Часть 4: Проекты программного обеспечения. Мн.: БГУИР, 2006. – 76с.

13. Рихтер Дж. CLR via C#. Язык программирования C# 2010 и платформа .NET 4.0. – СПб.: Питер, 2012. – 928 c.

14. Шильдт Г. C# 4.0. Полное руководство. – М.: ООО «И. Д. Вильямс», 2011. – 1056 с.

15. Albahari, Joseph. C# 5.0 in a Nutshell / Joseph Albahari, Ben Albahari. – 5th edition. – O’Reilly Media, Inc, 2012. – June. – 1062 P.

16. Троелсен Э. Язык программирования С# 2010 и платформа .NET 4. 5–е издание. – М.: ООО «И. Д. Вильямс», 2010. – 1392 с.;

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг исходного кода некоторых классов приложения**

**EventObject.cs**

using System.Collections.Generic;

using UnityEngine;

using System.Collections;

using Newtonsoft.Json;

public abstract class EventObject: StreamedSettings{

public float wait;

public string description;

public List<string> eventChainKeyTag;

public string callbackName;

public EventObject[] exitNodes;

public string[] exitNamesArray;

public string variableName;

[JsonIgnore]public EventObject parent;//ignored to avoid cyclic ref;

[JsonIgnore]public virtual string[] exitNames{get{return exitNamesArray;}}

[JsonIgnore]public virtual int exitCount{get{return exitNames.Length;}}

public abstract EventObject GetNewInstance();

public void OnPreEnter(List<string> eventChainKeyTag){

this.eventChainKeyTag = eventChainKeyTag.Clone();

EventManager.Instance.RegisterObject(this);

if(wait<=0f){

OnEnter();

}else{

TimeManager.Instance.updateEvent+=Update;

}

}

protected virtual void OnEnter(){

}

public void DebugPrintTags(){

Debug.Log("Entered EventObject: " + key);

Debug.Log("----TAGS----");

foreach(var o in eventChainKeyTag){

Debug.Log(o);

}

Debug.Log(description);

Debug.Log("----END----");

}

public virtual void Update(){

wait-=TimeManager.realDeltaTime;

if(wait<0f){

TimeManager.Instance.updateEvent-=Update;

OnEnter();

}

}

public virtual void OnExit(){

EventManager.Instance.DeregisterObject(this);

}

public virtual void OnRemove(){

TimeManager.Instance.updateEvent-=Update;

EventManager.Instance.DeregisterObject(this);

}

public virtual EventObject Clone(){

var clone = this.MemberwiseClone() as EventObject;

clone.wait = wait;

clone.eventChainKeyTag = eventChainKeyTag.Clone();

clone.description = description;

clone.callbackName = callbackName;

if(exitNodes!=null){

clone.exitNodes = new EventObject[exitNodes.Length];

clone.exitNamesArray = new string[exitNodes.Length];

for (int i = 0; i < exitNodes.Length; i++) {

if(exitNodes[i]!=null && exitNamesArray[i]!=null){

clone.exitNodes[i] = exitNodes[i].Clone();

clone.exitNamesArray[i] = exitNamesArray[i];

}else{

GameUtilities.LogWarning("ExitNodes or ExitNamesArray " + i + " are null");

}

}

}

return clone;

}

}

**EventChain.cs**

using UnityEngine;

using Newtonsoft.Json;

public class EventChain : EventObject {

[JsonIgnore]public string sub\_name = "General";

[JsonIgnore]public bool is\_location\_bound = false;

public EventObject rootNode;

protected override void OnEnter() {

//Do not call base, eventchain should not be registered in active obj

GameUtilities.Log("[Event::EventChain] chain: " + key + " started");

if(!eventChainKeyTag.Contains(key)){

eventChainKeyTag.Add(key);

}

rootNode.OnPreEnter(eventChainKeyTag);

}

public EventChain(){

key = "EventChain";

}

public override void OnExit() {

//TODO:callback for random event ended

// EventManager.Instance.LaunchRandomEvent();

}

public override EventObject GetNewInstance() {

return new EventChain();

}

public override EventObject Clone() {

var clone = base.Clone() as EventChain;

clone.rootNode = rootNode.Clone();

return clone;

}

}

**LocationSettings.cs**

using System.Collections.Generic;

using Newtonsoft.Json;

[System.Serializable]

public class LocationAISettings:Settings {

public List<WeightedSettings> aiCharacterSet;

public List<int> enemyFilter;

public int faction;

public float attackDelay;

public int maxMoney;

public float wallet;

public string GetRandomCharacter() {

return aiCharacterSet.Count > 0 ? aiCharacterSet.WeightedRandom() : null;

}

public LocationAISettings() {

aiCharacterSet = new List<WeightedSettings>();

enemyFilter = new List<int>();

attackDelay = 0f;

wallet = 0f;

maxMoney = 1000;

}

public override object Clone() {

var clone = MemberwiseClone() as LocationAISettings;

clone.aiCharacterSet = aiCharacterSet.CloneSettings();

clone.enemyFilter = enemyFilter.Clone();

return clone;

}

}

[System.Serializable]

public class LocationBackgroundSettings:StreamedSettings {

public List<LocationBackgroundLayerSettings> layers = new List<LocationBackgroundLayerSettings>();

public List<string> citizens;

public int citizenCount;

public string groundDigTile;

public string groundTile;

public string skyGradientKey;

public string weatherEffect;

}

[System.Serializable]

public class LocationBackgroundLayerSettings: Settings {

public List<string> sprites = new List<string>();

public int quantity = 10;

public int order;

public override object Clone() {

var clone = MemberwiseClone() as LocationBackgroundLayerSettings;

clone.sprites = sprites.Clone();

return clone;

}

}

[System.Serializable]

public class LocationSettings:StreamedSettings {

public string nameKey;

public string descKey;

public int initialMoney;

public int globalCharacterCounter;

[JsonIgnore] public int nextCharacterId{ get { return globalCharacterCounter++; } }

[JsonIgnore] public bool randEventsEnabled = true;

public int aiComplexity;

public string backgroundKey;

public List<string> buildings;

public List<float> buildingsPos;

public List<string> characters;

public List<string> rooms;

public List<string> items;

public List<string> junks;

public List<string> upgrades;

public List<string> cops;

public List<string> trucks;

public Dictionary<int,LocationAISettings> locationAISettings;

[JsonIgnore]public string mainQuest{ get { return key + "\_main\_quest"; } }

[JsonIgnore]public LocationBackgroundSettings background{ get { return Database<LocationBackgroundSettings>.GetInstance().GetItem(string.IsNullOrEmpty(backgroundKey) ? Location.defaultLocationKey : backgroundKey); } }

public override object Clone() {

var clone = MemberwiseClone() as LocationSettings;

clone.buildings = buildings.Clone();

clone.buildingsPos = buildingsPos.Clone();

clone.characters = characters.Clone();

clone.rooms = rooms.Clone();

clone.items = items.Clone();

clone.junks = junks.Clone();

clone.upgrades = upgrades.Clone();

clone.cops = cops.Clone();

clone.trucks = trucks.Clone();

if (locationAISettings != null) {

clone.locationAISettings = new Dictionary<int, LocationAISettings>();

foreach (var o in locationAISettings.Keys) {

clone.locationAISettings.Add(o, locationAISettings[o].Clone() as LocationAISettings);

}

}

return clone;

}

}

[System.Serializable]

public class Settings: IClonable {

public string key;

public Settings SetKey(string k) {

key = k;

return this;

}

public virtual object Clone() {

return MemberwiseClone();

}

public override string ToString() {

return key;

}

}

[System.Serializable]

public class StreamedSettings: Settings {

public bool isDefault;

[JsonIgnore]public string path;

}

**BuildingSettings.cs**

using System;

using System.Collections.Generic;

using UnityEngine;

using Newtonsoft.Json;

[Serializable]

public class BuildingLayoutData {

public List<CellSettings> cellSettings;

public List<RoomData> roomData;

public BuildingLayoutData() {

cellSettings = new List<CellSettings>();

roomData = new List<RoomData>();

}

public BuildingLayoutData Clone() {

var clone = new BuildingLayoutData();

clone.cellSettings = cellSettings.CloneSettings();

clone.roomData = roomData.CloneSettings();

return clone;

}

}

[Serializable]

public class BuildingCharacterData {

public List<CharacterData> charData;

public List<int> charAssignments;

public List<Vector3> charPositions;

public BuildingCharacterData() {

charData = new List<CharacterData>();

charAssignments = new List<int>();

charPositions = new List<Vector3>();

}

public BuildingCharacterData Clone() {

var clone = new BuildingCharacterData();

clone.charData = charData.CloneSettings();

clone.charAssignments = charAssignments.Clone();

clone.charPositions = charPositions.Clone();

return clone;

}

}

[Serializable]

public class BuildingDecorData {

public List<Vector3> decorPos;

public List<string> decorId;

public List<bool> decorFlip;

public string facadeNameKey;

public BuildingDecorData() {

decorPos = new List<Vector3>();

decorId = new List<string>();

decorFlip = new List<bool>();

}

public BuildingDecorData Clone() {

var clone = new BuildingDecorData();

clone.decorId = decorId.Clone();

clone.decorPos = decorPos.Clone();

clone.decorFlip = decorFlip.Clone();

return clone;

}

public void Flip(int index){

decorFlip[index] = !decorFlip[index];

}

public void MoveUp(int index){

if(index>0){

var v3 = decorPos[index];

decorPos.RemoveAt(index);

decorPos.Insert(index-1, v3);

var s = decorId[index];

decorId.RemoveAt(index);

decorId.Insert(index-1, s);

var f = decorFlip[index];

decorFlip.RemoveAt(index);

decorFlip.Insert(index-1,f);

}

}

public void MoveDown(int index){

if(index<decorPos.Count-1){

var v3 = decorPos[index];

decorPos.RemoveAt(index);

decorPos.Insert(index+1, v3);

var s = decorId[index];

decorId.RemoveAt(index);

decorId.Insert(index+1, s);

var f = decorFlip[index];

decorFlip.RemoveAt(index);

decorFlip.Insert(index+1, f);

}

}

public void Remove(int index){

decorPos.RemoveAt(index);

decorId.RemoveAt(index);

decorFlip.RemoveAt(index);

}

}

[Serializable]

public class BuildingAIData {

public float AIDailyIncome;

public int AIDefenceCount;

public int AIDropChance;

public int AIMinAttackCount;

public int AIMaxAttackCount;

public float AIAttackTimeLimit;

public BuildingAIData Clone() {

return MemberwiseClone() as BuildingAIData;

}

}

[Serializable]

public class BuildingSettings: StreamedSettings {

public string nameKey;

public string customName;

public bool attackable = true;

public int upperFloors;

public int verticalSize;

public int horizontalSize;

public int fraction;

public int initialElectricity;

public BuildingAIData ai;

public BuildingLayoutData layout;

public BuildingCharacterData characters;

public BuildingDecorData decor;

public List<UpgradePropertyChange> buildingUpgrades;

[JsonIgnore]public string sub\_name = "General";

[JsonIgnore]public bool is\_location\_bound = false;

public override object Clone() {

var clone = MemberwiseClone() as BuildingSettings;

clone.layout = layout.Clone();

clone.characters = characters.Clone();

clone.decor = decor.Clone();

clone.ai = ai.Clone();

return clone;

}

}

**\_ImportSettings.json**

{

"$type": "SpriteSheetImportSettings, Assembly-CSharp-Editor",

"animationSubfolder": "Trucks",

"defaultRectWidth": 80,

"defaultRectHeight": 40,

"alignment": 7,

"repeating": {

"$type": "SpriteSheetRow[], Assembly-CSharp-Editor",

"$values": [

{

"$type": "SpriteSheetRow, Assembly-CSharp-Editor",

"isAnimation": true,

"isLooped": true,

"key": "move"

}

]

},

"key": "Trucks"

}

**LocationExample.json**

{

"$type": "LocationSettings, Assembly-CSharp",

"nameKey": "story\_01\_tutorial",

"descKey": "story\_01\_tutorial\_desc",

"initialMoney": 875,

"globalCharacterCounter": 0,

"aiComplexity": 0,

"backgroundKey": "a\_lvl\_1\_tutorial",

"buildings": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"story\_01\_tutorial\_building\_0",

"story\_01\_tutorial\_building\_1",

"story\_01\_tutorial\_building\_2"

]

},

"buildingsPos": {

"$type": "System.Collections.Generic.List`1[[System.Single, mscorlib]], mscorlib",

"$values": [

42.84,

50.2,

56.6

]

},

"characters": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"employee\_damage",

"employee\_sell"

]

},

"rooms": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"employee",

"rest",

"production\_green\_fnt",

"electricity\_fnt",

"storage\_fnt",

"sell1\_fnt",

"money\_storage\_fnt",

"rest\_fnt"

]

},

"items": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"tier4\_Melee\_Hands"

]

},

"junks": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"junkie4",

"junkie3",

"junkie2",

"junkie1"

]

},

"upgrades": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"upgrade\_employee1"

]

},

"cops": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": []

},

"trucks": {

"$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",

"$values": [

"0\_moto"

]

},

"locationAISettings": {

"$type": "System.Collections.Generic.Dictionary`2[[System.Int32, mscorlib],[LocationAISettings, Assembly-CSharp]], mscorlib"

},

"isDefault": false,

"key": "story\_01\_tutorial"

}