Save System

for Opsive Character Controllers

Инструкция

v1.0

Save System for Opsive Character Controllers

ВАЖНО!

Данный перевод любительский, может содержать различные ошибки и не точности перевода. Правильность перевода не является истинной последней инстанции. Перевод сделать для облегчения процесса использования ассета. Автор это делает только для себя, но не против поделится переводом и с другими.

Содержание

Часть 1: Добро пожаловать в Систему сохранении (Save System)

Спасибо за использование Save System для Opsive Character Controllers! Этот asset добавляет сохранение одного игрока поддерживает (сохранение и загрузку игр и сохранение изменений при смене сцен) для контроллеров персонажей Opsive.

Это asset подходит к контролером:

- Ultimate Character Controller
- First Person Controller
- Third Person Controller (version 2)
- UFPS: Ultimate First Person Shooter (version 2)
- UFPM: Ultimate First Person Melee
- UTPS: Ultimate Third Person Shooter
- UTPM: Ultimate Third Person Melee

Если вы используете старую версию контроллера (UFPS version 1) то нужно использовать Save System for UFPS, установите этот пакет (asset).

Как получить помощь

Пишите нам сюда! Если вы застряли, у вас есть вопросы или вы хотите попросить функцию, пожалуйста свяжитесь с нами:

Email: support@pixelcrushers.com
Forum: https://pixelcrushers.com/phpbb
Discord: https://discord.gg/Nt9dVj

Примечание. Этот ресурс разработан и поддерживается Pixel Crushers, а не Opsive.

Демонстрация (Demo)

Что бы запустить демо:

1. Импортируйте контроллер Opsive и систему сохранений.
2. Добавьте следующие сцены в список сцен (уровней) в настройках проекта (сборки):
   - Pixel Crushers ► Save System for Opsive ► Demo ► DemoScene1
   - Pixel Crushers ► Save System for Opsive ► Demo ► DemoScene2
3. Откройте обе сцены и повторно сохраните их. Это сохранит тип контроллера который вы используете.
4. Запустив DemoScene1. Вы можете следующее:
   - Нажмите Escape, чтобы открыть демонстрационное меню, содержащие кнопки для сохранения и загрузки игры.
   - Использовать переходы (порталы) между уровнями DemoScene1 и DemoScene2.

В оставшейся части этого инструкции описывается как настроить сцены и объекты для сохранения игры. А также описание других функций которые не входят в демо сцены.

Часть 2: Настройки

Следуйте шагам описанным в этой граве, чтобы настроить систему сохранений в своем проекте. Вы можете переменять компоненты системы сохранения к префабам, а также к объектам на вашем уровне.

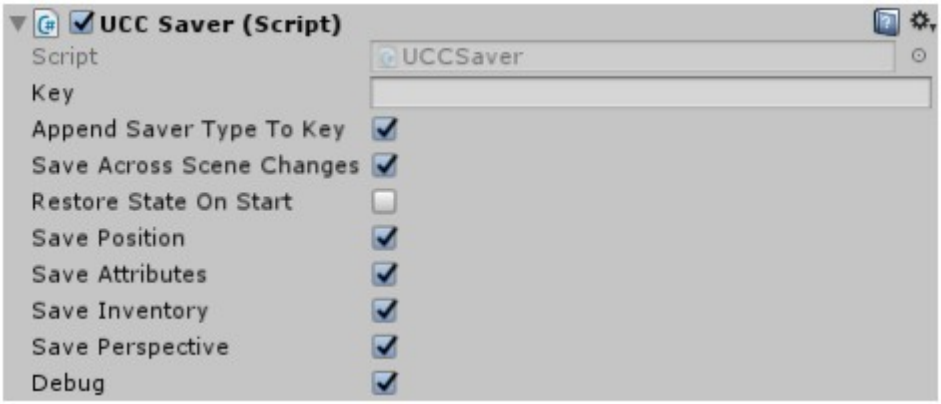
Настройка сцены

Добавьте на сцену префаб Save System из Pixel Crushers ► Save System for Opsive ► Prefabs to your starting scene. При желании вы также можете добавить его в другие игровые сцены. Но это должно быть в первой сцене который вы загрузите в игру (например, сцена главного меню).

Если вы хотите добавить демонстрационное меню в вашу собственную сцену, добавьте префаб Save System Test Menu.

Игрок

Добавьте **UCC Saver** к своему игроку.



Это сохранит статистику игрока, инвентарь, позицию и текущий вид, если использовать контроллер, который может переключаться между видами от первого и третьего лица.

Данные в сохраненных играх хранятся под ключами. Вы можете указать значение в поле Ключ (**Key**) или просто оставить его пустым по умолчанию имя GameObject (например, Nolan). Убедитесь, что галочка Сохранить изменения между сценами (**Save Across Scene Changes**) установлена.

Чтобы записать отладочную информацию в консоль, установите флажок Отладка (**Debug**).

Если вы заинтересованы только в сохранении информации об игроке, вот и все. Остальная часть инструкции объясняет, как настраивать другие функции, такие как порталы смены сцен, заставки для других типов объектов, контрольные точки сохраняет и скриптовый API.

Сохранения и ключи

Система сохранения использует компоненты, называемые «хранителями» (**savers**), для сохранения и восстановления информации. **UCC** игрока компонент «Храниетль» (**Saver)** является одним из примеров. Система сохранения также включает несколько других типов.

Каждый компонент нуждается в уникальном ключе для записи своих данных в сохраненных играх. Если ключ (**Key**) поле пустое, оно будет использовать имя **GameObject** (например, «Orc»). Если вы установите флажок Добавить тип записи к ключу, ключ также будет использовать тип записи (например, «Orc_PositionSaver»). Это полезно, если **GameObject** имеет несколько компоненты заставки.

Если **GameObjects** имеют одинаковое имя, вам нужно будет назначить уникальные ключи. В противном случае они все будут пытаться записывать свои данные под одним ключом, перезаписывая друг друга. Чтобы автоматически назначать уникальные ключи каждый хранитель (**saver**) в сцене, выберите пункт меню Tools > Pixel Crushers > Common > Save System > Assign Unique Keys….

Вы также можете написать свои собственные компоненты Saver. Скрипт стартового шаблона в Plugins / Pixel Crushers /Common / Templates содержат комментарии, которые объясняют, как написать свои собственные вкладчики.

Позиция, Актив, выключатели и сохранение анимации

Чтобы сохранить позицию GameObject, добавьте **Position Saver**.

Чтобы сохранить состояние аниматора в GameObject, добавьте **Animator Saver**.

Чтобы сохранить активное / неактивное состояние GameObject, добавьте **Active Saver**. Чтобы сохранить включенное состояние компонента, добавить **Enabled Saver**. В обоих случаях поместите заставку на объект GameObject, который гарантированно будет активным. Для сохранения состояния пикапа (подбираемых предметов) или разрушаемого объекта, используйте инструкции ниже.

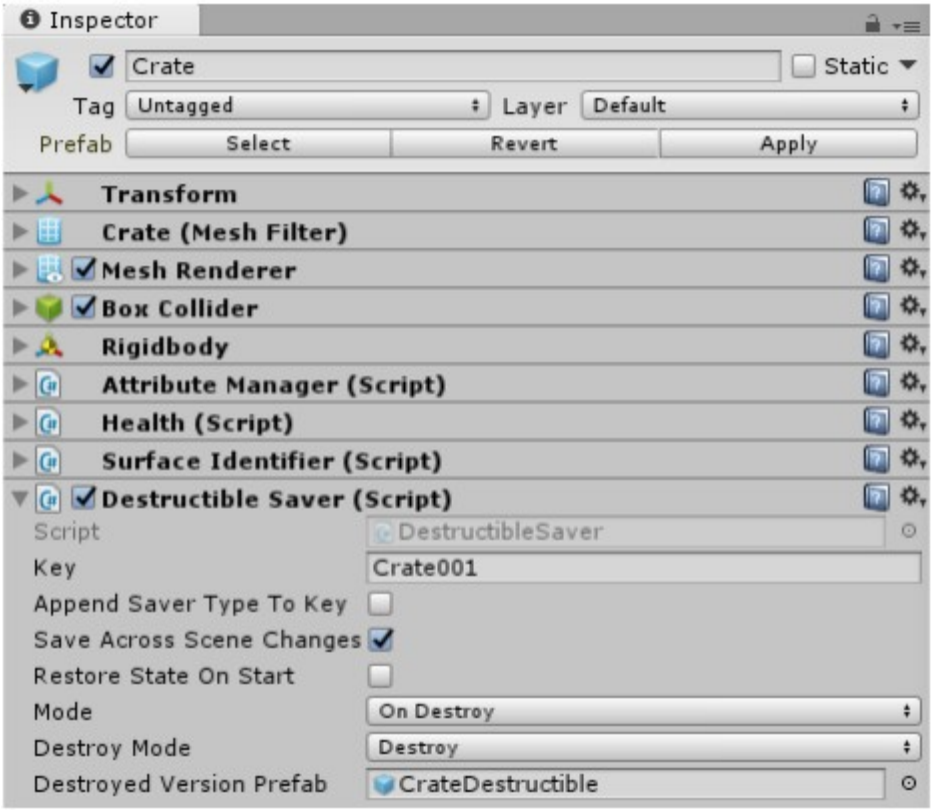
Пикапы (подбираемые объекты) и разрушения

Что бы сохранить пикап или разрушаемый объект, добавьте к нему **Destructible Saver**. Некоторые объекты такие как турели могут не правильно обрабатываться или отображаться, в этом случает установите **Destroy Mode** на деактивировано вместо активно.

Если разрушаемый объект оставляет после себя свою разращенную версию, назначьте prefab с этим вариантом в **Destroyed Version Prefab**. Обычно такой же вариант префаба используют в **Spawned Objects On Death** в компоненте **Health**.

Установите флажок **Save Across Scene Changes**, чтобы запомнить изменения при выходе из сцены и возвращении к ней. Обратите внимание, что это увеличит размер сохраненного игрового файла.

Скриншот ниже показывает настройки для разрушаемого ящика в качестве примера.



Спавн объектов (Spawned Objects)

В этом разделе описывается, как настроить спавн объектов для их сохранения.

Объекты которые создаются вовремя игры контролируются так называемым **Spawned Object Manager**. Хоть и в его название есть слово **Spawned** «порожденный» он не имеет нечего общего с компонентом Opsive's **Respawner**. Spawned «порожденный» в этом случае обозначает, что объект был создан во время игры после уничтожения уничтожаемого объекта например ящика. **Spawned Object Manager** отслеживает объекты которые могут быть разращены. Когда вы запускаете игру он восстанавливает эти объекты.

Ниже описан процесс настройки:

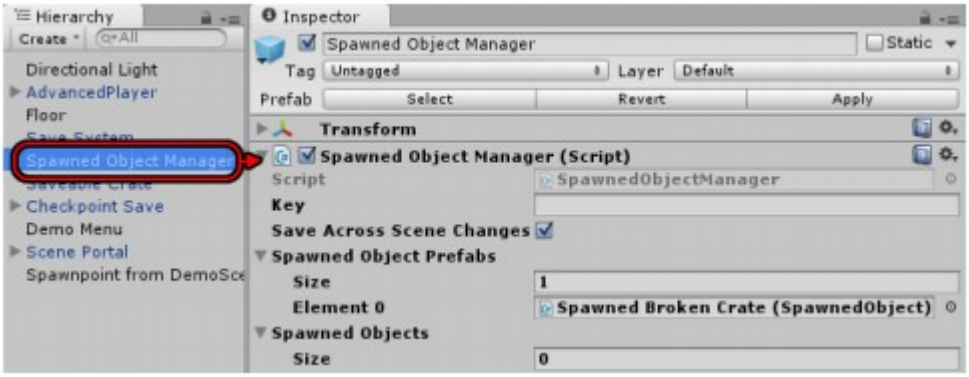
1. Создайте копию префаба целый / уничтоженный и добавьте Spawned Object.
2. Добавьте на сцену **Spawned Object Manager** и добавьте в него префаб разрушаемого объекта.
3. Настройте объекты для удаления экземпляров этого префаба вместо обычного префаба Opsive.

Создание Spawned object префабам

Найдите префаб (например CrateDestructible) и сделайте копию, например с именем Spawned Broken. Добавьте к нему компонент **Spawned Object**. Повторите это для всех предметов которые игрок может создать.

Создание Spawned Object Manager

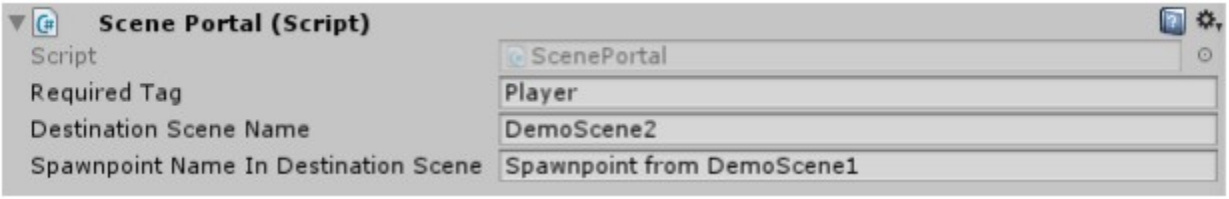
Перетащите префаб **Spawned Object Manager** из папки Prefabs или создайте постой GameObject и добавьте компонент **Spawned Object Manager** как показано ниже. Каждая сцена (уровень) должны иметь собственный **Spawned Object Manager.**



Добавьте все создаваемые объекты (Spawned Object) которые игрок может создать (в данном примере разрушить). Если объект отсутствует в списке то менеджер не сможет перезапустить его при загрузке игры или возвращении на сцену.

Переходы между уровнями (Scene Portals)

Чтобы настроить переход на другую сцену (уровень), перетащите префаб **Scene Portal** из папки Prefabs на свою сцену (уровень) или создайте пустой GameObject с триггером и добавьте компонент Scene Portal на него.



Если установить обязательный тег, то портал будет реагировать только на GameObjects с этим тегом. **ВАЖНО:** Контролер персонажей Opsive помечает объект (модель) игрока как Player но не его коллайдер на дочернем объекте в виде капсульного коллайдера. Установите для этого коллайдера тег Player.

Установите **Destination Scene Name** для сцены, к которой ведет этот портал. Убедитесь, что сцена назначения находится в настройки сборки вашего проекта.

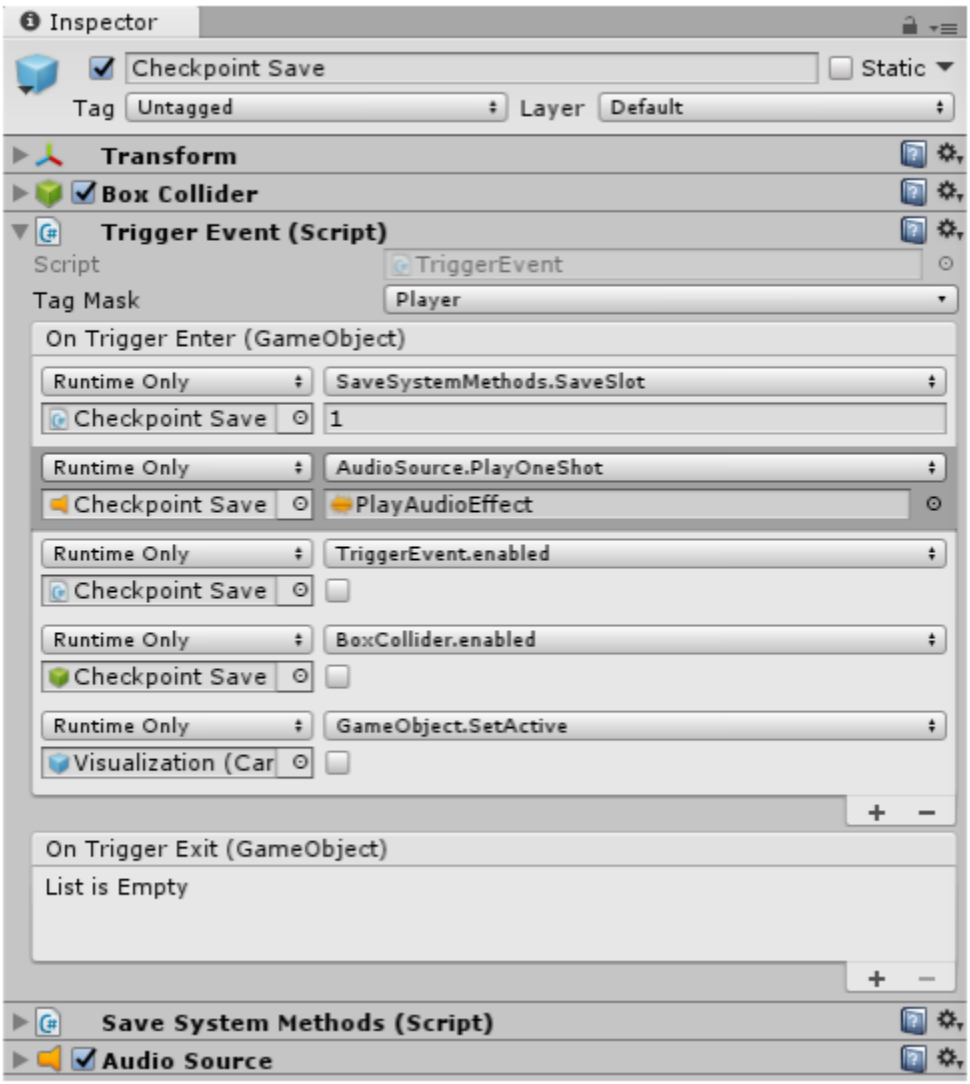
В сцене назначения создайте пустой GameObject, где должен появиться игрок. Это называется точка возрождения. В компоненте **Scene Portal** исходной сцены установите имя **Spawnpoint In Destination** поле сцены с именем этой точки появления.

Убедитесь, что ваша точка возрождения находится достаточно далеко от порталов сцены (перехода на другую сцену) в сцене назначения, чтобы не вызывать немедленный переход на другую сцену. (*Просто не ставьте точку появления игрока рядом с переходом не предыдущую сцену!*)

Если вы не хотите использовать триггеры, вы можете вручную вызвать метод ScenePortal.UsePortal.

Точки сохранения (Checkpoints Saves)

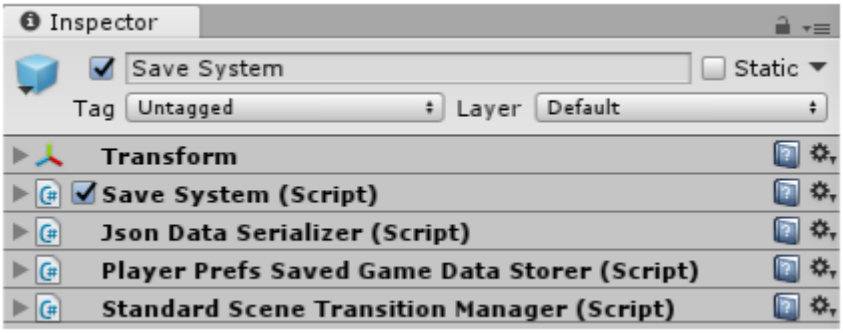
Чтобы настроить триггер точки сохранения перетащите префаб **Checkpoint Save** из папки Prefabs в сцену иди добавьте к пустому GameObject триггер и **Trigger Event** с **Save System Methods**. Префаб предварительно сконфигурирован для сохранения в слоте 1, воспроизведения аудио клипа и скрытия контрольной точки.



Если вы добавляете компоненты вручную, настройте событие «Trigger Event’s» на «Trigger Enter» для вызова SaveSystemMethods.SaveSlot и укажите номер слота. При желании отключите триггер после сохранение контрольной точки путем добавления событий, как показано выше.

Если вы хотите, чтобы игрок автоматически возродился на последней контрольной точке, сохраните ее после смерти, замените компонент **Character Respawner** персонажа игрока на **Checkpoint Character Respawner**.

Система сохранения (Save System)



Система сохранения GameObject действует как постоянный синглтон, что означает, что он переживает изменения сцены и как правило, есть только один экземпляр. Если вы переключитесь на сцену, которая также имеет Save System GameObject, исходная система сохранения GameObject уничтожит объект в новой сцене, чтобы обеспечить наличие только одного объекта такого вида.

Система сохранения основана на двух типах компонентов:

- **Data Serializer**: преобразует двоичные сохраненные игровые данные в сохраняемый формат.
- **Saved Game Data Storer**: записывает сериализованные данные в постоянное хранилище, такое как PlayerPrefs или файлы локального диска.

По умолчанию система сохранения использует Json Data Serializer. Если вы хотите использовать другой сериализатор, вы можете добавить собственную реализацию класса **DataSerializer** в **Save System GameObject.**

По умолчанию система сохранения использует **PlayerPrefs Saved Game Data Store** для сохранения игр в PlayerPrefs. Система сохранения также поставляется с **Disk Saved Game Data Storer**, который сохраняет игры в зашифрованные файлы на локальном диске; чтобы использовать его, удалите **PlayerPrefs Saved Game Data Store** (если имеется) и добавьте **Disk Saved Game Data Storer**. Если вы хотите хранить игры другим способом, вы можете добавить собственную реализацию класса SavedGameDataStorer.

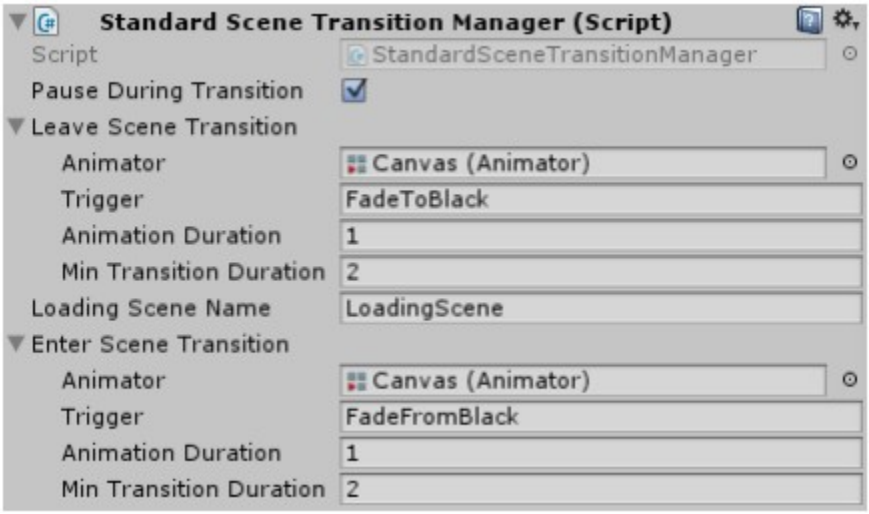
Чтобы загрузить и сохранить игры, вы можете вызвать методы, описанные в разделе Скриптинг (Scripting), чтобы загрузить и сохранить игры, а также изменить сцены.

Если вы не хотите создавать какие-либо сценарии, добавьте компонент **Save System Methods** к своим кнопкам пользовательского интерфейса и настройте их события OnClick () для вызова соответствующих методов, таких как LoadFrom Slot или Save Slot.

Система сохранения также имеет дополнительный диспетчер перехода сцены, описанный ниже.

Менеджер смены сцен (Scene Transition Manager)

Система сохранения имеет **Standard Scene Transition Manager**, который позволяет воспроизводить анимации outro и intro и / или отображать сцену загрузки при загрузке следующей фактической сцены. Это необязательно. Если вы не хотите использовать его, вы можете удалить его.



Если установлен флажок **Pause During Transition**, убедитесь, что аниматору (ам) задано значение **Unscaled Time**.

Если присутствует Scene Transition Manager, система сохранения будет:

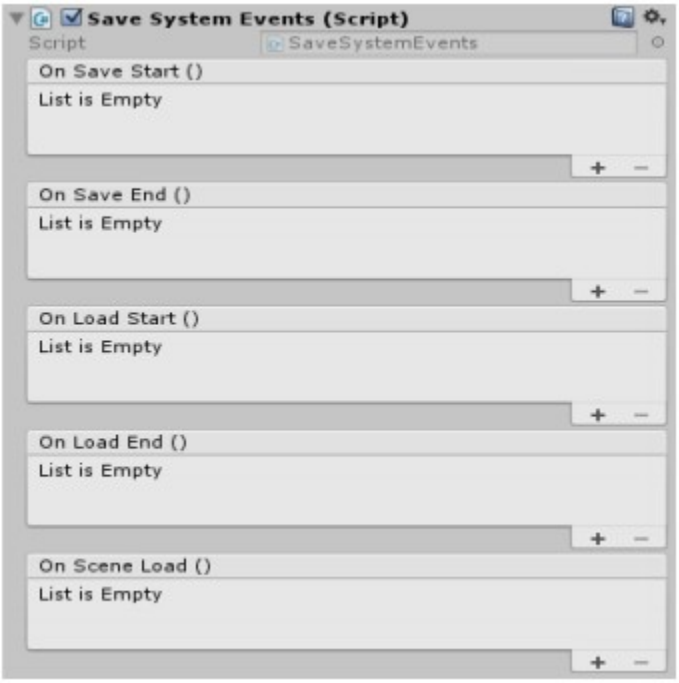
1. Установит триггер аниматора Leave Scene Transitions (если указан).
2. Загрузит сцену загрузки (если указано).
3. Асинхронно загрузит следующую актуальную сцену.
4. После того, как фактическая сцена будет загружена, установит триггер Enter Scene Transitions (если указан).

Стандартный префаб Save System включает одну секунду постепенного затухания при выходе из старой сцены и одну секунду постепенного исчезновения черного при входе в новую сцену. Это позволяет системе сохранения переместить объекты GameObject в новую сцену, пока она покрыта черным.

Если вы хотите использовать разные эффекты между сценами, вы можете написать свой собственный подкласс **Scene Transitions Manager** и добавьте его в свою систему сохранения.

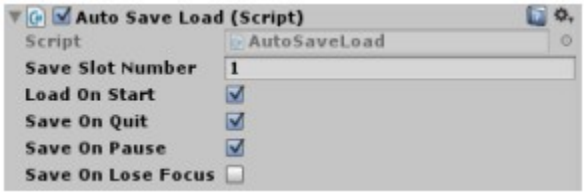
Система сохранения событий (Save Systeam Events)

Компоненты Save System Events позволяют подключать события в инспекторе.



Авто сохранения и загрузка

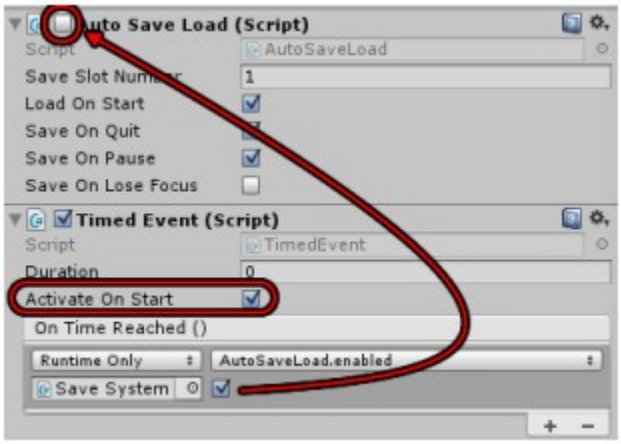
Мобильные игры обычно автоматически сохраняются, когда игрок закрывает игру, и автоматически загружаются, когда игрок возобновляет игру. Чтобы добавить это поведение в вашу игру, добавьте компонент Auto Save Load в систему Save System:



Установите флажок **Load On Start**, чтобы загрузить сохраненную игру (если она существует) при запуске, и **Save On Quit**, чтобы сохранить игру при выходе из приложения. Поставьте галочку **Save On Pause**, чтобы также сохранить игру, когда игрок делает паузу / минимизирует ее. Таким образом, игра будет сохранена правильно, если игрок приостановит приложение и затем убьет его, вместо того, чтобы нормально выходить из него в самом приложении. Установите флажок **Save On Lose Focus**, чтобы сохранить игру, когда приложение теряет фокус.

Если в вашей игре используется заставка, возможно, вы не захотите включать Auto Save Load до тех пор, пока первая сцена фактически не загрузится. В противном случае, если игрок свернет приложение во время заставки, компонент **Auto Save Load** может сохранить игру в исходном

состоянии до загрузки ранее сохраненных данных. Чтобы настроить это, отключите автосохранение загрузки и используйте компонент **Timed Event**, чтобы включить его при запуске:



Часть 3: Скриптинг (Scripting)

Система сохранения имеет очень простой интерфейс сценариев.

Классы SavaSystem

Эти методы доступны в классе SaveSystem:

public static void LoadFromSlot(int slotNumber)
Загружает ранее сохраненную игру из слота.

public static void SaveToSlot(int slotNumber)
Сохраняет текущую игру в слот.

public static void DeleteSavedGameInSlot(int slotNumber)
Удаляет данные в сохраненном игровом слоте.

public static void LoadScene(string sceneNameAndSpawnpoint)
Загружает сцену, опционально позиционируя игрока в указанной точке появления. Параметр является строка, содержащая имя сцены для загрузки, за которой может следовать "@spawnpoint", где «spawnpoint» - это имя GameObject в этой сцене. Игрок будет порожден при этом Позиция GameObject. Этот метод неявно вызывает RecordSavedGameData () перед выходом из текущей сцены и вызывает ApplySavedGameData () после загрузки новой сцены.

public static SavedGameData RecordSavedGameData()
Возвращает объект SavedGameData, содержащий сохраненные данные из текущей игры.

public static void ApplySavedGameData(SavedGameData savedGameData)
Применяет объект SavedGameData к текущей игре.

public static void LoadGame(SavedGameData savedGameData)
Загружает игру, ранее сохраненную в объекте SavedGameData

public static void LoadScene(string sceneName, string spawnpointName = null)
Загружает сцену, опционально помещая игрока в точку появления в новой сцене.

public static void RestartGame(string startingSceneName)
Удаляет текущие сохраненные игровые данные и перезапускает игру с указанной сцены.

public static void LoadAdditiveScene(string sceneName)
Аддитивно загружает сцену в текущую среду и говорит ее Savers применять их состояния из текущих сохраненных игровых данных.

public static void UnloadAdditiveScene(string sceneName)
Выгружает ранее загруженную сцену.

Класс Saver

Saver - это базовый класс для любых компонентов, которые записывают данные для сохраненных игр. Вы можете создавать подклассы для расширения данных, сохраняемых системой сохранения. Начальный шаблон с подробными комментариями предоставляется в Plugins ► Pixel Crushers ► Common ► Templates ► SaverTemplate.cs. Комментарии содержат подробные инструкции. Вы также можете ссылаться на другие подклассы, такие как DestructibleSaver.cs и PositionSaver.cs для примера.

DataSerializer class

DataSerializer - это базовый класс для сериализаторов данных, который система сохранения может использовать для сериализации и десериализации сохраненных игровых данных. Подкласс по умолчанию — JsonDataSerializer.

Класс SavedGameDataStorer

SavedGameDataStorer является базовым классом для хранения данных. Хранилище данных пишет и читает SavedGameData объект из некоторого места хранения, например PlayerPrefs или файл на диске. Подкласс по умолчанию - PlayerPrefsSavedGameDataStorer, но система сохранения также включает DiskSavedGameDataStorer, который сохраняет в зашифрованные файлы на поддерживаемых платформах (например, на рабочем столе).

События (Events)

Вы можете зарегистрировать слушателей для этих событий C # в классе SaveSystem:

```
public static event SceneLoadedDelegate sceneLoaded = delegate { };
public static event System.Action saveStarted = delegate { };
public static event System.Action saveEnded = delegate { };
public static event System.Action loadStarted = delegate { };
public static event System.Action loadEnded = delegate { };
```

Методы системы сохранения

Чтобы получить доступ к методам **SaveSystem** без использования сценариев, например, в событии OnClick () кнопки пользовательского интерфейса, добавьте в сцену компонент **Save System Methods**. Этот компонент предоставляет методы **SaveSystem** класс для инспектора.

Часть 4: Сериализаторы (Serializers)

Сериализатором по умолчанию является Сериализатор данных JSON. Однако вы можете удалить это и добавить сериализатор двоичных данных, если хотите сериализовать в двоичный формат. Если ваши Savers используют типы, которые не сериализуемы, вы потребуется создать подкласс Bial Data Serializer и добавить суррогаты сериализации. Для примеров, смотрите сценарии BinaryDataSerializer.cs, Vector3SerializationSurrogate.cs и QuaternionSerializationSurrogate.cs.

Если вы хотите сериализовать в другой формат, вы можете написать новый подкласс DataSerializer.

Часть 5: Сохранение игровых данных

По умолчанию игры сохраняются в PlayerPrefs Saved Game Data Store. Однако вы можете добавить Disk Saved Game Data Store, если вы хотите сохранять игры на локальных дисковых файлах.

Если вы хотите сохранить другим способом, например, в базе данных или по сети, вы можете написать новый подкласс SavedGameDataStorer.