Sketch Dialogue

Шаблон для диалогов

Документация для Unity3D

Версия: 1.0

Автор: Perfect Human Apps

Обновление: 02 мая 2021

Сайт: Perfect-Human.com

Спасибо за покупку и использование шаблона Sketch Dialogue!

Данный шаблон позволяет в кратчайшие сроки внедрить в ваш проект модуль ведения диалогов между игроком и игровыми персонажами. Кроме того, функционал шаблона позволяет напрямую влиять на характеристики персонажа по результатам решений игрока.

Фронт-энд выполнен на языке **C#** (в среде Unity3D), серверный бэк-энд работает в связке **MySQL+PHP**. Клиент-Серверная архитектура является опциональной, все сценарии могут быть описаны внутри кода для полностью автономного приложения или подгружаться с вашего сервера по требованию.

Важно! Если вы планируете использовать клиент-серверную архитектуру вам потребуются базовые знания о **клиент-серверном** взаимодействии. Кроме того, для тонкой настройки шаблона крайне желательно знание MySQL и PHP. Данная документация снабжена развернутым путеводителем по каждому этапу работы как на фронт-энде (в Unity), так и на бэк-энде (PHP скрипты и обвязка с MySQL сервером). Однако, такие этапы как развертывание своего собственного сервера (на локальной машине или хостинге), регистрация и/или получение токена авторизации на партнерских сайтах будут упомянуты лишь вскользь, со ссылками на ресурсы, где подобные этапы разобраны наиболее подробно. В любом случае, вы должны чётко понимать какие данные и откуда вы будете получать для формирования своего набора сценариев.

В случае каких-либо затруднений вы можете связаться со мной. В разделе "**Обратная связь и поддержка**" указаны все контакты.

Пожалуйста, оставьте отзыв на Asset Store. Обратная связь с вами крайне важна и позволит своевременно улучшать продукт.

СОДЕРЖАНИЕ

1. КРАТКИЙ ОБЗОР КОМПОНЕНТОВ

1.1 Условные обозначения.

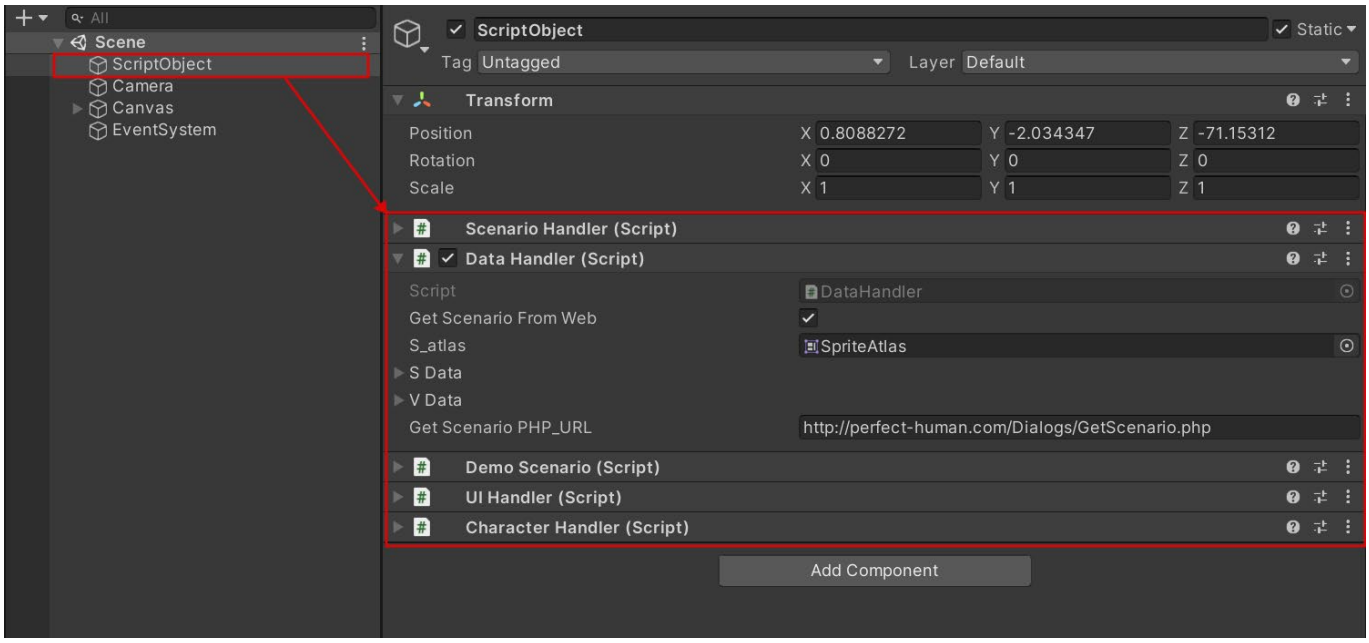
Для облегчения ориентирования в тексте документации предусмотрены следующие цветовые коды для выделения объектов:

Класс
Процедура
Переменная C#
Объект Unity Editor
Скрипты PHP
Таблица SQL
Переменная PHP

Обратите внимание, что некоторые объекты могут иметь несколько цветовых кодов. Например, **ScenarioHandler**.StartScenario() означает, что процедура StartScenario() будет вызвана из синглтон-класса **ScenarioHandler.**

1.2 Набор управляющих классов.

Для работы шаблона применяется набор из управляющих C# скриптов-синглтонов, прикрепленных к единому объекту ScriptObject.



DataHandler. Задает структуру данных, хранит информацию о полученном сценарии, а также отвечает за получение данных с сервера.

ScenarioHandler. Обработчик всех действий сценария.

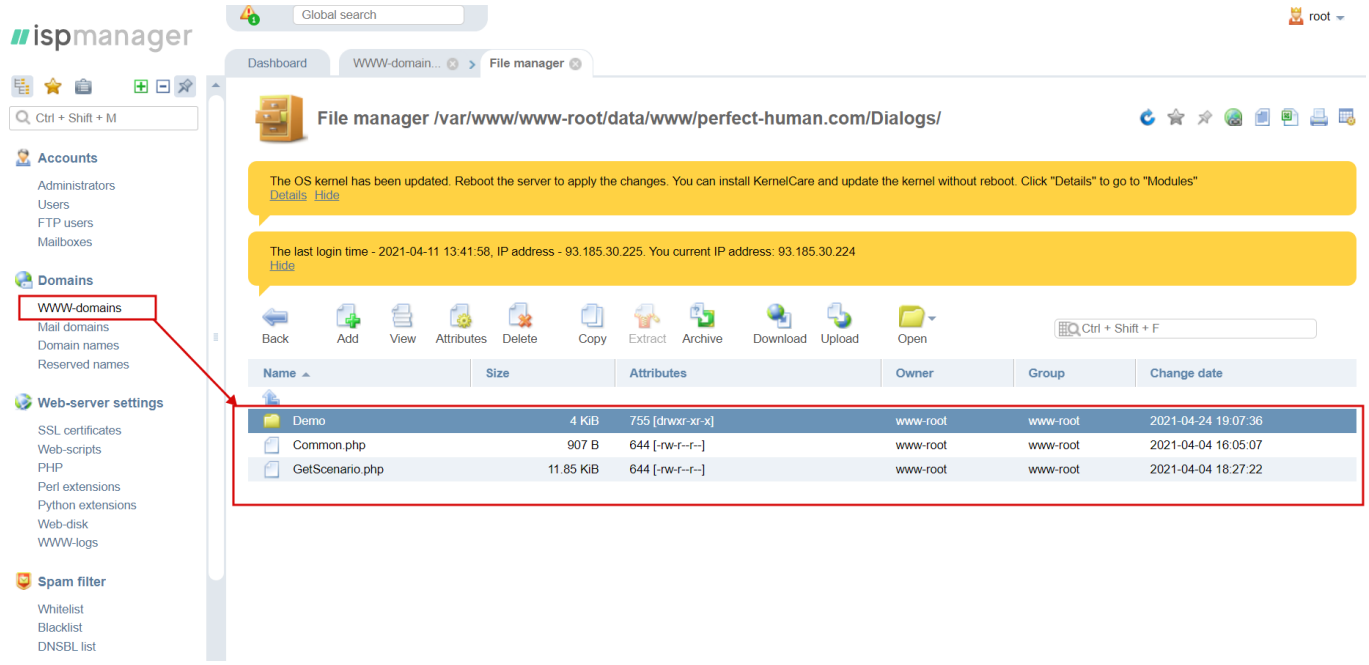
UIHandler. Обработчик для вывода меню и служебных сообщений.

DemoScenario. Содержит демонстрационный сценарий.

StatInfo. Компонента для отображения характеристик персонажа в списке генерации.

SnapScrolling. Компонента управления горизонтальным слайдером (при загрузке).

1.3 Серверные PHP скрипты для взаимодействия с Unity.



Данные скрипты обеспечивают взаимодействие между клиентом и сервером MySQL. Соединение напрямую с сервером не происходит по соображениям безопасности.

GetScenario.php Получение демонстрационного сценария.

Common.php Общий скрипт, обеспечивает служебную информацию для соединения с базой данных и безопасность от SQL-инъекций.

2. НАСТРОЙКА КЛИЕНТСКИХ МОДУЛЕЙ

2.1 Модуль хранения и получения данных.

2.1.1 Основные функции.

Вся обработка и получение данных осуществляется в классе **DataHandler**.

Get_Scenario_from_Web()

Отвечает за получение данных с сервера. В демонстрационных целях загрузка осуществляется небольшого сценария. После получения JSON по API результат укладывается в стек классов сценария с инициализацией всех переменных. В конце процедуры осуществляется вызов демонстрационного сценария **ScenarioHandler**.StartScenario(sData.scenarios[0]).

GetSpriteFromAtlas(string **SpriteID**, string **Prefix** = "")

Поиск спрайтов в атласе спрайтов с помощью **GetSpriteFromAtlas**(string **SpriteID**, string **Prefix** = ""), где SpriteID – идентификатор спрайта в атласе, а Prefix – опциональный префикс в случае если у вас много повторяющихся наименований картинок.

AddNewScenarioCharacter(List<ScenarioCharacter> **cc**, int **id**,string **s_name**, int **s_place**,string **s_image**)

Функция для добавления персонажа в сценарий.

Параметр **id** – идентификатор персонажа.

Параметр **s_name** – строка с именем персонажа.

Параметр **s_place** – число, порядковый номер стартовой позиции персонажа. 0 – персонаж игрока. 1 – положение слева. 2 – Справа, сверху. 3 – Справа, снизу.

Параметр **s_image** – строка-идентификатор для изображения персонажа. Желательно поместить изображение в папку с атласом.

AddNewStage(List<ScenarioStage> cl, int **s_StageOrder**, int **s_PersonID**, string **s_StageType** = "Talk", int **s_Special** = 0, float s_**Delay** = 0)

Функция для добавления этапа в сценарий.

Параметр **s_StageOrder** – порядок этапа в стеке этапов сценария.

Параметр **s_PersonID** – идентификатор персонажа.

Параметр **s_StageType** – тип этапа (1 – проверка на характеристики, 2 – проверка на реплики).

Параметр **s_Special** – указатель на системный тип этапа с особым обработчиком.

Параметр **s_Delay** – задержка перед новым сценарием.

AddReplica(List<ScenarioReplica> **cr**, int **s_id**, string **s_Text**, string **s_ReplicaType** = "Story")

Функция для добавления реплик для определенного этапа сценария.

Параметр **s_id** – идентификатор реплики.

Параметр **s_Text** – строка, текст реплики.

Параметр **s_ReplicaType** – внешний вид “облака” реплики.

AddNewCondition(List<CommonConditions> **cc**, int **s_IDCheck** = 0, int **s_TypeOfCondition** = 2, string **s_StatToCheck** = "Replica", int **s_Min** = 1, int **s_Max** = 0)

Функция для добавления условий для этапов или реплик. Может быть применена как для определения условий возникновения, так и для определения количества генерируемых характеристик персонажа.

Параметр **s_IDCheck** – идентификатор проверки.

Параметр **s_TypeOfCondition** – тип проверки.

Параметр **s_StatToCheck** – название для проверки.

Параметр **s_Min** – минимальное значение для проверки/добавления.

Параметр **s_Max** – максимальное значение для проверки/добавления.

2.2 Модуль обработки сценариев.

2.2.1 Обработка этапов сценария.

Запуск этапа выполняется через процедуру **StartStage**(int **stageID**). Единственный параметр в ней – порядковый идентификатор этапа сценария. Данная процедура последовательно выполняет следующие действия:

1. Проверяет условия этапа через **CheckConditions**(**CurScenarioStage**.Conditions) в случае если проверка не пройдена выполняется переход к следующему этапу **NextStage**().
2. Заполнение списка реплик, подходящих для данного этапа. К ним так же применяется процедура **CheckConditions**(**curReplica**.CheckConditions) для отбора подходящих по условию реплик. В случае если ни одной реплики не найдено и это не особый этап, выполняется переход к следующему этапу.
3. Если данный этап имеет пометку особый, выполняется **SystemAction**(**CurStageType**, **CurScenarioStage**.PersonID).
4. Если этап обычный, то в зависимости от того кому принадлежит реплика воспроизводится анимация для персонажей через **StartNPC**().
5. Если реплика принадлежит игроку – создается набор из реплик через **MakeVariants**(). В случае если реплика всего одна – выводится обычное диалоговое окно с указанным типом реплики. Если реплик несколько (до 3) выводится окно вариантов на выбор.
6. Если реплика принадлежит неигровому персонажу – выводится соответствующая типу реплика через **MakeReplica**(**CurScenarioStage**.PersonID, **ReplicaToSend**, **cr**.ReplicaType).
7. Когда набор этапов исчерпан демонстрационный сценарий сбрасывается к началу.

2.2.2 Применение условий этапов, реплик и генерации характеристик.

Обработка условий – одна из важнейших операций при выполнении сценария. Она служит сразу для трех целей – 1) отбора этапов 2) отбора реплик 3) подбора характеристик для генерации характеристик персонажа. Условия бывают двух типов:

1. Проверка на ранее выбранные реплики. Условие считается выполненным если требуемый идентификатор реплики был выбран игроком или неигровым персонажем.
2. Проверка на текущие характеристики персонажа. Условие считается выполненным если идентификатор характеристики StatToCheck выбранного персонажа входит в диапазон Min и Max класса условий.

Оба типа отборов могут быть применены для составления абсолютно произвольных деревьев этапов/реплик с учетом контекста повествования.

Кроме того, класс условий применяется при генерации характеристик персонажа в системном этапе Resolve_Quiz. Условия для генерации определяются в классе реплик **ScenarioReplica**. SuccessOutcome в случае успешной проверки и реплик **ScenarioReplica**.FailOutcome если проверка реплики не пройдена. Генерация характеристик выполняется по ключу StatToCheck.

2.2.3 Обработка особых событий.

Обработка особых условий осуществляется в процедуре **SystemAction**(string **StageType**, int **ReplicaSender**). В качестве первого параметра она принимает тип этапа сценария, второй этап необходим для определения персонажа, к которому применяется событие. На момент написания документации существуют следующие типы этапов.

1. **Exit**. Данный этап убирает персонажа из стека диалогового окна.
2. **Player_Name**. Открывает панель для ввода имени персонажа.
3. **Player_Avatar**. Открывает панель для выбора аватара персонажа.
4. **Player_Gender**. Открывает панель для выбора пола персонажа.
5. **Quiz**. Открывает цепь этапов-опроса игрока.
6. **Resolve_Quiz**. Закрывает цепь этапов-опроса игрока и открывает панель генерации заработанных игроком характеристик.

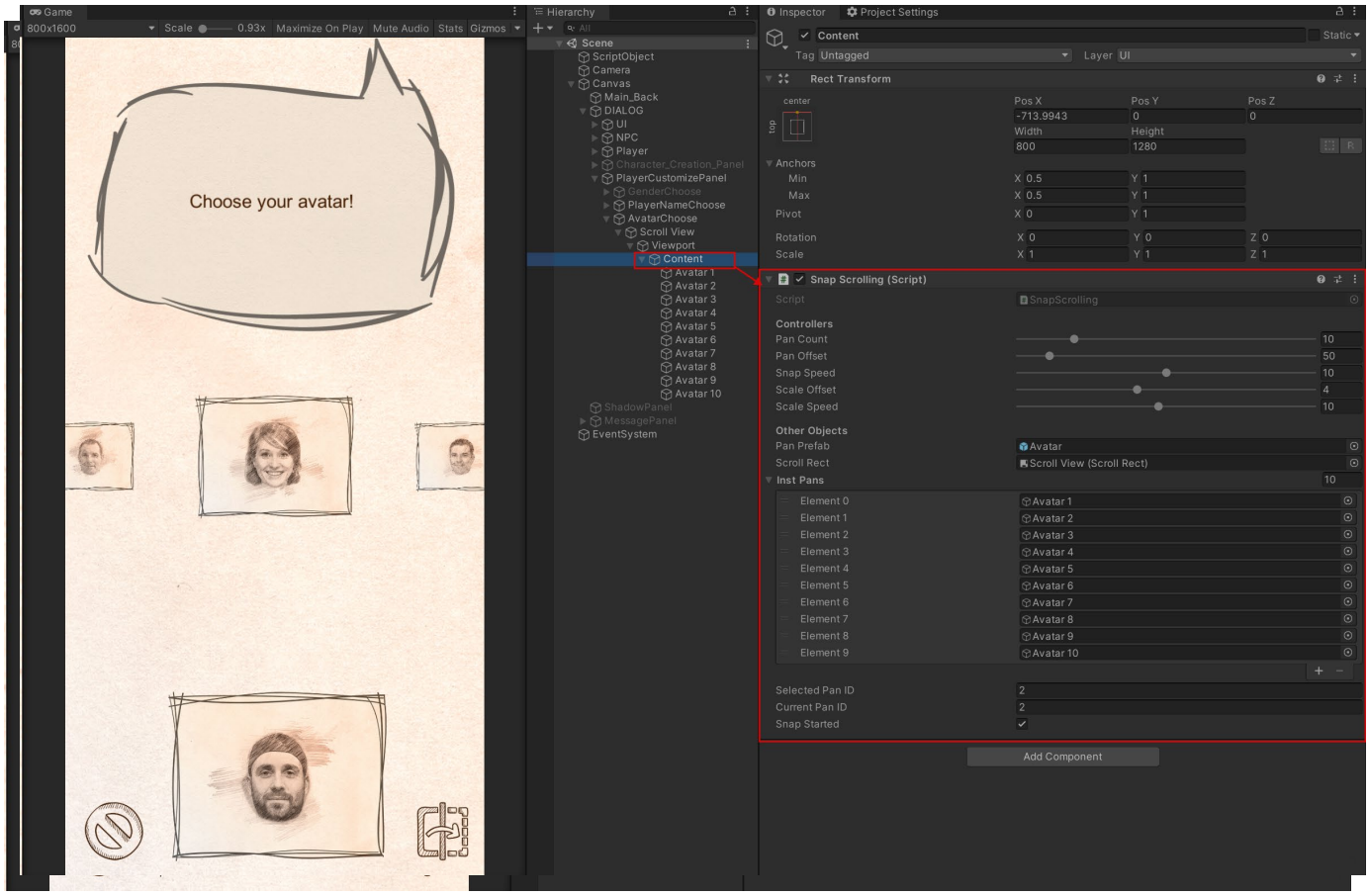
Finish. Завершает сценарий, вне зависимости от наличия последующих этапов сценария.

2.2.4 Начисление характеристик.

Начисление характеристик происходит в процедуре **EventsActionsResolver**() после подтверждения окна генерации характеристик, вызванного через **SystemAction**() с типом Resolve_Quiz. Данная процедура анализирует все выбранные реплики этапов (CurScenarioStagesList) и ассоциированные с ними массивы характеристик в поле классе **ScenarioReplica**.SuccessOutcome.

2.3 Дополнительные модули.

2.3.1 Модуль выбора аватара.



Выбор аватара происходит при обработке этапа с типом Player_Avatar в процедуре **SystemAction**(). Результат ввода обрабатывается в процедуре **SystemActionsResolver**() и помещается в переменную **DataHandler**.AvatarID, которую можно использовать в дальнейшем. Панель выбора аватара представляет собой специальный скроллинг-объект карусельного типа. Вы можете использовать его как самостоятельный модуль в своих проектах, он позволяет достаточно гибко и красиво организовывать просмотр слайдов. За его работу отвечает класс **SnapScrolling**. В качестве картинок вы можете использовать либо уже готовые слайды, либо создавать их программно. В случае ручного добавления необходимо задать размер массива из слайдов в переменной instPans и закрепить в нем элементы через инспектор Unity. Запуск слайдера происходит через **StartSnap**(int **curElem** = 0). Переменная curElem указывает на элемент массива, с которого начнётся показ слайдов.

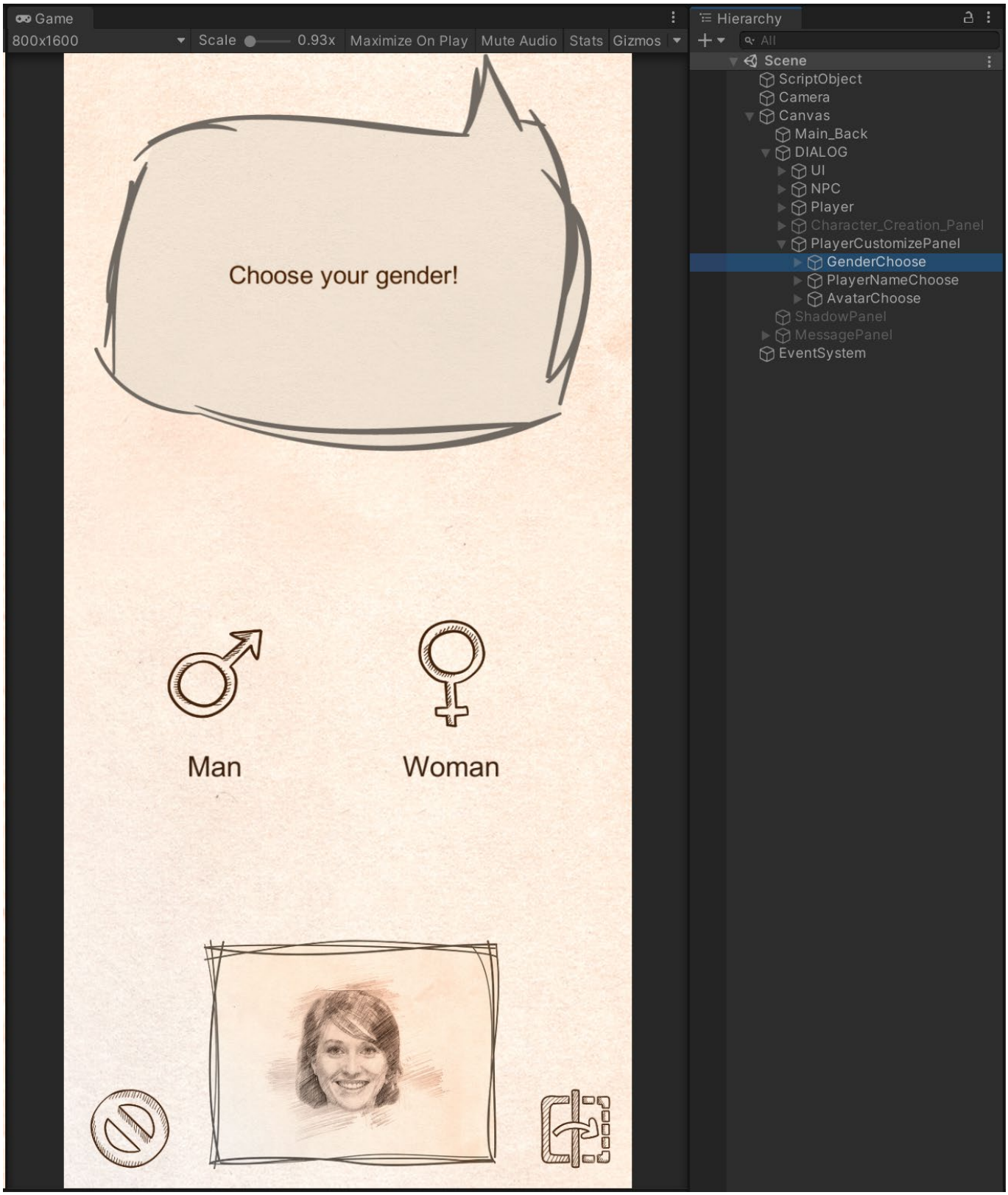
Вы можете регулировать некоторые аспекты слайдера с помощью переменных:

1. panOffset. Отвечает за расстояние между слайдами.
2. snapSpeed. Управляет скоростью прокрутки.
3. scaleOffset. Влияет на размер слайдов.
4. scaleSpeed. Регулирует плавность уменьшения и увеличения элементов слайда.

2.3.2 Выбор имени.

Выбор имени происходит при обработке этапа с типом Player_Name в процедуре **SystemAction**(). Результат ввода обрабатывается в процедуре **SystemActionsResolver**() и помещается в переменную **DataHandler**.UserName, которую можно использовать в дальнейшем. Для этого в тексте реплики можно использовать специальную метку [UserName].

2.3.3 Выбор пола.



Выбор имени происходит при обработке этапа с типом Player_Gender в процедуре **SystemAction**().
Результат ввода обрабатывается в процедуре **GenderChoose**(int **gender**) и помещается в переменную **DataHandler**.Gender, которую можно использовать в дальнейшем.

2.4 Программный ввод сценария.

2.4.1 Описание демонстрационного сценария.

Демонстрационный сценарий составлен таким образом, чтобы проиллюстрировать основные возможности шаблона. В нем реализовано дерево решений, реакция на выбор данного дерева впоследствии, опросник с генерацией характеристик персонажа, а также вспомогательные функции, такие как выбора аватара, имени и пола. Данный список системных функций может быть легко расширен и встроен в исходный функционал. Программный ввод данного сценария реализован в классе **DemoScenario**.

Ввод всех элементов сценария осуществляется с помощью статических функций в классе **DataHandler**. Они подробно описаны в разделе 2.1.1. Готовый сценарий помещается в переменную **DataHandler**.sData. В ней хранится список всех сценариев, полученных как программно, так и с сервера.

Далее мы подробно остановимся на основных элементах программного ввода.

2.4.2 Создание персонажей сценария.

Создание персонажа осуществляется с помощью функции **DataHandler**.AddNewScenarioCharacter(). Добавьте от 2 до 4 персонажей в сценарий. Причем 0 значение для параметра Place — это персонаж игрока. Остальные порядковые номер отвечают за порядок их вывода при стартовой инициализации.

Кроме того, вы можете назначить необходимые характеристики каждому их них с помощью **player**.CharValues.Add(new DataHandler.ScenarioCharacterValues(int **ValueID_s** = 0, int **amount_s** = 0))

В данной функции параметр **ValueID_s** — это идентификатор характеристики, определенный в базе данных, либо заведенный вручную в процедуре **DemoScenario**.AddDefaultValueData().

2.4.3 Ввод этапов сценария.

После определения персонажей приступаем к созданию этапов сценария. Выполняется это с помощью функции **DataHandler**.AddNewStage(**IntroScenario.Stages**, 1, 2), она возвращает ссылку на созданный этап сценария. Первый параметр данной функции — список этапов в сценарии, второй — порядковый номер этапа, третий — действующее лицо этапа.

При вводе этапов необходимо заранее представлять себе сюжет сценария. Прежде всего нужно понимать будут ли в нем системные этапы, требующие дополнительной обработки или же будет сплошной диалог с фиксацией последствий выбора. При составлении цепочки этапов держите в уме, что любой из этапов может быть пропущен при несоблюдении условий этапа, если они заданы. Например, если в предыдущем этапе вы выбрали первую реплику, а не вторую — следующий этап (выход персонажа со сцены) может быть пропущен. Это позволяет максимально гибко реализовывать не только реплики, но и все интерактивные действия.

2.4.4 Добавление реплик к этапам.

Реплики — основной строительный блок диалоговой системы. Задается с помощью функции **DataHandler**.AddReplica(**s3.ReplicaList**, 4, "Okay, then let's get started!", "Action");

Первый параметр — список реплик этапа, в который она будет добавлена. Данный список может содержать сколько угодно элементов. Однако выведены будут только три или менее из них. Соответственно остальные должны быть отфильтрованы при помощи условий - либо на реплики, либо на характеристики персонажа (Более подробно смотри раздел 2.2.2).

Второй параметр — идентификатор реплики.

Третий — текст реплики.

Четвертый — внешний вид реплики.

Обратите внимание, что в тексте реплик допускаются специальные вставки такие как "Good choice, [UserName]!". Они служат для пост обработки текста и замены их на ключевые слова. В данном примере — замена на ник персонажа.

При добавлении реплик помните, что именно они — двигатель всей истории. От их выбора будут зависеть условия для последующих этапов, реплик, начисления характеристик или ресурсов.

3. НАСТРОЙКА СЕРВЕРНЫХ МОДУЛЕЙ

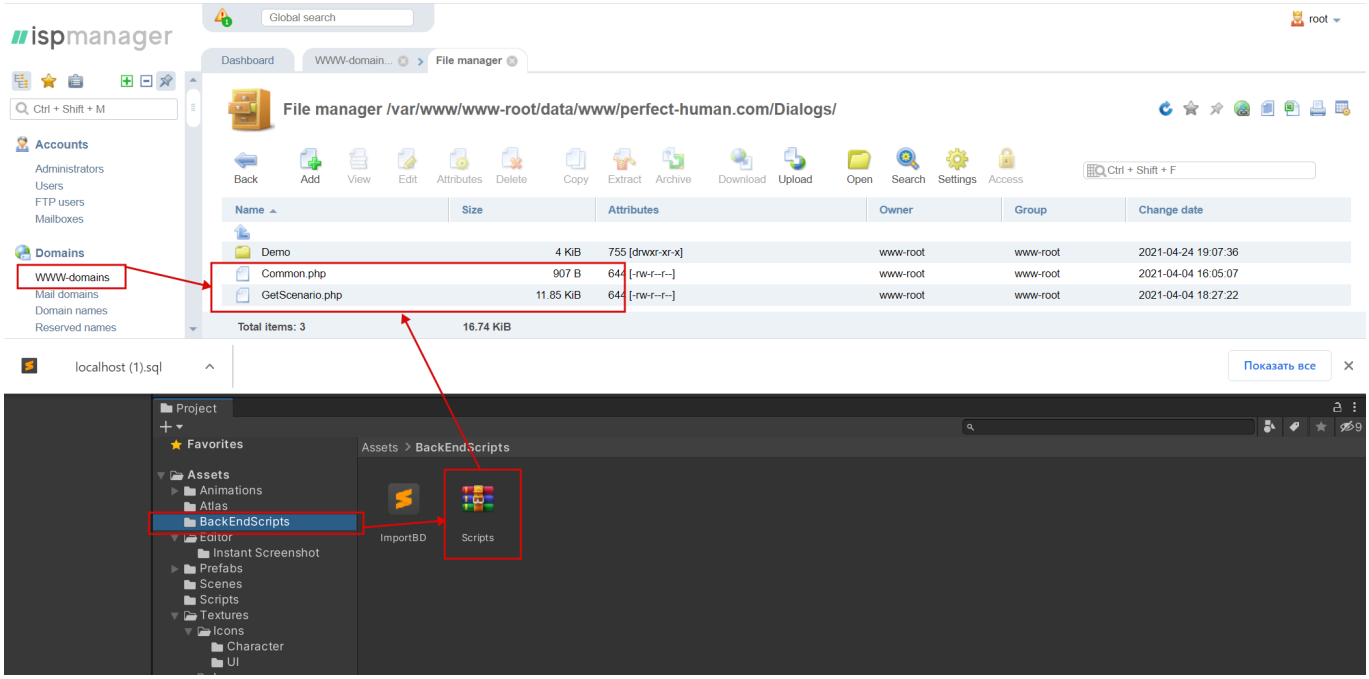
3.1 Общие данные по работе с хостингом.

3.1.1 Покупка хостинга и получение данных для работы.

Вы можете разместить бэк-энд сервер как локально на своей машине (с помощью Apache или Nginx) так и воспользовавшись услугами хостинга. Первый вариант потребует от вас обеспечения постоянного доступа к широкополосной сети и сетевого администрирования, он более предпочтителен, когда ваш проект будет достаточно развит или на стадии тестирования. Однако для начала гораздо более предпочтителен вариант с хостингом. Его мы и рассмотрим.

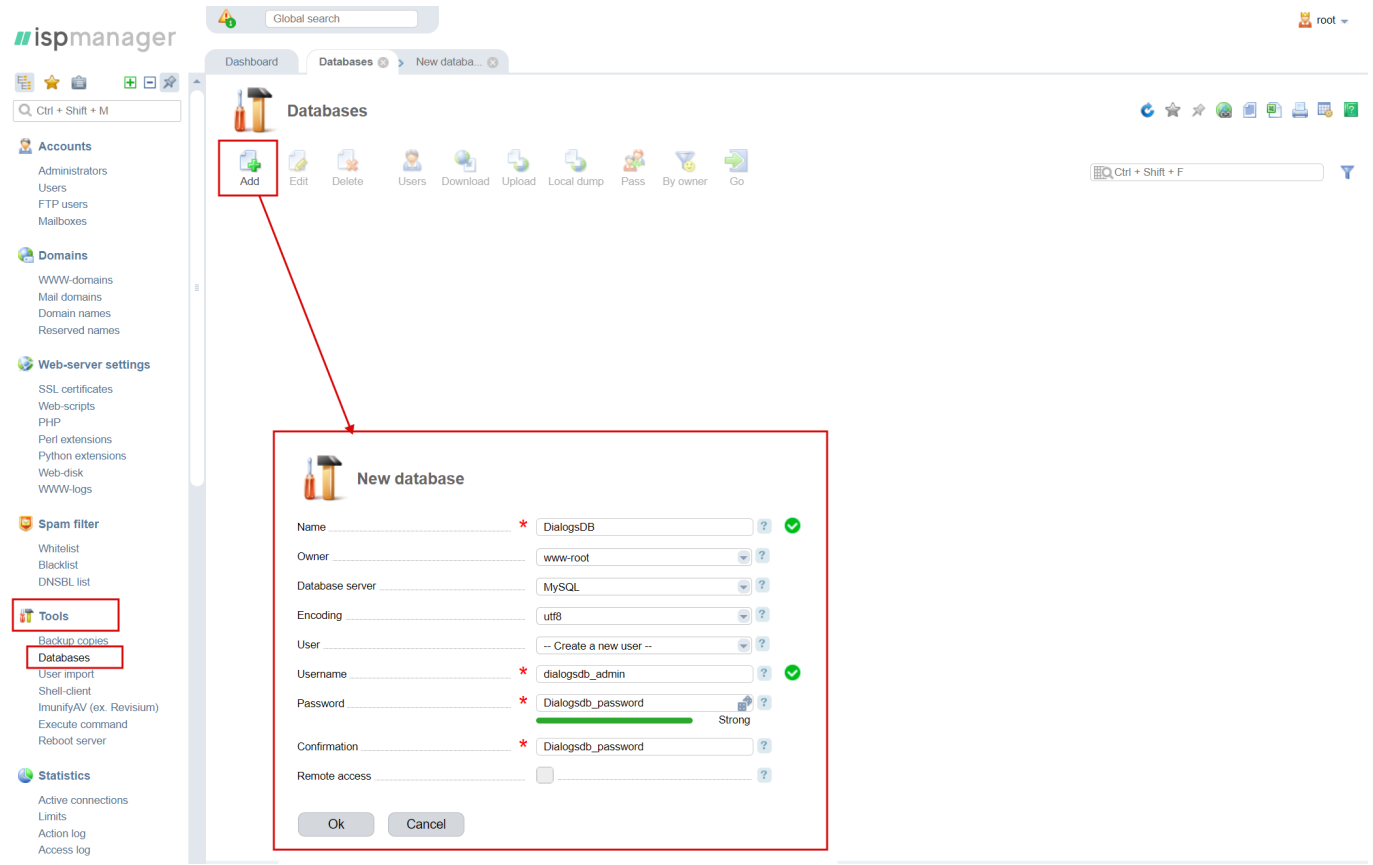
Для начала вам нужно приобрести место на хостинге. Данные услуги предоставляют множество компаний, выбирайте любую, которая вам больше подходит. Для этого достаточно в браузере набрать “услуги хостинга”. В данной документации все примеры разбираются в ISP panel для файлового администрирования и PhpMyAdmin для администрирования баз данных. Поэтому если вам важно пошагово повторить процесс развертывания файлов и базы данных убедитесь, что они у вас будут установлены. Если же у вас уже есть хостинг и навыки работы с ним – пропустите этот раздел.

3.1.2 Запись файлов скриптов на сервер через ISP panel.

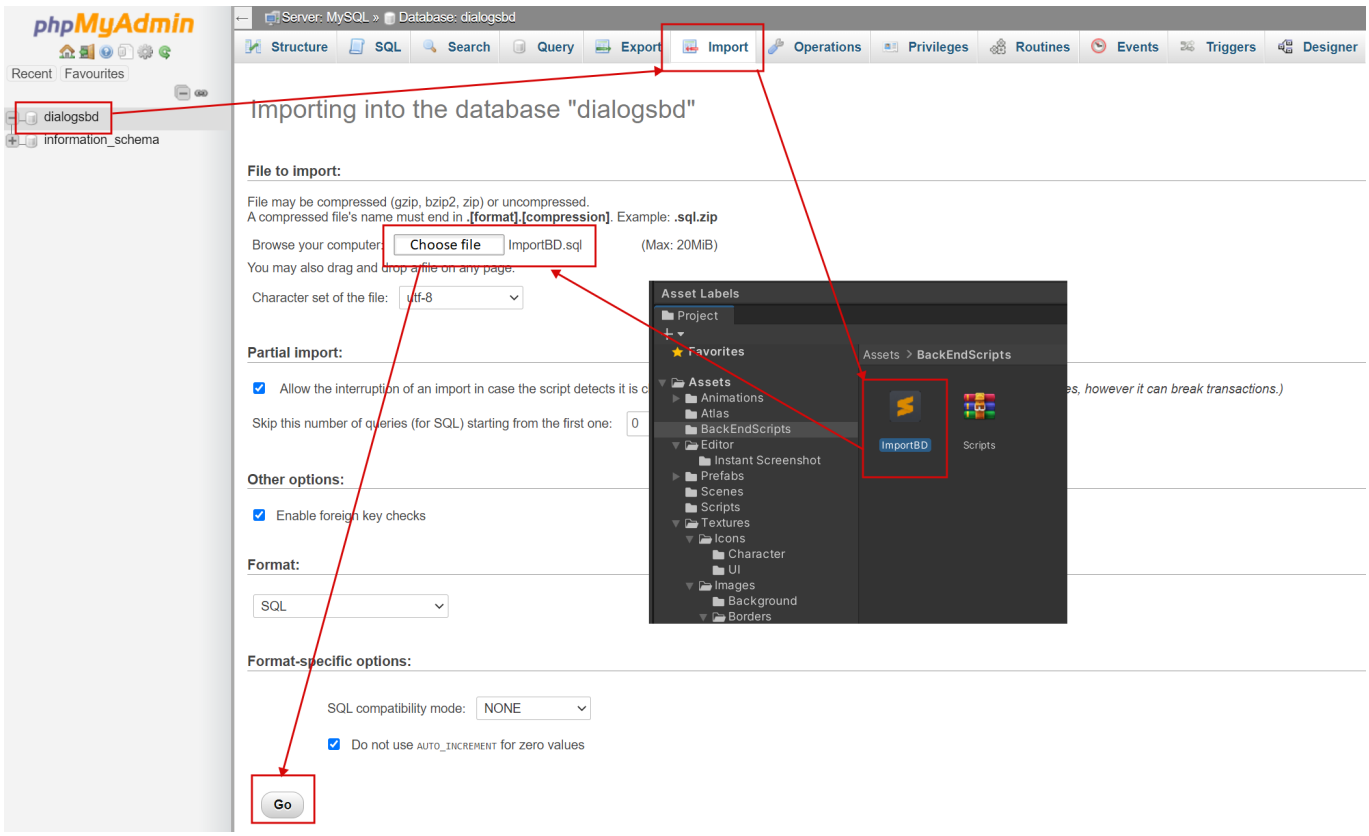


Архив со всеми скриптами для сервера хранится в каталоге BackEndScripts. Скачайте его и разместите в пространстве, которое вам подходит. Разархивируйте содержимое.

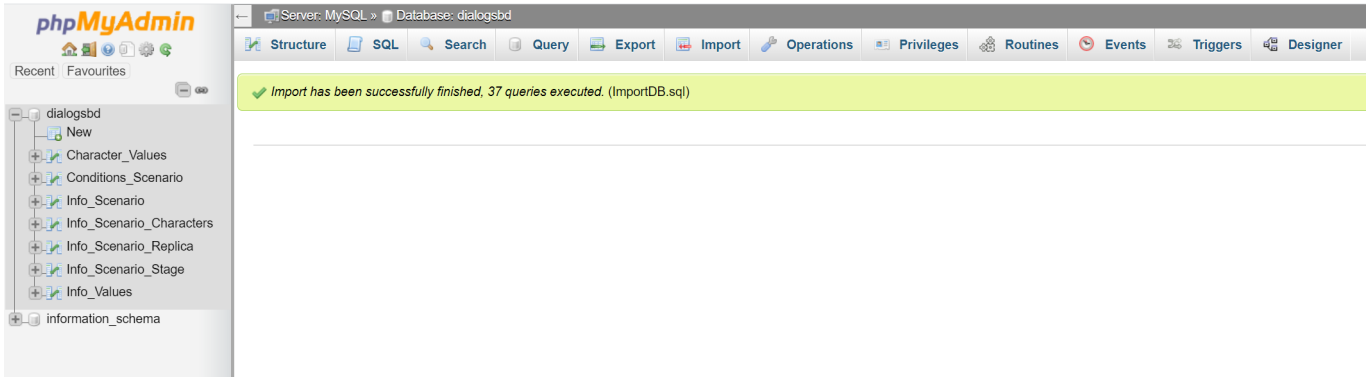
3.1.3 Импортирование базы данных в PhpMyAdmin.



Создайте новую базу данных, пользователя и пароль к нему. Обязательно запомните, а лучше запишите где-нибудь эти данные. Они нам потребуются для записи в скрипте соединения с БД.



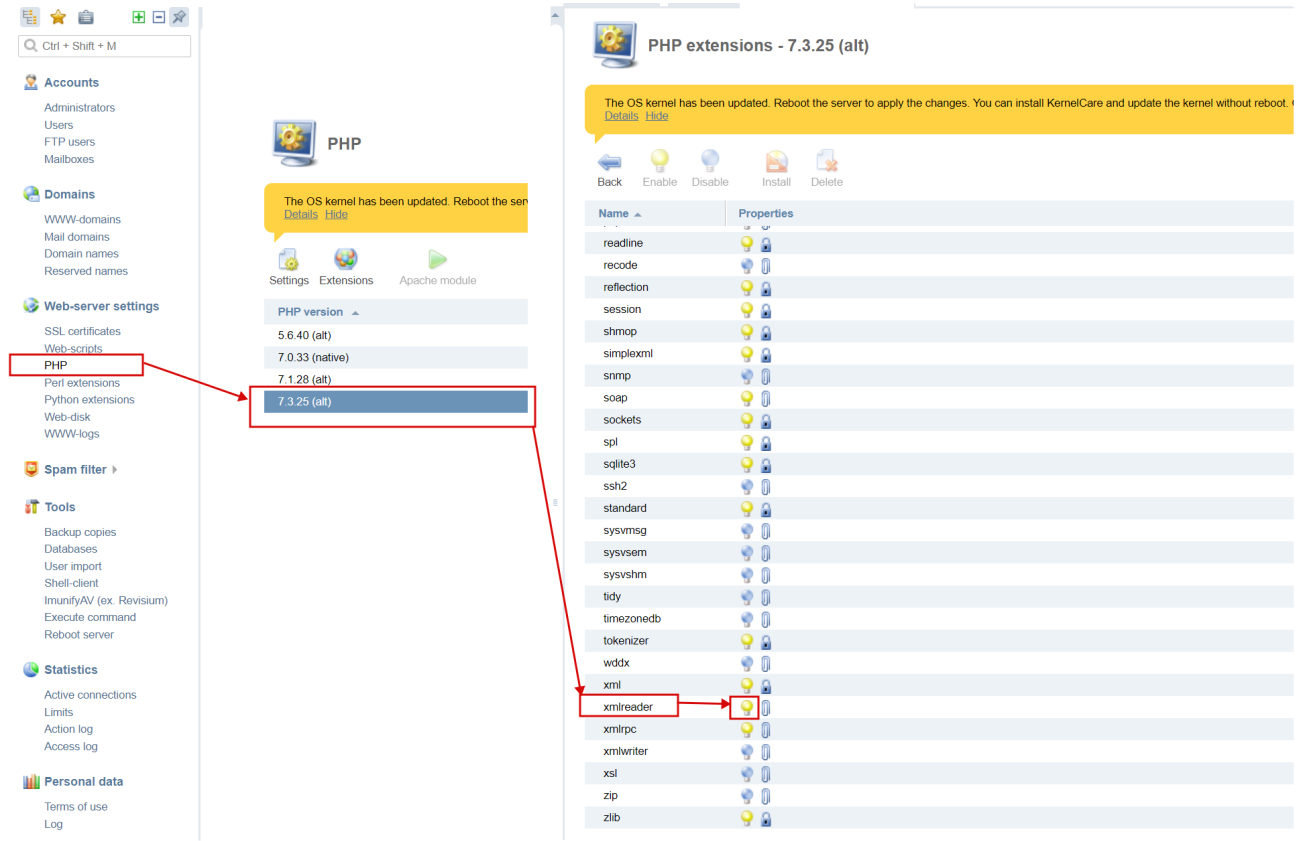
Импортируйте файл со структурой Базы Данных и первичными настройками. В ней ещё нет самих реплик и этапов, мы загрузим их позднее. Когда импорт завершится, вы увидите такой экран:



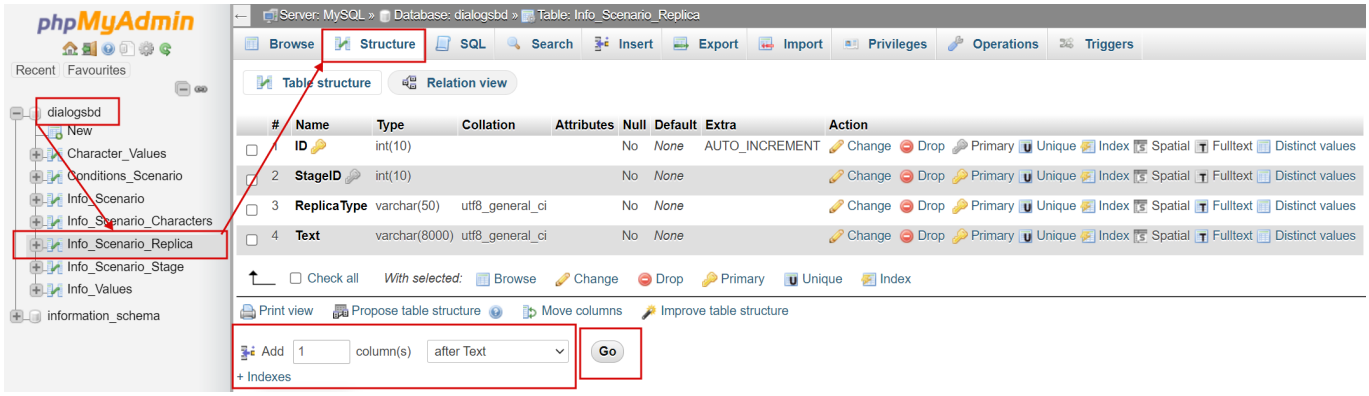
Поздравляем! База данных успешно загружена. Рекомендую хорошо изучить структуру базы, чтобы представлять все особенности хранения данных. Когда вы поймете как структура коррелирует со структурой внутри приложения можно начинать подгонку структуры под ваши нужды.

3.1.4 Версия PHP и необходимые библиотеки.

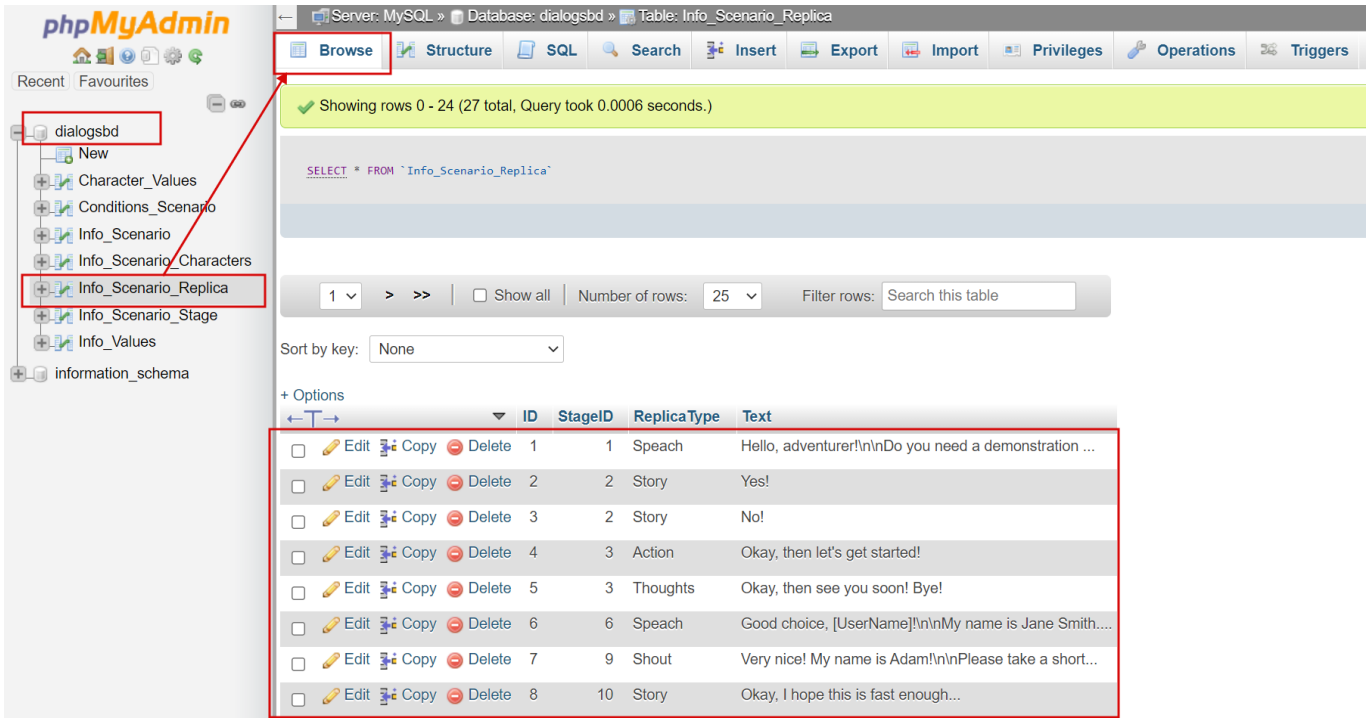
Важно! Обязательно обратите внимание на вашу версию PHP. В случае если у вас версия 7+ изменения не потребуются. Однако в случае если у вас более старая версия PHP используйте скрипты из папки (PHP 5+). Это важно, поскольку обращения к SQL серверу отличаются в этих версиях.



3.1.5 Структура базы данных MySQL.



Структура Базы Данных определяется совокупностью таблиц и колонок в этих таблицах. На примере выше показана структура таблицы с репликами. В ней 4 колонки. Сами реплики представлены в ней строками, где каждое значение колонки отражает какую-то характеристику реплики. Посмотреть их можно в меню Browse.



Обратите внимание, что колонки в точности повторяют структуру класса **DataHandler**.ScenarioReplica в Unity. Это не случайно, поскольку в общем виде однородность данных должна сохраняться и на сервере и на клиенте. Старайтесь избегать лишних полей (колонок) в своей структуре. Для добавления новой колонки вы можете воспользоваться меню внизу, определите в нем количество колонок и стартовое положение и нажмите Go. Для редактирования колонок используйте кнопку Change напротив каждой колонки.

Старайтесь без нужды не менять структуру, т.к. без четкого понимания есть вероятность рассогласования ваших скриптов и структуры базы данных, что приведет к ошибкам при загрузке приложения или импорта данных. Всегда осознанно изменяйте структуру. Если вы непоправимо

испортили базу попробуйте восстановить её, для этого используйте почаще используйте бэкапы таблиц. Для этого воспользуйтесь опцией Export.

Теперь давайте разберем для чего используется каждая таблица.

1. Info_Scenario. Данная таблица служит для хранения всех сценариев. В ней вы задаете название и идентификатор сценария. Повторяет структуру класса **DataHandler**.ScenarioData.
2. Info_Scenario_Stage. Таблица с этапами сценария. Содержит ссылку на ид сценария. Повторяет структуру класса **DataHandler**.ScenarioStage.
3. Info_Scenario_Replica. Таблица с репликами этапов сценария. Содержит ссылку на ид реплики. Повторяет структуру класса **DataHandler**.ScenarioReplica.
4. Conditions_Scenario. Таблица с полным описанием условий для этапов и реплик. Повторяет структуру класса **DataHandler**.CommonConditions.
5. Info_Scenario_Characters. Эта таблица служит хранилищем для всех персонажей, связанных через идентификатор сценария. Повторяет структуру класса **DataHandler**.ScenarioCharacter.
6. Character_Values. Таблица с набором характеристик персонажей. Содержит ссылку на ид персонажей. Повторяет структуру класса **DataHandler**.ScenarioCharacterValues.
7. Info_Values. Таблица-справочник с описанием всех характеристик и/или ресурсов. Повторяет структуру класса **DataHandler**.ValueData.

3.2 Модули взаимодействия с клиентом.

3.2.1 Общее описание модулей.

Рассмотрим подробнее PHP скрипты, которые мы разместили ранее на нашем сервере. Они обеспечивают взаимодействие клиента с сервером через класс **DataHandler**.

В принципе Unity позволяет напрямую работать с удаленным MySQL сервером. Для этого достаточно открыть порт на сервере и указать данные для подключения на стороне Unity. С точки зрения быстродействия такой вариант даже более оптимален, поскольку убирает накладные расходы на сериализацию и десериализацию JSON. Однако с точки зрения безопасности открытие свободного доступа к базе данных крайне опасно. Опыт подсказывает, что единственно верным решением остается доступ через localhost самого сервера. Именно поэтому нам требуется специальный транспорт для передачи Post и Get запросов нашему SQL серверу. В общем виде структура этих php скриптов похожа:

1. Принимаются параметры $_POST['YOUR_VARIABLE'] в качестве входных данных.
2. Производится сверка ключа безопасности с помощью MD5 шифрования. Данный ключ указывается в скрипте Common.php (переменная $secretKey).
3. Открывается соединение с базой данных dbConnect(). Параметры подключения и сама функция описана в скрипте Common.php.

4. Производится чтение или модификация базы данных.
5. Обрабатывается ответ MySQL сервера в виде массива.
6. Кодируется ответ в формат JSON через директиву json_encode()
7. Печать результата через метод echo. В случае успеха, как правило, добавляется ключевое слово SUCCESS, которое служит критерием для начала разбора результата на клиенте.
8. Закрытие соединения с базой данных через mysql_close($link);

Важно! Перед выполнением любых запросов убедитесь, что вы правильно заполнили все переменные в данном скрипте.

1. Переменная $secretKey должна в точности соответствовать переменной **DataHandler**. SKey в редакторе Unity.
2. Переменная $dbName должна соответствовать названию базы данных, которую вы создали в разделе 3.1.3.
3. В директиве $link = mysql_connect('localhost', $dbName, 'YOUR_PASSWORD'); замените пароль, заданный в разделе 3.1.3
4. Убедитесь, что путь в классе **DataHandler** соответствуют полному пути до вашего скрипта. По умолчанию, там стоит путь до сервера Perfect-Human с демонстрационными данными.

3.2.2 Скрипт GetScenario - получения данных о сценарии.

Скрипт GetScenario.php является самым важным, поскольку отвечает за получение данных о сценариях. Мы будем двигаться по функциональным блокам, начиная сверху вниз.

include ("Common.php"); Данная строка выполняет подключение общего модуля, в котором хранится строка подключения к базе данных и секретный ключ.

Параметры задаются в WWWForm на стороне клиента. На стороне сервера они принимаются так:

$hash = safe($_POST['hash']); Это специальный хэш для шифрования секретного ключа. Как правило он применяется в связке с ID пользователя. Вы можете добавить свой токен в это поле.

Следующая конструкция сверяет хэш на стороне сервера и на стороне клиента. В случае если они не совпадают (кто-то постучался на сервер извне) срабатывает возврат из скрипта с сообщением "Access Denied!".

```
$real_hash = md5($secretKey);

if ($real_hash != $hash)

{
   die("Access Denied!");
}
```

Далее следует запрос для получения справочника характеристик.

```
 //--------------------------------------------------------------------------------------------------------------------
  //echo("VALUES INFO");
  //--------------------------------------------------------------------------------------------------------------------

$query = "SELECT * FROM `Info_Values` AS Info_Values";
$Info_Values_Result = mysqli_query($link, $Info_Values_query) or die(mysqli_error());

$Val_response = array();
$Val_data = array();

        while($row_val = mysqli_fetch_assoc($Info_Values_Result))
        {
                $Val_ValueID = $row_val['ValueID'];
                $Val_Name = $row_val['Name'];
                $Val_ImageID = $row_val['ImageID'];

                $Val_data[] = array('ValueID'=> $Val_ValueID, 'Name'=> $Val_Name, 'ImageID'=> $Val_ImageID);
        }
$Val_response['values'] = $Val_data;
$Val_json = json_encode($Val_response);

echo $Val_json;
echo "|";
```

На что здесь следует обратить внимание:

1. SELECT со знаком * означает, что нас интересуют все поля из таблицы Info_Values.
2. Ключевое слово FROM указывает на основной источник данных для запроса, в нашем случае это таблица `Info_Values` AS Info_Values. Обратите внимание на псевдоним этой таблицы Info_Values, это название может быть произвольным, но достаточно осмысленным чтобы запрос проще читался.
3. После запроса идет сборка массива с ответом для сборки ответа в формат JSON с помощью json_encode($Val_response).

Далее следует 2 запроса для получения персонажей и их характеристик.

```
//--------------------------------------------------------------------------------------------------------------------
//echo("SCENARIO CHARACTERS");
//--------------------------------------------------------------------------------------------------------------------

$CharacterScenQuery = "

SELECT
Characters.ID ID
,
Characters.ScenarioID ScenarioID
,
Characters.Place Place
,
Characters.Name Name
,
Characters.ImageID ImageID

FROM
`Info_Scenario_Characters` AS Characters";

$CharactersScenResult = mysqli_query($link, $CharacterScenQuery) or die(mysqli_error());

//--------------------------------------------------------------------------------------------------------------------
//echo("FILLING CHARACTERS");
//--------------------------------------------------------------------------------------------------------------------

while($rowch = mysqli_fetch_assoc($CharactersScenResult))
{
        $charid = $rowch['ID'];

        $character_stats = array();

        while($rowstat = mysqli_fetch_assoc($CharactersStatsResult))
        {
```

```
                if($charid == $rowstat['CharacterID'])
                {
                        $character_stats[] = array('ValueID'=> $rowstat['ValueID'], 'Amount'=> $rowstat['Amount']);
                }
        }

        $character_data[] = array('ID'=> $charid, 'ScenarioID' => $rowch['ScenarioID'], 'Name'=> $rowch['Name'], 'Place'=> $rowch['Place'], 'ImageID'=>
        $rowch['ImageID'], 'CharValues' => $character_stats);
}

$character_response['characters'] = $character_data;
$character_json = json_encode($character_response);

echo $character_json;
echo "|";
```

Обратите внимание, что при сборке последнего элемента используется вложенный цикл по массиву характеристик персонажей.

```
//--------------------------------------------------------------------------------------------------------------------
//echo("SCENARIOS");
//--------------------------------------------------------------------------------------------------------------------

$ScenarioQuery = "
SELECT
Info_Scenario.ID ScenarioID
,
Info_Scenario.Name Name
,
Conditions_Scenario.ID Conditions_ID
,
Conditions_Scenario.TypeOfCondition TypeOfCondition
,
Conditions_Scenario.StatToCheck StatToCheck
,
Conditions_Scenario.IDCheck IDCheck
,
Conditions_Scenario.Min Min
,
Conditions_Scenario.Max Max
        FROM
`Info_Scenario` AS Info_Scenario
LEFT JOIN
Conditions_Scenario AS Conditions_Scenario ON Info_Scenario.ID = Conditions_Scenario.LinkID AND Conditions_Scenario.TypeOfLink = 1";

$scenarioResult = mysqli_query($link, $ScenarioQuery) or die(mysqli_error());
```

На что здесь следует обратить внимание:

1. SELECT и поля за ним говорят о том, какие поля таблиц нас интересуют. Обратите внимание, что данные берутся из разных таблиц. Например, **Conditions_Scenario**.ID. Здесь идентификатор условия берется из таблицы Conditions_Scenario. Правила объединений двух и более таблиц мы рассмотрим в 3 пункте.

2. Конструкции LEFT и INNER JOIN необходимы для присоединения данных к основной выборке. Например, в нашей таблице характеристик персонажей есть ID характеристик. Но нет информации о самих характеристиках (названиях и свойствах). Чтобы сделать их доступными в выборке SELECT необходимо соединить две таблицы в которых общим является идентификатор характеристики.

3. Таким образом, конструкция: LEFT JOIN Conditions_Scenario ON Info_Scenario.ID = Conditions_Scenario.LinkID AND Conditions_Scenario.TypeOfLink = 1 позволяет нам соединить ID сценария из таблицы Info_Scenario и ID условий из таблицы Conditions_Scenario.

4. Отличия LEFT от INNER грубо говоря состоят в следующем: LEFT или левое соединение означает, что вы к основной таблице (Info_Scenario) **по возможности** подбираете данные из таблицы, которую вы хотите присоединить (Conditions_Scenario). А INNER или внутреннее соединение означает, что записи основной таблицы обязательно должны совпадать с таблицей Conditions_Scenario по правилу их соединения. В случае если они вообще не совпадают, вам вернется пустой ответ на запрос. Таким образом, при определенных условиях INNER JOIN можно использовать в качестве фильтра.

Кроме того, здесь мы видим еще условие TypeOfLink = 1. Это условие отбирает только условия, подходящие по типу. Это ОЧЕНЬ важный момент. Идентификаторы могут пересекаться в пределах Сценариев, Этапов и Реплик:

TypeOfLink = 1 - соединение по идентификаторам Сценариев.

TypeOfLink = 2 - соединение по идентификаторам Этапов.

TypeOfLink = 3 - соединение по идентификаторам Реплик.

TypeOfLink = 4 - соединение по идентификаторам успешных исходов Реплик.

TypeOfLink = 5 - соединение по идентификаторам неудачных исходов Реплик.

Следующие два запроса — к Этапам и Репликам выполняются по той же схеме. Обратите внимание, что возвращаем мы сразу несколько массивов с данными разделенный вертикальной чертов. Это символ-сепаратор. На клиенте мы обработаем этот результат в процедуре **DataHandler**.Get_Scenario_Web_Process():

```
// Разделение двух массивов данных.
string[] mSplit = result.Split("|"[0]);

// Конвертация JSON из ответа сервера сразу в класс DataManager.ValuesData.
vData = JsonUtility.FromJson<ValuesData>(mSplit[1]);
```

В самом конце скрипта необходимо закрыть соединение с MySQL.

```
mysqli_close($link);
```

4. ОБРАТНАЯ СВЯЗЬ И ПОДДЕРЖКА.

Спасибо за покупку и использование шаблона Sketch Dialogs.

Надеюсь, вам понравится ваша покупка и её имплементация не вызовет больших трудностей. Если у вас есть какие-либо вопросы, пожалуйста, напишите мне по почте PerfectHumanApps@gmail.com или в раздел поддержки Sketch Dialogs на форуме Unity. Это поможет не только вам, но и мне самому.

Кроме того, я был бы признателен, если бы вы нашли время оставить отзыв на странице Asset Store. Ещё раз спасибо вам!