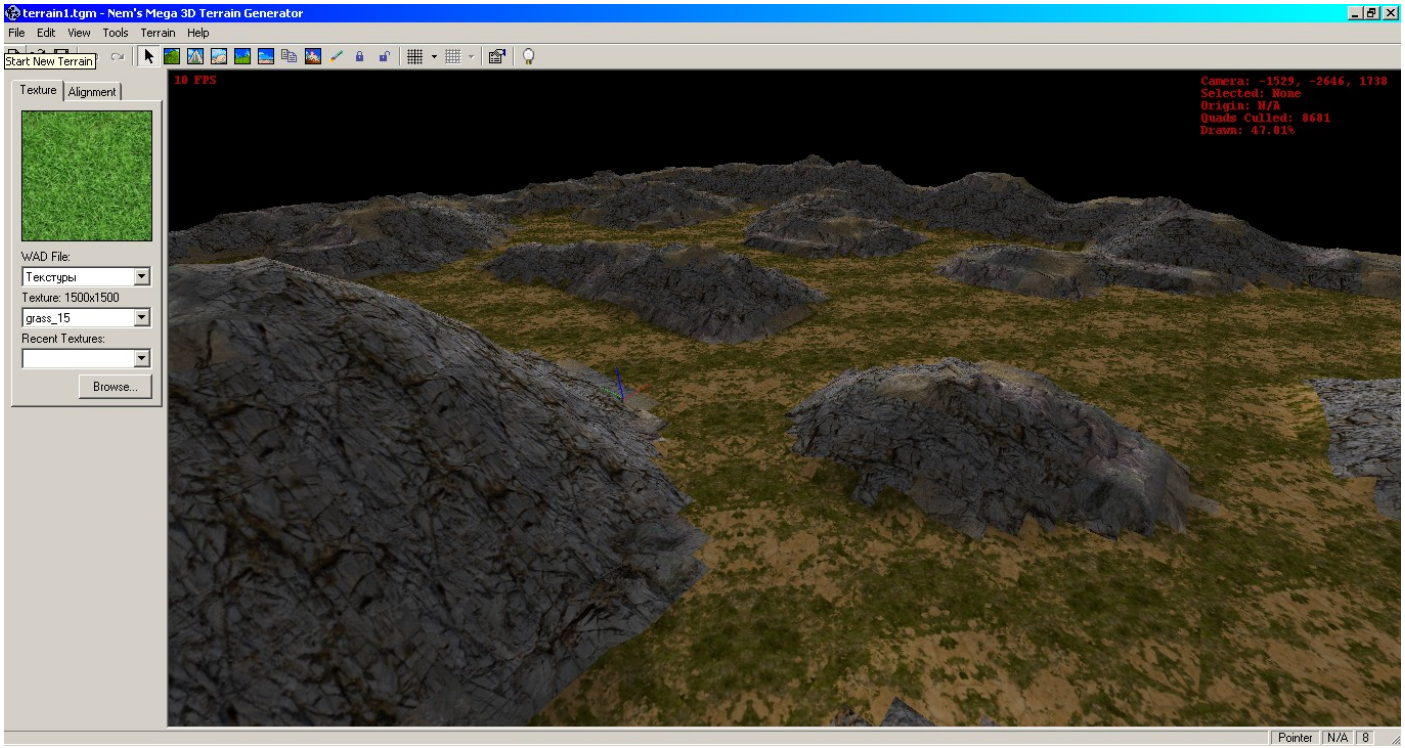
Более сложный проект автомобильного симулятора

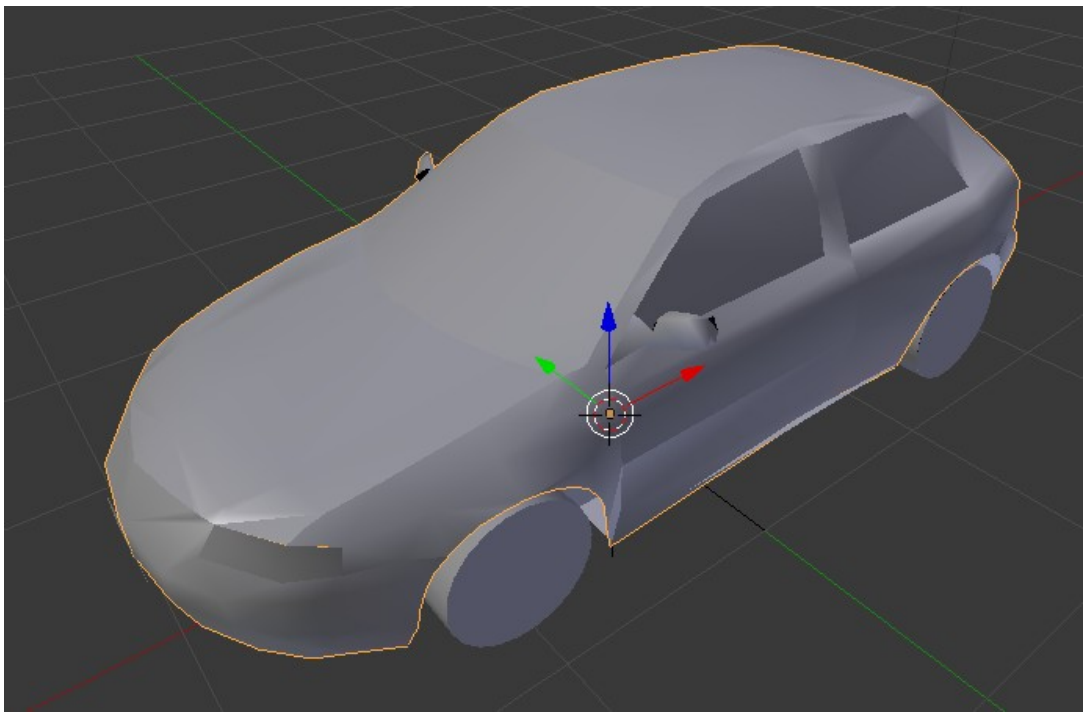
Работа над сложным проектом требует терпения и времени. В компаниях по разработке игр количество задействованных людей может доходить до ста человек. А времени может потребоваться от года до трёх. Естественно, что и ресурсы такой компании не сопоставимы с ресурсами индивидуального разработчика. Ведь «инди» нужно стать и художником, и моделистом, и программистом, и режиссером-сценаристом. При этом необходимо овладеть хотя бы в общих чертах несколькими программами. На первый взгляд, «инди..» - это уникальный человек, владеющий богатым набором компьютерных (и не только) знаний. Но главный враг такого уникума – время. Поэтому не стоит ставить перед собой грандиозных замыслов и задач. Достаточно сосредоточиться на небольших игрушках с наименьшим количеством уровней. Не пытайтесь приблизиться к графике таких игр, как Quake. Это совершенно не нужно. Для маленькой игры важнее интересный сюжет. В игре должен присутствовать смысл и логика. Игра должна затягивать, заставлять возвращаться на уровень и проходить его снова, но иначе или быстрее. Поэтому запасёмся терпением и начнём работать над более сложным проектом автомобильного симулятора. Кому-то на это потребуется несколько дней, а кому-то и пару недель. Но мы ведь не привыкли отступать ! ☺

Начнём с разговора о ландшафте. Какой вы видите свою игру? Будет ли это гоночный трек, или городской массив? А может быть это будет горная дорога, с длинными тоннелями? В любом случае там будет присутствовать земная твердь. Естественно, можно (и желательно научиться) сделать ландшафт средствами Blender. Но можно и сильно упростить себе задачу, используя сторонние программы для этих целей. А время стоит экономить ☺. Одна из простеньких и бесплатных программ – Terrain Generator. Растительности там не создашь, но «землю» можно сделать очень быстро. В разделе «Полезное» я разместил архив с программой и пару уроков, найденных мной на просторах интернета. На Win 7 она ставится без проблем, а возможность экспорта в формат ..OBJ – это всё, что нам необходимо. Правда, текстуры придётся класть вручную рядом с готовым файлом OBJ. Программа на английском, но прочитав уроки, вы поймёте, насколько она проста в работе. Радует её быстрота и конечно бесплатность. Хотя, я не нашел возможности плавного наложения текстур друг на друга… Из-за чего переходы между ними могут быть довольно резкие. Вывод – более тщательно подбирать картинки текстур по рисунку и цвету. Но это всё не сильно напрягает и для наших целей вполне достаточно. Ниже показано, как она выглядит:

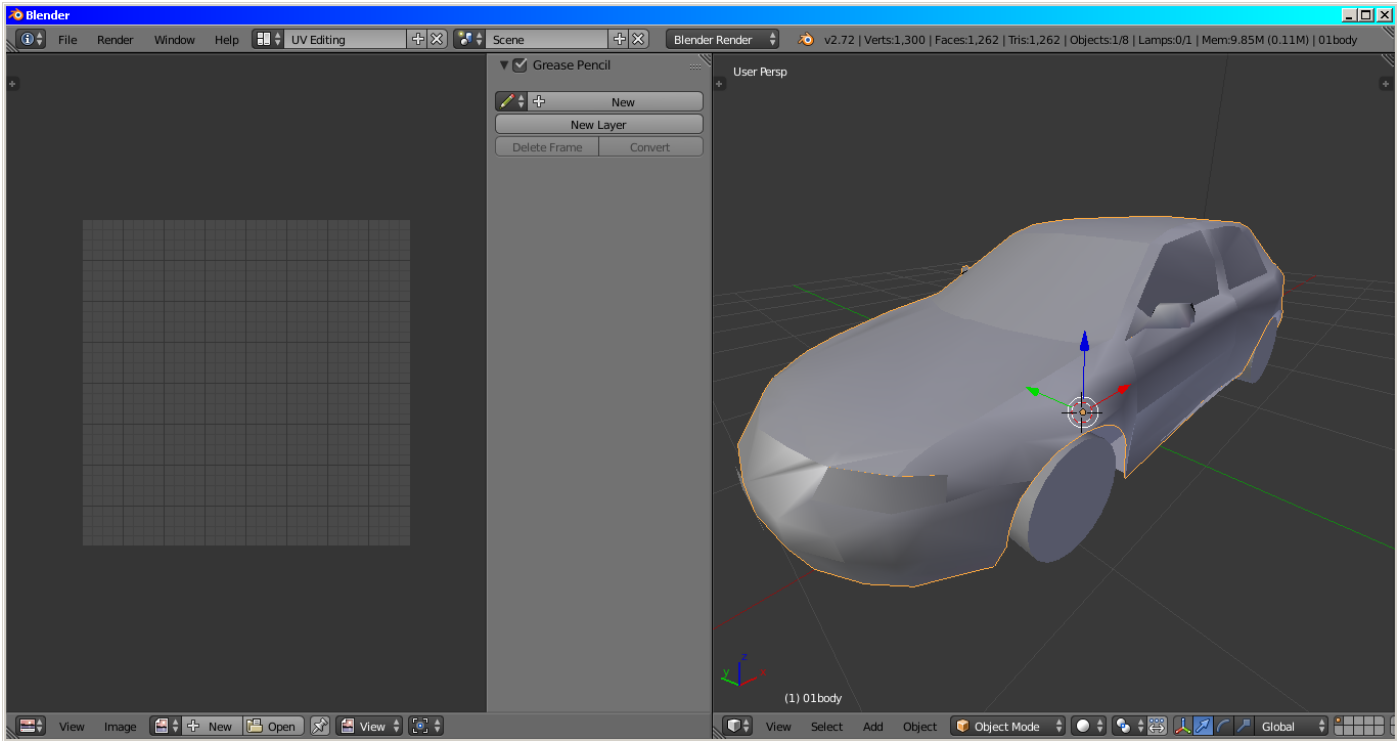


И ещё… Даже маленький для этой программы ландшафт (32x32) становится гигантским при импорте в Blender. Так что нам всё равно придётся его масштабировать. Но, чем больше полигонов, тем меньше вероятность, что наша машина при разгоне пройдёт сквозь стену, и качественнее пейзаж ☺. Надеюсь, что с ландшафтом вы разберётесь сами.

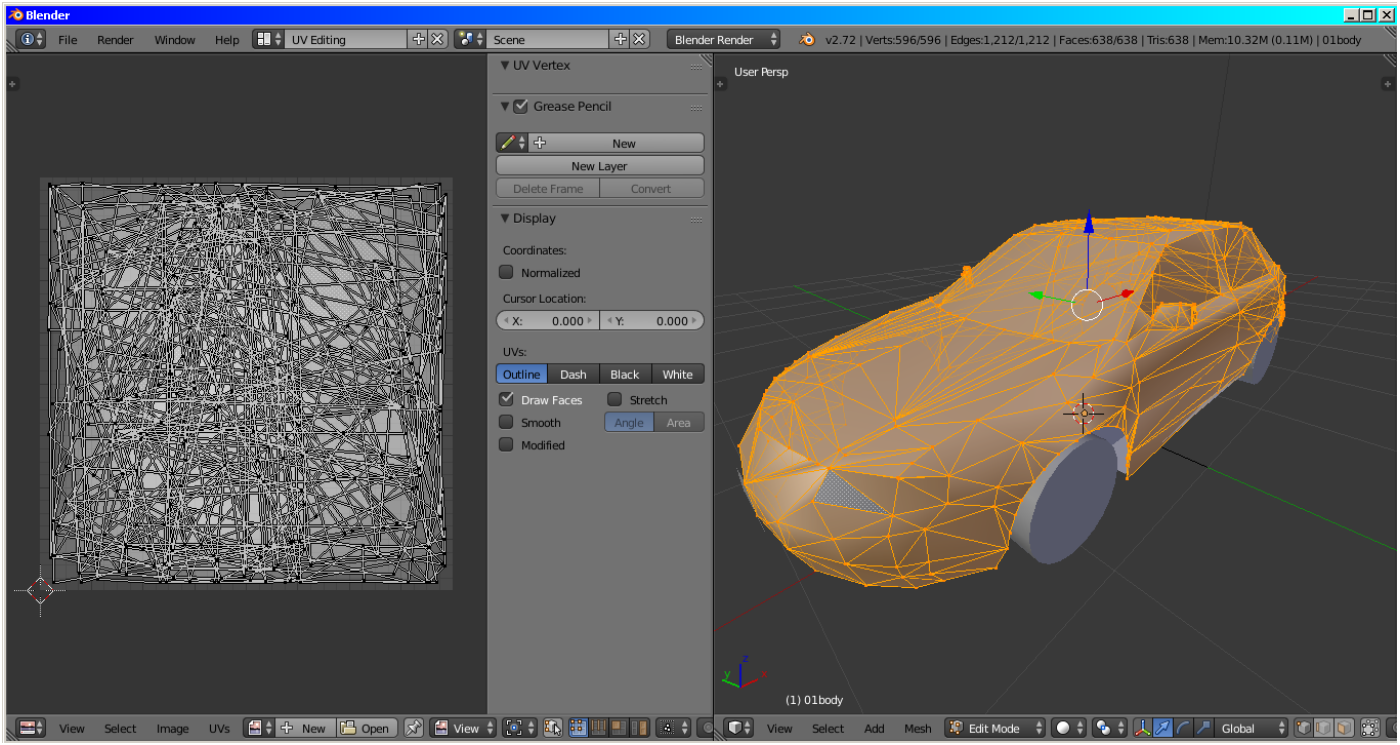
Теперь я хочу показать ещё один способ текстурирования модели 3ds. Импортируем модель:



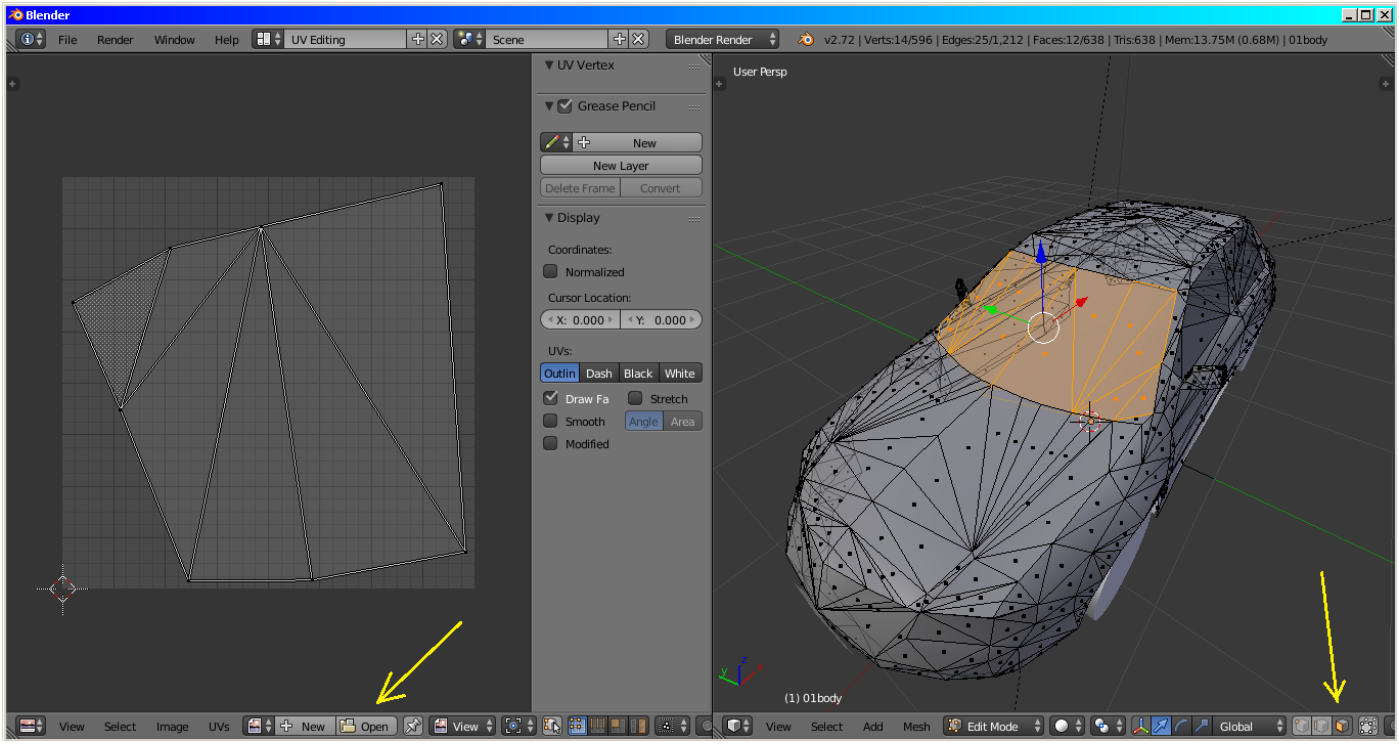
И переходим в UV Editing:



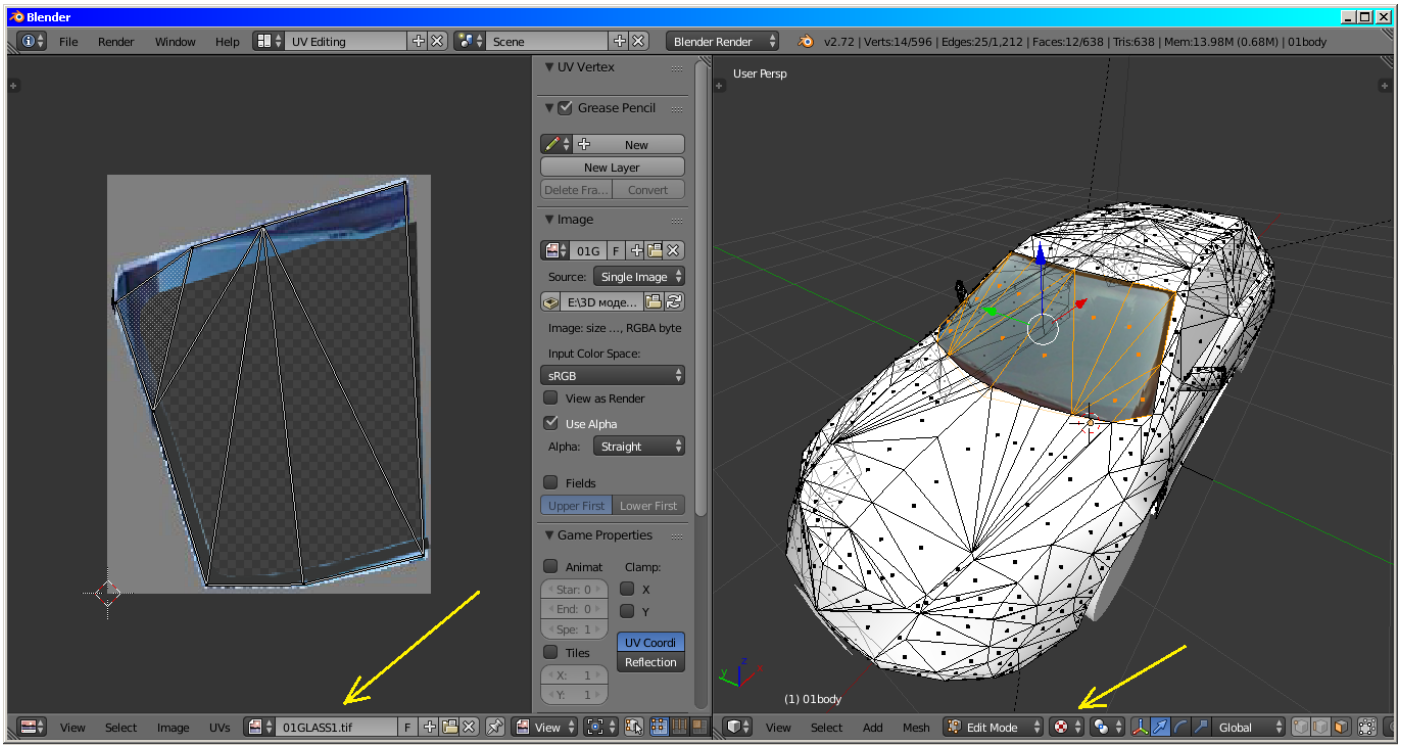
Где, не загружая ни одной картинки, справа переходим в режим редактирования и снимаем выделение со всего (лат. «A»):



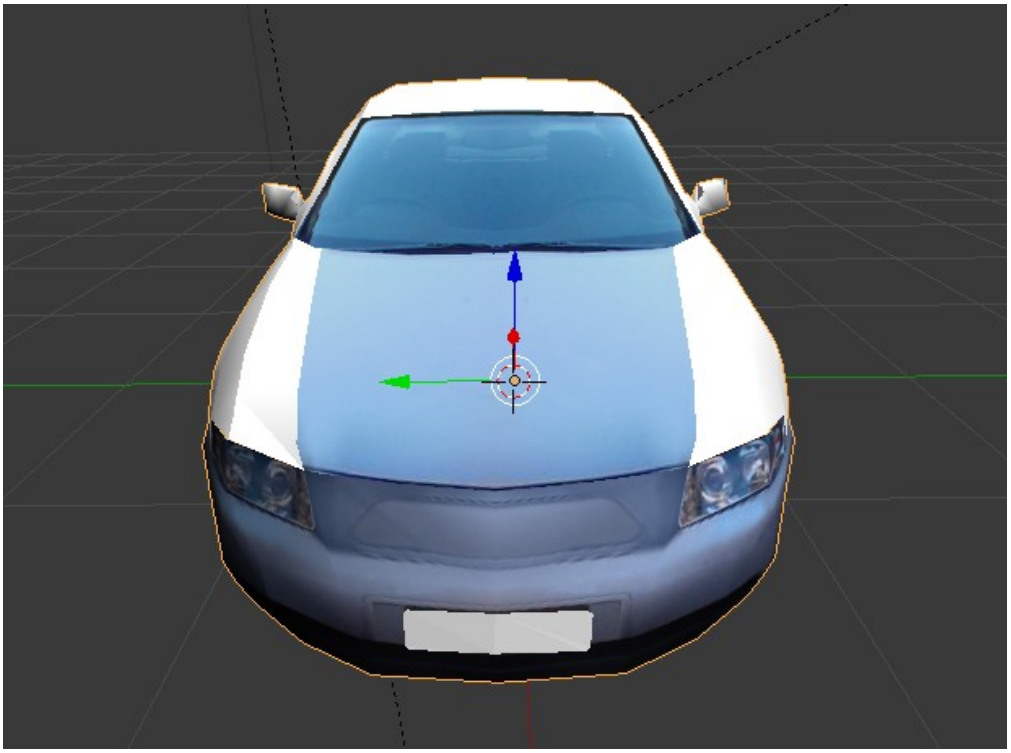
Теперь переходим в режим выделения полигонов и выделяем все полигоны лобового стекла. Слева у нас аккуратно лягут на плоскость все полигоны. Достаточно наложить любую текстуру, как она тут же замечательно покроет лобовое стекло. А если у вас есть готовая текстура лобовика, тогда вообще прекрасно – мы просто открываем её картинку и всё:



Включаем отображение текстур и любуемся:

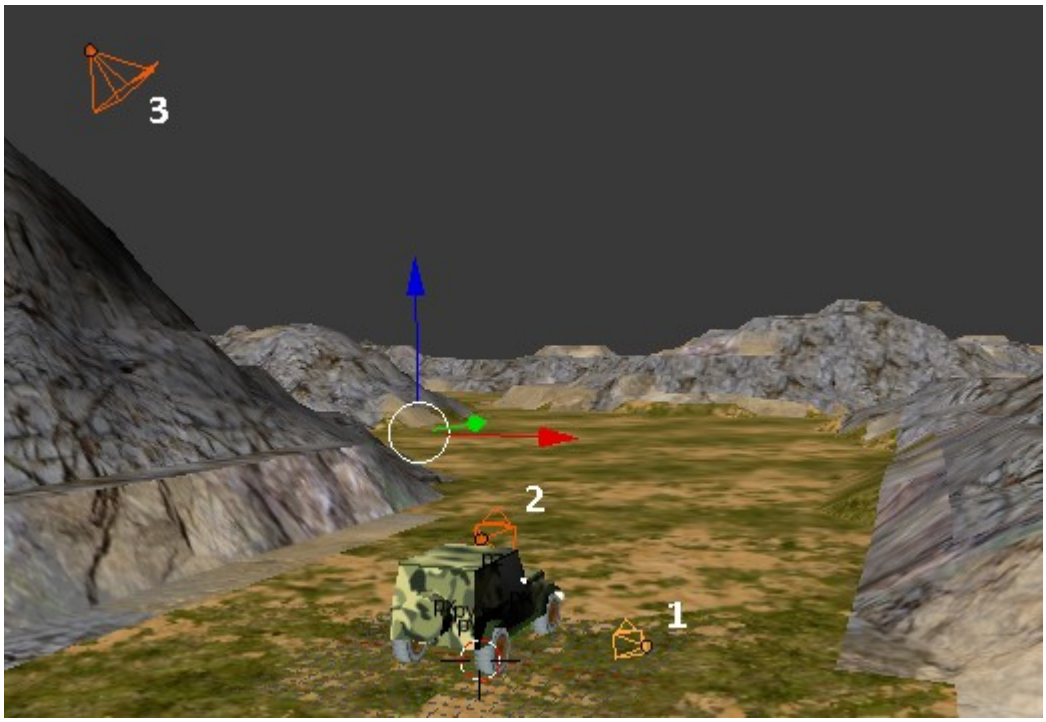


Если мы хотим покрыть всю машину одноцветной текстурой, то просто выделяем все нужные нам полигоны. Если же у нас предусмотрена иная раскраска, то выделяем лишь то, что нам необходимо. Это нужно делать отдельно с каждой деталью и всякий раз сбрасывать выделение перед следующей деталью. Не забудем наложить на каждую деталь свой материал и настроить его (желательно сделать прозрачным). Иначе мы ничего не увидим при включении игры. Например, я покрыл текстурой лобовое, фары, капот и бампер. Вот что вышло:



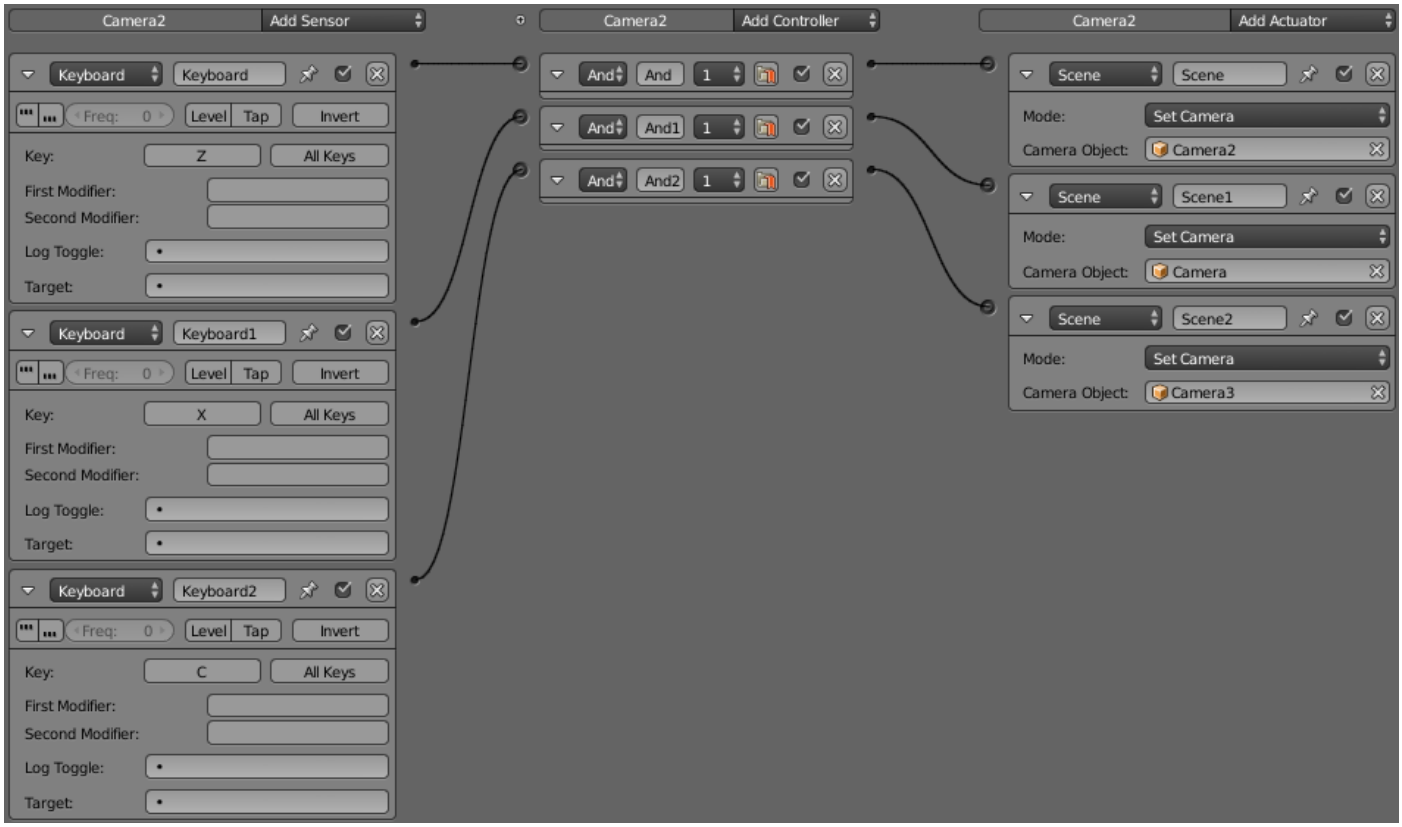
Ну, вроде всё по подготовке ☺. Далее создаём новый проект и рядом с ним папку, в которую бросим 3ds машины и Obj ландшафта, со всеми сопутствующими им текстурами. А так же, все звуки в ту же папку. Если мы предварительно сохраняли скрипты отдельными файлами ..PY , то их тоже бросаем туда. У меня вышло вот что:

Загружаем скрипты настройки автомобиля и управления. В дополнение к основной камере добавляем ещё две. Камеру № 2 устанавливаем над капотом и связываем её жестко с кузовом (потомок->родитель). Камеру № 3 устанавливаем гораздо выше первой и немного левее. Настраиваем как и первую. Эта камера будет обзорной:

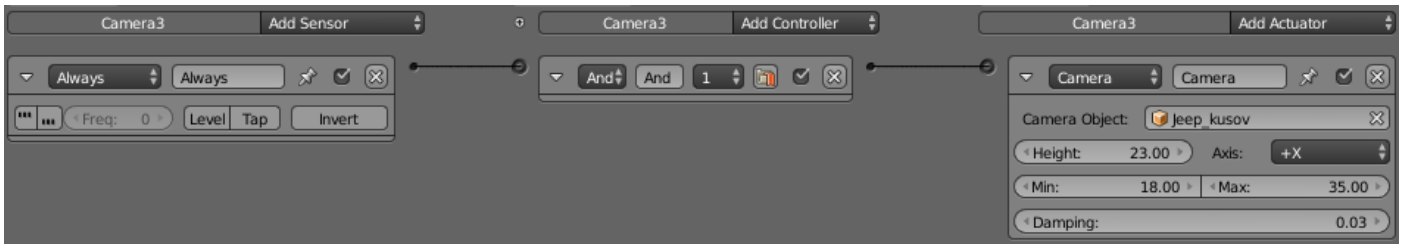


Настройка камеры 1:

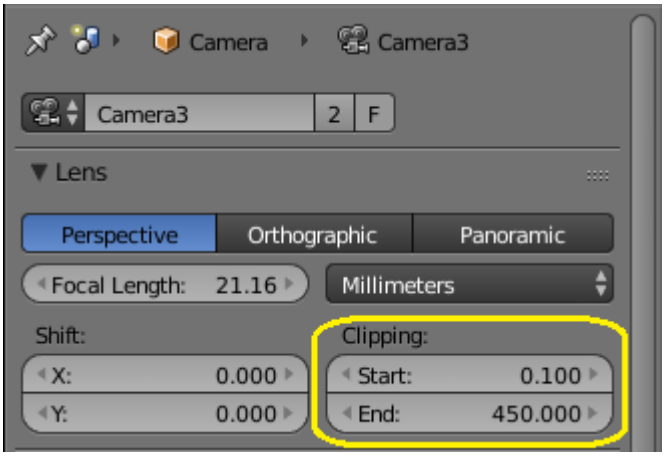
Заодно сделаем переключение между камерами по нажатию клавиш «Z», «X» и «C».
Настройка камеры 2:



Настройка камеры 3:



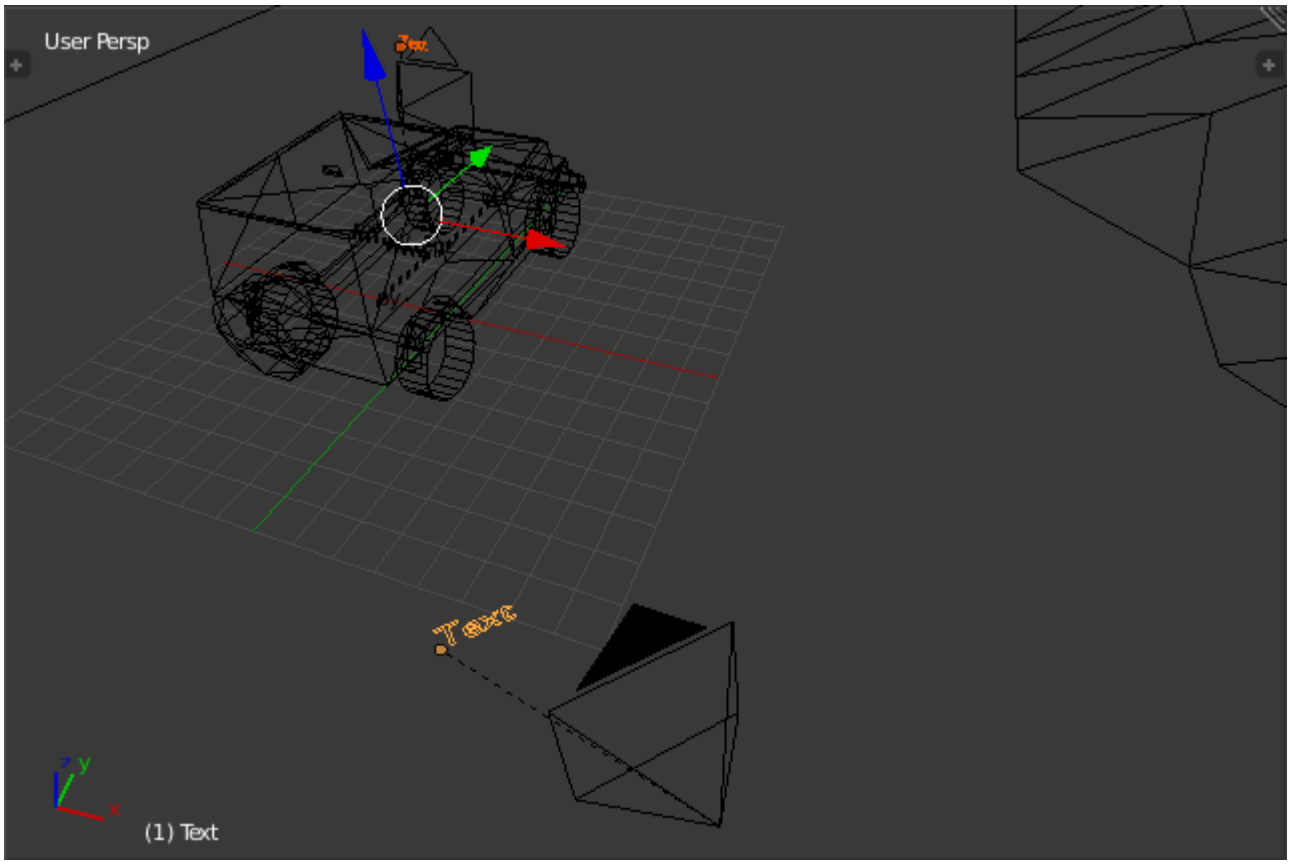
Обязательно увеличим дальность видимости камер, иначе горы будут выплывать из-за горизонта:



Теперь вам придётся «покататься» на машинке, чтобы опытным путём установить, какой должна быть масса и ускорение (gasPower). Массу устанавливаем в физике корпуса, а ускорение в скрипте управления. Заметьте, что чем тяжелее машина, тем

она устойчивее. Но и ускорение нужно больше. Иначе в гору она не поедет. А скорость мы ограничим в скрипте спидометра ☺.

Далее необходимо к каждой камере прикрепить по текстовому объекту. Хотя, это можно было сделать и сразу, в момент создания камер. Все текстовые объекты будут объединены в одном скрипте, и все будут показывать скорость:

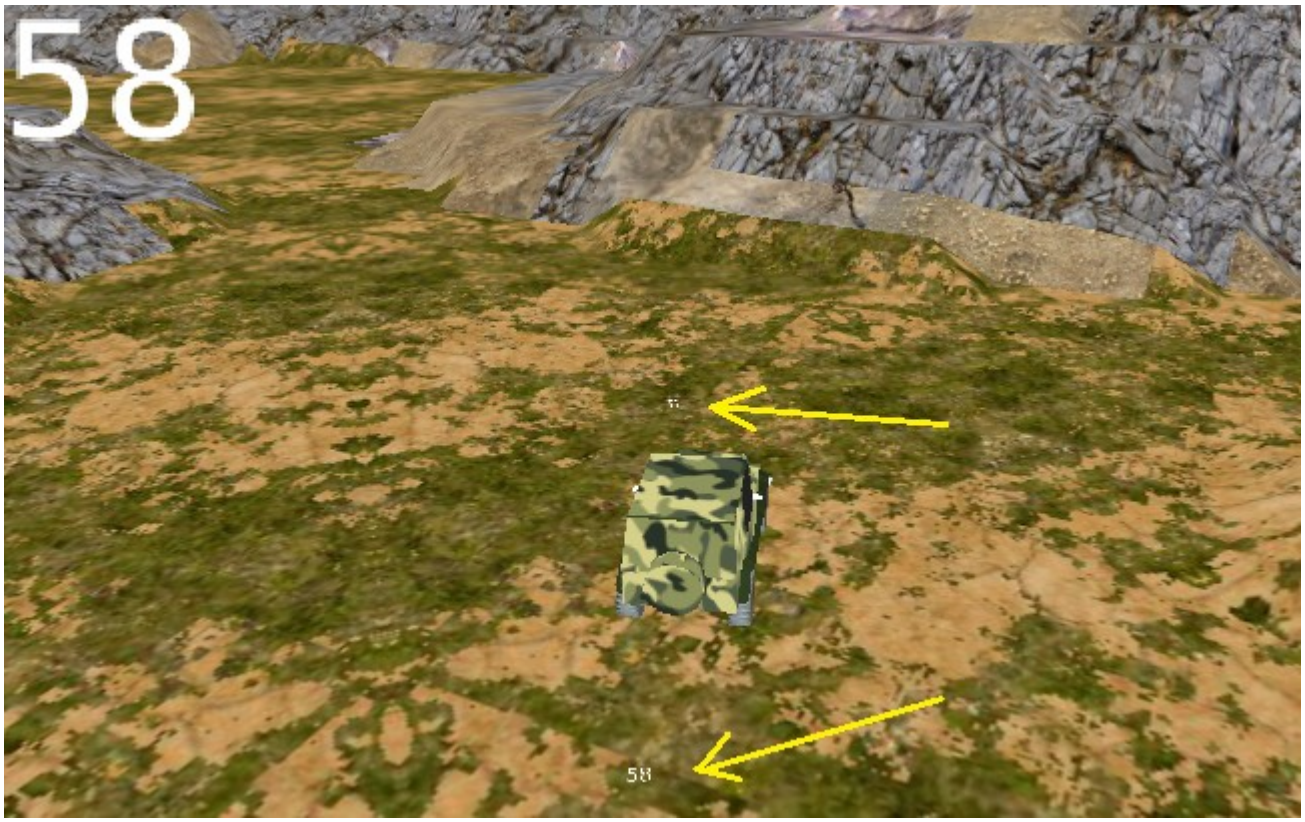


Пронумеруем их в соответствии с камерами: Text, Text2, Text3. Выделяем любую камеру, например 3 и вставляем скрипт спидометра с некоторыми добавками:

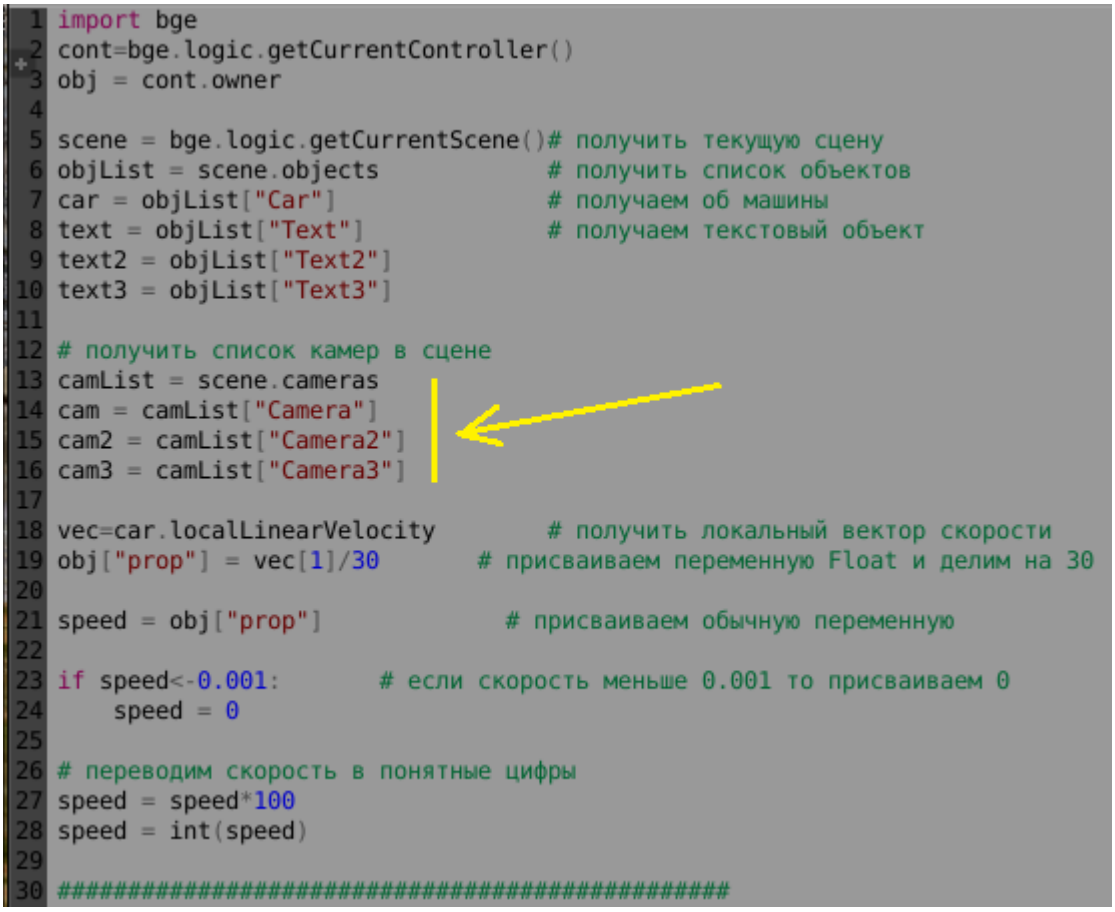
```
import bge
cont=bge.logic.getCurrentController()
obj = cont.owner

scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов
car = objList["Car"]               # получаем об машины
text = objList["Text"]             # получаем текстовый объект
text2 = objList["Text2"]
text3 = objList["Text3"]

vec=car.localLinearVelocity        # получить локальный вектор скорости
obj["prop"] = vec[1]/30       # присваиваем переменную Float и делим на 30

speed = obj["prop"]              # присваиваем обычную переменную

if speed<-0.001:       # если скорость меньше 0.001 то присваиваем 0
    speed = 0

# переводим скорость в понятные цифры
speed = speed*100
speed = int(speed)

# конвертируем цифры в строку
text.text = str(speed)         # выводим на экран
text2.text = str(speed)
text3.text = str(speed)

if speed > 65:  # если скорость больше 65 то
    car.localLinearVelocity = [ 0.0, 19.5, 0.0]
    # линейная скорость по оси Y равна 19.5

```

View Text Edit Format Templates Spidometr_Standart...

Но если посмотреть на всё это взглядом камеры 3, то получается очень некрасивая картина. Видны все текстовые объекты других камер, а нам это совершенно не нужно:



Выход из положения есть. Необходимо включать только тот текстовый объект, который привязан к активной в данный момент камере. Другие же должны быть не видны. Не знаю, как вам, а мне нравится думать над логикой скрипта даже больше, чем над дизайном. А если попробовать включить в скрипт оператор IF, который будет проверять, какая камера активна, и выводить данные в нужный текстовый объект? Значит, в первой части скрипта нужно дополнительно получить список камер в сцене:

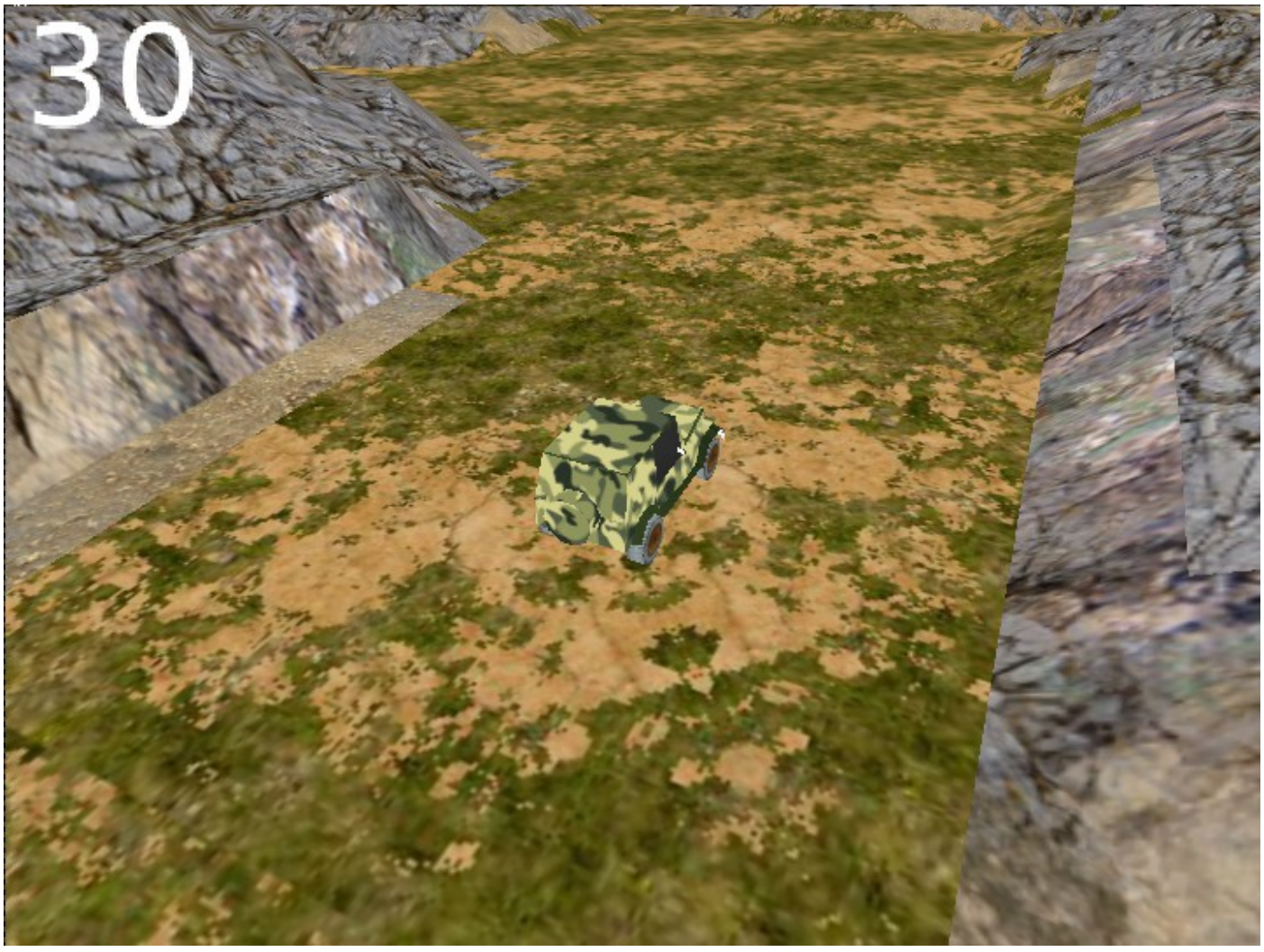


```
import bge
cont=bge.logic.getCurrentController()
obj = cont.owner

scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов
car = objList["Car"]               # получаем об машины
text = objList["Text"]             # получаем текстовый объект
text2 = objList["Text2"]
text3 = objList["Text3"]

# получить список камер в сцене
camList = scene.cameras
cam = camList["Camera"]
cam2 = camList["Camera2"]
cam3 = camList["Camera3"]

vec=car.localLinearVelocity        # получить локальный вектор скорости
obj["prop"] = vec[1]/30       # присваиваем переменную Float и делим на 30

speed = obj["prop"]             # присваиваем обычную переменную

if speed<-0.001:       # если скорость меньше 0.001 то присваиваем 0
    speed = 0

# переводим скорость в понятные цифры
speed = speed*100
speed = int(speed)

################################################
```

А во второй части скрипта установить блок проверки активности камер и печати в соответствующем текстовом объекте:

```
##########################################
# Блок проверки камер

if cam == scene.active_camera:  #Если активна камера1 то
    # конвертируем цифры в строку
    text.text = str(speed)        # выводим на экран
    text2.text = " "   #Пишем пустую строку
    text3.text = " "   #Пишем пустую строку
if cam2 == scene.active_camera:  #Если активна камера2 то
    # конвертируем цифры в строку
    text.text = " "  #Пишем пустую строку
    text2.text = str(speed) # выводим на экран
    text3.text = " " #Пишем пустую строку
if cam3 == scene.active_camera:  #Если активна камера3 то
    # конвертируем цифры в строку
    text.text = " "  #Пишем пустую строку
    text2.text = " " #Пишем пустую строку
    text3.text = str(speed) # выводим на экран

################ Ограничение скорости

if speed > 65:  # если скорость больше 65 то
    car.localLinearVelocity = [ 0.0, 19.5, 0.0]
    # линейная скорость по оси Y равна 19.5

```

Обновлённый скрипт спидометра готов! Запускаем игру и видим, как замечательно у нас всё получилось:



Хочется напомнить вам, что все скрипты, которые мы с вами пишем, желательно сохранять для себя в отдельной папке. Тогда, начиная новый проект, вам не придётся писать их заново. Это не только сокращает время разработки, но и даёт возможность подсмотреть отдельные строки, если вы пишите новый скрипт и что-то подзабыли ☺.

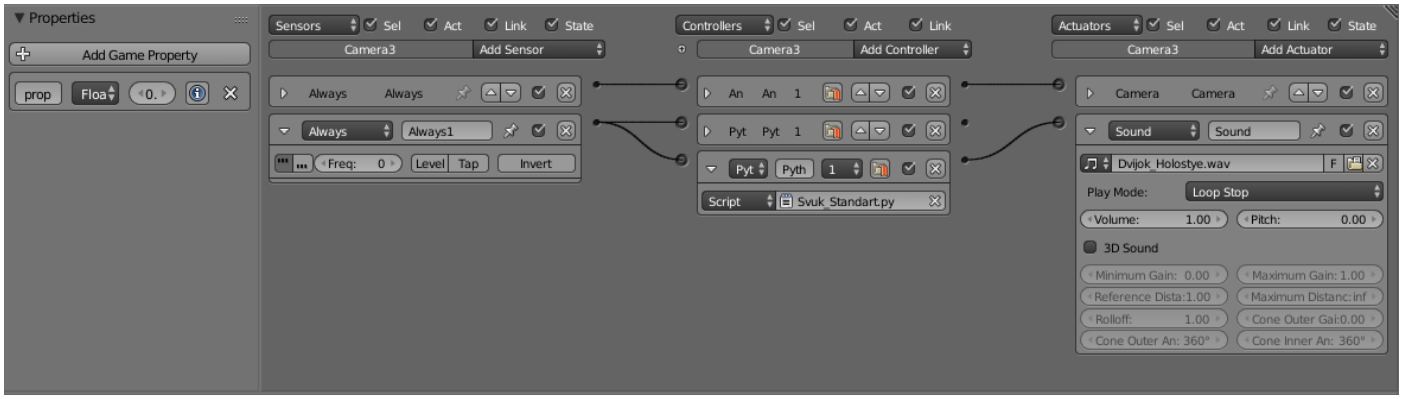
Теперь о звуке. Возможно ли сделать переключение передач? Естественно! Это же Blender с замечательным Python ☺ ! В нём всё возможно. Что мешает нам написать скрипт, который при достижении определённой скорости, будет сбрасывать свойство pitch до нуля, не ограничивая скорости? Выглядит это так:
скорость 0 -> 20 – pitch 0.5 -> 2

скорость 21 -> 40 – pitch 0.5 -> 2

скорость 41 -> 65 – pitch 0.5 -> 2

Получится автоматическая коробка передач. Для ручной коробки достаточно использовать сенсор Keyboard.

Я могу написать код скрипта. Но моё предложение к вам – попробуйте подумать сами ☺. Тем более, что суть я уже разложил по полкам ☺. А для этого учебного проекта мне достаточно одной скорости. Поэтому просто вставляем звук мотора. В логике камеры 3 ставим пару дополнительных блоков:



И не забудем туда впихнуть скрипт для звука из предыдущего урока. Заодно можно и звук тормозов добавить. Для этого выделим объект Car и к сенсору Brake и EBrake добавим контролёр And. А к нему актуатор Sound, со звуком тормозов.

Теперь, если мы хотим сделать EXE файл игры, необходимо все текстуры и звуки запаковать в blend файл. Иначе, даже если мы разместим папку текстур рядом с EXE файлом, мы можем их не увидеть на другом компьютере. Поскольку пути к текстурам будут не совпадать. Для этого в начале делаем так:

File -> External Data -> Pack All Into.blend

А затем: File -> Export -> Save As Game Engine Runtime

и сохраняем в нужную папку.

Ну вот мы и подошли к критической точке нашего игростроения, после которой встаёт вопрос «А что же дальше». А дальше мы должны придумать цель игры. Без неё всё ,что мы делали, не имеет никакого смысла.

Но об этом будем рассуждать в следующем уроке.

Составил **Niburiec** для сайта **http://blender-game.ucoz.ru**