Известная истина гласит о том, что повторение – мать учения. И конечно лучший способ чему-нибудь научиться – это постоянно практиковаться. Естественно, будут ошибки. Но ведь не ошибается только тот, кто ничего не делает. Значит, повторяя один и тот же проект в разных вариантах, мы лишь оттачиваем знания и приобретаем опыт, попутно исправляя ошибки и улучшая общую концепцию построения игры.

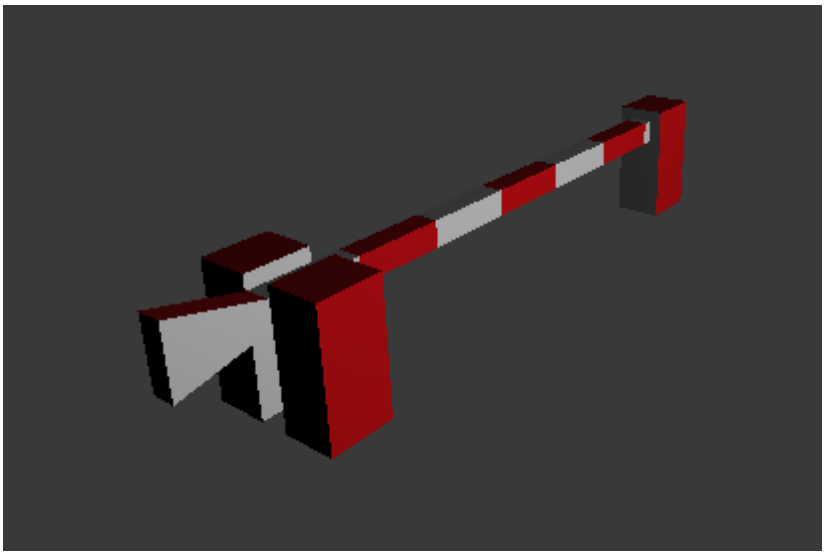
В этот раз мы поговорим о цели игры. Самый простой способ для начинающего игродела - не мудрить с сюжетом, а попробовать в общих чертах скопировать чью-то готовую игру. Желательно по возможности точно, вплоть до текстур. Это не будет плагиатом, поскольку вы это будете делать только для приобретения опыта в понимании стратегии игры, а не в коммерческих или иных целях. Сделав таким образом с десяток простеньких проектов, возможно вы приблизитесь к созданию своего собственного неповторимого творения. И не важно, что в нём может не быть крутой графики и спецэффектов. Зато будет индивидуальность и новый подход, а может и неизбитый сюжет…

Но это в будущем, а пока будем учиться дальше ☺. Попробуем использовать в проекте простую анимацию. Благо, что Blender позволяет анимировать почти всё. Можно сделать открывающиеся двери без применения ограничений и лишней логики. Можно анимировать платформы, автомобили, пропеллеры, объекты, взрывы или смену текстур и формы. Это всё достаточно просто в понимании и практическом применении. Немного сложнее костная анимация человека, например. Хотя, впоследствии мы разберёмся и с ней.

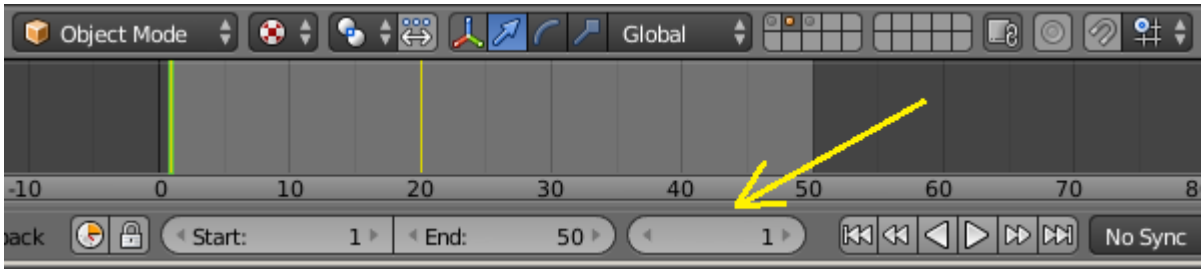
Создаём новый проект (с автомобилем и его скриптами) полностью готовый к употреблению ☺. Мой выглядит так:

Ничего особенного в нём нет. Бордюр с травой немного выше асфальта. Он будет окружён прозрачными панелями, чтобы машина не вылетала за пределы квадрата. Я не вставлял звуки, поскольку это не конечный продукт, а лишь учебный проект. Мой замысел заключается в следующем. Машина выезжает через автоматически открывающийся

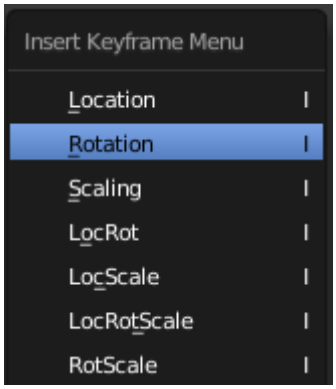
шлагбаум (анимация) и проезжает «змейку» из дорожных фишек до конечного пункта, обозначенного чем угодно. Если фишка задета, то это заносится в ошибку и отображается на экране. Цель - проехать без ошибок. Ещё, было бы неплохо поставить сенсоры, отслеживающие отклонение от трассы или движение в обратную сторону. Начнём по порядку. Сделаем анимацию шлагбаума. Для этого переходим на следующий слой и создадим модель шлагбаума. При этом, сама перекладина должна быть отдельным элементом модели:



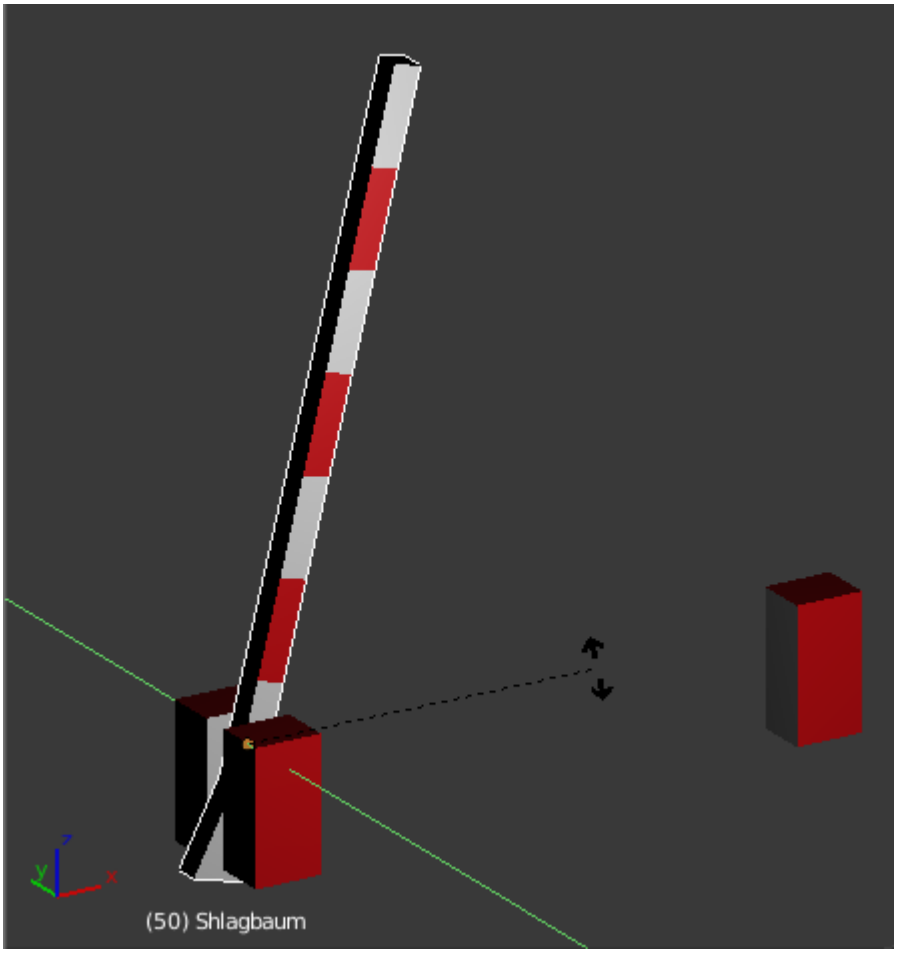
Выделяем перекладину и внизу, на линии анимации, устанавливаем первый кадр:



Теперь в режиме отображения Default с курсором в 3d окне жмём лат. клавишу «I» и Rotation:

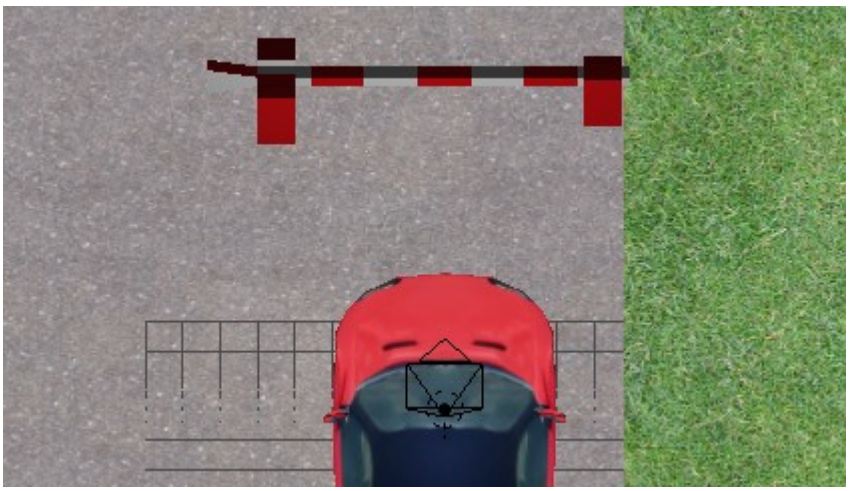


Таким образом мы установили первый ключевой кадр. Далее, устанавливаем 20 кадр на линии анимации. Устанавливаем курсор в 3d окно. Замечаем, что нам необходимо повернуть балку шлагбаума по оси Y. **Надеюсь, вы уже сместили центр шлагбаума в точку поворота?** Жмём клавишу «R» и следом клавишу «Y»(необходимую нам ось поворота) и поворачиваем балку:



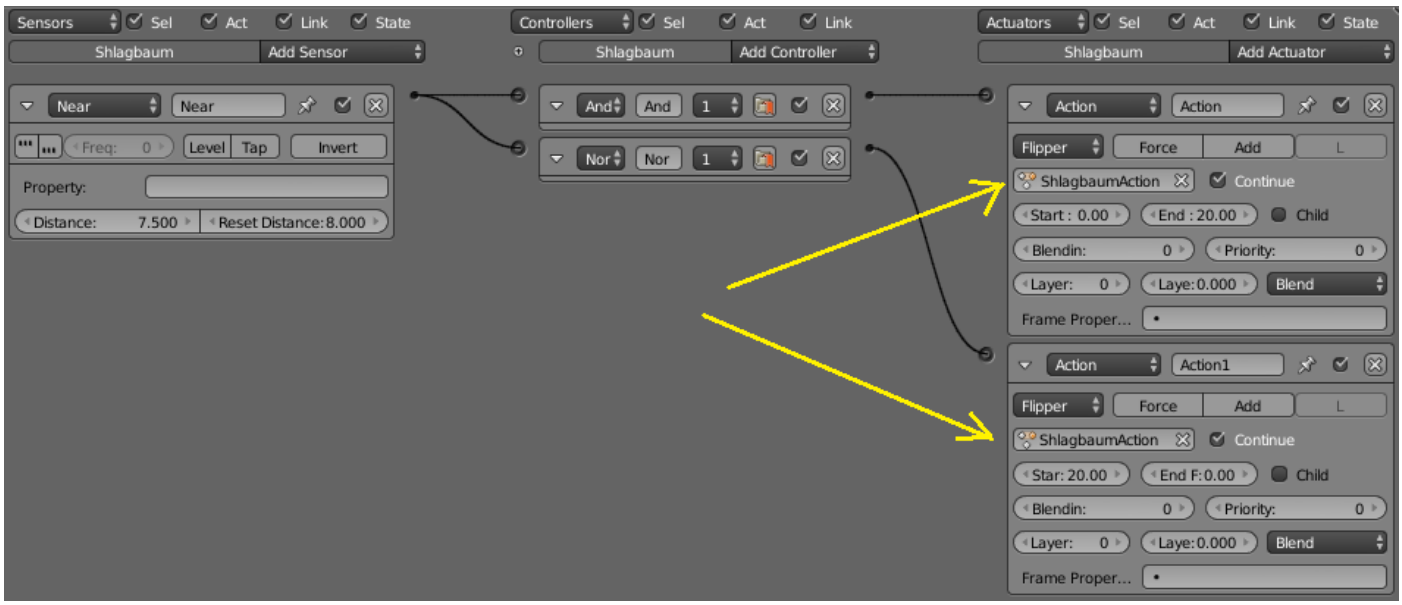
Снова жмём клавишу «I» и Rotation, закрепив таким образом ключевой кадр в 20 кадре. Готово! Теперь можно проверить, как работает анимация, нажав Play:

Сейчас необходимо применить созданную анимацию. Включаем два слоя одновременно и устанавливаем шлагбаум перед автомобилем на расстоянии:

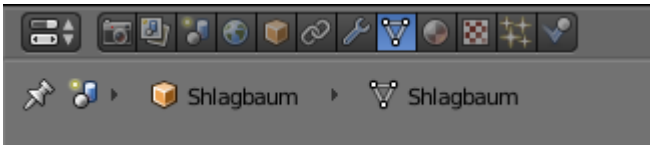


Выделяем балку и переходим в логику. Для того, чтобы шлагбаум открылся, ему необходимо обнаружить объект Rigid Body. Сделать это возможно с помощью двух сенсоров – Near и Radar. Near может обнаруживать любой динамический объект (либо конкретный) на определённом расстоянии вокруг себя. А Radar ещё и в строго определённом направлении. Это полезно, если вы хотите открыть дверь только с одной

стороны, например. В нашем случае, для упрощения задачи, достаточно сенсора Near. К нему, через котроллер And мы подключаем актуатор Action. Он регулирует работу анимации в BGE. Таким образом работа всей логики выглядит так. Если сенсор обнаружил объект, то передаётся положительный импульс на первый актуатор и срабатывает последовательность анимации с 0 по 20 кадр. Как только сенсор перестал обнаруживать объект, то передаётся отрицательный импульс на второй актуатор, через контролёр Nor (для отрицательных импульсов). Срабатывает обратная последовательность с 20 по 0 кадр и шлагбаум закрывается:



Следует учесть, что я назвал словом Shlagbaum и сам объект и сетку меша. Нужно это или нет – честно, я не знаю. Но в каком-то из двух случаев анимация не работала, поэтому я так и поступил:

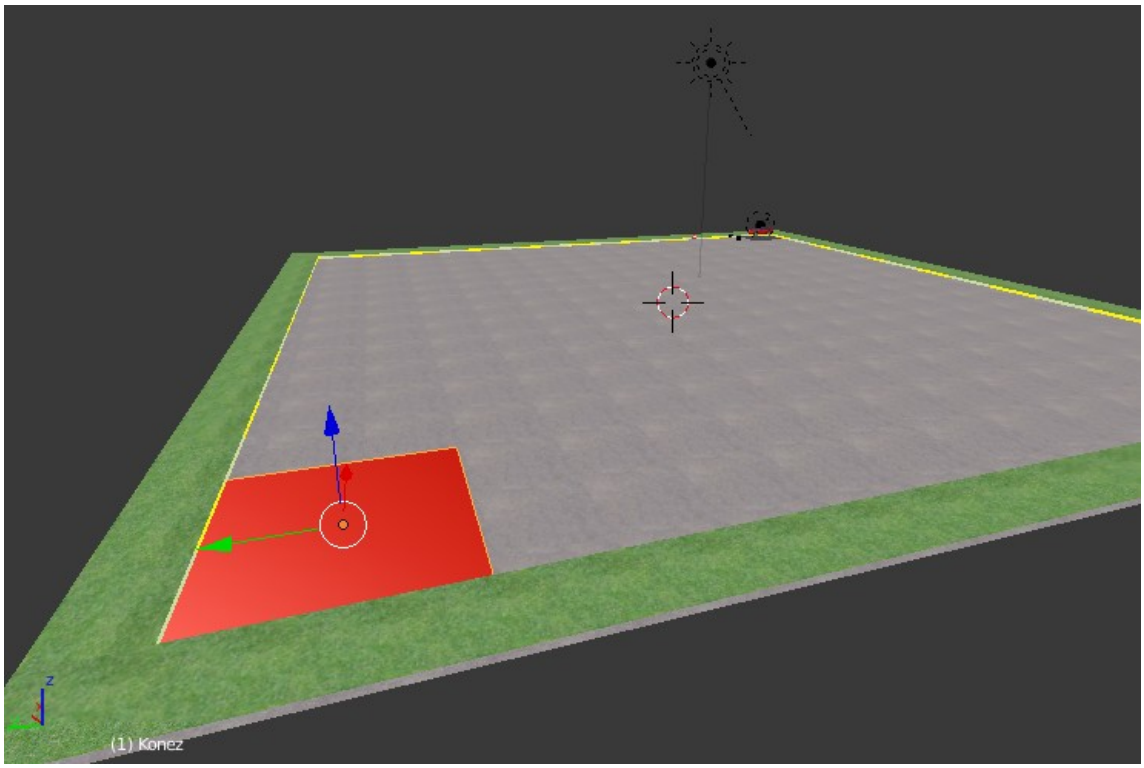


Проверяем игру. При приближении машины к шлагбауму, последний должен открыться. После проезда, шлагбаум должен закрыться. Правда, он откроется даже тогда, когда машина будет сбоку от него ☺. Но в данном обучающем примере это не важно.

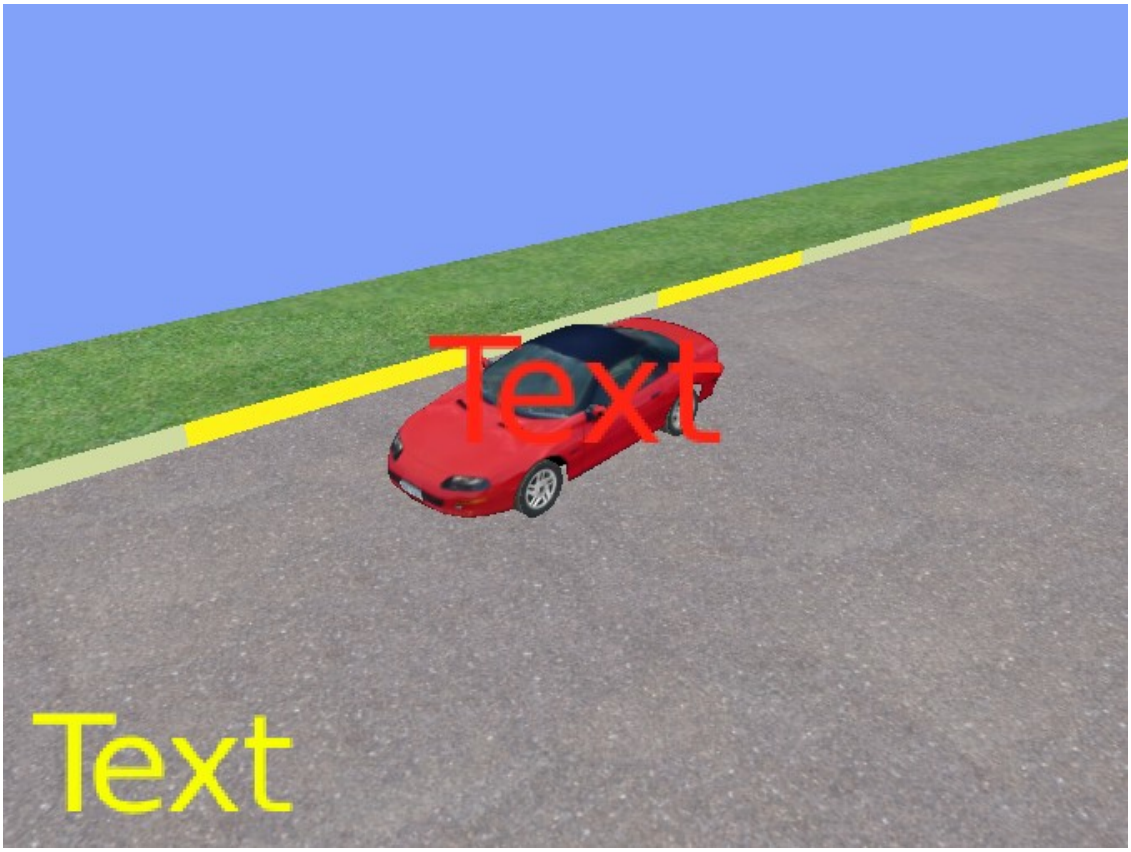
Вообще, вы должны были уже привыкнуть к тому, что каждое новое действие в логике или моделировании игры нужно постоянно проверять. Я называю это «тестирование». Тестировать нужно буквально всё и по многу раз. Нельзя запрограммировать игру и ни разу в неё не играв быть уверенным, что всё пойдёт нормально. Это нонсенс! Иногда это надоедает. Но у меня нашелся выход из положения – это мой сынишка. Я даю ему задание, протестировать игру и он начинает над ней издеваться. Являясь самым непредвзятым критиком, он моментально выявляет все недостатки ☺.

Теперь создадим конечный пункт с сенсором Near. Т.е., когда автомобиль въезжает в определённую зону, то срабатывает переход в другую сцену. Либо просто выводится текстовое сообщение о конце игры. Давайте разместим небольшую плоскость в противоположном по диагонали углу асфальта. Я назвал её «Konez». Это и будет конечный пункт. Хотя, его можно было просто нарисовать текстурой, но тогда пришлось бы вставлять «пустышку», чтобы к ней прикрепить сенсор Near. В физике плоскости выставим No

Collision, чтобы она не реагировала на машину на физическом уровне, и раскрасим её в красный цвет (для заметности):

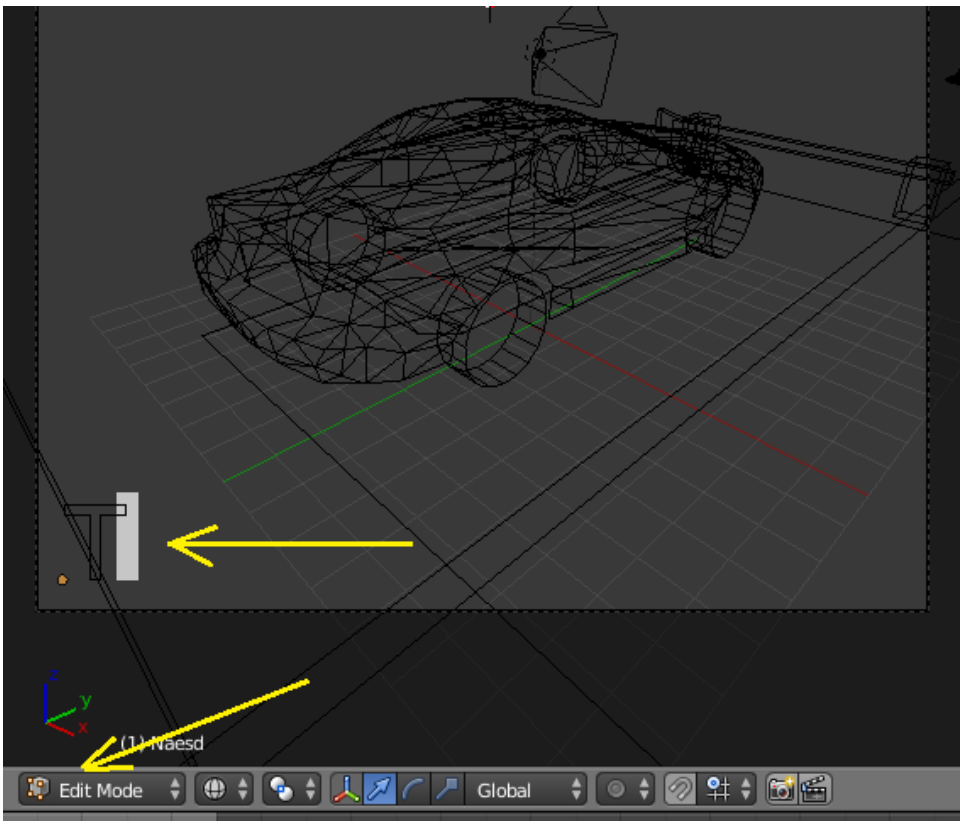


К основной камере прикрепим два текстовых объекта:

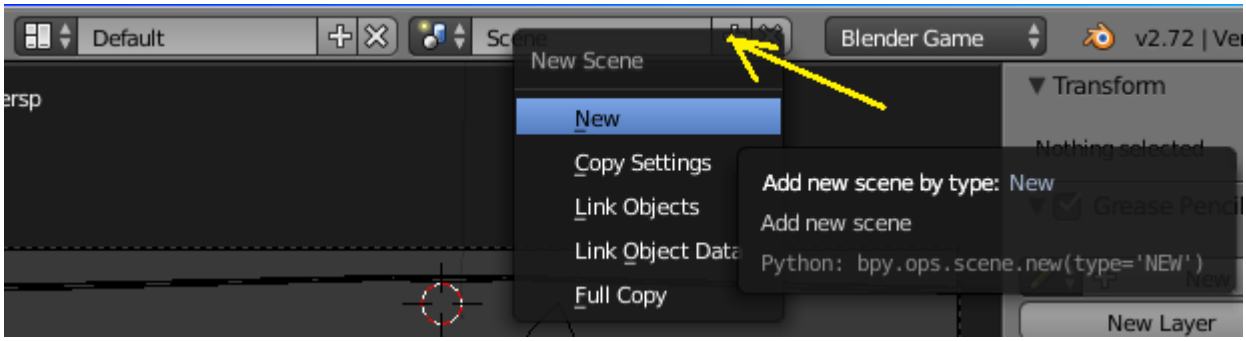


Центральный я назвал «Info», а нижний «Naesd». По названиям понятно, для чего они. Info будет выводить различную текстовую информацию, а Naesd подсчитывать количество сбитых фишек. Возможно, для Info больше подойдёт белый цвет. Я сознательно не стал создавать текстовые объекты во второй камере, прикреплённой к капоту (и переключаемой клавишей «X»), чтобы не усложнять логику. Переключение камер и вывод информации

только в активной мы рассмотрели в предыдущем уроке. Здесь наша задача разобраться с анимацией и целью игры. Поэтому я позволил себе сделать анимацию для текстового объекта Info. Для красоты, т.с. ☺ Я развернул Info на -90 градусов по X и создал первый ключевой кадр. По замыслу, при выводе информации Info будет возникать с поворотом и также уходить в поворот на бок. Всего получилось 50 кадров. Ключевые кадры: 0 – лежит на боку(-90 по X), 20 – поворот(0 по X), 30 – (0 по X), 50 – обратный поворот (-90 по X). Хотя, вы можете этого и не делать, ведь это лишь для красоты ☺. Теперь зайдём в режим редактирования текстовых объектов и удалим текст (Backspace):



Создадим ещё одну пустую сцену и назовём её «Vyhod»:



В ней нужно раскрасить фон и создать камеру.

Переходим назад, в первую сцену.

Выделяем нашу красную плоскость и переходим в логику, где вставляем сенсор Near (назовём его Finish), контролёр Python и пишем скрипт (уже не боимся ☺) под названием Near_perehod:

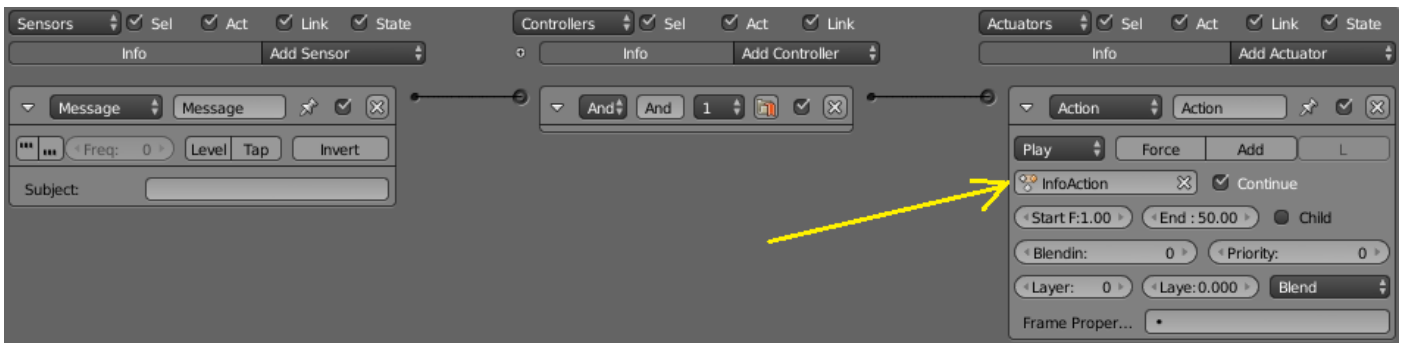
```
1  # -*- coding: utf-8 -*-
2
3  import bge
4
5  # get current scene
6  scene = bge.logic.getCurrentScene()
7  # get the current controller
8  cont = bge.logic.getCurrentController()
9
10 finish = cont.sensors["Finish"]
11
12 # get object list
13 objList = scene.objects
14 # определяем текстовый объект
15 text_info = objList["Info"]
16 text_info.size = 0.8 # меняем размер текста
17 text_info.text = " " # меняем сам текст
18
19 if finish.positive == True:
20     text_info.text = "Уровень пройден"
21
```

View Text Edit Format Templates Near_per

Первая строка необходима для вывода русского шрифта в игре. В строке 10 определяем сенсор. В строке 15 определяем текстовый объект. В строках 19-20: если сенсор позитивен (т.е. определил динамический объект), то выводим текст такой-то…

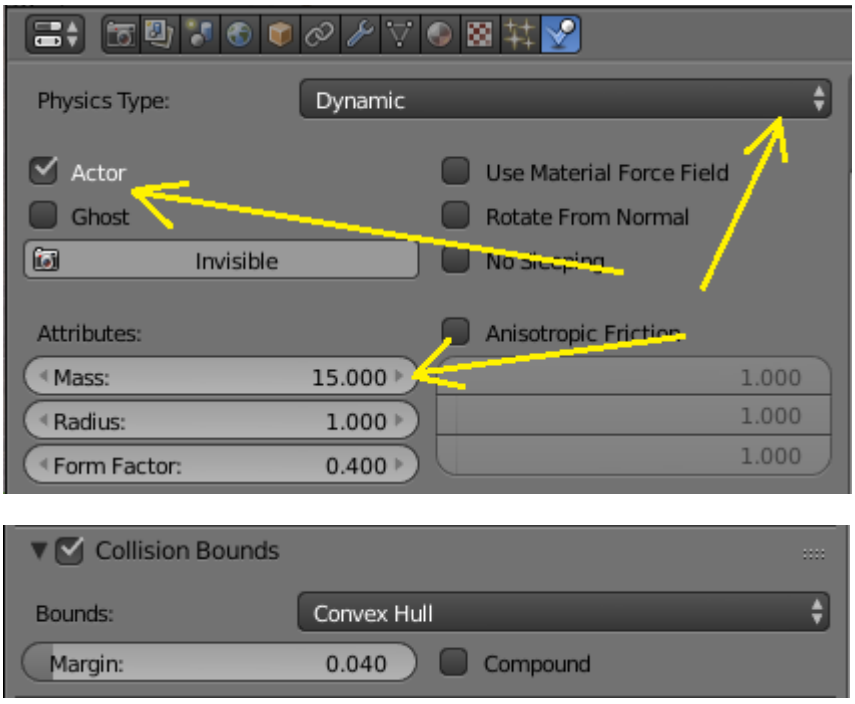
Теперь при наезде машины на красную плоскость будет выводиться текст «Уровень пройден». А вот с анимацией текста всё немного сложнее. Дело в том, что в логике мы не можем вставить анимацию текста в объект «Konez». Иначе анимироваться будет не текст, а сама плоскость (парадокс!). Поэтому мы должны послать сообщение объекту «Info». А уж там принять это сообщение и преобразовать его в анимацию текста. Давайте посмотрим, как это выглядит на практике. Выделяем объект «Konez» и вот наша логика:

Теперь выделяем текстовый объект «Info» и принимаем сообщение реализуя анимацию:

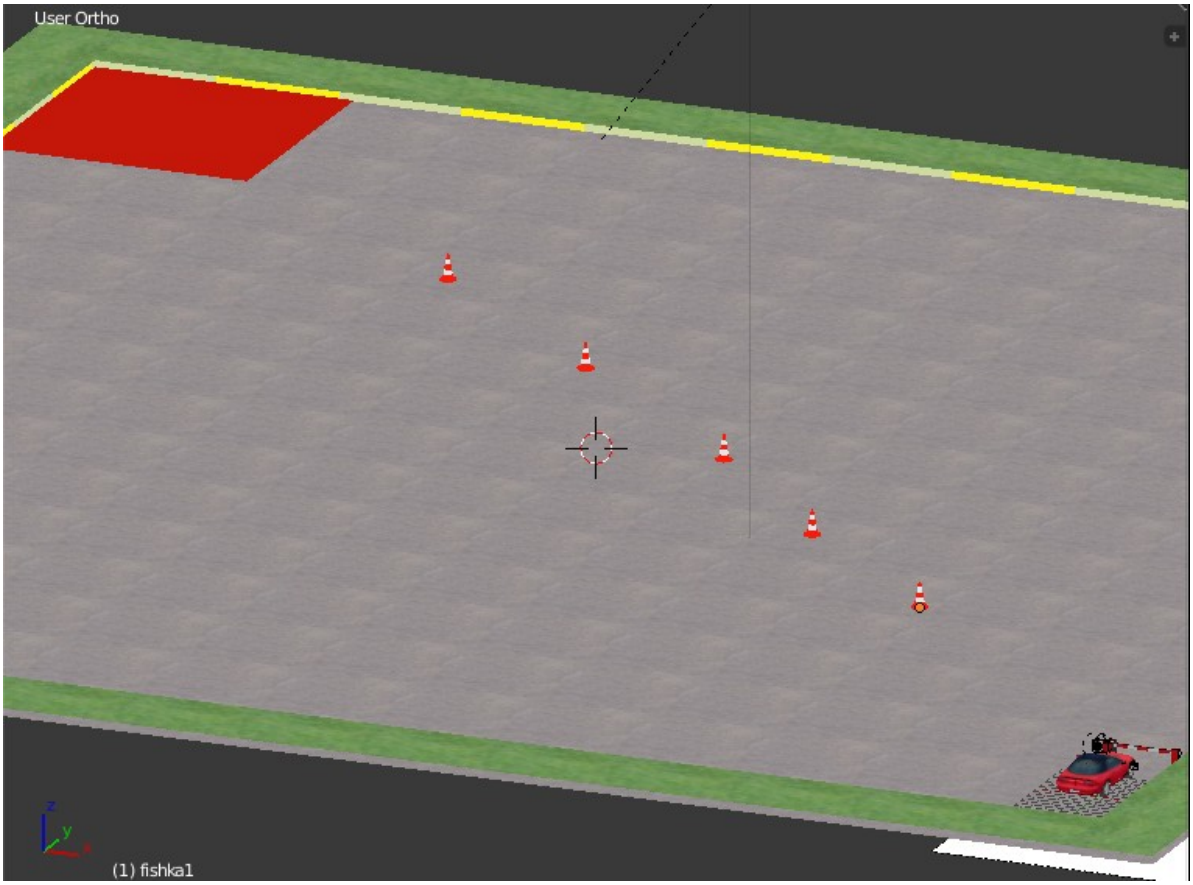


Но, если это для вас слишком сложно ☺, тогда просто сделайте переход в другую сцену. А уже там напишите, что уровень пройден ☺.

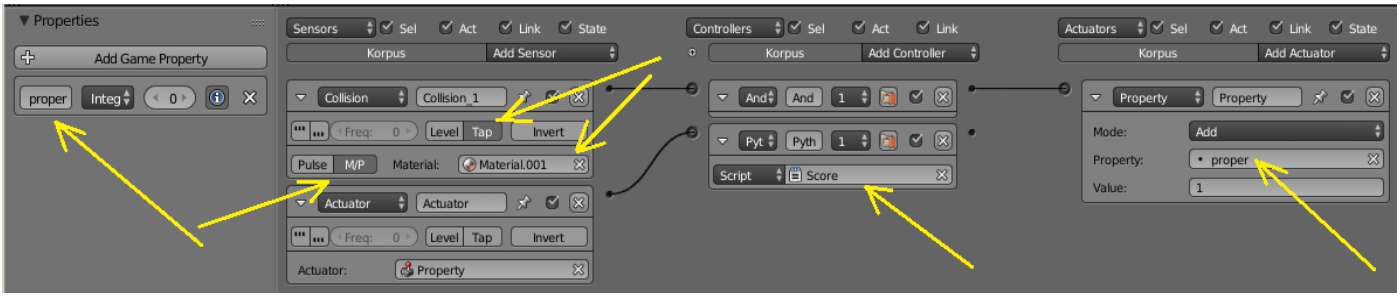
Осталось расставить фишки и настроить подсчёт столкновений. По поводу «расставить..» вы и сами разберётесь. Я сделал готовую фишку формата 3ds и размножил её. А вот по поводу столкновений нужно сделать несколько пояснений. Дело в том, что сенсор Collision замечательно работает только с одним материалом. Т.е. если каждой фишке назначить свой материал, то посчитать столкновения не сложно. Но замусоривать проект «кучей» материалов – огромная расточительность и непрактичность. Ведь мы просто продублировали один объект, а значит и материал у дубликатов один и тот же. Что же произойдёт при столкновении? Сенсор сработает только один раз, считая все остальные фишки тем же материалом (т.е. объектом). Однако Python снова спешит нам на помощь! У сенсора Collision есть функция сброса в первоначальное состояние. Как только сенсор сработал, он успел передать информацию текстовому объекту и после сбросился в начальное состояние – вот что нам нужно! Таким образом можно сосчитать сколько угодно объектов с одним материалом ☺. Будем реализовывать. Загрузим фишку в проект (через импорт 3ds) , разместив её чуть выше плоскости асфальта. Настроим физику, сделав её динамическим объектом:



Теперь продублируем её четыре раза, чтобы получилось 5 фишек. Разместим их на одной линии, по диагонали от машины, до конечного пункта. Расстояние между фишками приблизительно равно двум корпусам автомобиля. Если хотите усложнить, то поставьте их ближе ☺. Выглядеть это будет примерно так:



Теперь выделяем корпус автомобиля и переходим в логику:



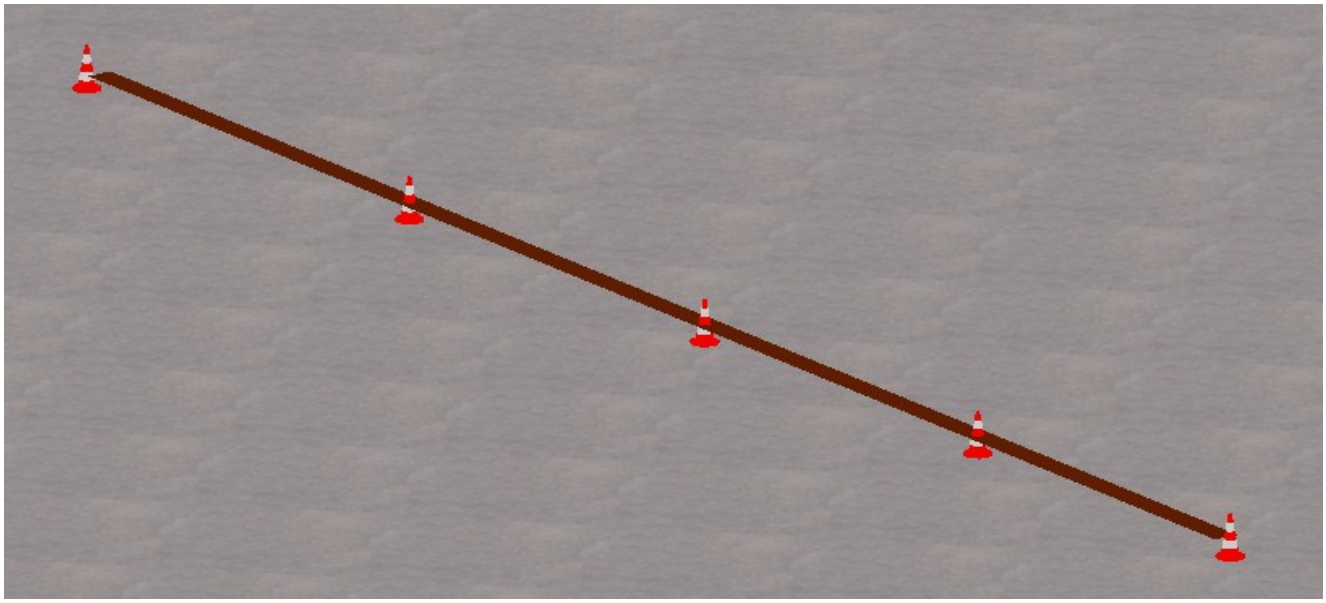
И пишем новый скрипт Score:

```
import bge

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

scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов

text_naesd = objList["Naesd"]              # получаем текстовый объект

score = obj["proper"]              # назначаем переменную

text_naesd.text = str(score)               # конвертируем цифры в строку

sen = cont.sensors["Collision_1"]   # получаем сенсор

sen.reset()  # сбросить сенсор
```

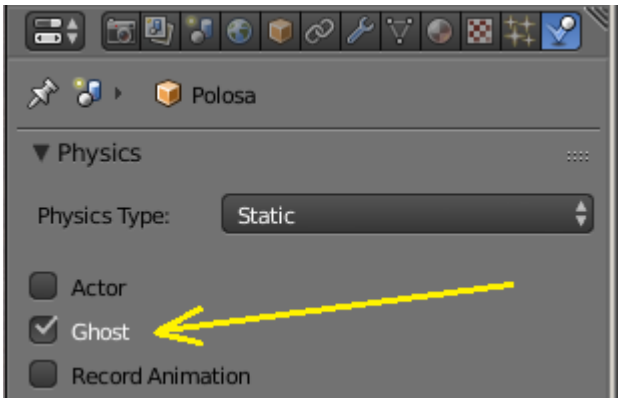

Не забываем, что сенсор можно получить только по имени, которое мы ему задали.

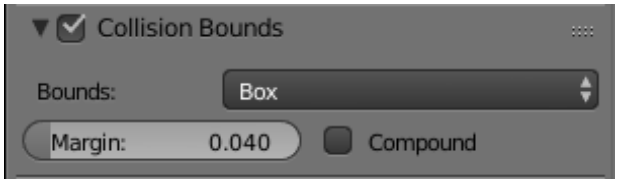
Ну вот. Подсчёт задетых фишек мы реализовали. Осталось несколько штрихов. Как, например, заставить игрока проехать «змейку», а не рядом с фишками? Как проверить, проехал ли игрок без ошибок, или он обманул и проехал прямо до финиша? Ставим задачу и решаем её!

Например, можно приклеить к каждой фишке сенсор Radar и отслеживать перемещение машины. Иногда такое усложнение может быть оправдано, но не здесь. Или, можно точно определить положение фишек и написать скрипт типа: «…если фишка сместилась по оси N, то делаем то и то…». Но это тоже не выход. Наша задача найти простое и действенное решение для данного конкретного случая. Поэтому мы поступим проще – будем снова отслеживать столкновения. С чем? Да с плоскостью, например! Создадим плоскость шириной с фишку и длинной от одной крайней фишки до другой. Опустим её так, чтобы она была где-то в середине высоты фишки:

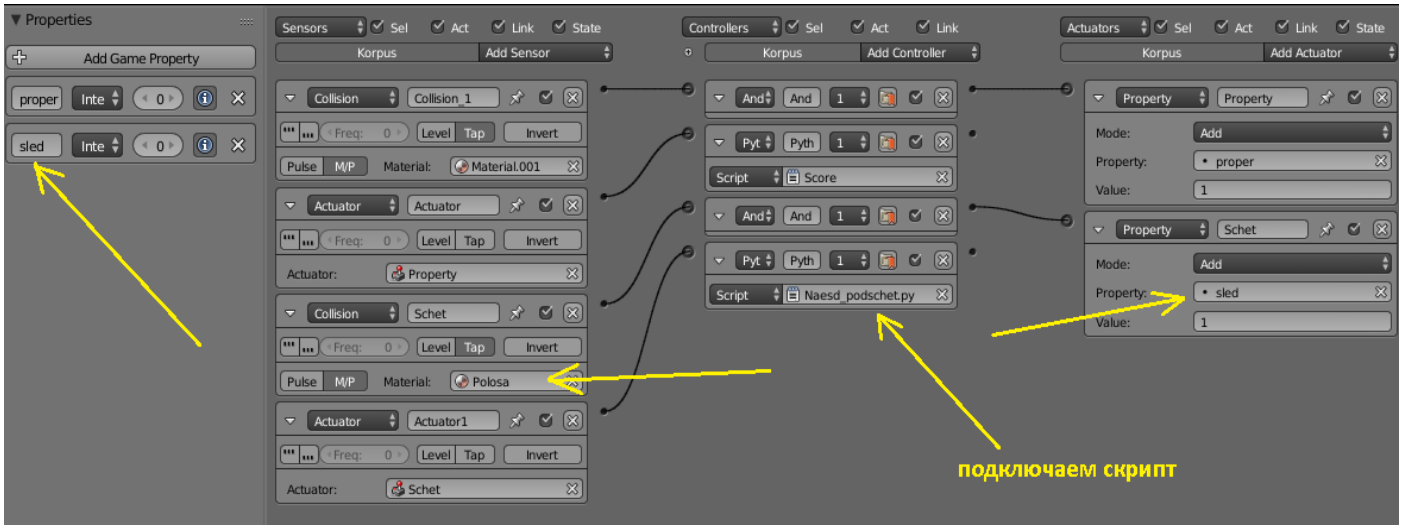


Назовём плоскость «Polosa» и материал ей назначим с таким же именем. А вот в физике нам нужно чтобы и столкновения отслеживались, и машина проходила сквозь плоскость. Для этого просто ставим галочку Ghost:





Выделяем Korpus автомобиля и, по факту, делаем то же, что и с фишками:



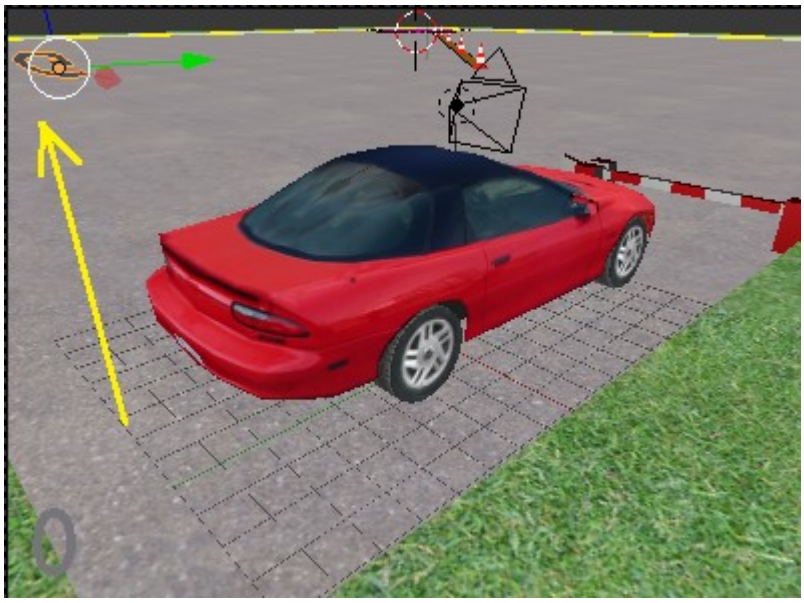
Пришем скрипт Naesd_podschet :

```
import bge

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

scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов

#text_naesd = objList["Naesd"]            # получаем текстовый объект

score = obj["sled"]             # назначаем переменную

#text_naesd.text = str(score)            # конвертируем цифры в строку

sen = cont.sensors["Schet"]   # получаем сенсор

sen.reset()  # сбросить сенсор

```

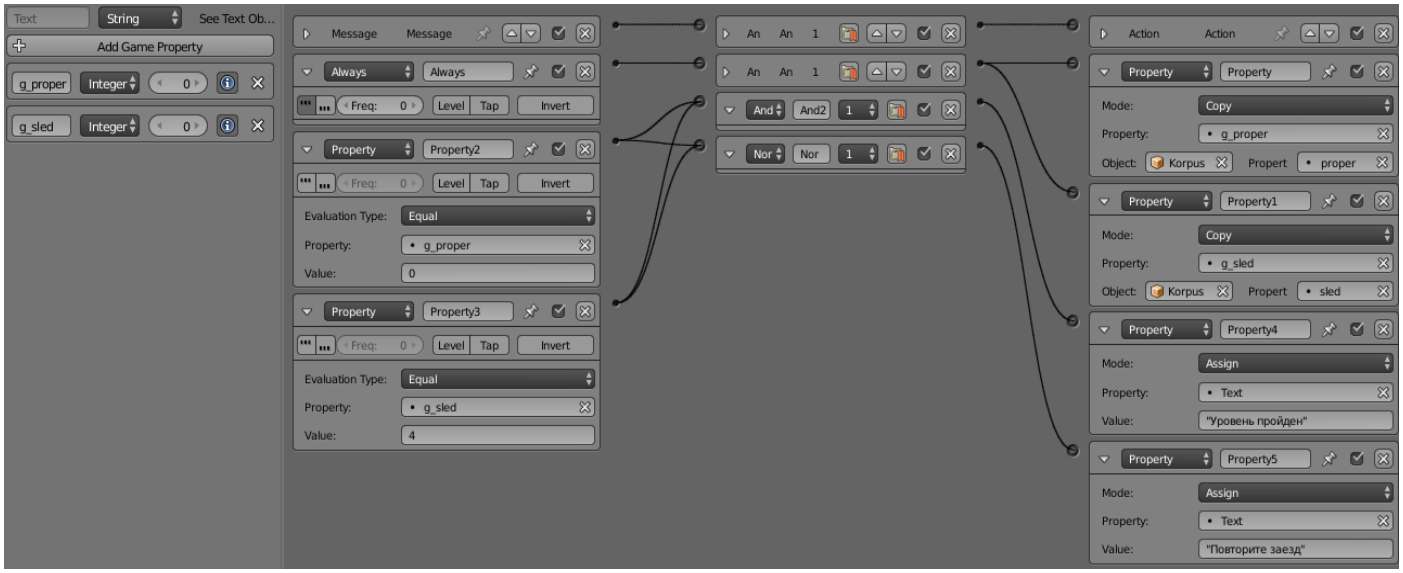
View Text Edit Format Templates Naesd_podschet.py

Готово. Если теперь машина будет пересекать плоскость, то скрипт будет считать количество пересечений. В данном случае это 4. Осталось дополнить логику сравнением количества пересечений: если машина не пересекла полосу или пересекла больше 4 раз, то выводится сообщение «..Не верно!». Если пройдено правильно, то «..Уровень пройден!». Для этого выделим красную квадратную плоскость финиша и в её логике удалим контролёр Python, со скриптом Near_perehod, поскольку здесь он больше не нужен:

Кстати, в актуаторе Message в поле Subject можно написать имя, для того, чтобы сенсор Message мог получать сообщения только этого актуатора сообщений (в сенсоре тоже указываем имя). Выделяем верхний текстовый объект (Info):

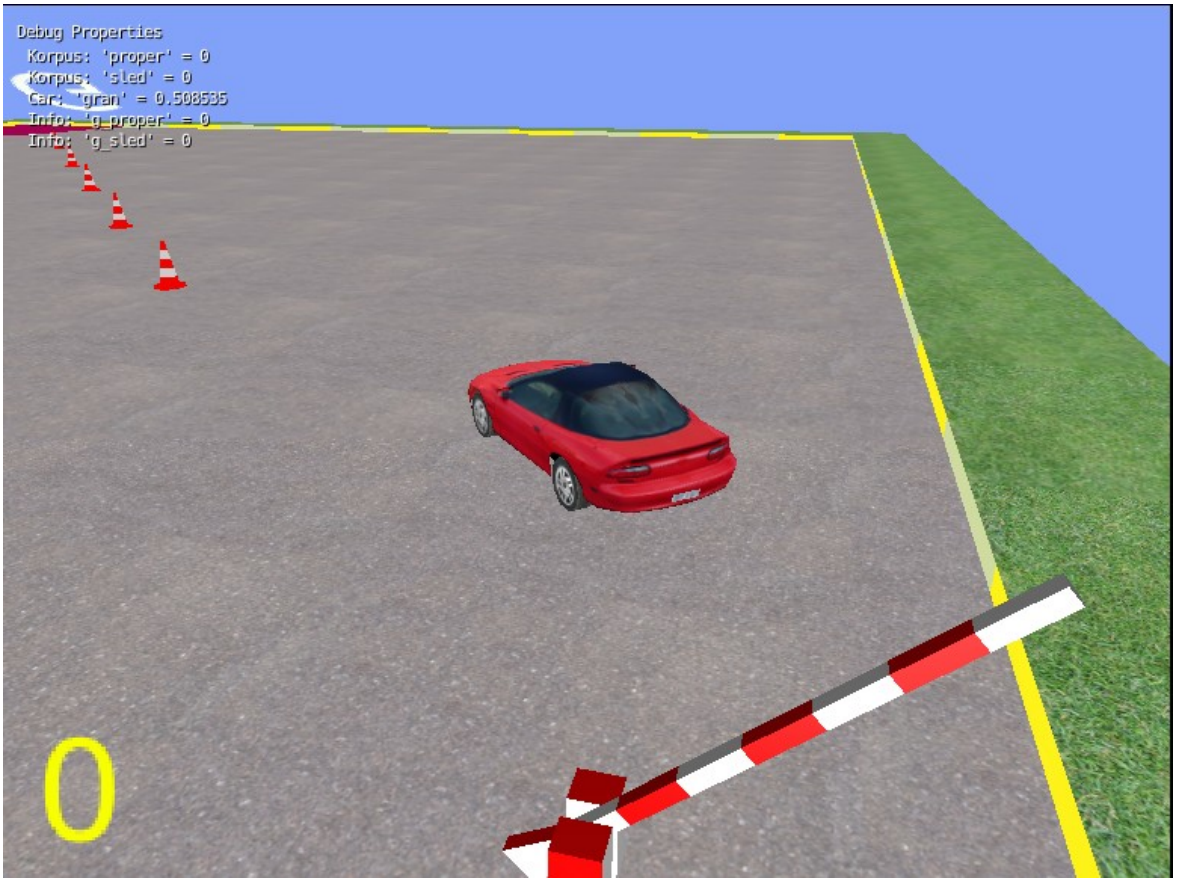


И создаём два новых свойства: g_proper и g_sled. В них (по способу, описанному в уроке 22) мы копируем свойства proper и sled:



И, в сенсорах property2 и property3, сравниваем полученные данные. Если g_proper = 0 , а g_sled = 4 , то пишем один текст. Иначе – другой. Всё просто! Кроме всего прочего, мы можем сделать полосу невидимой (в материале включим Invisible) ☺.

Я не гарантирую, что игра будет работать так, как ожидалось. Задача была проста – показать возможность использования простой анимации и взаимодействия с динамическими объектами. Возможно, нужно ещё подумать и что-то доделать. Но принцип я объяснил. Всё зависит от вас ☺. Далее мы попробуем разобраться с тем, как создать соперника. А ещё – о примитивном варианте искусственного интеллекта.



24 мая 2015 года.

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