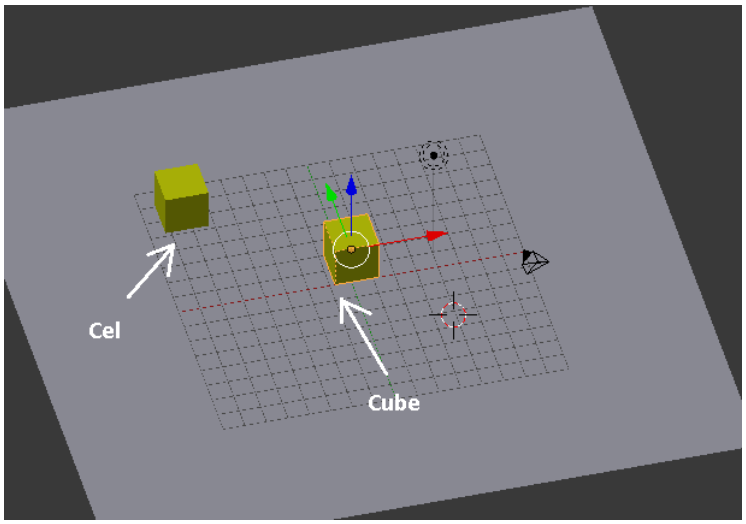
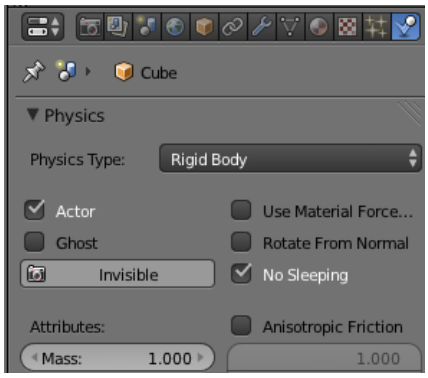
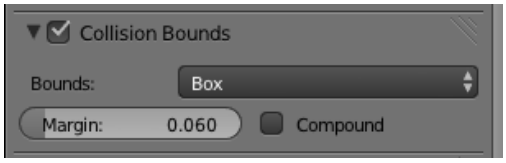
Пример простого слежения за объектом и движения к нему

Для начала создадим план и два объекта на нём. Кубик в центре будет актёром, а внешний кубик будет целью, к которой будет двигаться актёр:

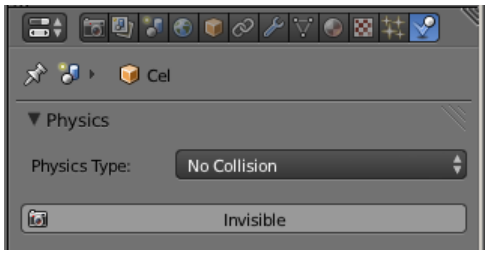


Выставим физические свойства у нашего актёра:



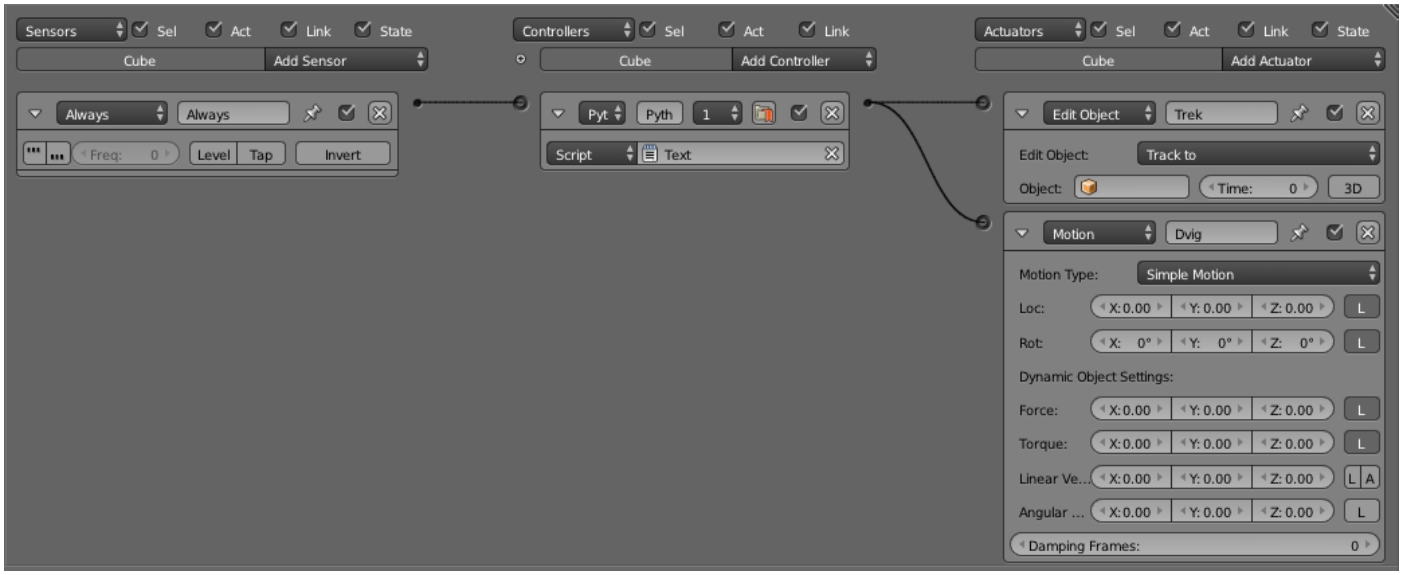


И у цели, тоже настроим физику:



Не забываем, что всё это в режим Blender Game. Переходим к логике. Создаём один сенсор, один контролёр и два актуатора (слежения и движения).

Контролёр Python. В нём мы позже впишем скрипт. Актуатор слежения назовём **Trek** , а актуатор движения назовём **Dvig** . Не забываем, что в скрипте используются имена актуаторов.



И пишем вот такой скрипт:

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

act_t = cont.actuators["Trek"] # Подключаем актуатор слежения
act_t.object = "Cel"               # Назначаем объект слежения
cont.activate(act_t)              # Активируем актуатор слежения

act_d = cont.actuators["Dvig"]  # Подключаем актуатор движения
act_d.useLocalForce = True      # Выставляем локальную ось
act_d.force  = [ 0.0, 5.0, 0.0]    # Двигаем объект
cont.activate(act_d)              # Активируем актуатор движения
```

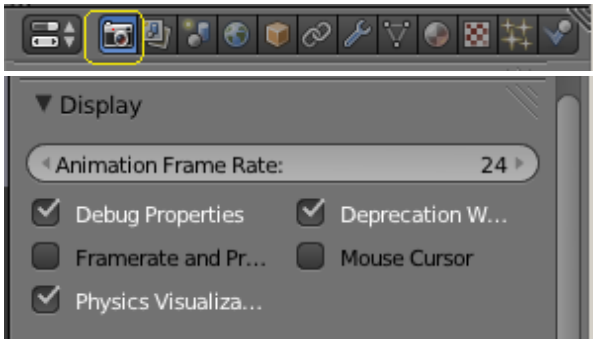
Вариант 2

##

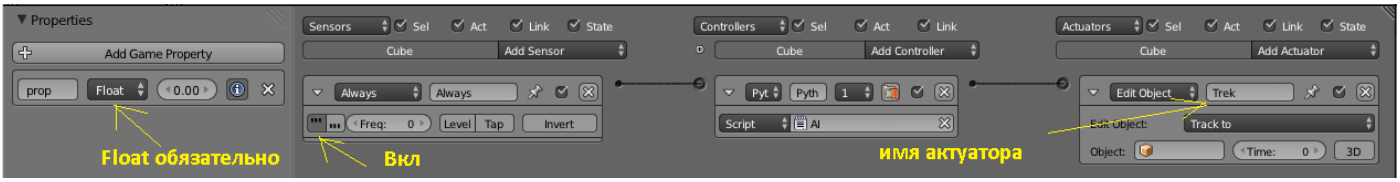
Однако, это очень упращенное представление. Есть много нюансов, которые заставляет нас усложнить скрипт. Ведь этот скрипт не позволяет нам организовать сложный путь передвижения. А именно это, зачастую, необходимый элемент игрового интеллекта. Например, в гоночном симуляторе, где соперники должны сами ехать к цели.

Поэтому, в новом варианте мы заставим объект цели мгновенно перемещаться по точкам, под управлением таймера. А актёр будет следить за целью и двигаться к ней.

Для начала в логике объекта актёра (Cube) мы удалим актуатор движения Motion. После нескольких попыток применить его по проще, я от него отказался. А скрипт актёра переименуем в **AI**. И добавим переменную **prop**. Она будет показывать нам изменение дистанции от актёра до цели. Это удобно при отладке. Правда, для этого нужно включить отображение Debug Propirties:



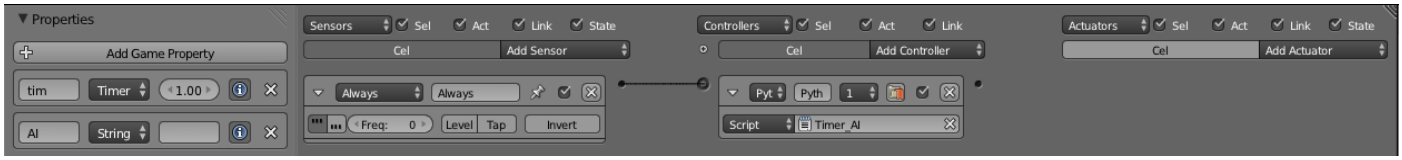
Ну, а Physics Visualization я включил для того, чтобы были видны границы физики объектов. И так – логика актёра:



Тут всё понятно. Актуатор назовём **Trek**. Теперь чуть изменённый скрипт:

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

scene = bge.logic.getCurrentScene()      # получить текущую сцену
objList = scene.objects                  # получить список объектов
akter = objList["Cube"]                  # получить объект актёра
zel = objList["Cel"]                     # получить объект цели


distance = obj.getDistanceTo(zel)        # получить дистанцию до цели

while distance > 1:                      # пока дистанция до цели больше единицы выполняется тело
   obj["prop"] = distance                      # применить значение дистанции к переменной PROP для отображения
   act = cont.actuators["Trek"]                # получить актуатор с именем Trek
   act.object = "Cel"                          # назначить объект для слежения
   cont.activate(act)                          # активировать актуатор слежения
   akter.localLinearVelocity = [ 0.0, 5.0, 0.0]   # придать линейную скорость актёру в локальных координатах
   if exitTest(): break                        # Экстренный выход из цикла

akter.localLinearVelocity = [ 0.0, 0.0, 0.0]     # Изменить скорость в ноль, по окончании цикла (дистанция меньше 1)
```

Здесь все объяснения в комментариях. Чуть не забыл. Волшебная кнопка с тремя точками в сенсоре нужна для того, чтобы скрипт работал непрерывно. Если её отключить, то скрипт сделает лишь один проход и остановится.
Переходим к следующему скрипту – скрипту таймера. Для удобства подключим его к объекту цели:



Как видим, актуаторы в данном скрипте нам не понадобятся. Переменная **tim** это и есть таймер. Переменная **AI** всего лишь вспомогательная, для отображения дополнительной информации. Новый скрипт назовём **Timer_AI**.

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

obj.worldPosition = [-6.2, 10.2, 1.0]          # первоначальная позиция объекта цели

timer = obj["tim"]                             # назначаем переменную таймер

if timer >3:                                    # Если число таймера больше 3
   obj.worldPosition = [-6.2, -11.2, 1.0]      # Меняем позицию цели
   obj["AI"] = "Tochka_1"                       # Выводим на экран сообщение Tochka_1
if timer >6:                                    # Если число таймера больше 6
   obj.worldPosition = [8.5, -11.2, 1.0]       # Меняем позицию цели
   obj["AI"] = "Tochka_2"                       # Выводим на экран сообщение Tochka_2 и т.д.
if timer >9:
   obj.worldPosition = [8.5, 10.2, 1.0]
   obj["AI"] = "Tochka_3"
if timer >12:
   obj.worldPosition = [-6.2, 10.2, 1.0]
   obj["AI"] = "Tochka_4"
if timer >15:
   obj["tim"] = 1
   obj["AI"] = "Stop"
```