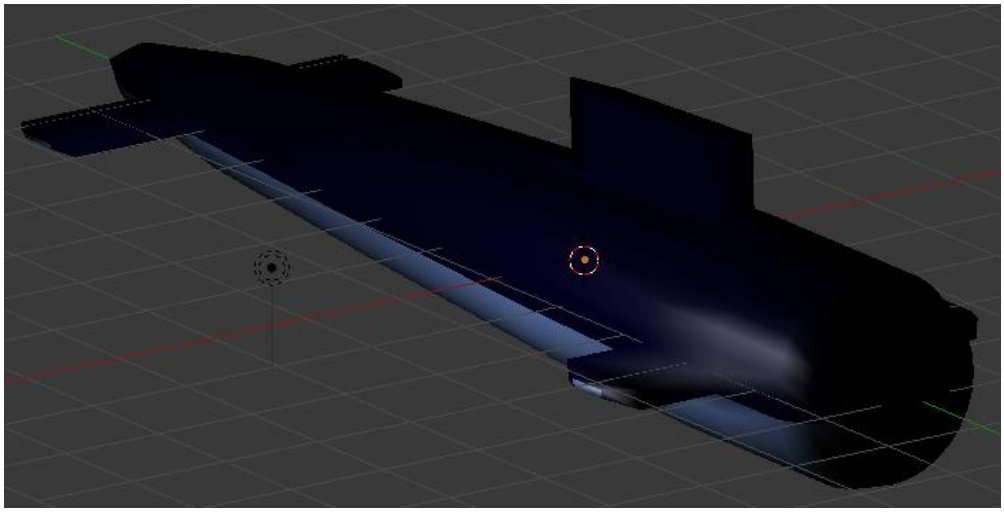
А теперь, на основании уже имеющихся у нас знаний, давайте попробуем создать заготовку простенькой игры. Например, вы находитесь в подводной лодке и должны уничтожить наибольшее количество вражеских субмарин. Естественно вы будете смотреть в перископ. А ещё в вашем распоряжении будет эхолот и строго определённое количество торпед. В конце урока размещены ссылки для скачивания проекта с текстурами и готовой игры, с исполняемым файлом EXE. Для ознакомления полезно изучать чужие «исходники», однако будет гораздо лучше, если вы сделаете всё своими руками. Включая картинки текстуры перископа, например в графическом редакторе GIMP. Напомню, она должна быть в формате PNG, с прозрачным фоном.

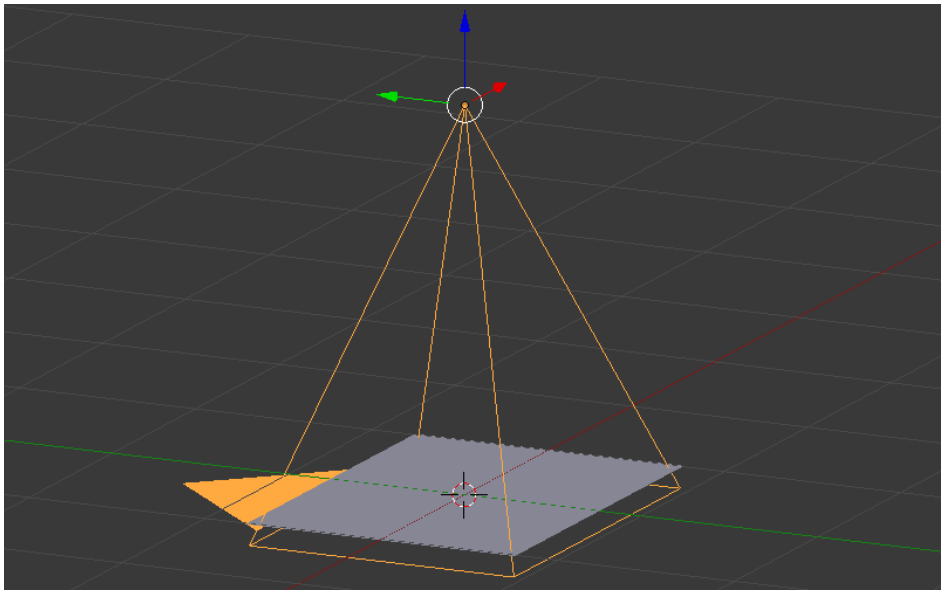
Опуская подробности моделирования (вы уже должны уметь создавать и окрашивать простейшие модели), будем останавливаться лишь на самом главном.

На первом слое  создадим и окрасим модель вражеской подводной лодки:

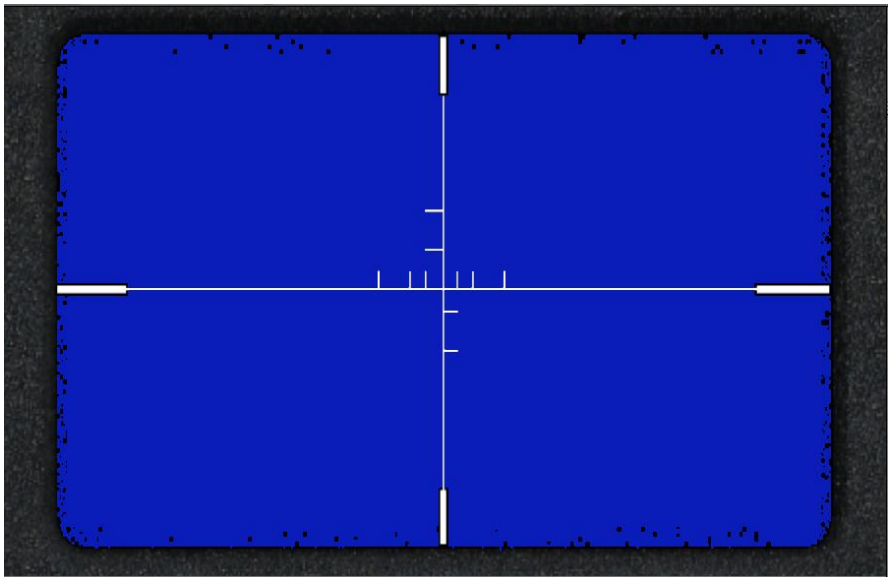


Сразу включим игровой режим Blender Game и отображение текстур. Заодно, удалим камеру, а свет можно пока оставить. Потом мы его изменим и переместим.

Переходим на второй слой  и создаём в нём камеру, в которой все значения поворотов Rotation выставляем в ноль. Наша камера должна смотреть строго вниз. А под ней создаём простой план, которому назначаем прозрачный материал и накладываем текстуру картинки перископа. Не забудем в материале выставить самосвечение, иначе картинку не увидим. Должно получиться приблизительно вот так:



Как видите, план полностью уместился в размер камеры. Для наглядности я выставил цвет неба окружающего мира World -> Horizon Color в синий цвет. Если включить вид с камеры в режиме отображения текстур и нажать лат. P, то это должно выглядеть приблизительно так:



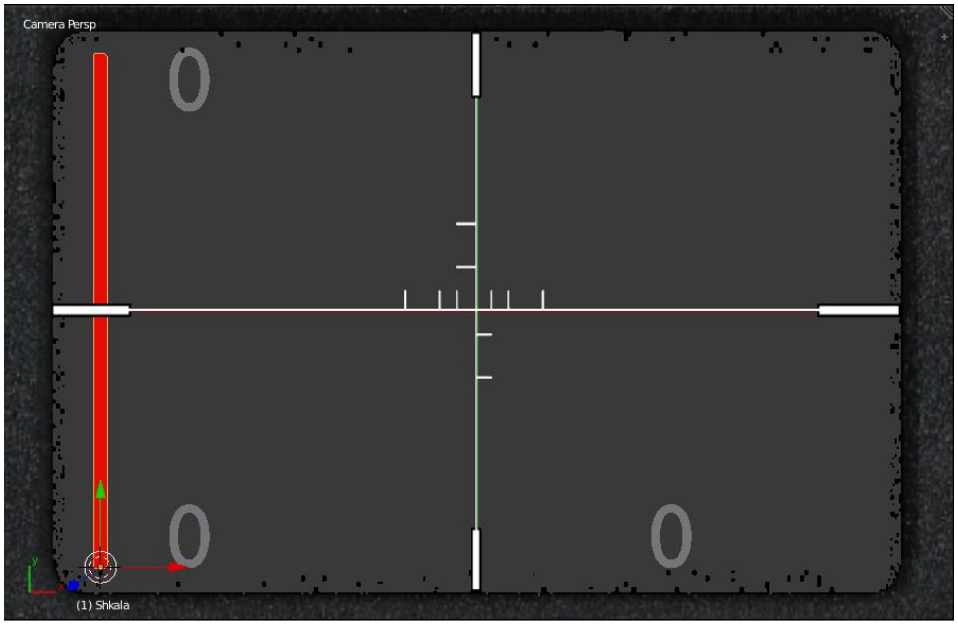
Конечно, для большей достоверности необходимо ещё и туман добавить. Но об этом поговорим позже.

Добавлю, что для упрощения наша камера (перископ) будет всегда неподвижна. Двигаться будут лишь вражеские субмарины. Если субмарина попала в перекрестие прицела и шкала эхолота показывает идеальное расстояние для выстрела, вам остаётся только вовремя нажать кнопку выстрела.

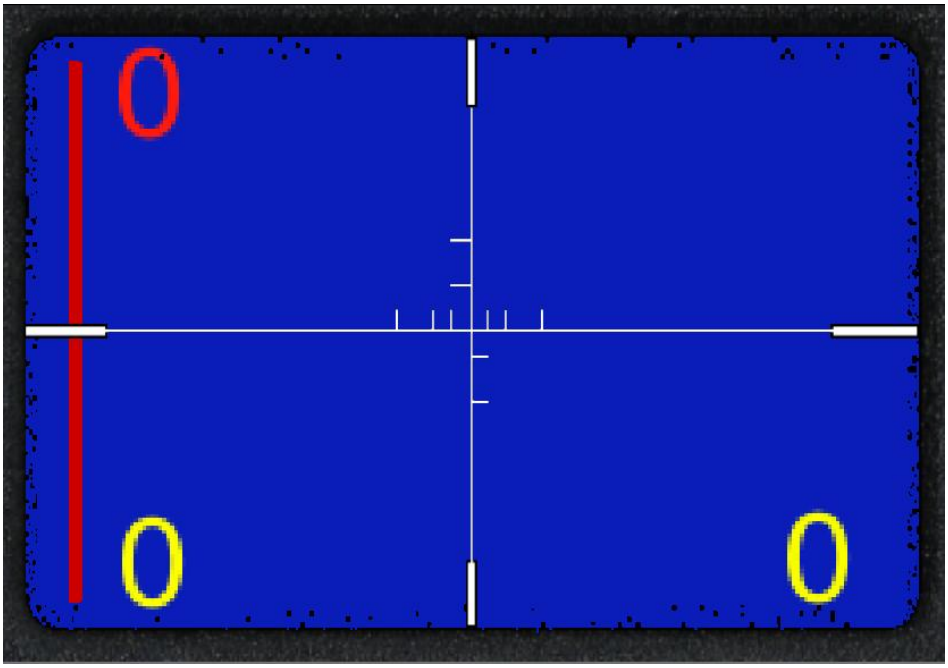
Давайте доукомплектуем наш прицел текстовыми объектами. Их нам понадобится целых три. Первый будет показывать расстояние до цели, второй – количество уничтоженных субмарин, и третий – количество оставшихся торпед. А для красоты, сделаем ещё и вертикальный индикатор эхолота. Это будет обычный объект «куб», нужного нам размера.

Переходим на второй слой, где у нас камера и выбираем вид сверху. Теперь добавляем текстовые объекты с цифрой «0». Просто кликаем в нужном месте лев.кноп.мыши, чтобы поставить указатель курсора:

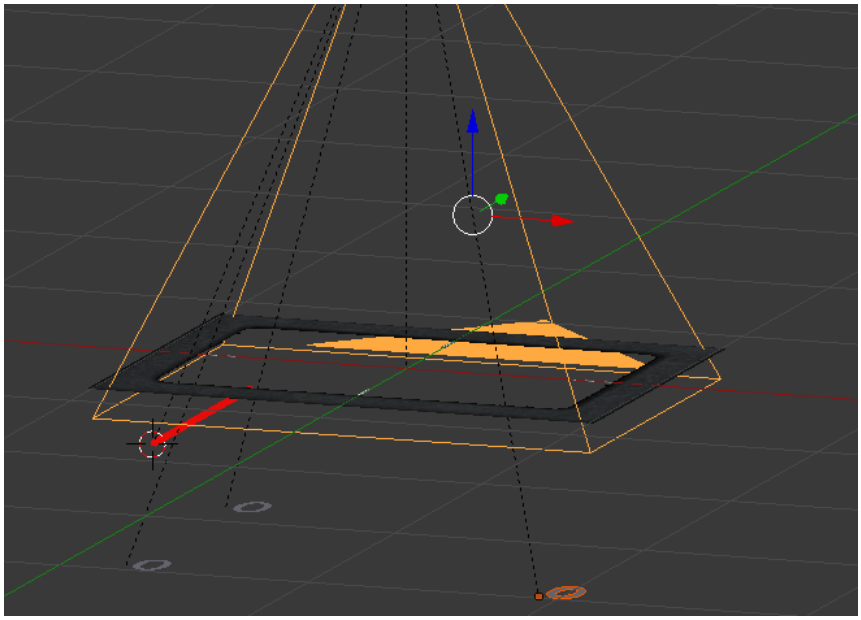
У меня размер текстового объекта Size получился 0.180 . Не забываем давать внятные названия текстовых объектов. Желательно по назначению. Затем просто дублируем текстовый объект и размещаем там, где нам необходимо. После этого создадим объект «куб» и разместим его слева, окрасив в красный цвет. У меня размеры получились такие (x—0.010 , y—0.560 , z—0.010). Обязательно переместим центр куба вниз:



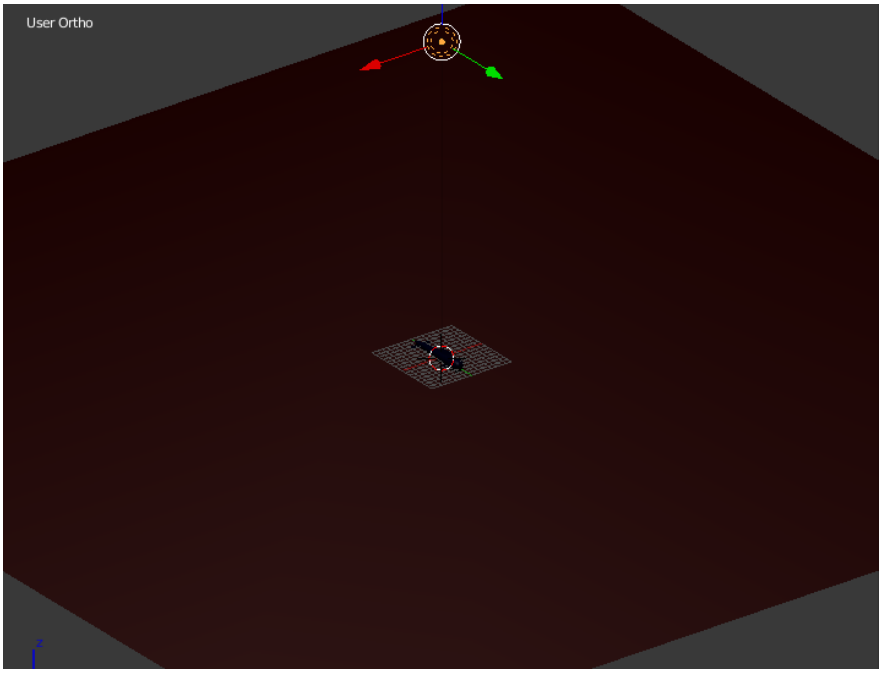
Если после нажатия лат. «P» цифры выглядят не очень привлекательно, то есть смысл выделить все текстовые объекты, переместить их вперёд (дальше от камеры) и увеличить размер:



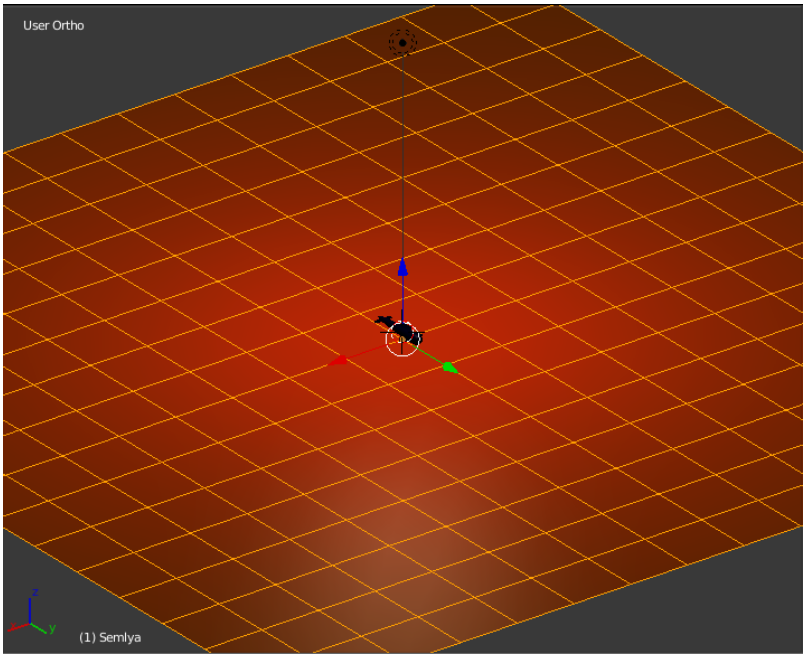
Теперь по очереди будем выделять каждый текстовый объект и соеденять их с камерой по типу «Потомок – Родитель». Так же поступим с кубом (шкалой) и с картинкой (планом) прицела. Это необходимо для совместного их перемещения. Т.е. когда мы перемещаем камеру, то все прикреплённые объекты перемещаются вместе с ней:

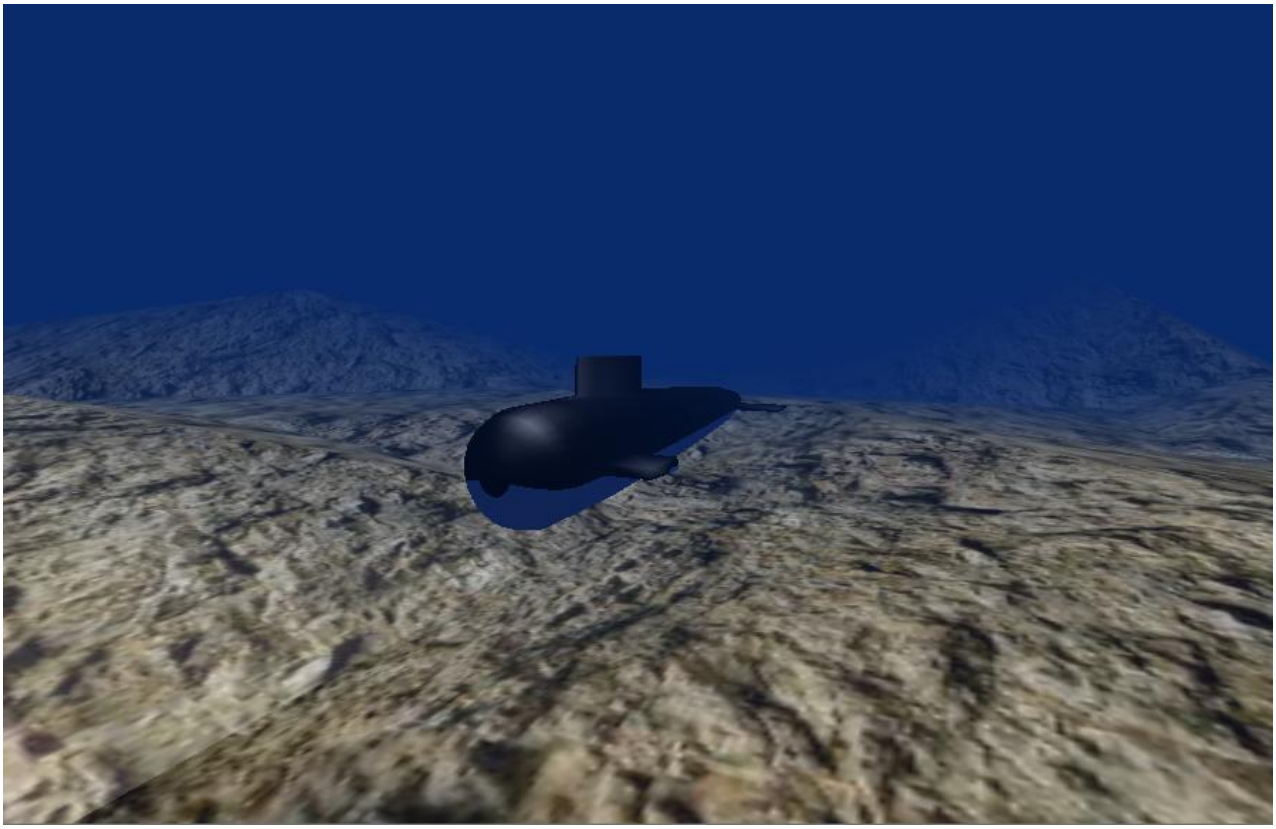


Снова перейдём на первый слой с подводной лодкой и создадим план – дно. Размеры произвольны, но мы ведь будем делать из него ландшафт, а значит разбивать на сектора командой Subdivide. Соответственно вырастет количество вершин. Поэтому не стоит делать план дна слишком большим. Теперь настроим освещение. Поставим наш свет по центру плана и поднимем его повыше:

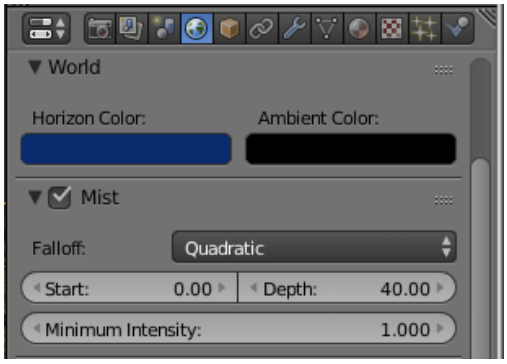


При этом размеры плана у меня получились 100 на 100 блендер-единиц. Высота света по Z где-то 57, а дистанцию для нормального освещения я увеличил до 100. Теперь осталось разбить план на ячейки, назначить прозрачный материал и наложить текстуру. Размер ячейки напрямую зависти от размера рисунка текстуры. Я не стал делать сложное текстурирование и упростил всё до максимума. Текстура должна находиться в папке для ресурсов, которую обязательно нужно создать рядом с файлом проекта. У меня это папка **Res_Submarin.** Там же лежит текстура картинки перископа, и туда же мы поместим все звуки для игры. Далее вид после разбивки на секции и после текстурирования и поднятия точек рельефа:

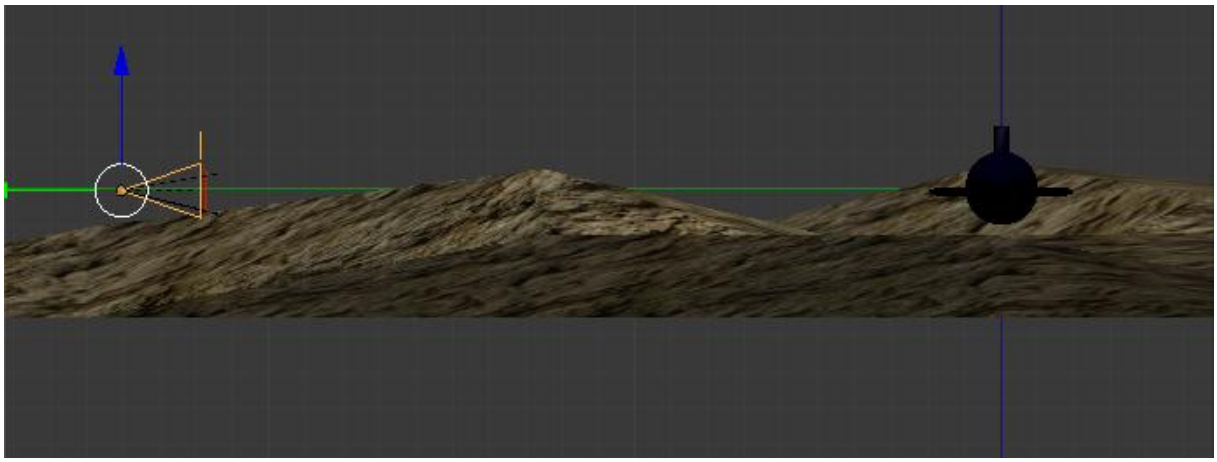




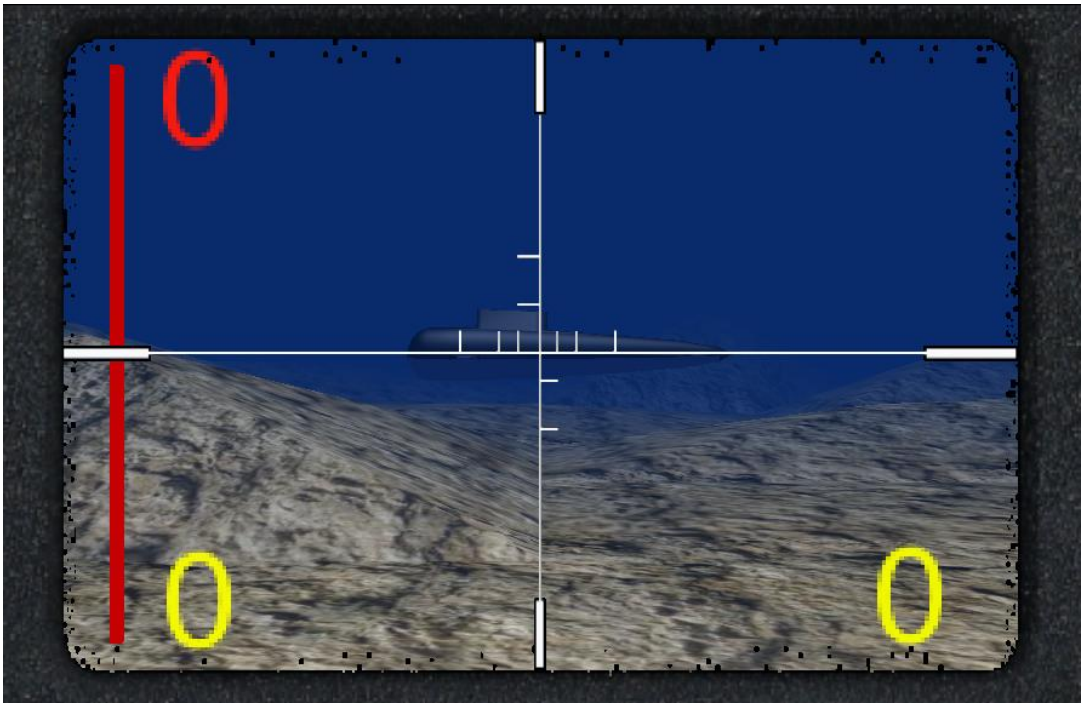
Для того, чтобы прозрачный материал (и вообще любой) хорошо реагировал на туман, не забываем ставить галочки в Use Mist и Sky. Если вы заметили, моя подводная лодка без хвостового вертикального киля ☺. В реальности, конечно, он присутствует ☺. Но не будьте строги, ведь это лишь вариант модели. Вы можете сделать по своему ☺. Ниже вариант моих настроек «тумана» и дисплея:



А сейчас включаем одновременно два слоя и, повернув камеру на 90 градусов по X (и на 180 по Z), размещаем её на комфортном для нас расстоянии. Не очень далеко от подлодки, но и не очень близко. Возможно, что дно придётся немного опустить. Развернём лодку по Z, чтоб видеть её бок:



Лодка должна занимать процентов 25 ширины экрана. Если, после этого её станет плохо заметно, уменьшим интенсивность тумана (т.е. увеличим Depth до 80). Позже, когда мы будем настраивать пуск торпеды, мы ещё не раз будем поворачивать камеру, чтобы совместить прицел с точкой попадания. А пока всё должно выглядеть приблизительно так:



Наконец, мы подходим к настройке логики игры. Как вы понимаете, это самый ответственный момент, поскольку и делает игру игрой. Поэтому сразу наметим, что необходимо будет сделать.

1. Лодка должна перемещаться справа на лево.
2. Выходя за пределы видимости она должна исчезать и снова появляться справа
3. Если происходит попадание, то она должна исчезать (появляясь справа), а счётчик попаданий должен прирасти на единицу.

Это только для лодки и без звуков. Теперь наметим действия для торпеды.

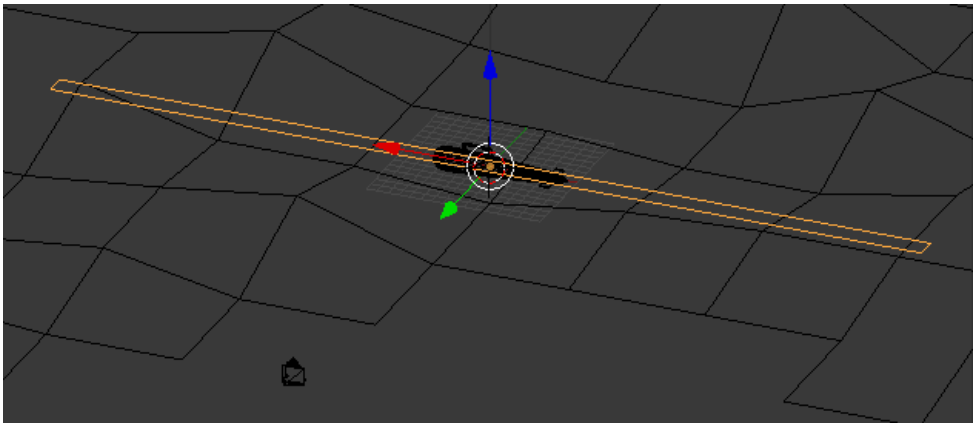
1. Торпеда должна перемещаться вперёд по нажатию клавиши. При этом счётчик торпед уменьшается на единицу (по умолчанию их 10).
2. Выходя за пределы видимости, торпеда должна исчезать и появляться в начале.
3. Соприкасаясь с лодкой, торпеда должна исчезать и появляться в начале.

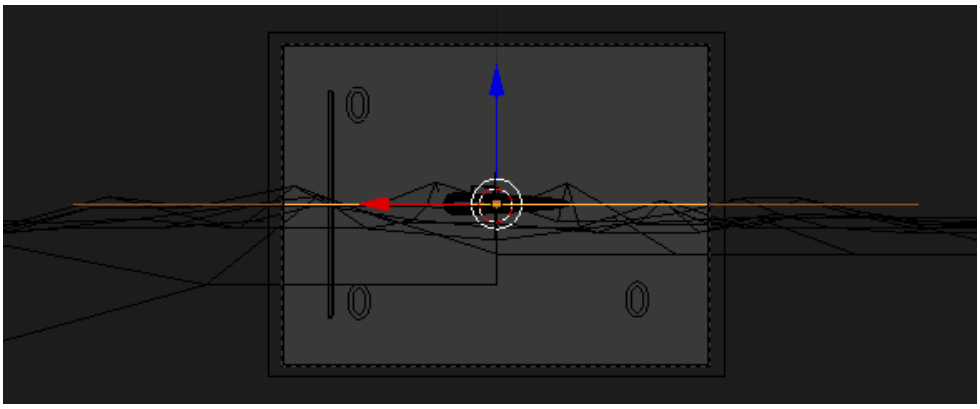
И, наконец, действия для эхолота.

1. Эхолот должен отслеживать расстояние до лодки, и отображать его в цифровом и графическом виде (т.е. должна изменяться высота красной полосы).

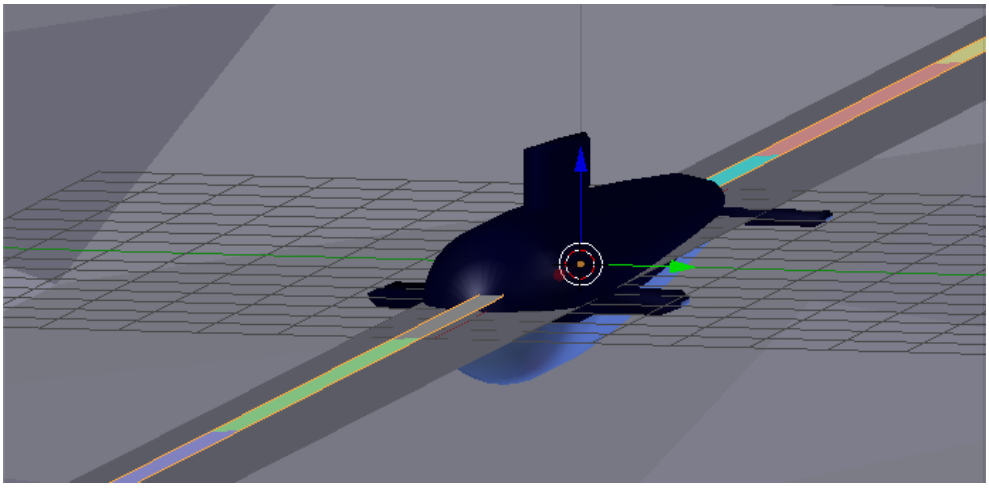
Вот сколько всего нам необходимо настроить. После отладки мы подключим звуки. Ну, и в завершении, сделаем меню начала игры и окончания игры (проигрыша и выигрыша). Если вы заметили, работа предстоит серьёзная. Поэтому, не стремитесь всё сделать за один день. Гораздо лучше, если вы будете тратить на проект по три часа в день. Ведь свежая голова гораздо лучше перегруженной ☺. Главное не забрасывать проект и довести его до конца.

Самый простой способ заставить двигаться лодку, это назначить ей актуатор движения Motion. Однако, мы возьмём чуть более сложный вариант. Создадим путь для лодки, строго определённой длины, с началом и целью в конце. Для этого по центру лодки создаём план и раздвигаем его по длине таким образом, чтобы его края выходили за пределы видимости камеры. Длина плана у меня получилась 40 blend единиц:

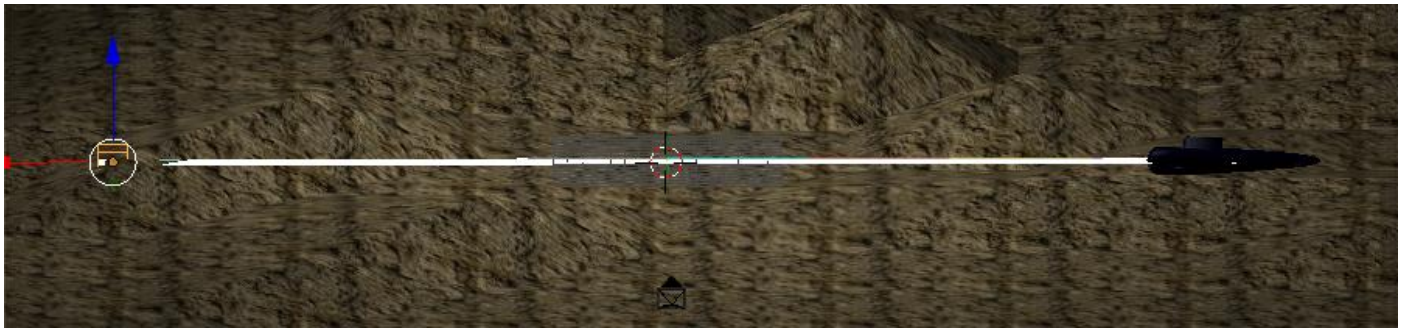




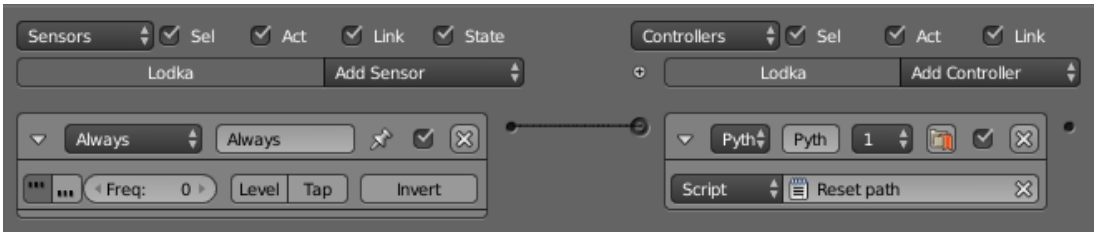
Не снимая выделения с плана пути, переходим на вкладку Scene. Ищем Navigation Mesh и жмём большую кнопку Build Navigation Mesh. Blender пересчитает план и сделает из него путь автоматический. При этом над планом пути появится ещё один разноцветный план поменьше:



Теперь сдвинем нашу лодку по горизонтальной оси в начало пути (вправо). А в конце пути (слева) вставим пустышку куба Add -> Empty -> Cube. Она и будет нашей целью, к которой будет двигаться лодка:



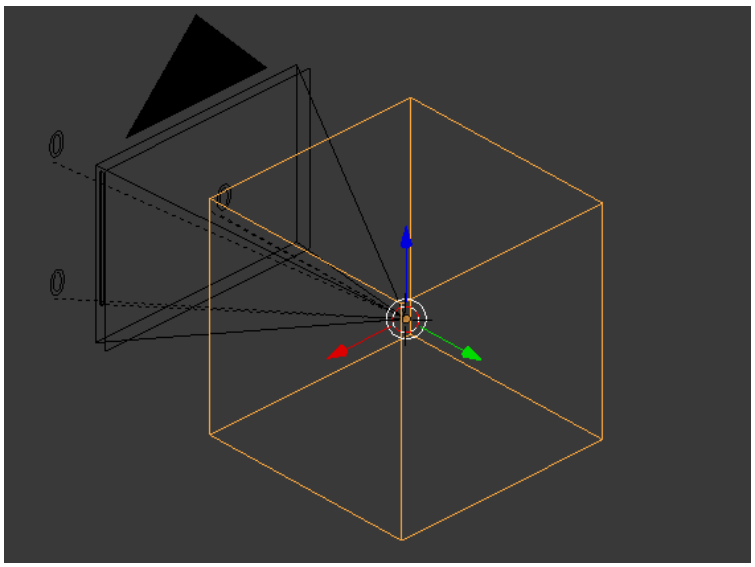
Осталось прописать логику движения. Выделим лодку и переименуем её в Lodka. А пустышку-цель переименуем в Cel. Заметим координаты центра цели. Нас интересует координата X, поскольку другие не так важны. Где-то 39 единиц. А у лодки эта же координата будет -39. Выделяем лодку и переходим в Game Logic. Далее, создаём сенсор Always и контролёр Python, в который мы вставим скрипт управления. Не бойтесь скриптов! Спасибо разработчикам Blender за Python !!! Он очень сильно облегчает жизнь и существенно ускоряет игровую логику. Выглядит это так:



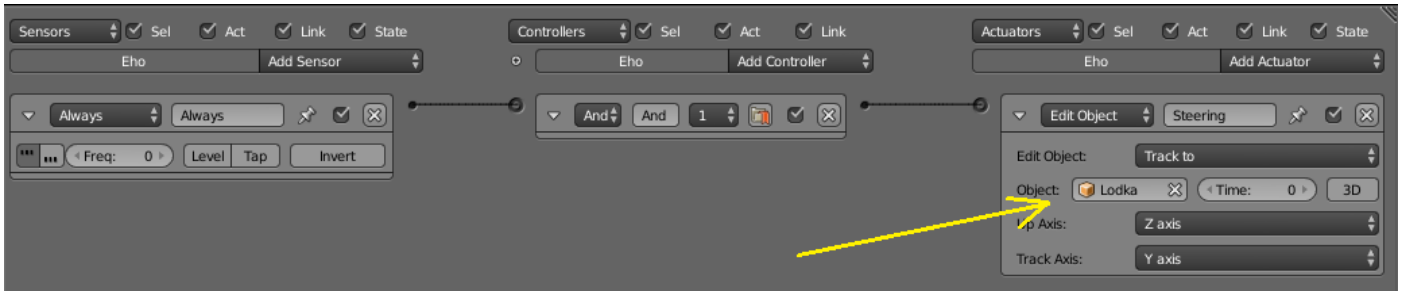
Скрипт управления лодкой назовём Reset path. Вот сам скрипт:

```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
if obj.worldPosition.x > -39 or obj.worldPosition.x == -39: # если координата X больше либо равна -39
   obj.worldPosition.x += 0.1   # то с каждым шагом перемещаем лодку по X на 0.1
if obj.worldPosition.x > 39 or obj.worldPosition.x == 39: # если координата X больше либо равна 39
   obj.worldPosition.x = -39   # то возвращаем лодку в начало
   obj.orientation = ([1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]) # с первоначальной ориентацией
```

Обязательно примените к лодке Object -> Apply -> Rotation. Чтобы она не плавала боком ☺.
Все строки закомментированы, поэтому вы сами разберётесь, что к чему. Не забывайте периодический сохранять свой проект, чтобы не пришлось всю работу делать заново ☺. Перейдите на вид с камеры, нажмите лат. «P» и убедитесь, что лодка проплывает справа на лево. Ура! Начало уже есть ! Если вы считаете, что она плывёт слишком быстро, то уменьшите в скрипте число приращения. Вместо 0.1 попробуйте поставить 0.07, например. Или ещё меньше…
Теперь давайте напишем скрипт слежения эхолота за объектом. Он должен нам показывать расстояние. В принципе, эхолот нам нужен только для красоты, чтобы добавить динамики к игре. Но всё же… Создадим за камерой ещё один пустой объект и назовём его Eho:



Перейдём в игровую логику. Выделим Eho и заставим его следить за лодкой. Вставим Always и контролёр And. Актуатор же возьмём Edit Object с аргументом Track to:



Если мы включим игру, то не увидим, как пустышка следит за лодкой. Ведь это пустышка. Но поверьте, что это так. Чтобы убедиться в этом, давайте вставим контролёр питона (добавим под And) и пропишем скрипт для текстового объекта эхолота. Это у нас первый объект Text. Мы даже название менять не будем. А сам скрипт так и назовём Eholot .

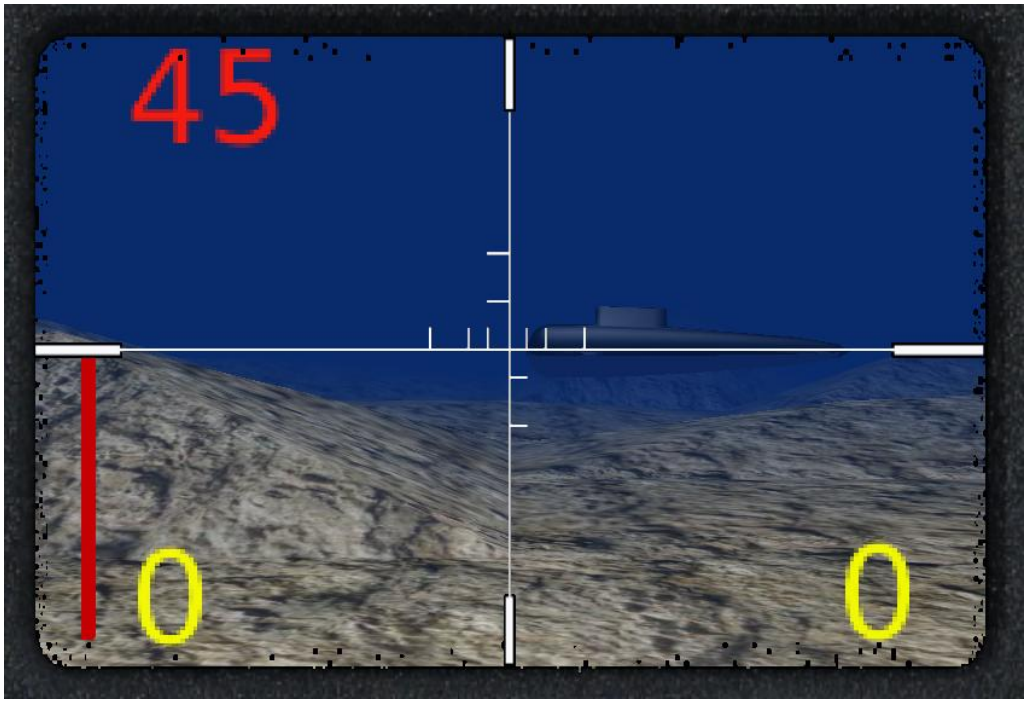
Для измерения расстояния от одного объекта до другого в BGE есть интересная (и полезная) функция с названием *getDistanceTo* . Вот её мы и применим:

```
import bge

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

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

box1 = objList["Lodka"]   # получаем объект лодки
distance = obj.getDistanceTo(box1) # узнаем дистанцию и присваиваем значение переменной
distance = int(distance)             # переводим в целые числа

text = objList["Text"]          # получаем текстовый объект
text.text = str(distance)          # конвертируем цифры в строку и отображаем
```

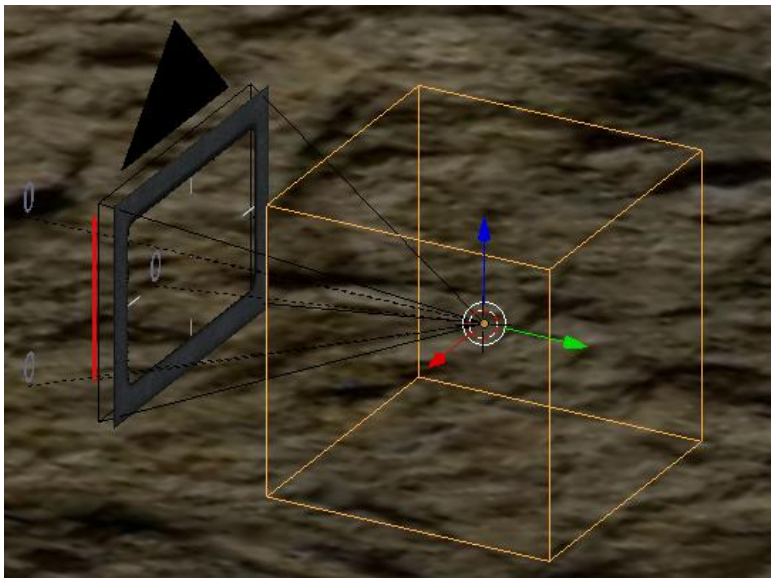
Собственно, чудо уже можно посмотреть, запустив игру. Но у нас ещё и графическая полоска должна изменять размер. Поэтому добавим в скрипт снизу ещё пару строк:

```
shkala = objList["Shkala"]  # получить объект шкалы
shkala.localScale.y = distance/158  # установить длину шкалы по вертикали
```

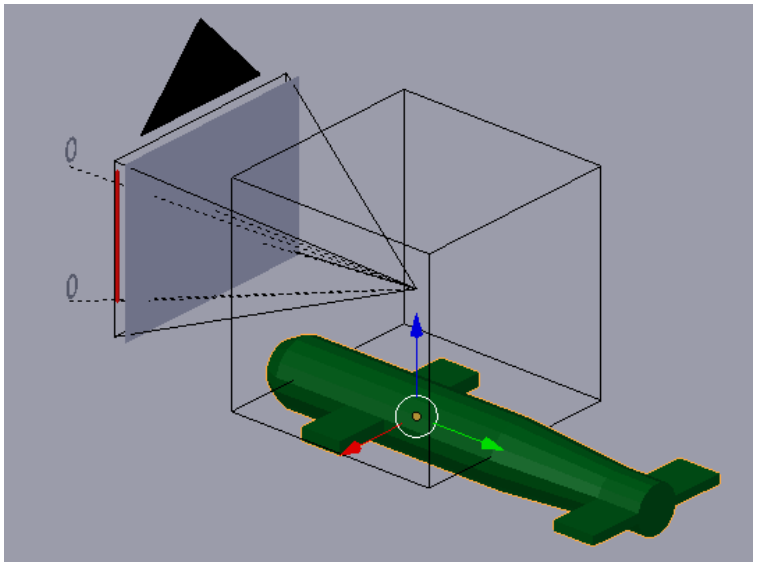
Число 158 найдено сугубо опытным путём и в каждой игре оно может быть разным. Вот собственно мы и разобрались с отображением расстояния эхолотом. Далее нас ожидает самое сложное, это запуск торпед. Ведь без них мы можем лишь наслаждаться мультипликацией проплывающей подводной лодки:



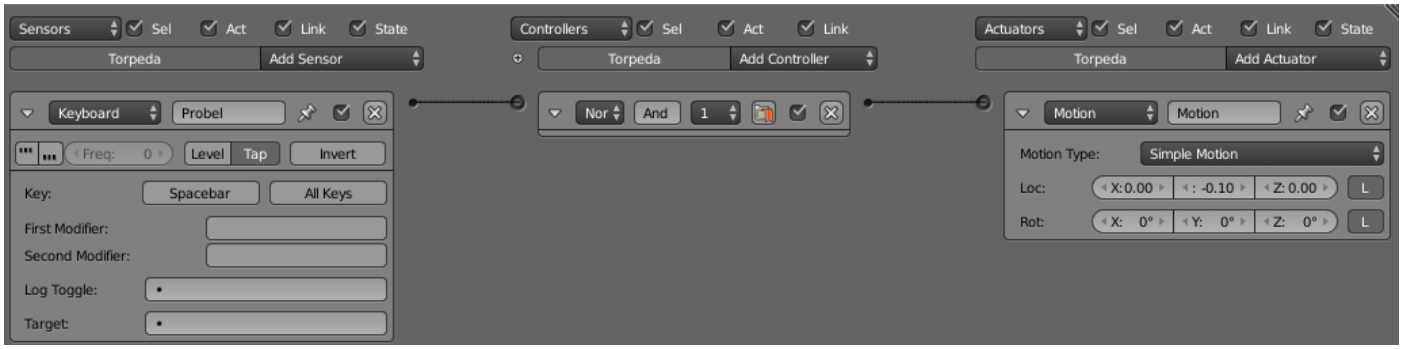
Торпеду начнём делать прямо в центре пустышки эхолота. Таким образом, она окажется точно по центру камеры и нам останется лишь сместить её чуть ниже:



Есть несколько способов моделирования. Вы можете выбрать любой, а я взял цилиндр и сделал заготовку торпеды. Обязательно применим Object -> Apply -> Rotation. Назначим ей материал.

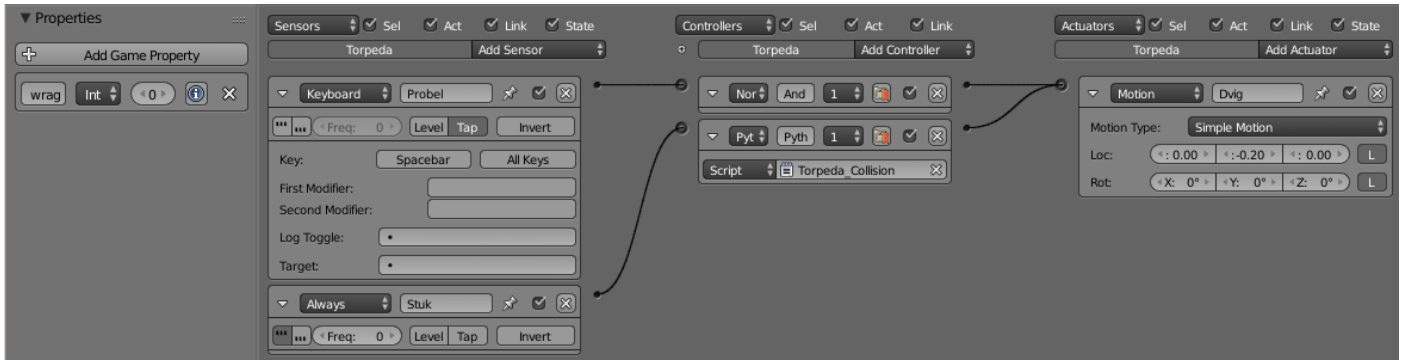


Теперь выделяем торпеду и переходим в логику. Вставляем сенсор нажатия клавиши пробела и контролёр Nor. В актуаторе движения выставляем отрицательное значение по оси Y. Всё это временно и нужно для того, чтобы посмотреть, будет ли торпеда попадать в лодку. Если нет, то возможно придётся немного изменить угол наклона торпеды.



В моём случае, когда я посмотрел на движение торпеды, оказалось что она проходит под днищем лодки. Поэтому я немного изменил угол Rotation X на -2 градуса. Осталось настроить столкновения.

Как же нам отследить столкновения торпеды с лодкой ? Самый простой путь это назначить столкновения по материалам. Как только торпеда будет соприкасаться с определённым материалом, сенсор Collision будет срабатывать. Но вот в чём загвоздка – необходимо чтобы хотя бы один объект был динамическим. У нас же два объекта типа Static. Статические объекты легче перемещать с одинаковой скоростью, и у них нет инерции при ударах друг о друга. Как же быть ? Честно говоря, я сам был в затруднении и перепробовал массу вариантов. И, наконец, меня посетила мысль. А что если на протяжении всего выстрела торпеда будет отслеживать расстояние до лодки ? И при определённом значении переместится в начало, зафиксировав попадание в цифрах ? Ведь это проще всего ! И главное – не нужно мудрить с физикой свойств ☺. И так, выделяем торпеду и делаем как у меня:



А вот скрипт Torpeda_Collision

```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects          # получить список объектов

act = cont.actuators["Dvig"] # получаем актуатор движения

text3 = objList["Text.002"] # получаем текстовый объект №3
box1 = objList["Lodka"]    # получаем объект лодки
distance = obj.getDistanceTo(box1) # узнаем дистанцию и присваиваем значение переменной


if distance < 4:   # Если от торпеды до лодки меньше 4 единиц
   cont.deactivate(act)     # Останавливаем актуатор движения

   obj.worldPosition.y = 45   # возвращаем торпеду в начало
   obj.worldPosition.x = 0
   obj.worldPosition.z = -1.4976

   obj["wrag"] += 1   # присваиваем 1 с каждым разом
   wrag = obj["wrag"]
   text3.text = str(wrag)  # Выводим на экран число убитых лодок
```

Что же произойдёт ? По нажатию клавиши пробела торпеда начинает двигаться. Если расстояние от торпеды до лодки меньше 4 единиц, то это равносильно попаданию. Тогда движение торпеды прекращается и она резко возвращается в исходное положение. При этом срабатывает счётчик попаданий. Ничего сложного ☺.

А что же с лодкой? Лодка должна исчезнуть при касании торпеды и переместиться в начало ☺. Для этого вносим некоторые дополнения в скрипт Torpeda_Collisio . Добавляем всего две строчки :

```
box1.worldPosition.x = -39   # то возвращаем лодку в начало
box1.orientation = ([1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]) # с первоначальной ориентацией
```

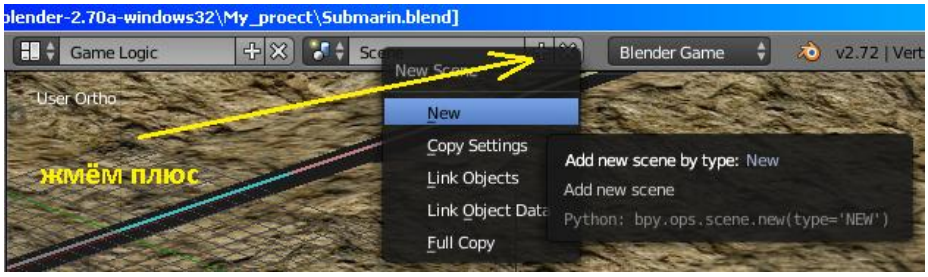
Вот весь скрипт:

```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов

act = cont.actuators["Dvig"] # получаем актуатор движения

text3 = objList["Text.002"] # получаем текстовый объект №3
box1 = objList["Lodka"]    # получаем объект лодки
distance = obj.getDistanceTo(box1) # узнаем дистанцию и присваиваем значение переменной


if distance < 4:   # Если от торпеды до лодки меньше 4 единиц
    cont.deactivate(act)      # Останавливаем актуатор движения

    obj.worldPosition.y = 45   # то возвращаем торпеду в начало
    obj.worldPosition.x = 0
    obj.worldPosition.z = -1.4976

    box1.worldPosition.x = -39   # то возвращаем лодку в начало
    box1.orientation = ([1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]) # с первоначальной ориентацией

    obj["wrag"] += 1   # присваиваем 1 с каждым разом
    wrag = obj["wrag"]
    text3.text = str(wrag)  # Выводим на экран число убитых лодок

```

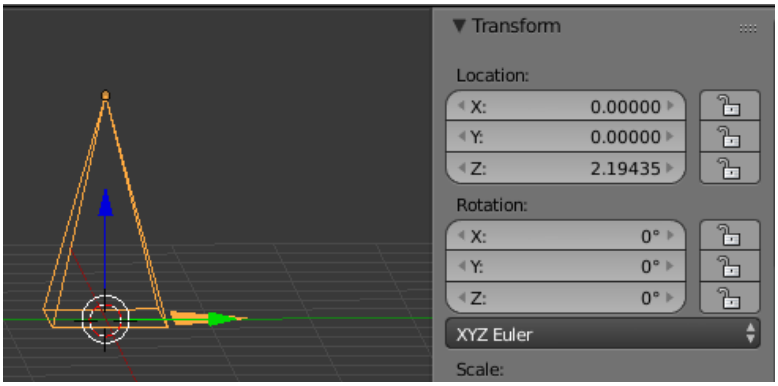
View Text Edit Format Templates Torpeda_Collision Run Script Register Text: Intern

Готово ! Теперь, если мы выстрелим в лодку и попадём, то и торпеда и лодка переместятся в свои начала, а счётчик покажет попадание.

Осталось посчитать торпеды и, когда они закончатся, перейти на другой уровень. Для этого нам нужно создать уровень и назвать его, например, Minimum (т.е. когда кончились торпеды):

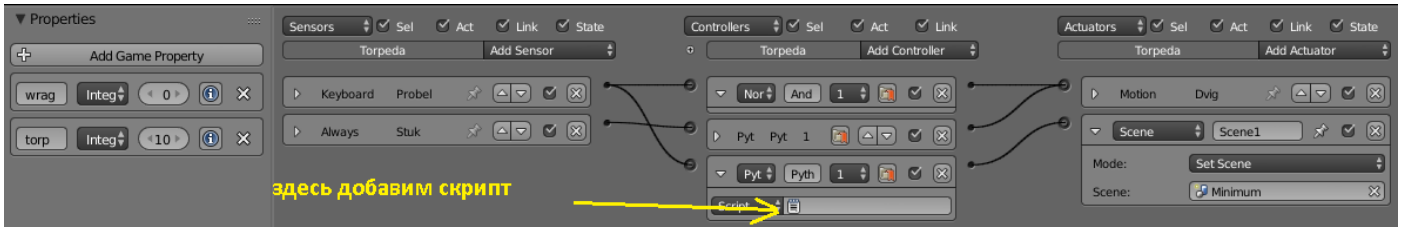


И в нём сразу же создадим камеру, с нулевыми значениями по Rotation :



Для чего ? Дело в том, что при переходе на другой уровень сразу включается вид с камеры. А мы туда поместим план с текстурой сообщения, типа «Вы проиграли !».

Переходим на первоначальную сцену и выделяем торпеду. В логике добавляем контролёр Python (для ещё одного скрипта) и актуатор Scene (для перехода в другую сцену – уровень). Соединяем всё, как на рисунке. В Pyt мы позже вставим новый скрипт, а в Scene вставим новую сцену Minimum. Слева создадим ещё одну переменную torp с числом 10 (это количество доступных выстрелов):



Жмём плюс в скриптах и создаём новый скрипт – Torpedy:


```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects          # получить список объектов

sens = cont.sensors["Probel"] # получаем сенсор
act1 = cont.actuators["Scene1"] # получаем актуатор

text2 = objList["Text.001"] # получаем текстовый объект №2
text3 = objList["Text.002"] # получаем текстовый объект №3

if sens.positive:
   obj["torp"] -= 1   # присваиваем -1 с каждым разом
   torp = obj["torp"]
   text2.text = str(torp)  # Выводим на экран число имеющихся торпед
   if torp < 1 and int(text3.text) < 10: # Если торпед 0 и убитых лодок меньше 10
      cont.activate(act1)   # То переходим на другую сцену - уровень
```

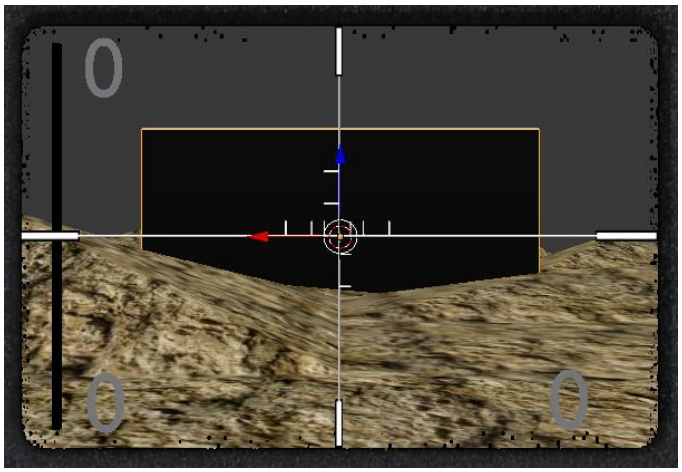
Пояснение! Здесь у нас в одно условие вложено другое. Что означает: «Если нажата клавиша пробел, то уменьшаем переменную на единицу до тех пор, пока торпед не станет меньше одной и при этом убитых лодок будет меньше десяти. Если это условие верно, то переходим на следующий уровень. Если нет – то ничего не делать.»

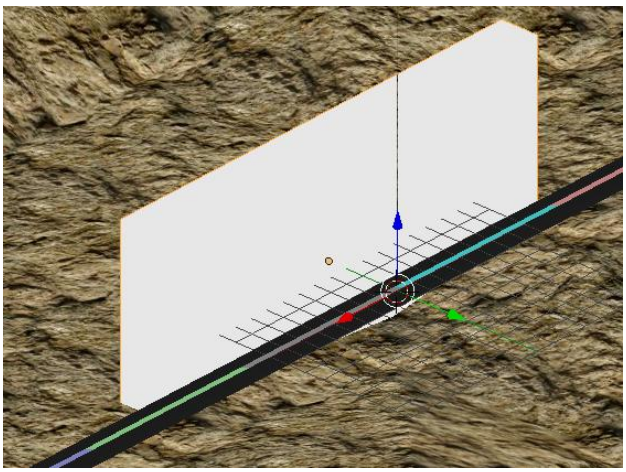
```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов

sens = cont.sensors["Probel"] # получаем сенсор
act1 = cont.actuators["Scene1"] # получаем актуатор

text2 = objList["Text.001"] # получаем текстовый объект №2
text3 = objList["Text.002"] # получаем текстовый объект №3

if sens.positive:
    obj["torp"] -= 1   # присваиваем -1 с каждым разом
    torp = obj["torp"]
    text2.text = str(torp)  # Выводим на экран число имеющихся торпед
    if torp < 1 and int(text3.text) < 10: # Если торпед 0 и убитых лодок меньше 10
        cont.activate(act1)   # То переходим на другую сцену - уровень

```

View Text Edit Format Templates Torpedy Run

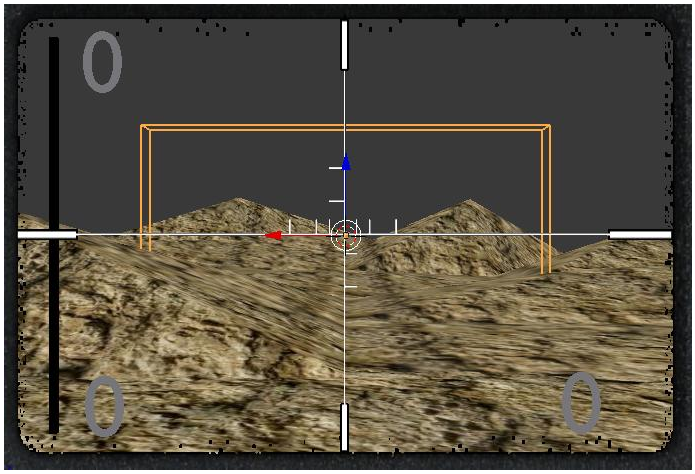
Вставляем скрипт в Pyt и запускаем нашу игру с видом из камеры. Вы можете убедиться, просто кликая на пробел, что по окончании всех выстрелов игра моментально переходит на новую сцену. Правда, после тестирования выяснилось, что в переменной torp количество выстрелов нужно выставлять не 10, а 11. ☺ ☺ ☺ Ну что же, на ошибках учатся… Хотя, если быть честным до конца, то алгоритм подсчёта необходимо немного изменить. Но это наша первая попытка знакомства с технологией постройки игры. Поэтому давайте оставим пока всё как есть. Вы можете спросить, что будет, если сбитых лодок 10 ? Мы просто допишем скрипт и сделаем переход в сцену победителя (которую мы позже обязательно создадим). А сейчас у нас есть ещё одна не решённая проблема. Если мы не попадаем в лодку, то торпеда не исчезает и снова не появляется, а продолжает свой бесконечный полёт в пространстве ☺. Что делать ? Сделаем самое простое – поставим за лодкой прозрачную стену. И напишем скрипт исчезновения торпеды при касании стены.

И так, переходим для удобства в Default и жмём Object -> Snap -> Cursor to Center . Вставляем куб и называем его Stena. Сдвигаем за путь и растягиваем на ширину видимости камеры (хотя можно и меньше). Наконец, назначаем стене прозрачный материал, такой как земле, например:





В GIMP создадим прозрачную текстуру формата PNG и сохраним её в папке ресурсов игры. Затем покроем этой текстурой нашу стену. Стена станет практический прозрачной:



Теперь допишем в скрипте Torpeda_Collision несколько строчек кода. Наша торпеда должна отслеживать расстояние и до стены:

stena = objList["Stena"] # получаем объект стены
distance2 = obj.getDistanceTo(stena) # узнаём дистанцию до стены
if distance2 < 1:   # Если от торпеды до стены меньше 1 единицы
  cont.deactivate(act)   # Останавливаем актуатор движения

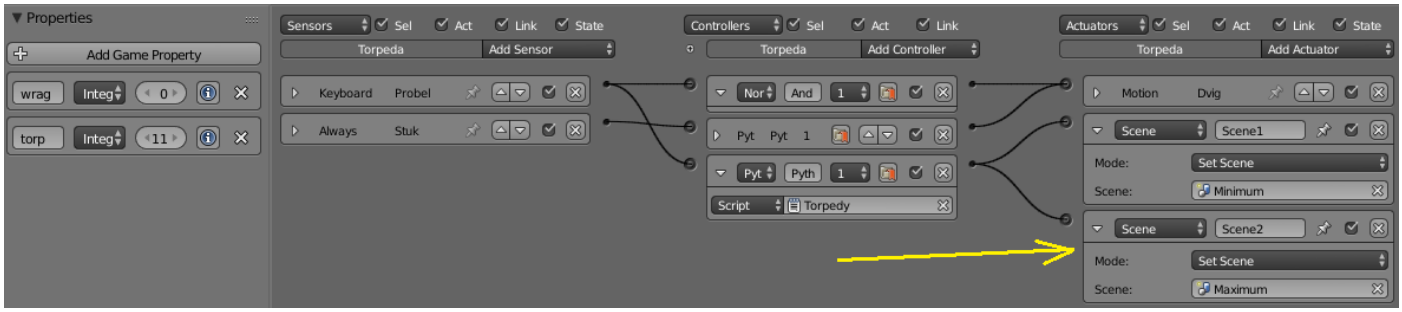
  obj.worldPosition.y = 45   # то возвращаем торпеду в начало
  obj.worldPosition.x = 0
  obj.worldPosition.z = -1.4976

Весь код теперь такой:

```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects            # получить список объектов

act = cont.actuators["Dvig"] # получаем актуатор движения

text3 = objList["Text.002"] # получаем текстовый объект №3
box1 = objList["Lodka"]    # получаем объект лодки
distance = obj.getDistanceTo(box1) # узнаем дистанцию и присваиваем значение переменной

stena = objList["Stena"]  # получаем объект стены
distance2 = obj.getDistanceTo(stena) # узнаём дистанцию до стены

if distance < 4:   # Если от торпеды до лодки меньше 4 единиц
    cont.deactivate(act)      # Останавливаем актуатор движения

    obj.worldPosition.y = 45   # то возвращаем торпеду в начало
    obj.worldPosition.x = 0
    obj.worldPosition.z = -1.4976

    box1.worldPosition.x = -39   # то возвращаем лодку в начало
    box1.orientation = ([1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]) # с первоначальной ориентацией

    obj["wrag"] += 1   # присваиваем 1 с каждым разом
    wrag = obj["wrag"]
    text3.text = str(wrag)  # Выводим на экран число убитых лодок

if distance2 < 1:   # Если от торпеды до стены меньше 1 единицы
    cont.deactivate(act)      # Останавливаем актуатор движения

    obj.worldPosition.y = 45   # то возвращаем торпеду в начало
    obj.worldPosition.x = 0
    obj.worldPosition.z = -1.4976
```

Конечно это не идеальное решение. На самом деле нужно было бы блокировать возможность запуска торпеды, пока существует хотя бы один её экземпляр. Ведь если быстро жать на пробел количество выстрелов стремительно сокращается, но торпеды не вылетают с такой же скоростью. Однако и этот недостаток нам пока придётся стерпеть. Ведь наша задача разобраться с самим принципом построения игры. Игра-то простейшая ☺.

Давайте создадим ещё одну сцену – уровень победителя. Мы перейдём на неё в случае уничтожения 10 ти лодок 10 тью торпедами. Сцену назовём Maximum. Не забудем вставить камеру, как и в сцене Minimum. Вернёмся в первую (основную) сцену с игрой и, перейдя в игровую логику, выделим торпеду. Там нам нужно добавить ещё один актуатор сцены и связать его со скриптом Torpedy . Назовём его Scene2:



Допишем скрипт Torpedy :

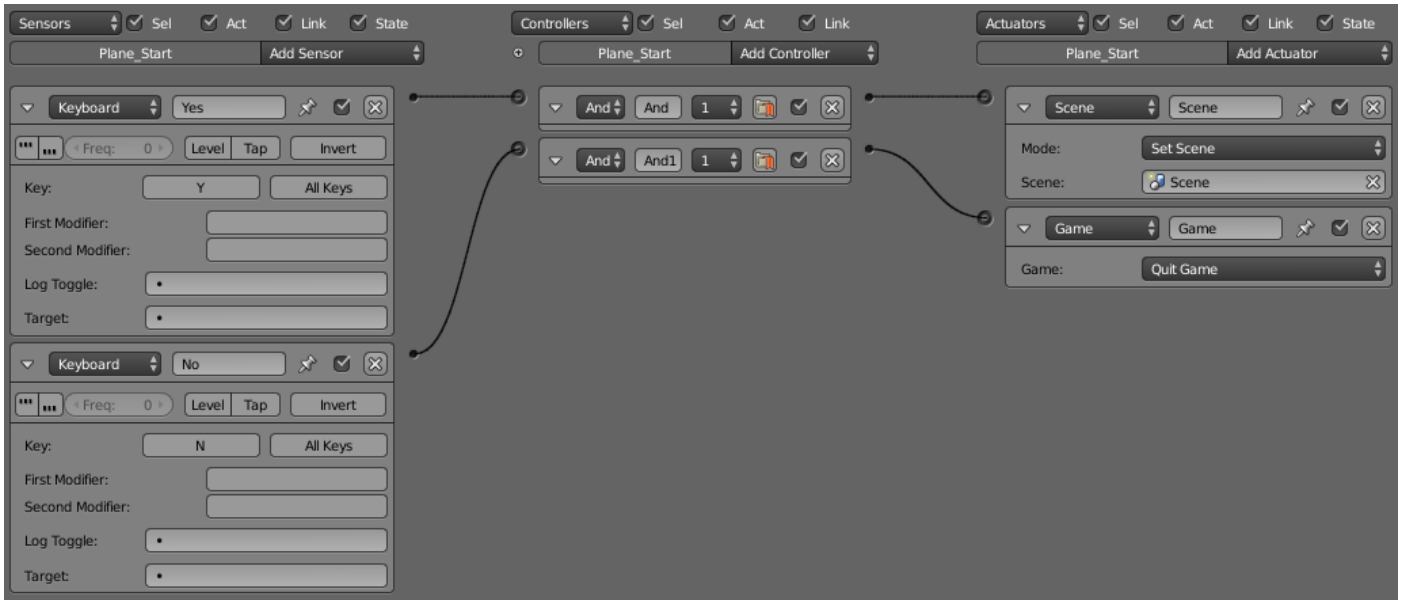
```
cont = bge.logic.getCurrentController() # получить текущий контролер
obj = cont.owner                        # получить выделенный объект
scene = bge.logic.getCurrentScene()# получить текущую сцену
objList = scene.objects             # получить список объектов

sens = cont.sensors["Probel"] # получаем сенсор
act1 = cont.actuators["Scene1"] # получаем актуатор
act2 = cont.actuators["Scene2"] # получаем актуатор

text2 = objList["Text.001"] # получаем текстовый объект №2
text3 = objList["Text.002"] # получаем текстовый объект №3

if sens.positive:
    obj["torp"] -= 1   # присваиваем -1 с каждым разом
    torp = obj["torp"]
    text2.text = str(torp)  # Выводим на экран число имеющихся торпед
    if torp < 1 and int(text3.text) < 10: # Если торпед 0 и убитых лодок меньше 10
        cont.activate(act1)   # То переходим на другую сцену - уровень Minimum
    if torp < 1 and int(text3.text) > 9: # Если торпед 0 и убитых лодок больше 9
        cont.activate(act2)   # То переходим на другую сцену - уровень Maximum

```


Как видно, мы добавили лишь ещё один актуатор (act2) , для управления Scene2 и прописали условие перехода на неё: если торпед меньше 1 , а сбитых лодок больше 9. С логикой самого игрового процесса мы, наконец, закончили. Нам осталось выполнить несколько пунктов:

1. Нарисовать в любом графическом редакторе (рекомендую GIMP) три картинки с надписями «Вы проиграли. Хотите попробовать ещё ? (Y / N)» «Поздравляем! Вы победили! Хотите пройти игру заново ? (Y / N)» «Подводная лодка. Играть (Y / N)». Размер картинки зависит от размера дисплея в настройках игры. У меня это 800*600.

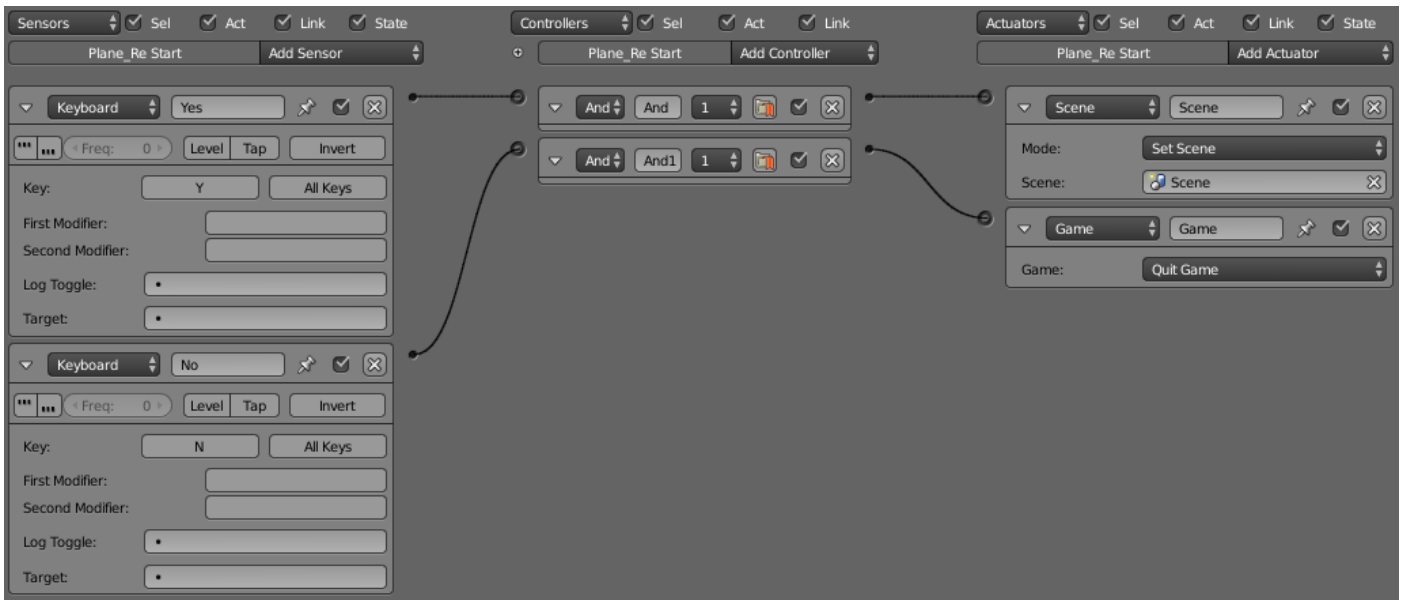
2. Создать стартовую сцену с названием Start . Настроить логику возврата в игру или выхода из неё в сценах Minimum , Maximum , Start.
3. Вставить в игру звуки (самое интересное, потому что оживляет игру!)
4. Создать исполняемый проект с EXE файлом для автономного запуска в Windows.

Картинки готовы. Создаём новую сцену Start. Вставляем камеру как и в сцене Minimum. Переходим в игровой режим (Blender Game). Настраиваем дисплей на 800*600. Вставляем плоскость. Переходим на вид с камеры и растягиваем плоскость по видимому размеру границы камеры (можно даже чуть шире). Назначаем прозрачный материал (он у нас уже есть). Накладываем текстуру картинки Start. Переименуем план в Plane_Start. Переходим в логику, выделяем план и делаем управление:



Если нажата клавиша Y , то срабатывает актуатор Scene и мы переходим в основную сцену игры (т.е. начинаем играть). Если нажата клавиша N , то срабатывает актуатор Game и игра завершается.

Переходим на сцену Minimum и делаем всё тоже, что и выше с картинкой Re_Start:



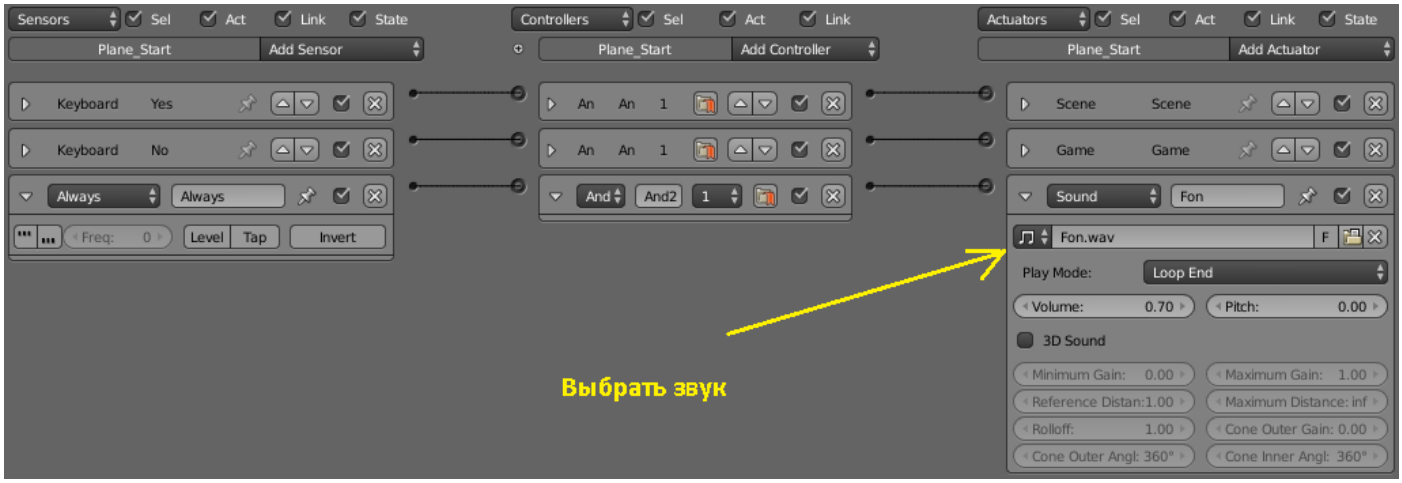
Переходим на сцену Maximum и делаем всё то же, что и выше с картинкой End_Game. Всё тоже самое, поэтому я даже не буду повторяться ☺.

Игра готова. Вы можете убедиться сами. Перейдите в сцену Start и нажмите лат. «P». Поиграйте и попробуйте все режимы.

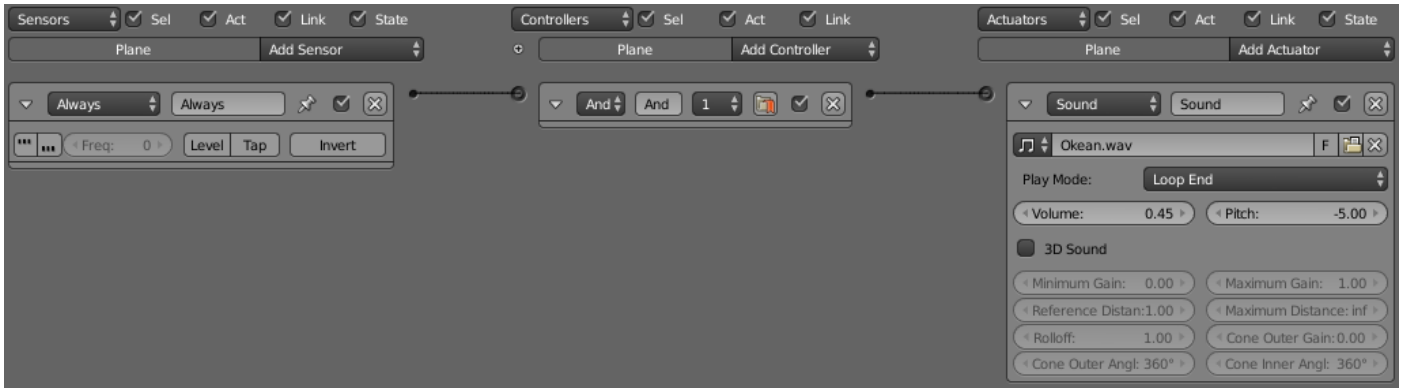
Осталось вставить звуки. Это моя любимая часть ☺.

Звуков в интернете полно. И это прекрасно. Можно найти звуки эхолота, шума воды, взрыва и т.д. Хочу сказать главное – берите звуки в формате Wav. Этот формат BGE обрабатывает без тормозов. Старайтесь искать файлы в 22 кГц. Они практический не отличаются на слух от звуков в 44 кГц, зато меньше нагружают логику. На своём опыте убедился, что звуки формата MP3 начинают заикаться и глючить, особенно если их много. В идеале, лучше всего обрабатывать звуки самим в каком-нибудь звуковом редакторе. Желательно точно обрезать звук (без пауз). Ещё лучше использовать короткие звуки. Когда их нужно удлинить, то лучше их зациклить (повторять).

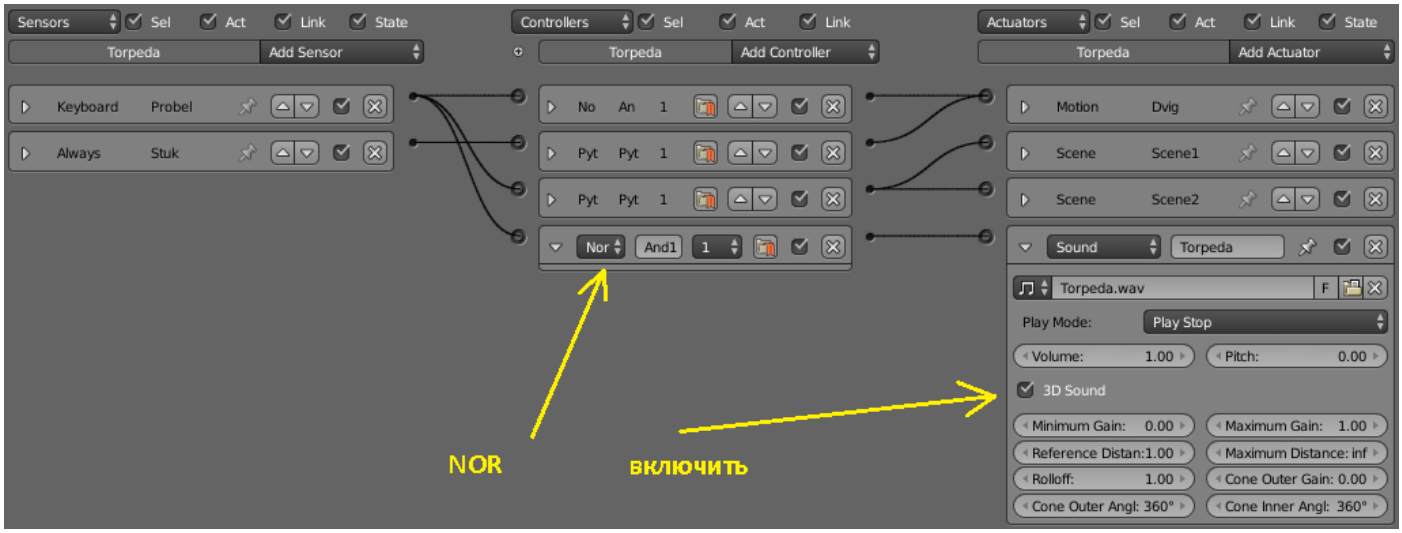
Переходим в сцену Start. Выделяем план и в логике добавляем звук фона:



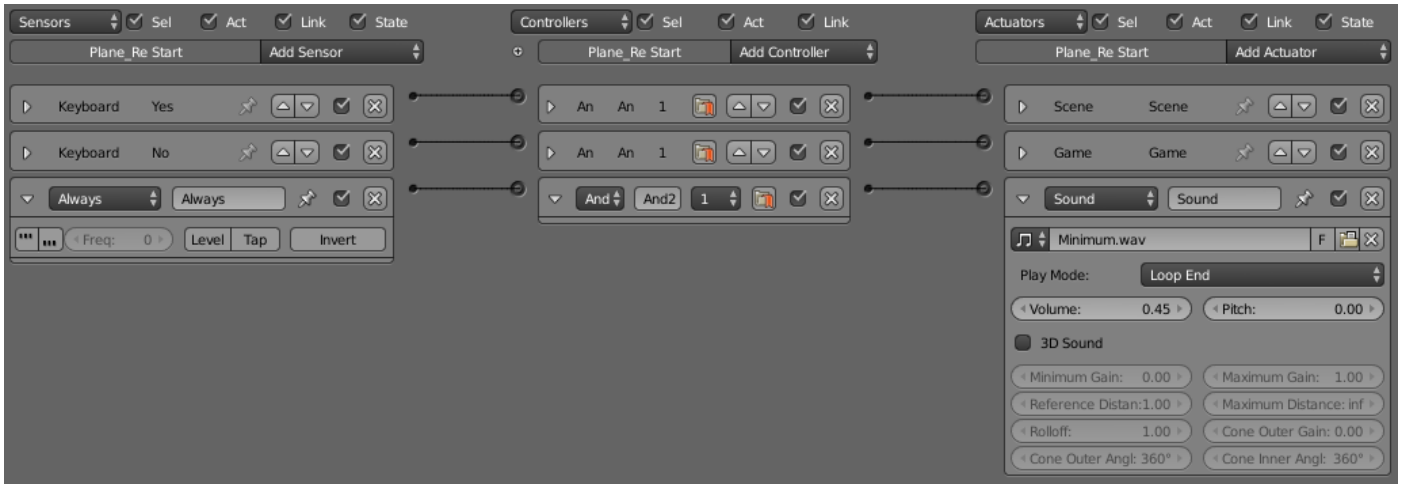
Здесь сложного ничего нет. Тоже делаем и в основной сцене, только в качестве постоянного объекта выделим план картинки перископа и добавим другой звуковой фон:

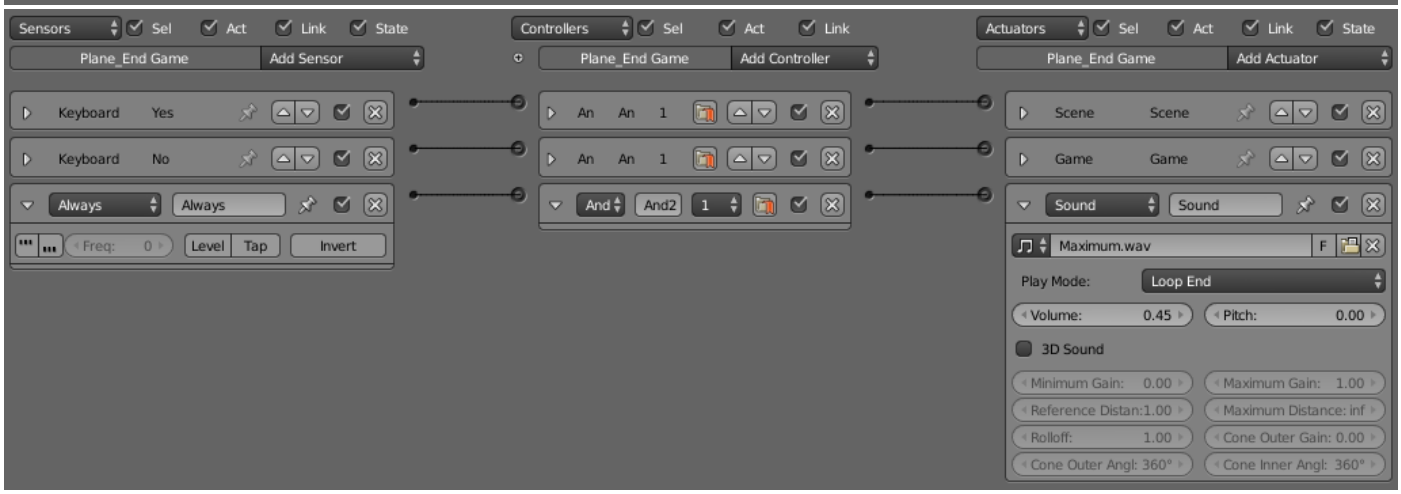


Выделяем торпеду и вставляем звук выхода торпеды. Правда, здесь мы уже подключим 3D. Этот эффект позволяет удаляться звуку вместе с торпедой:



Наконец, переходим по очереди в сцену Minimum и Maximum и так же, как и в сцене Start, вставляем определённые фоновые звуки:





В итоге, переходим в сцену Start и запускаем игру, ведь она готова полностью.

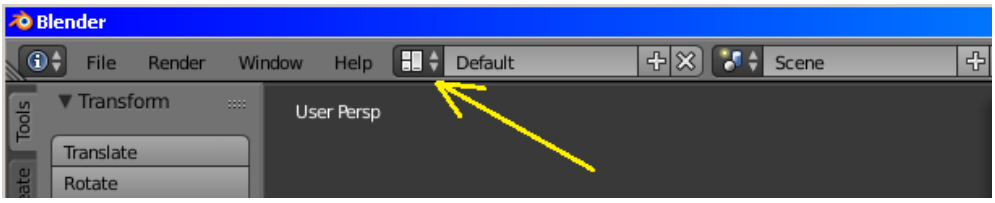
Недостатков в игре много. Например, нет анимации взрыва. Но мы ведь ещё её и не рассматривали в наших уроках. Дойдёт дело и до анимации. Не блокируется кнопка пуска торпеда, пока торпеда идёт к цели, нет таблицы рекордов, столкновения сделаны лишь приблизительно и т.д. ☺ А разве ставилась задача сделать крутую игрушку? Нет. Самое главное было опробовать на практическом примере принципы построения игры. А мы научились многому:

1. Научились делать простейшее меню

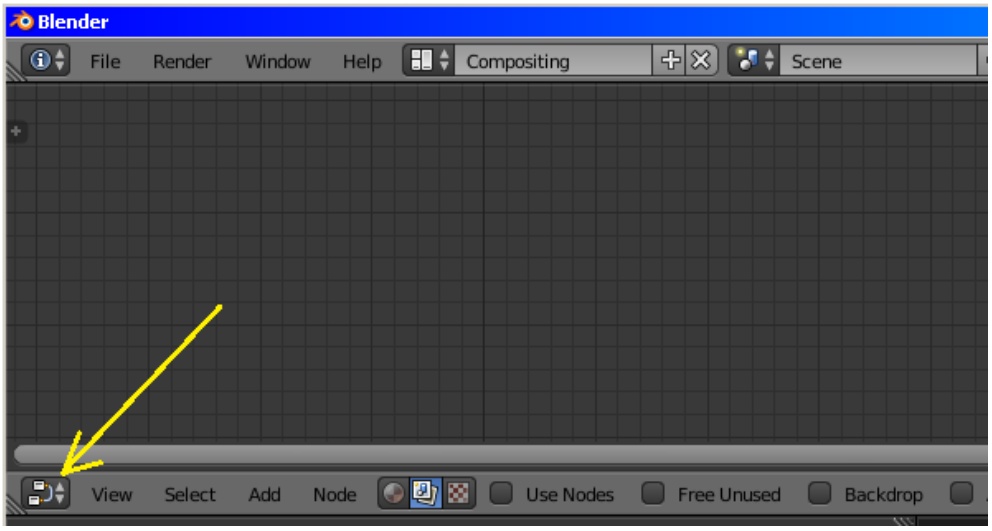
2. Научились делать переходы между сценами
3. Узнали принцип построения элементарной логики поведения объектов при столкновении или обнаружении друг друга.
4. Написали несколько скриптов и научились немного в них разбираться
5. И ещё много разных мелочей…

Поэтому задача данного урока выполнена. Одни только скрипты будут серьёзной помощью в будущем. Ведь их можно модифицировать и применять в других играх. Остаётся создать EXE, для запуска на Windows, поскольку на Линуксах проект можно запускать и так.

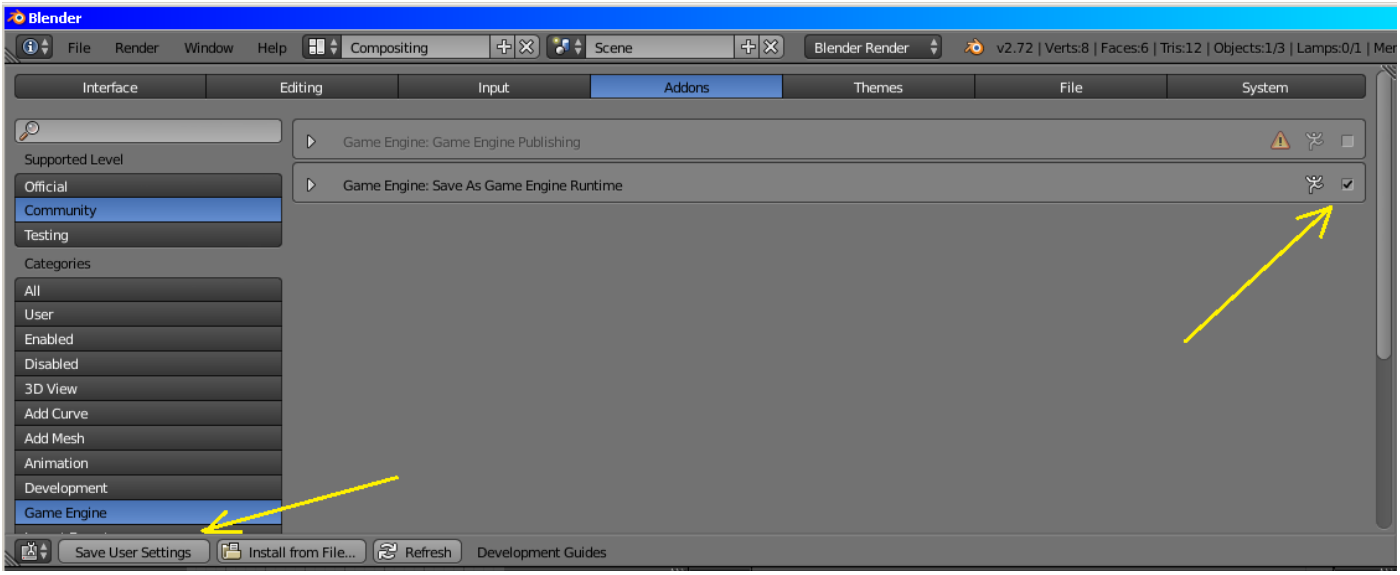
Для этого нужно включить саму возможность экспорта в исполняемый файл. Для своей работы Blender не требует инсталляции в систему. Достаточно скачать ZIP архив с программой, открыть его и запустить файл blender.exe . Такой способ имеет ряд преимуществ, главное из которых это возможность запускать любую версию Blender без инсталляции. Лично я так и делаю. Далее, открываем File -> User Preferences… Вот здесь могут возникнуть проблемы. Дело в том, что Blender явно недолюбливает некоторые видеокарты. У меня стоит Intel . Стоит мне перейти на «User Preferences…» , как программа мгновенно вылетает, не допуская меня ни к каким настройкам. Но добрые люди есть, и в интернете я нашел, как обойти данное неудобство. Так вот пошагово я делаю так. Выбираю вверху Compositing:



И слева выбираю «User Preferences…»:



После открытия окна, выбираем во вкладке Addons (раздел Community) раздел Game Engine. Справа отобразится Game Engine : Save As Game Engine Runtime. Ставим галочку и сохраняем, нажав Save User Settings. После всех манипуляций возвращаем всё назад в Default. Теперь если выбрать File -> Export появится возможность экспорта в EXE с названием Save As Game Engine Runtime. Нажимаем и экспортируем в нужную нам папку. Желательно для игры создать отдельную папку. Ещё один нюанс заключается в том, что Blender экспортирует только файл проекта и файлы Python. Если запустить такую игру, то ничего кроме белого экрана мы не увидим. Поэтому не следует забывать переносить папку с текстурами и звуками в готовую игру. Она должна быть рядом с EXE файлом. В нашем случае это папка **Res_Submarin**.



Вот мы и закончили создание своей первой игрушки. Теперь её можно запустить обычным EXE файлом в любой системе Windows. Кому-то может показаться, что папка с игрой получилась великовата (около 75 мегабайт). Уверяю вас, это очень маленький размер. Ведь сам проект весит в разы меньше. Больше всего места занимают файлы Python и ваши текстуры и звуки. Программы и игры написанные на C++ малы потому, что все необходимые им библиотеки уже находятся в системе Windows. А Blender действует по принципу «Всё своё ношу с собой!». Это очень удобно. И лишних 75 мегабайт при нынешних мощностях компьютеров совершенно не обременяет ☺.

Вот ссылка на раздел сайта с готовым проектом и игрой -> Подводная лодка .

Урок составил Niburiec для сайта http://blender-game.ucoz.ru/ 17.01.2015.