Part two of a series of tutorials for the Blender Game Engine by Smoking_mirror.

Document Description:

This is part two of a set of tutorials, it covers setting up a helicopter type player object for use in the BGE*.

*** Blender Game engine**

Document Aim:

The aim of the tutorials is to introduce new users of the BGE to some of the concepts needed to produce a moderately complex game. The aim of this second tutorial is to show how to set up the logic bricks, objects and python scripts to give complex mouse driven movement to a player object.

Document Goals:

- Describe how to arrange the various meshes, objects and “empties” used by the player.
- Describe how to set up simple actions as well as property driven actions.
- Give some advice about naming conventions.
- Show how to add a pre-written script.
- Give an overview of how the script was written.

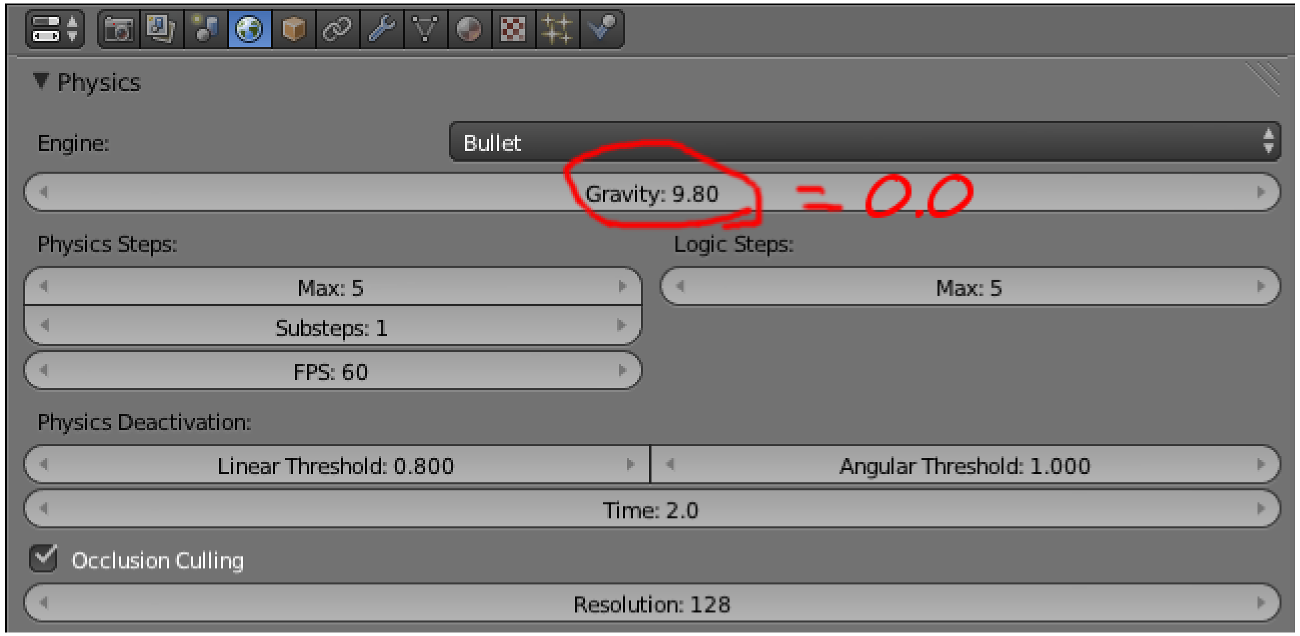
Introduction:

I'm going to go through the steps needed to get your player object moving around the game word. I'll be using quite easy concepts at first and scaling up to more difficult things later. Some steps are place holders for future development, so you shouldn't skip parts although they may seem over complicated. There is a reason for each step. Once you are finished you should have a simple chopper that can fly around the game world.

Document Index:

Part One – Setting up the game world

The first step is to set up our game world. We don't want to use gravity in the game because our physics are going to be very simple, and where gravity needs to be used it will be faked.
So you can tun off gravity by going to the world panel in properties:


Physics

Engine: Bullet

Gravity: 9.80 = 0.0

Physics Steps:
Max: 5
Substeps: 1
FPS: 60

Logic Steps:
Max: 5

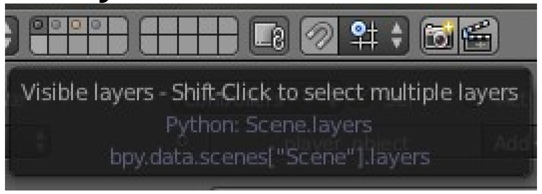
Physics Deactivation:
Linear Threshold: 0.800
Angular Threshold: 1.000
Time: 2.0

✓ Occlusion Culling
Resolution: 128

You can also add a ground object at this point so you can enjoy flying around over it. In a later tutorial we will be covering how to create an environment and more details about level design, but for now a simple textured plane should be enough.

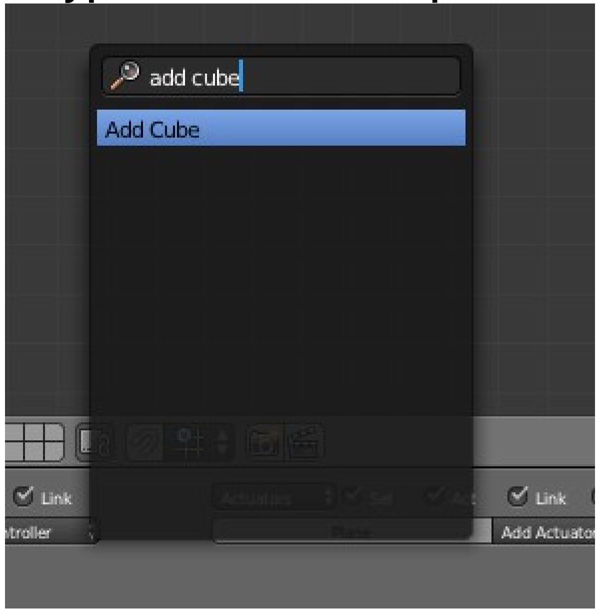
Part Two- Creating the player objects and empties.

So lets go to an empty layer of your project. It's always easier to work on your game objects away from the main layer. Press any number key to move to a different layer.



Visible layers - Shift-Click to select multiple layers
Python: Scene.layers
bpy.data.scenes["Scene"].layers

Next we want to add a cube. This will be the parent element of out game object. This cube will move around the screen, it will be used for collisions, and everything else will be parented to it. To add the cube just press space and type "add cube" (capitals are not needed).

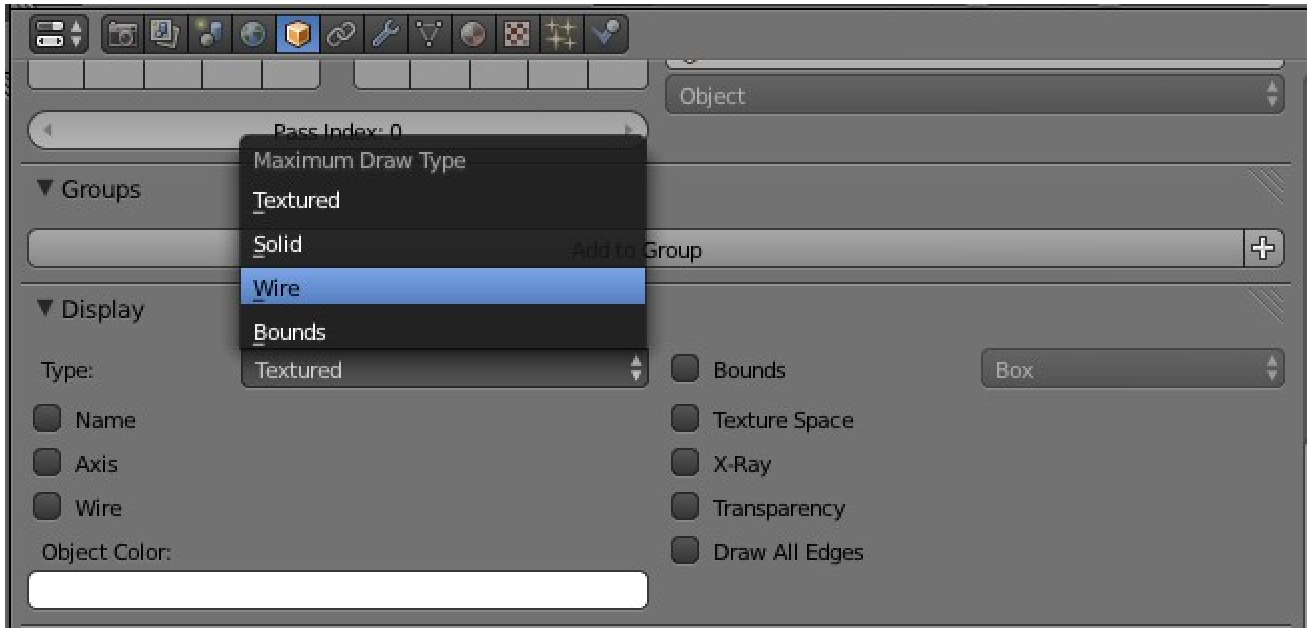


add cube

Add Cube

You will want to go in to edit mode by pressing "tab" and change the shape of the cube so it is about the same size as your chopper.

We don't want to see a solid box in the scene while working on the player object so we will set the cube visibility to wireframe:

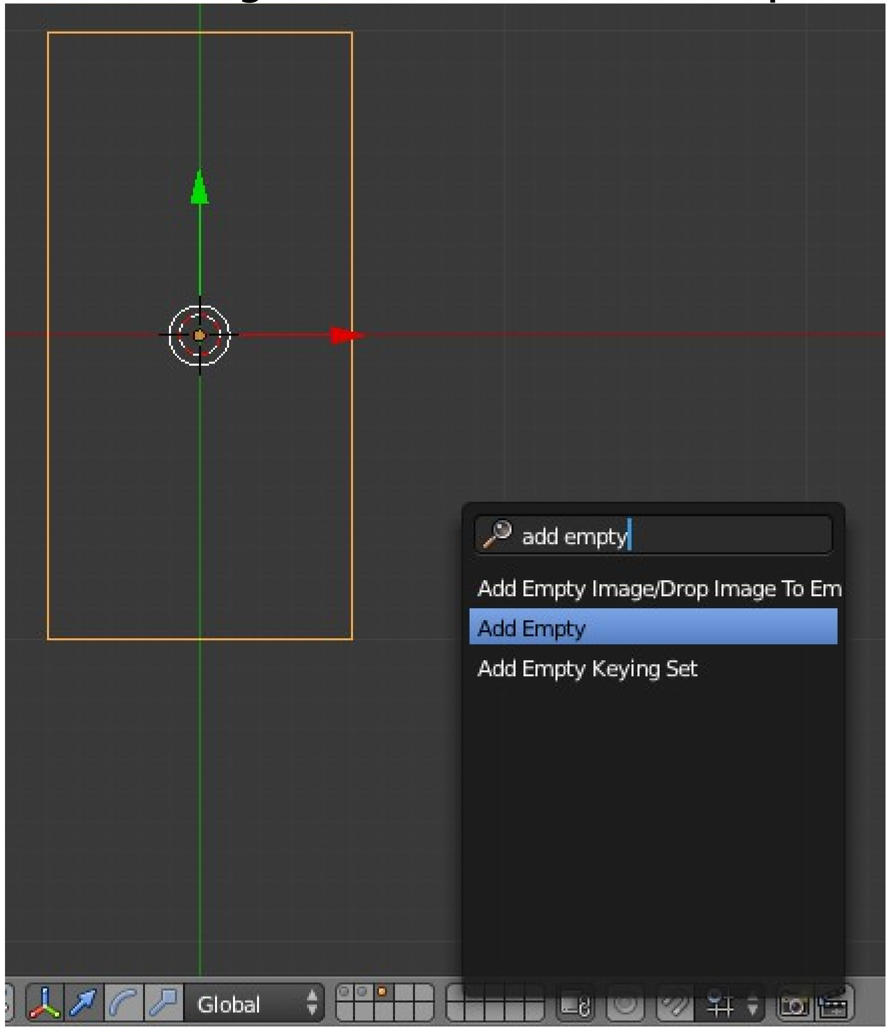


You can also set the object's name, I'd recommend something meaningful like “player_object”. It's good practice to link two word names with an underscore or use capitals on each word and no spaces, like this “PlayerObject”. Choose one naming convention and stick with it, it will make your life much easier in future.

Next we are going to add some empties to handle the roll and pitch of the aircraft.

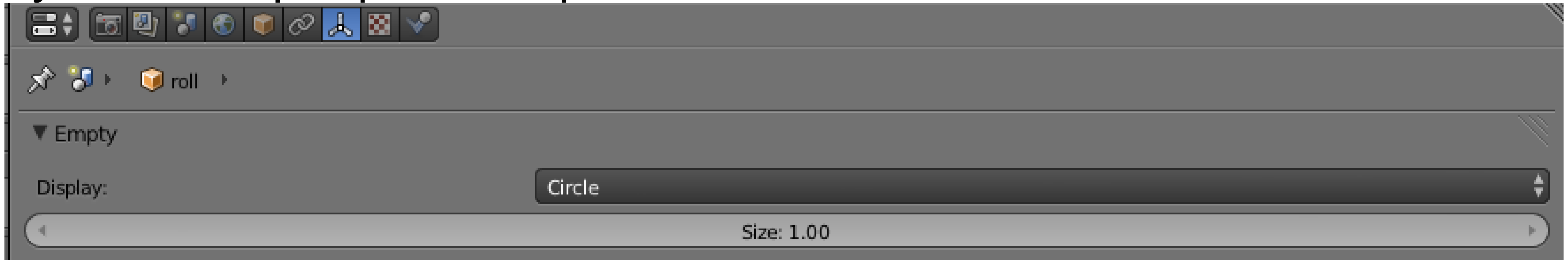
- Roll is when the helicopter leans left or right, like when you are strafing.
- Pitch is when you tilt forward or back, this is how most older helicopters handle forward and backwards movement.

Add the empties and give them meaningful names, like “roll” and “pitch”.



When adding the empty using space, be sure to scroll down past the other items on the search list and highlight “Add Empty”, otherwise nothing will happen when you hit enter.

It is also possible to change the way the empties are displayed in the 3d view. I chose a circle for the roll object and a simple up arrow for pitch.



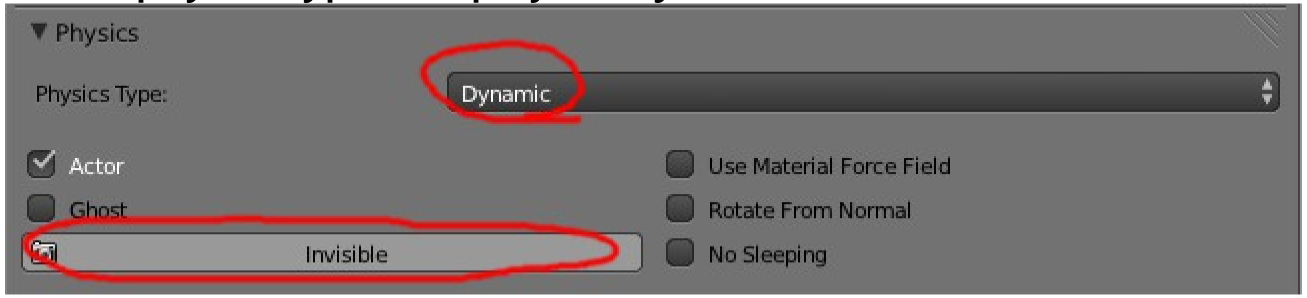
roll

▼ Empty

Display: Circle

Size: 1.00

Now we should set the physics type for “player_object”:

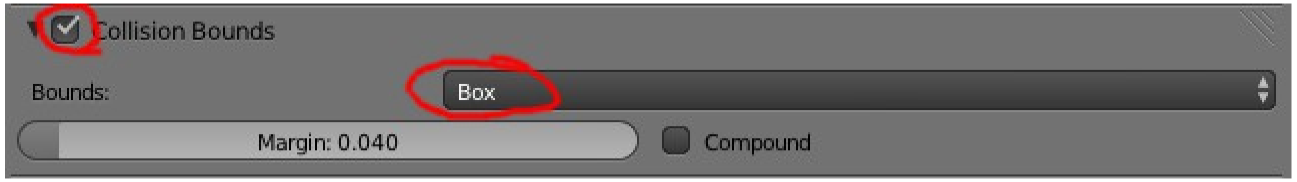


▼ Physics

Physics Type: Dynamic

- [✓] Actor
- [ ] Ghost
- Invisible
- [ ] Use Material Force Field
- [ ] Rotate From Normal
- [ ] No Sleeping

Make it dynamic and invisible. You should also set the collision bounds:



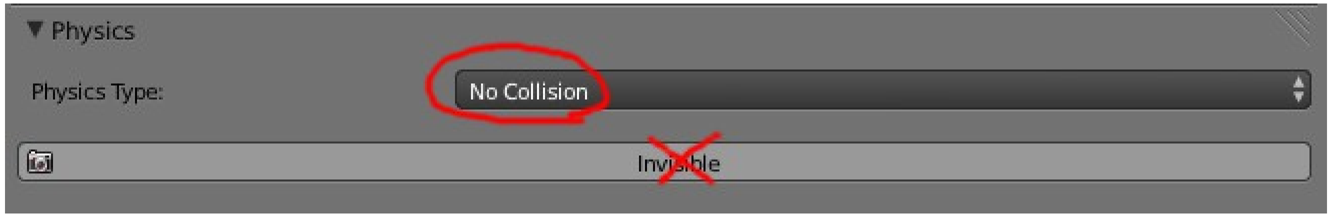
▼ [✓] Collision Bounds

Bounds: Box

Margin: 0.040

- [ ] Compound

All other objects should be set to “no collisions”:

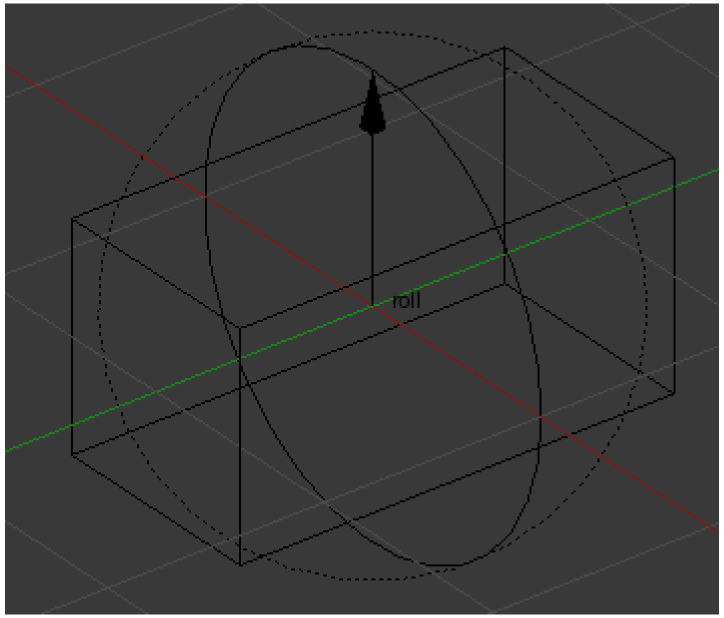


▼ Physics

Physics Type: No Collision

~~Invisible~~

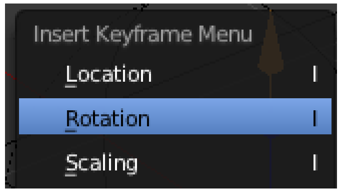
Our object should now look like this:



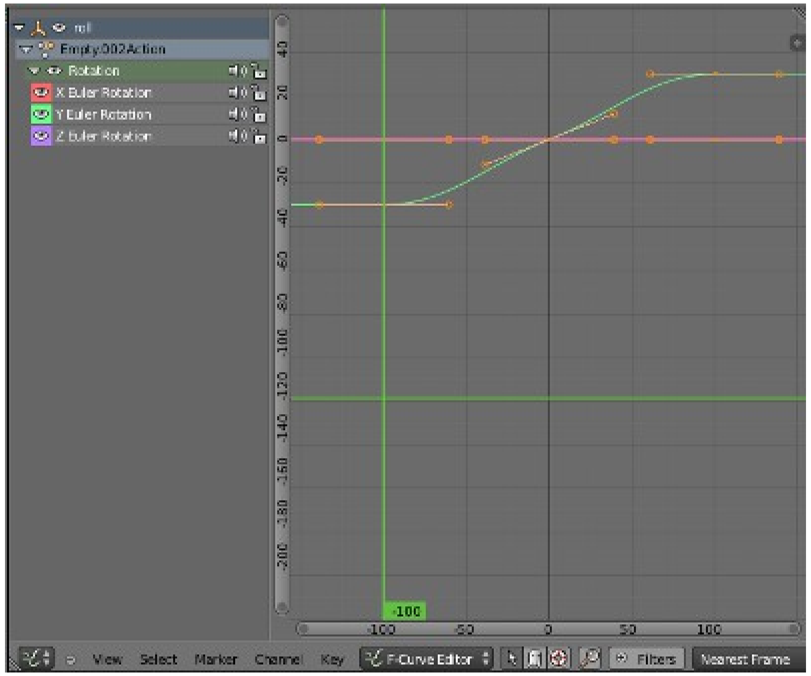
Make sure everything except the “player_object” is set to no collision otherwise you'll be getting problems later. This includes the player mesh once you add it.

Part three- Key framing the empties.

Next we need to set up the empties so that they will tilt in game to show the rotational movement of our aircraft. Select the roll empty and press “i”:

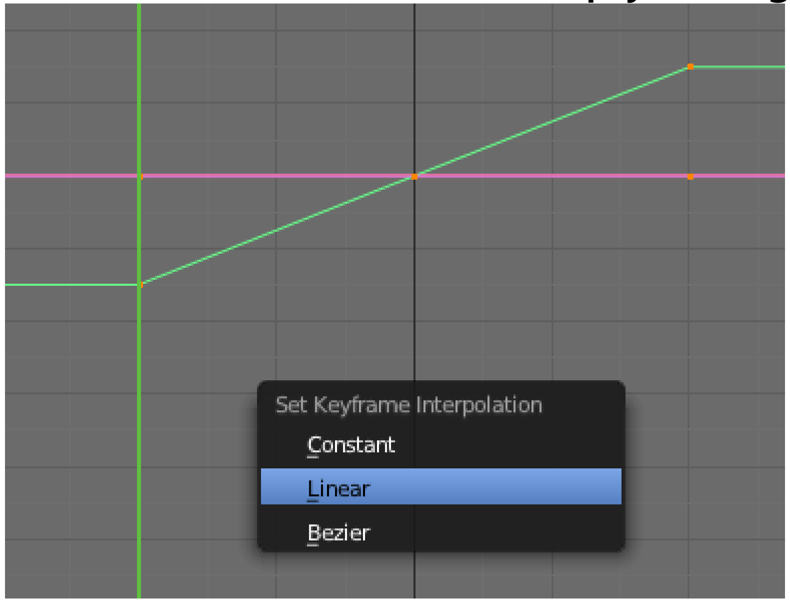


Select rotation to add your first keyframe. You should make sure the animations are set to frame zero when you do this. You can find the animations we need in the graph editor which you should open up in another window:



Go to frame 100 and rotate the “roll” empty 30 degrees left, then add a keyframe.

Do the same at frame -100 but this time rotate the “roll” empty 30 degrees right.

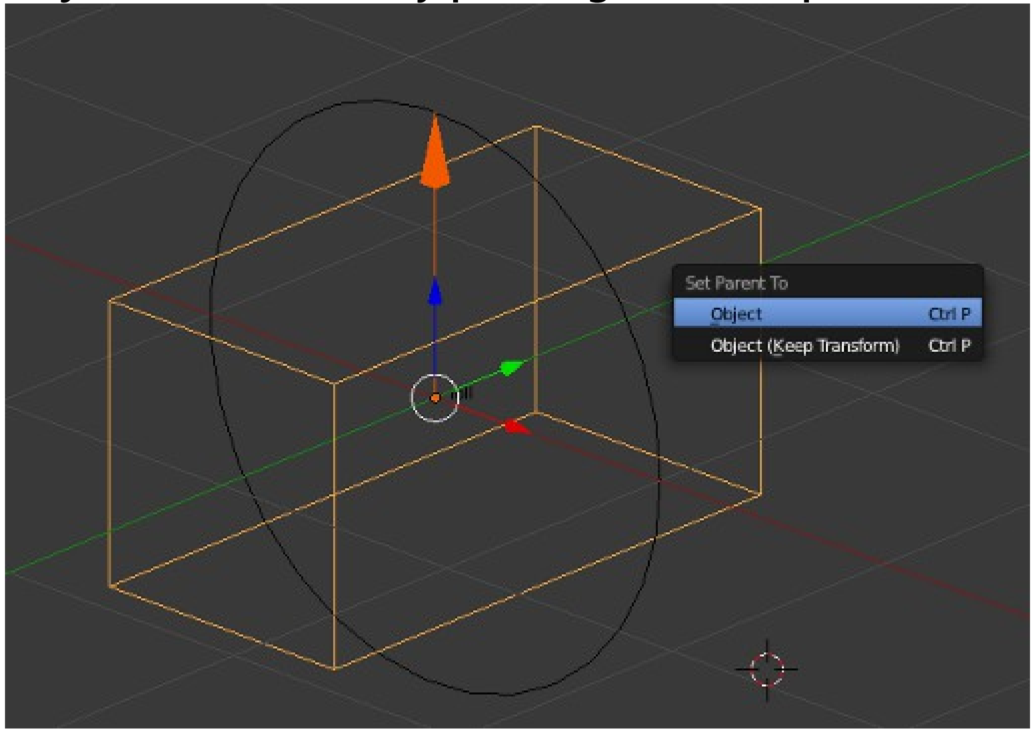


With the cursor in the graph editor window, press “t” and set interpolation to linear. You should do this with all the actions that you will make.

Now do the same with the “pitch” empty, but instead tilt it forward and back before adding key frames at 100 and -100 frames. Scrub the current frame back and forward to check that your empties are tilting correctly before moving to the next part.

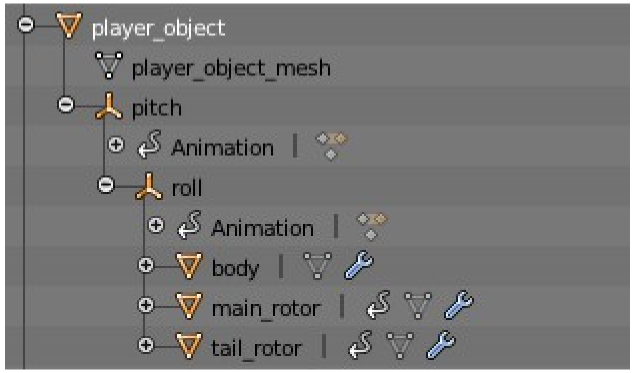
Part four- Parenting.

Next you have to set up your parenting relationships. We want the helicopter mesh to be parented to the "roll" object, the "roll" object to be parented to the "pitch" object and the "pitch" object to be parented to the "player_object". We do this by pressing "Ctrl" + "p".

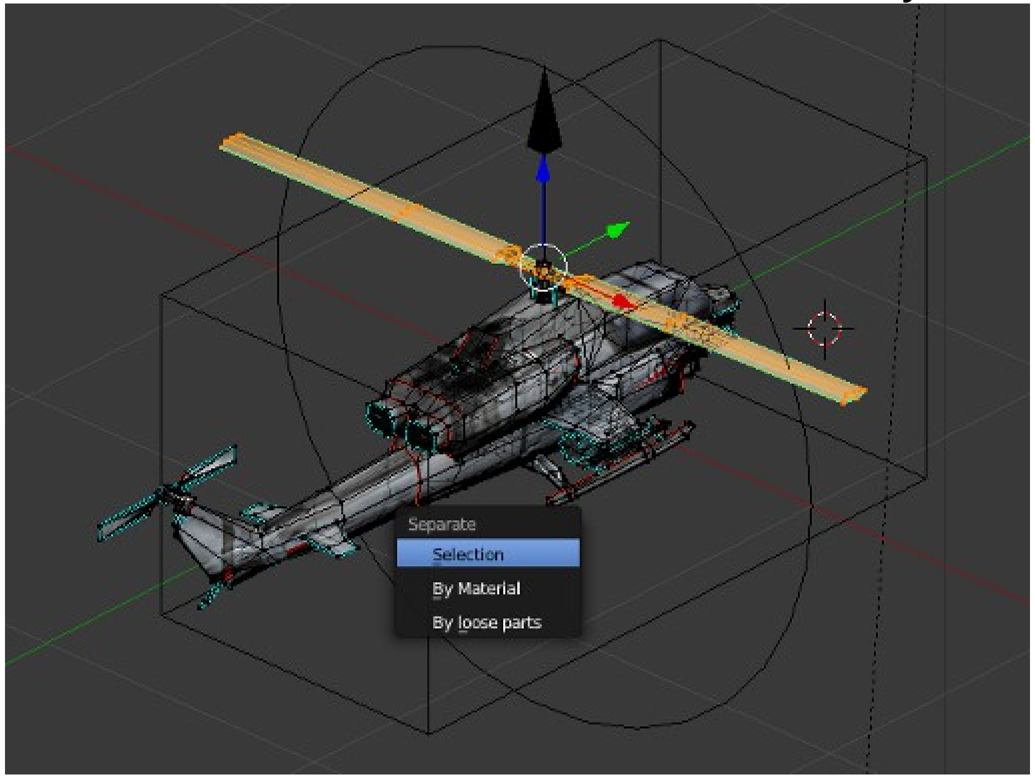


If you need to un-parent an object due to a mistake you can use "Alt" + "p".

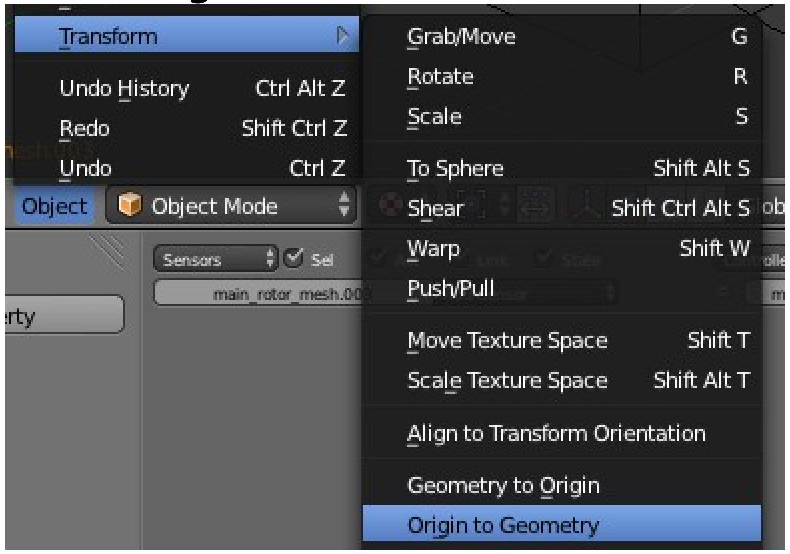
It should be organized like this:



We will also separate the main rotor and tail rotors from the main body mesh:



We do this because they are going to be given animations to make them rotate. Press “p” to separate the selected vertices. Don't forget to make the mesh “no collisions”.



After separating the rotors you want to set their center by going to the object menu and using “transform” and “origin to geometry”.

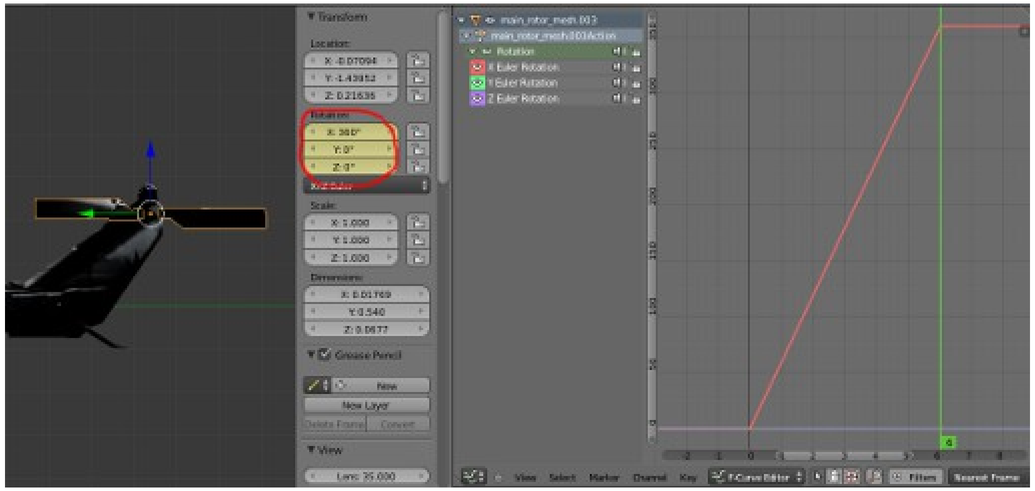
Now your objects should all be in place and correctly parented. Try grabbing and moving the main “player_object” to make sure everything else moves along with it.

If you rotated anything or scaled any of the objects you had better go to the object menu and select apply rotation/ scale at this point to avoid unwanted effects later.

Part five- Rotor set up.

We want our rotors to rotate in game and look like real chopper rotors. Later will will be adding functionality for taking off and landing, or for crashing so we need to be able to control how the rotors behave and look in game. For now we are just going to set up a basic always/and logic to get them moving and looking like moving blades.

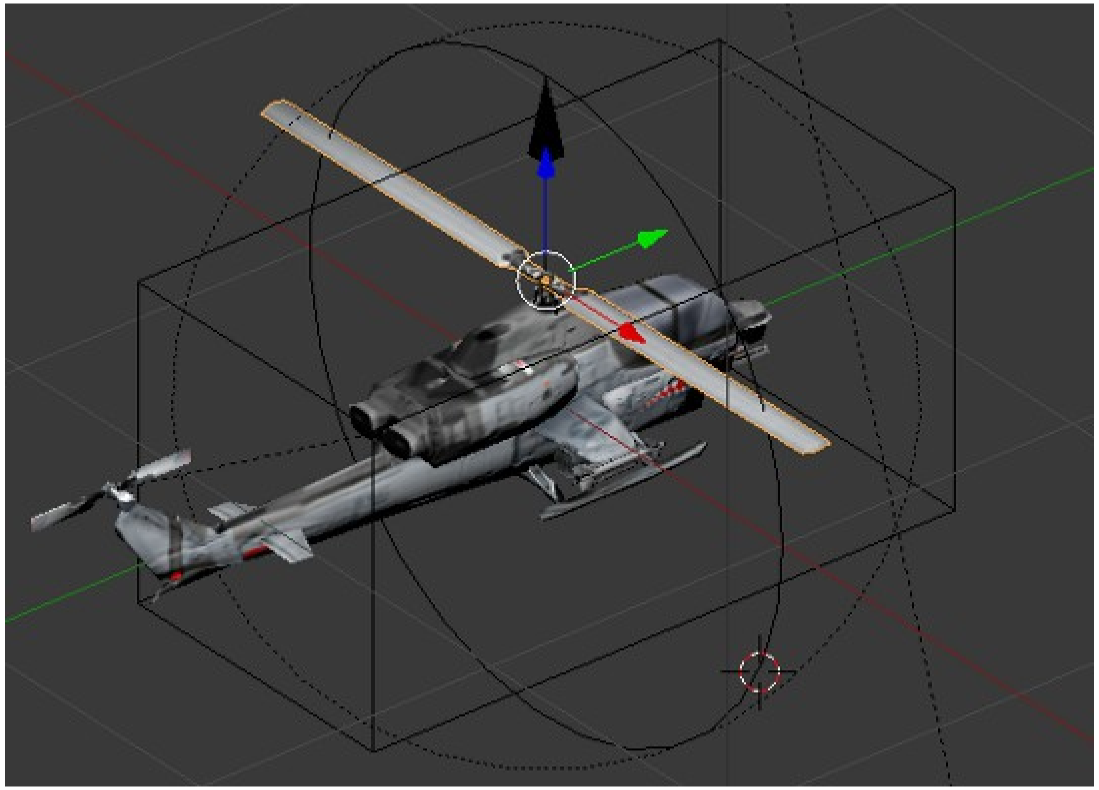
First add key frames for actions on the main rotor and tail rotor:



I find it easier to add the rotation manually in the rotation box in the main 3d window. Press “n” to bring it up if it's hidden.

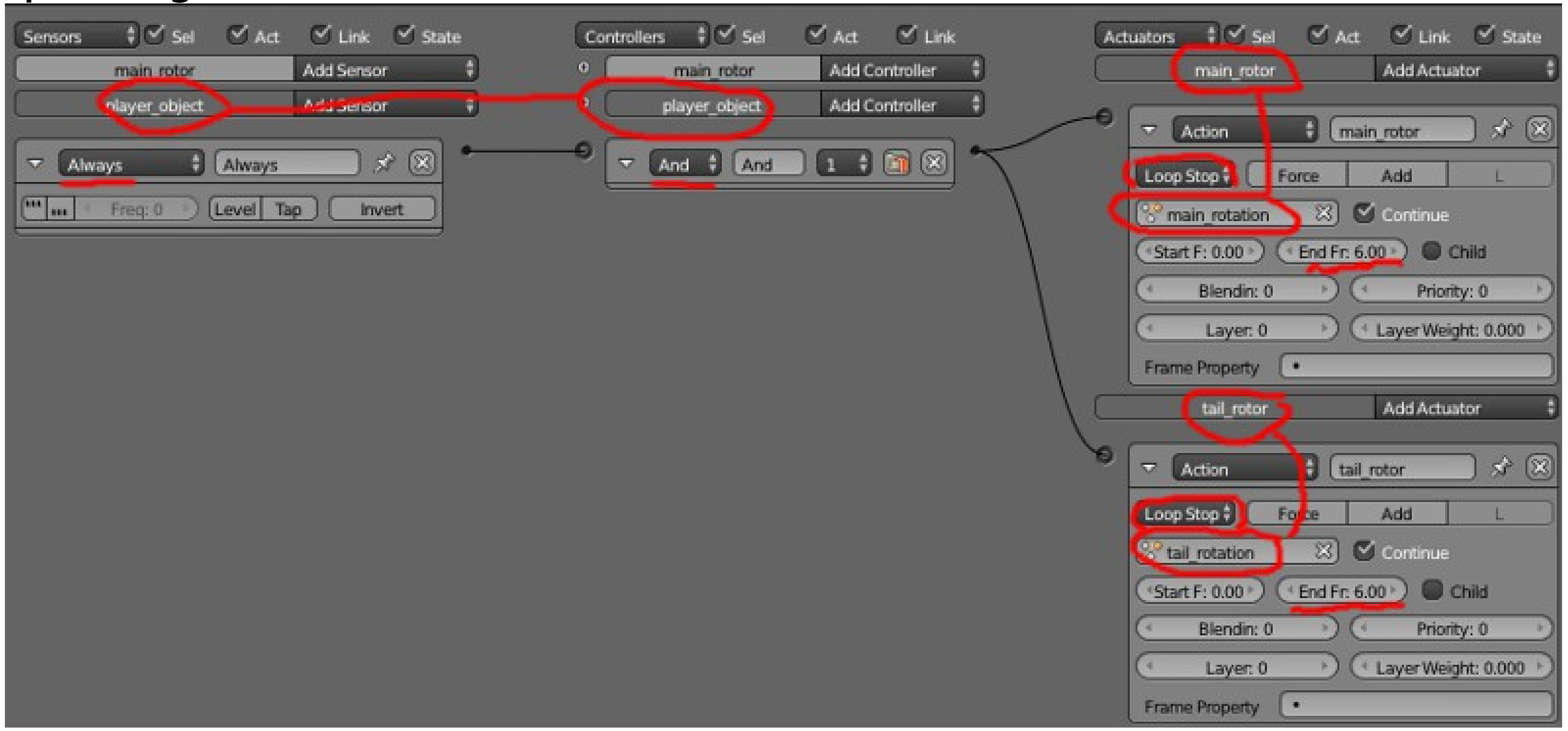
The rotors should turn around a full 360 degrees in about 5-7 frames. Make sure you set the interpolation to “linear”.

Your player object should now look like this:



Remember the mesh should be facing along the "y+" axis, away from the camera. The rotors should be parented to the "roll" empty.

Now set up the logic bricks like this:

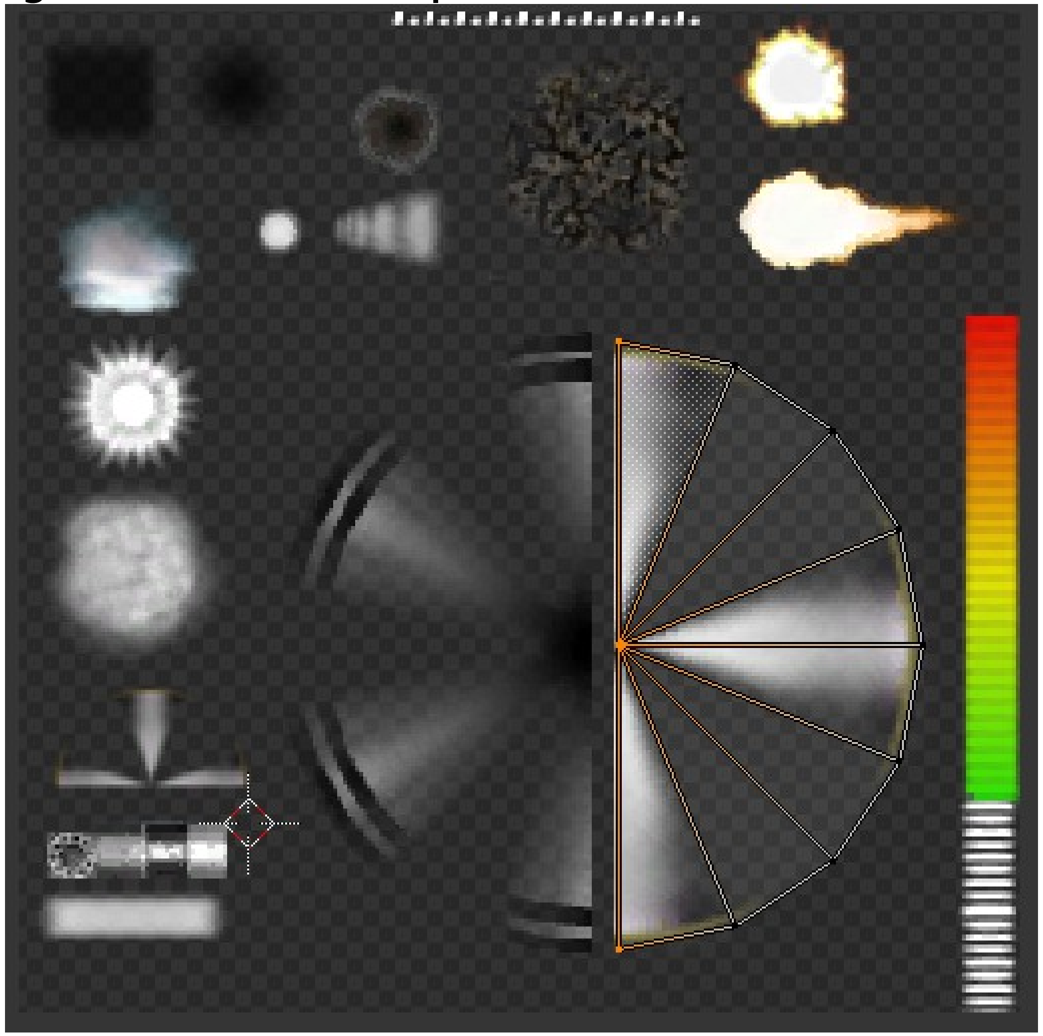


The player object needs an "always" sensor linked to an "And" controller. The controller should be linked to two actions, one for each rotor. Notice how all the actions and objects are named meaningfully, this makes it much easier to sort out problems.

- The action should be set to "loop stop".
- End frame should be the last frame of the rotor animation.
- Don't forget to select the action in the box which shows three balls under loop stop.

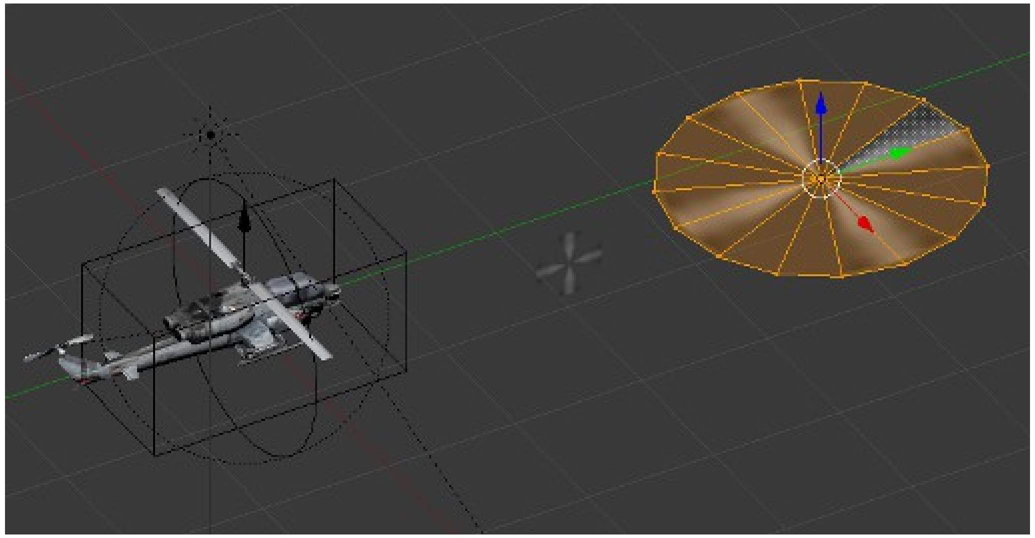
Everything else can be left as it is. You don't need to set the always sensor to pulse mode.

Next you need a texture for your rotating blades. Here's the one I use, along with other effects I will use in the game such as gun flashes, smoke, sparks and HUD elements:

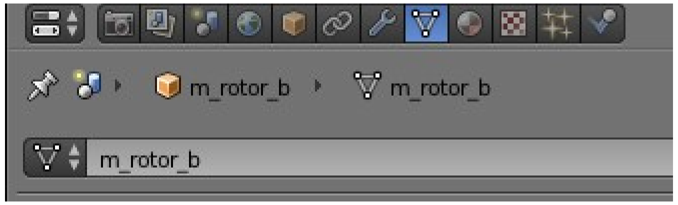


It's good to keep all your effects on one sheet if you can, makes organization much easier.

Create rotor objects which are simple 2d circles with a diameter the same as your chopper's blades.



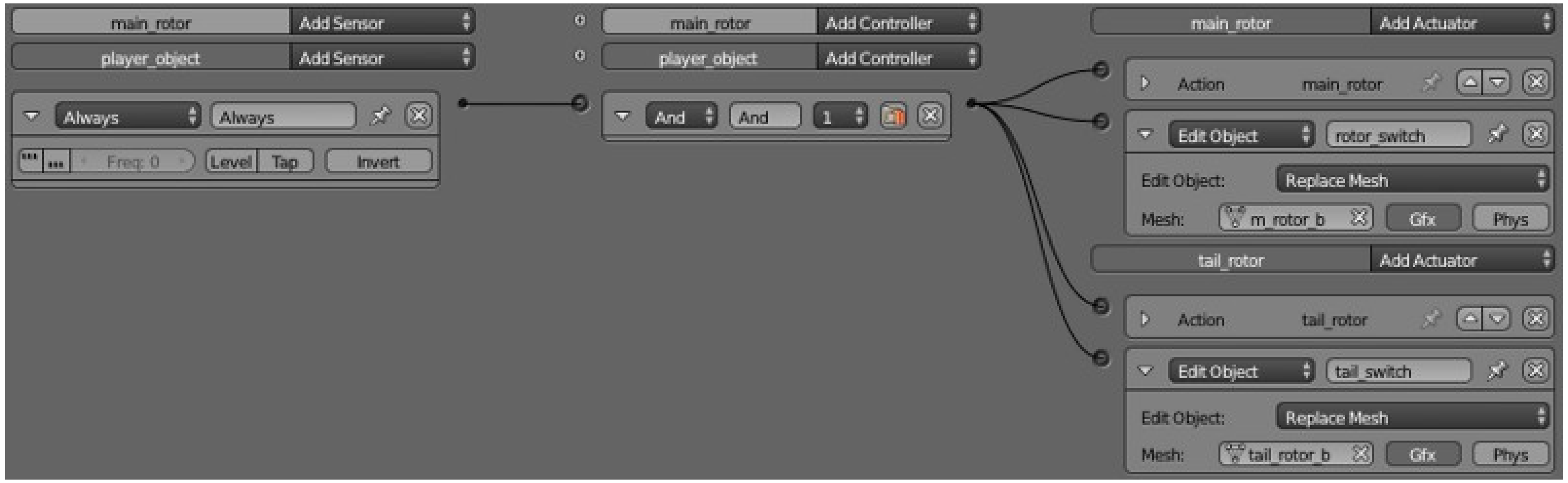
Name them with meaningful names. Don't forget to name the mesh, as well as the object.



m_rotor_b ▸ m_rotor_b

m_rotor_b

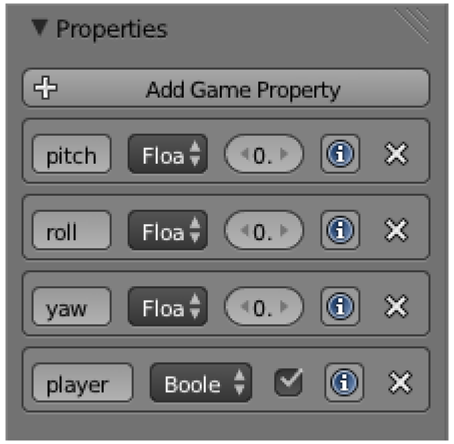
Next we need actuators to switch the mesh to use your spinning blades:



Use the edit object actuator and replace only “Gfx”.
If you press “p” now the game will start and your chopper should hang in space with it's rotors spinning. If not, then something is missing, so go back and check all the logic bricks and actions.

Part six- Movement logic.

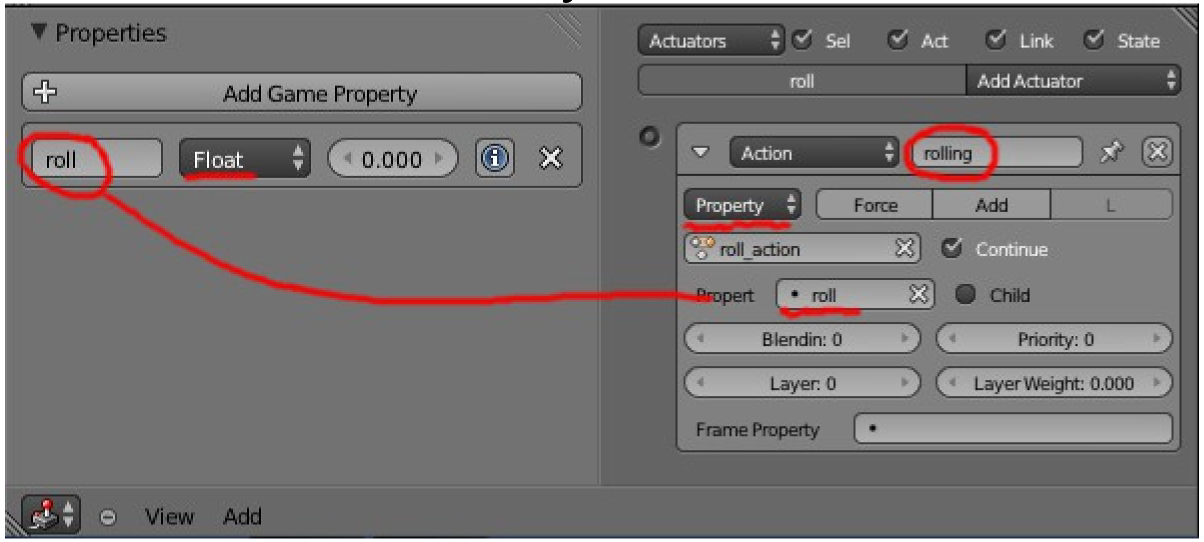
Now we are going to make the chopper move. Start by adding some properties to the "player_object":



pitch	Floa	0.	
roll	Floa	0.	
yaw	Floa	0.	
player	Boole	✓	

"player" is used to find the player object in the game world. Sometimes we want to track to the player with missiles, or focus the camera on it. Adding a property makes this possible.

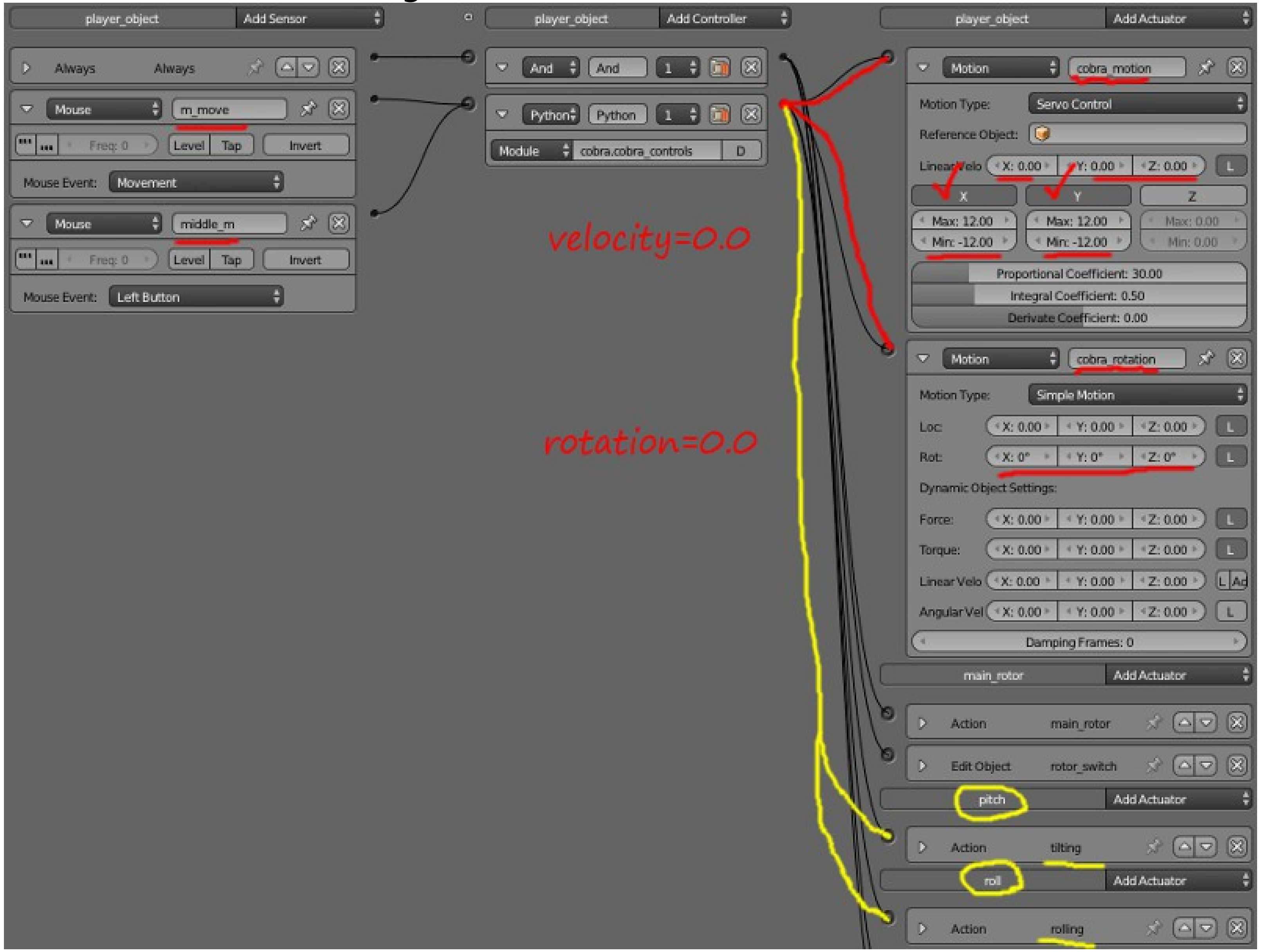
Next we want to add an actuator to the “roll” object:



Name it “rolling” and set it to property. You will also have to add a “roll” property to the empty.
The name of the actuator becomes important here because it is referenced in the script.
Do the same with the pitch object, but add a “pitch” property and call the actuator “tilting”.

*Actually “tilting” is not a good name for the actuator, but “tilt is what I initially chose for “y” axis movement. Only during development did I change to “pitch/roll/yaw”. Sometimes you will make changes during development and have to go back and change things if you want to keep them well organized. However if I change this now some of the illustrations in the next section will become more confusing. It's good practice to change and re-order things as soon as you can, otherwise chaos can quickly build up.

Now we need to add some more logic bricks:



player_object — Add Sensor

- Always — Always
- Mouse — m_move — Freq: 0 — Level — Tap — Invert — Mouse Event: Movement
- Mouse — middle_m — Freq: 0 — Level — Tap — Invert — Mouse Event: Left Button

player_object — Add Controller

- And — And — 1
- Python — Python — 1 — Module — cobra.cobra_controls — D

player_object — Add Actuator

- Motion — cobra_motion
  - Motion Type: Servo Control
  - Reference Object:
  - Linear Velo — X: 0.00 — Y: 0.00 — Z: 0.00 — L
  - X — Max: 12.00 — Min: -12.00
  - Y — Max: 12.00 — Min: -12.00
  - Z — Max: 0.00 — Min: 0.00
  - Proportional Coefficient: 30.00
  - Integral Coefficient: 0.50
  - Derivate Coefficient: 0.00
- Motion — cobra_rotation
  - Motion Type: Simple Motion
  - Loc: X: 0.00 — Y: 0.00 — Z: 0.00 — L
  - Rot: X: 0° — Y: 0° — Z: 0° — L
  - Dynamic Object Settings:
  - Force: X: 0.00 — Y: 0.00 — Z: 0.00 — L
  - Torque: X: 0.00 — Y: 0.00 — Z: 0.00 — L
  - Linear Velo X: 0.00 — Y: 0.00 — Z: 0.00 — L — Ad
  - Angular Vel X: 0.00 — Y: 0.00 — Z: 0.00 — L
  - Damping Frames: 0

main_rotor — Add Actuator

- Action — main_rotor
- Edit Object — rotor_switch

pitch — Add Actuator

- Action — tilting

roll — Add Actuator

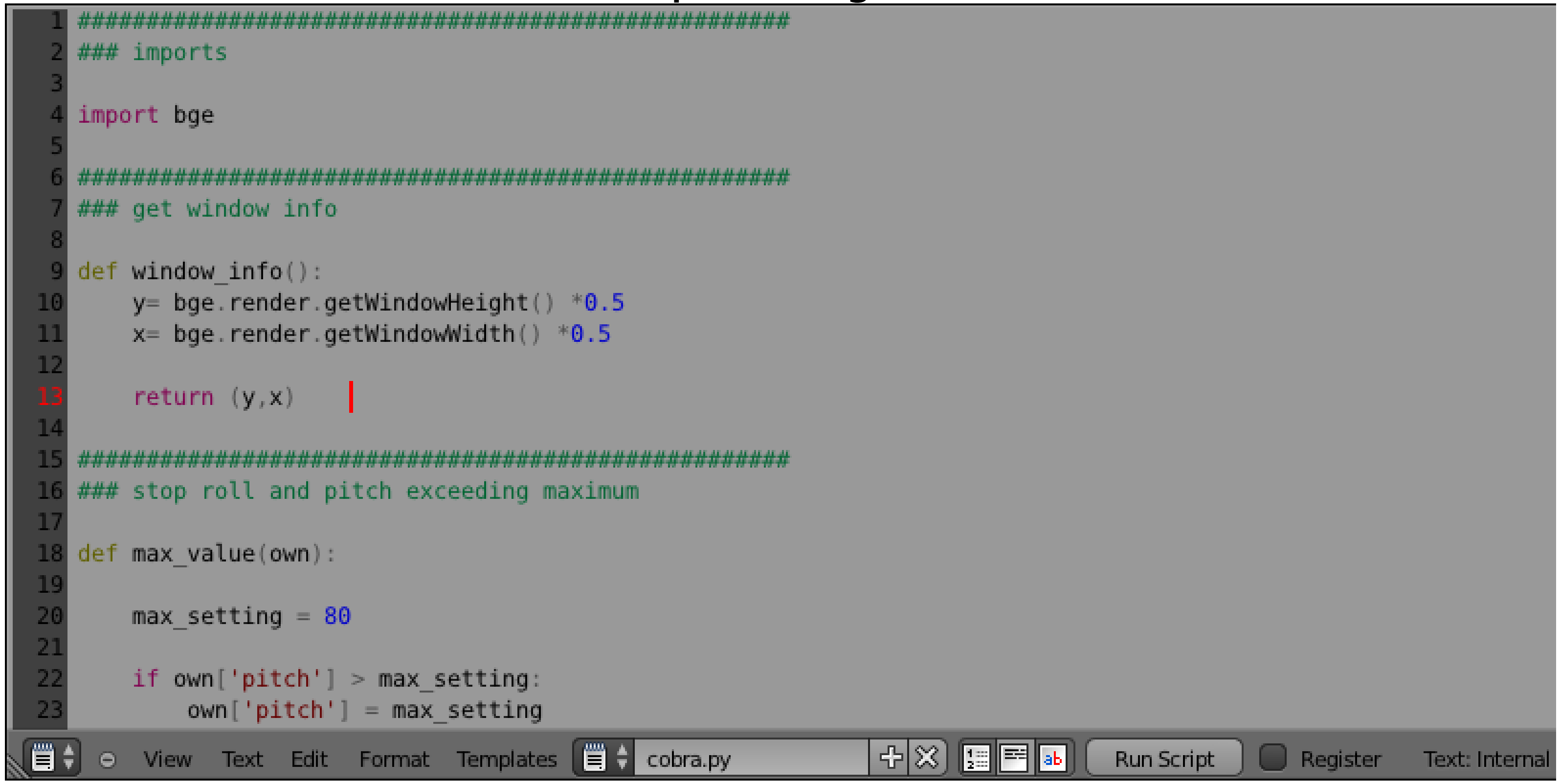
- Action — rolling

You need two mouse sensors, one for movement and one for the middle mouse button (mistakenly set to left mouse in the above screen shot, something which caused me confusion later). Don't set them to pulse mode.
You need a python controller set to module mode, not script mode. The module name is:

```
cobra.cobra_controls
```

You also need actuators for movement. One should be a servo movement actuator, set the max “x” and “y” forces to 12 and minimums to -12. The other actuator is for rotation and doesn't need changing.
Call the actuators “cobra_motion” and “cobra_rotation”. The sensors should be “m-move” and “middle_m”. You should also link the “tilting” and “rolling” actuators to the python controller.

You will now need to add the movement script to the game.



```
##################################################
### imports

import bge

##################################################
### get window info

def window_info():
    y= bge.render.getWindowHeight() *0.5
    x= bge.render.getWindowWidth() *0.5

    return (y,x)

##################################################
### stop roll and pitch exceeding maximum

def max_value(own):

    max_setting = 80

    if own['pitch'] > max_setting:
        own['pitch'] = max_setting
```

The script can be found in the forum where you found this tutorial.

You can add it by pasting it in to your text window and naming it:

`cobra.py`

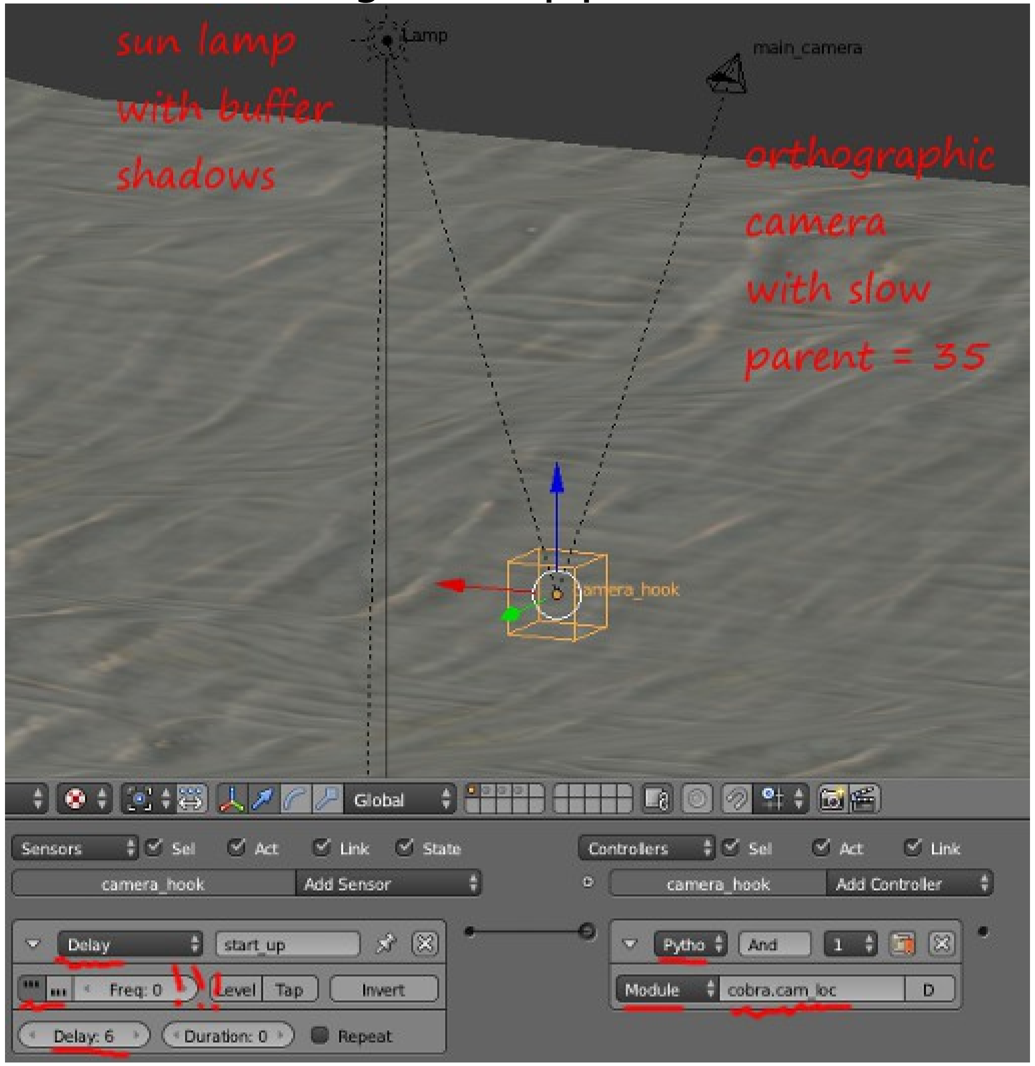
I'll go in to more detail about the script later. Once you have the script added, your helicopter should fly as intended. Else you may get this error:

```
ImportError: No module named 'cobra'
```

Check the name of your script and any other things that may have been misnamed.

Part seven– Camera set up.

Now go in to your first layer, where you have the ground plane. You will need a new empty with an orthographic camera and a shadow casting sun lamp parented to it, like this:



Sensors ✓ Sel ✓ Act ✓ Link ✓ State
camera_hook | Add Sensor
Delay | start_up
Freq: 0 | Level | Tap | Invert
Delay: 6 | Duration: 0 | Repeat

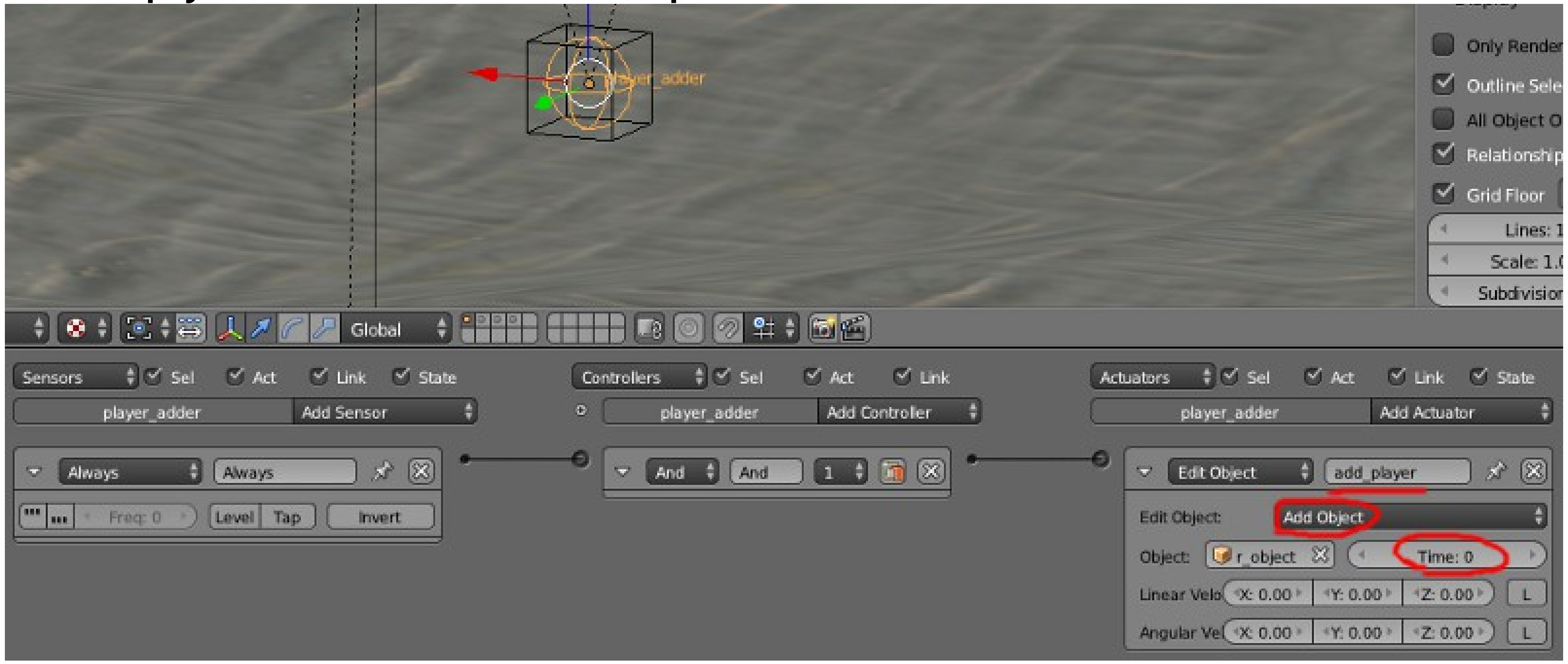
Controllers ✓ Sel ✓ Act ✓ Link
camera_hook | Add Controller
Pytho | And | 1
Module | cobra.cam_loc | D

Set up the logic bricks as shown in the image above. Use a python controller with the following module: `cobra.cam_loc`

This will make your camera follow your chopper's position but not copy it's rotation. Add a string property to the empty called “player_ob”.

*It's not good practice to use a sensor on pulse mode with “Freq:” of 0. It can be a big logic drain if many objects have this. However, we have just one object doing this and the script is not that long. You could also try parenting the camera to a vertex on the “player_object”, but we want the camera to remain in scene and in position even if the player is destroyed.

Add another empty to the scene and set it up like this:



Sensors: player_adder — Always (Always), Freq: 0, Level, Tap, Invert

Controllers: player_adder — And (And), 1

Actuators: player_adder — Edit Object (add_player)
Edit Object: Add Object
Object: r_object   Time: 0
Linear Velo X: 0.00 Y: 0.00 Z: 0.00 L
Angular Vel X: 0.00 Y: 0.00 Z: 0.00 L

It should be adding the “player_object”. Now enjoy flying around your game world.

Hopefully your game is running as intended. If you have any problems following the tutorial post in the thread where you found this tutorial or take a look through the example blend to see what could be different. In the next tutorial I'll be looking at setting up our chopper's weapons and some simple enemies for target practice.

Smoking Mirror AKA Pickledtezcat AKA John Topple.

23rd June 2013

Tutorial posted here:

Blender Artists Forum

If you want to write your own tutorials to be included in the series you should use the following formatting conventions:

Main Title: Segoe Print 26 point, Centered.
Subtitles: Segoe Print 16 point, left aligned.
Main text: Segoe UI Semibold 16 point, left aligned.
Image captions: Segoe Print 16 point, Centered.

The document should be landscape, not portrait for better viewing on wide screen monitors without splitting pages, and images included should be of a maximum with of 512 pixels and should be centered. There should be a roughly 50/50 mix of text and images

You can use the following image for title text background:

It's a simple gradient from pure white to “404040” gray on the right.