Panda3D BVW Reference

some of what you need to know to survive BVW and Panda3D

model conversion

make a model:

```
clean, collapsed geometry.
no extra geometry
no extraneous history
```

3DSMax:

```
There is a plugin for 3ds Max.
Once this has been installed, you choose
"File"->"Save as" and pick *.EGG as the filetype
```

Maya:

```
save a .mb file from maya
open a command prompt (C:\WINDOWS\system32cmd.exe)
maya2egg <Maya File>.mb <Egg File>.egg

-a model (model only)
-a chan (animation only)
-a both (model with animation
-ps strip (egg looks for textures in same dir)
-uo ft (converts to the scale used in panda3d)
```

types of .EGG:

```
to access the bones of a model in panda:
  export the model and animation together
otherwise:
  just the model is fine

for multiple animations
  -export each separately and attach to an Actor
```

config.prc short guide

number of sounds you can load:	`audio-cache-limit 32`
displays your frame rate:	`show-frame-rate-meter 1`
window properties:	`win-width 640`
	`win-height 480`
	`win-origin-x 100`
	`win-origin-y 100`
	`fullscreen 0`
	`undecorated 0`
	`cursor-hidden 0`
	`window-title 'Panda'`

Tracking System

Most of the hard work has been done for you in the SimpleTracking.py module. Using this, you can attach up to 4 objects to the tracking system.

panda Module:

```
import SimpleTracking
```

initialize:

```
serverIP = "machinename.etc.cmu.edu:4500"
spacePad = SimpleTracking.SpacePadSystem(serverIP)
spacePad.enable()
```

control objects:

```
camera.reparentTo( spacePad.getHMDHelper() )
obj1.reparentTo( spacePad.getYellowHelper() )
obj2.reparentTo( spacePad.getGreenHelper() )
obj3.reparentTo( spacePad.getBlueHelper() )
```

jam-o-drum

panda Modules:

```
import spinners #handles spinner hardware
import drumpads #handles drumpad hardware
from globals import * #useful global variables
```

initialize:

```
d=drumpads.DrumPads()
s=spinners.Spinners()
d.Start()
s.Start()
```

a task:

```
def polling( task):
     d.Poll()
     s.Poll()
     return Task.cont
```

things to accept:

```
for x in range (NUM_STATIONS):
     temp="SPIN_"+str(x)+"_"+str(SPIN_CCW)
     self.accept(temp, self.spinCCW, [x])
     temp="SPIN_"+str(x)+"_"+str(SPIN_CW)
     self.accept(temp, self.spinCW, [x])
     temp="HIT_"+str(x)
     self.accept(temp, self.hit, [x])
```

http://www.python.org/ has all of the real answers.

python basics

White space matters in Python, which means that your code will not run if the indention is wrong, and it will also affect things such as the logic of if statements.

```
#this is a comment
```

basic python logic

if:

```
if test:
     #true case
elif test 2:
     #second true case
else:
     #false case□
```

while:

```
while test:
     #do other stuff□
```

for:

```
for x in range(10):
     #do stuff

for x in range(5,10):
     #do stuff

for x in aList:
     #do stuff
```

python tuples

A tuple consists of a number of values separated by commas. They are useful for ordered pairs and returning several values from a function.

creatuion:

```
empty= ()
single = 'thing',
tuple= 12, 89, 'a'
```

accessing: `tuple[0] □` returns 12

python dictionaries

A dictionary is a set of key:value pairs. All the keys in a dictionary must be unique.

creatuion:

```
empty= {}
dict= {'a':1, 'b':2, 'c':3}
```

accessing:	`dict['a'] □ □`	returns 1
deleting:	`del dict['b']`	
finding:	`dict.has_key('e')□`	returns 0
	`dict.keys()□ □`	**returns ['a','c']**
	`dict.items□ □`	**returns [('a',1), ('c',3)]**

python list manipulation

One of the more important data structures in python are lists. Lists are very flexible and have many built in control functions.

creatuion:	`list= [5,3,'p',9,'e']`		`[5,3,'p',9,'e']`
accessing:□	`list[0]□`	**returns 5**	`[5,3,'p',9,'e']`
slices:	`list [1:3]`	**returns [3, 'p']**	`[5,3,'p',9,'e']`
length:	`len(list)`	**returns 5**	`[5, 3, p',9,'e']`
sort:	`list.sort()`		`[3,5,9,'e','p']`
add:	`list.append(37)`		`[3,5,9,'e','p',37]`
return & remove:	`list.pop()`	**returns 37**	`[3,5,9,'p']`
	`list.pop(1)`	**returns 5**	`[3,9,'p']`
remove:	`list.remove('e')`		`[3,9,'e','p']`
	`del list[0]`		`[9,'p']`
insert:	`list.insert(2, 'z')`		`[9,'z','p']`
concatination:	`list + [0]`	**returns [3,5,'a',9,0]**	`[9,'z','p']`

python class and function definition

function:

```
def myFuncName(param1, param2):
□       #code goes here
```

class:

```
clas myClassName(inheritanceClasses):
     def __init__(self):
          #initilixation code here
          self.x=0□
     def myFunnName(self):
          #my function code goes here
c=myClassName()
```