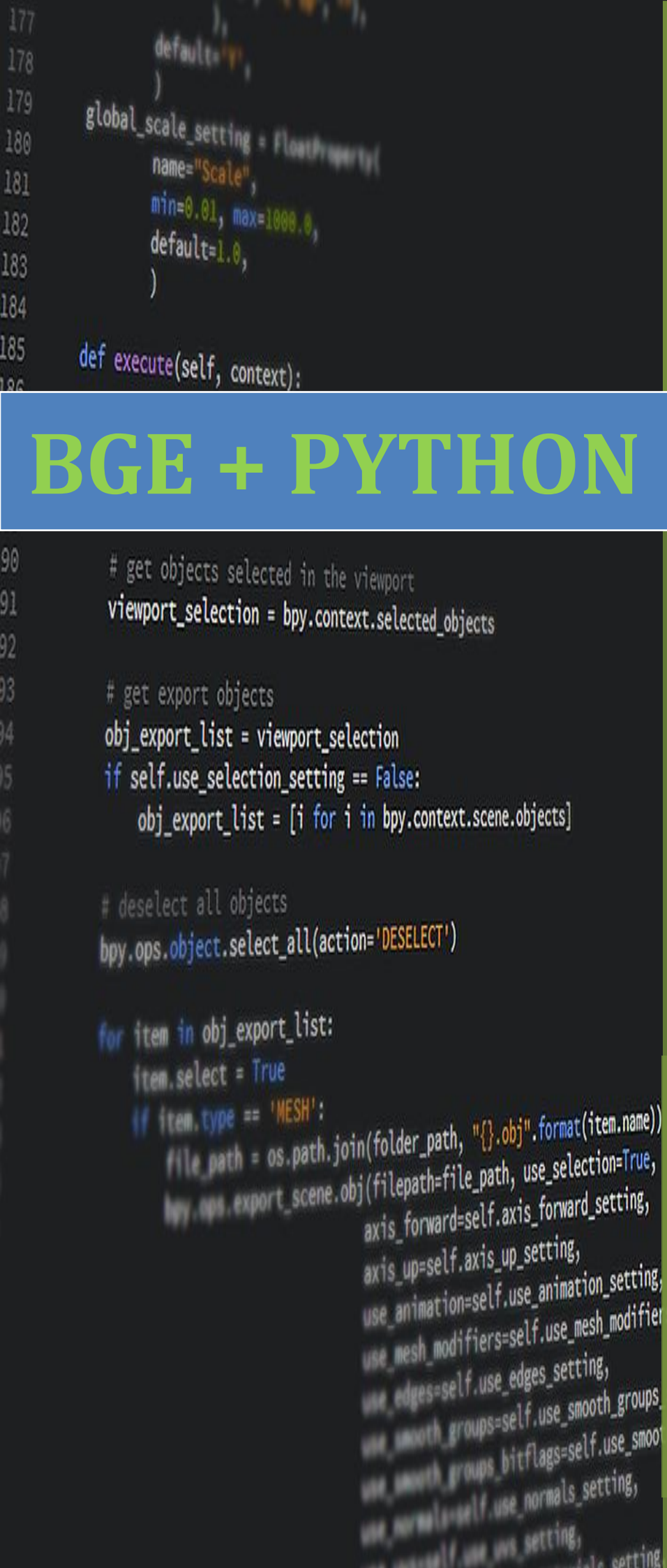
2020

BGE + PYTHON



Edinaldo Cicero

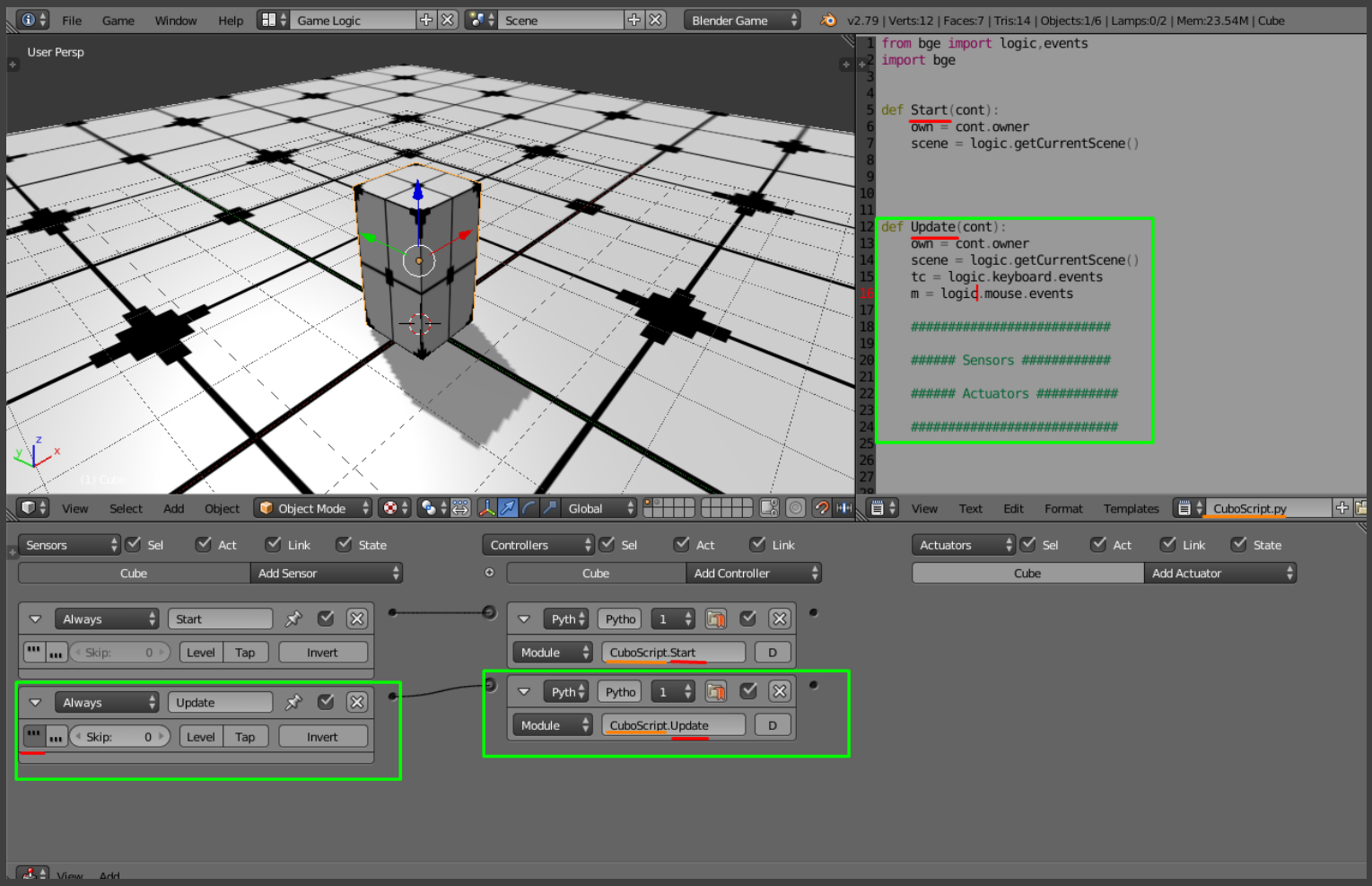
ÁtomoGames.

28/07/2020

BGE + PYTHON

Começando com Funções para Módulos:

Ao criar uma Função Start ou Update você há usa como Modulo no objeto que desejas usar no script, ou seja, o dono do script o próprio objeto.



Para chamar o script pelo Modulo vc pega o nome do (script. Nome da Função) como visto acima no controlador python.

Lá na função Update (cont), esse (cont) corresponde á (controller) para quer serve, para chamar o logicbricks de controlador do objeto, ou seja, o que está em verde com isso você tem acesso aos sensores e actuators dos objetos que estiverem ligados nesse controlador.

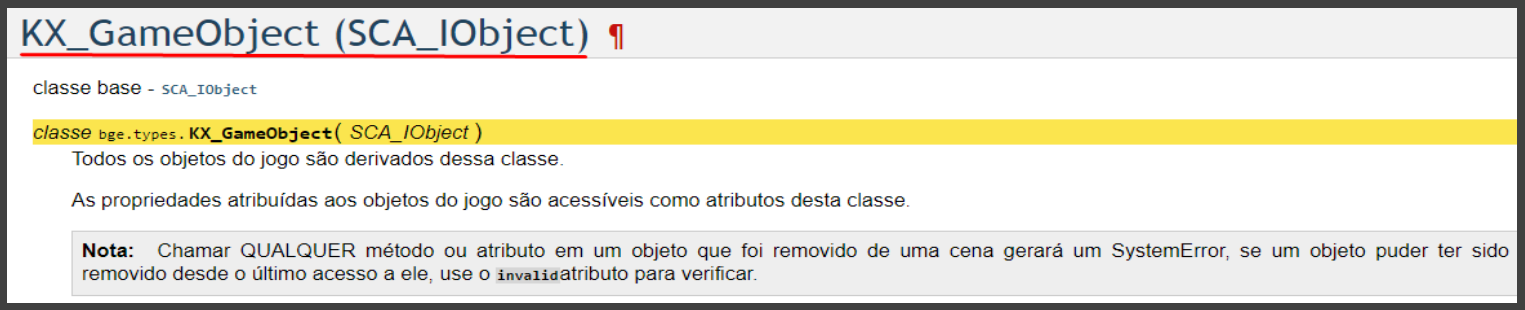
Own ou (owner = cont.owner) aqui corresponde ao próprio objeto proprietário do script com isso em mão você pode manipular como quiser o próprio objeto exemplo:

Para movimentar o objeto own:

```
owner.applyMovement([x,y,z],True)
```

Veja como eu aplico o movimento para o dono do próprio script(owner.) depois de colocar o ponto você deve chamar um função da biblioteca, da class(KX_GameObjects) que pertence á todos os objetos do blender com isso tem varias funções para todos os objetos.

API BLENDER 2.79:



KX_GameObject (SCA_IObject) ¶

classe base - SCA_IObject

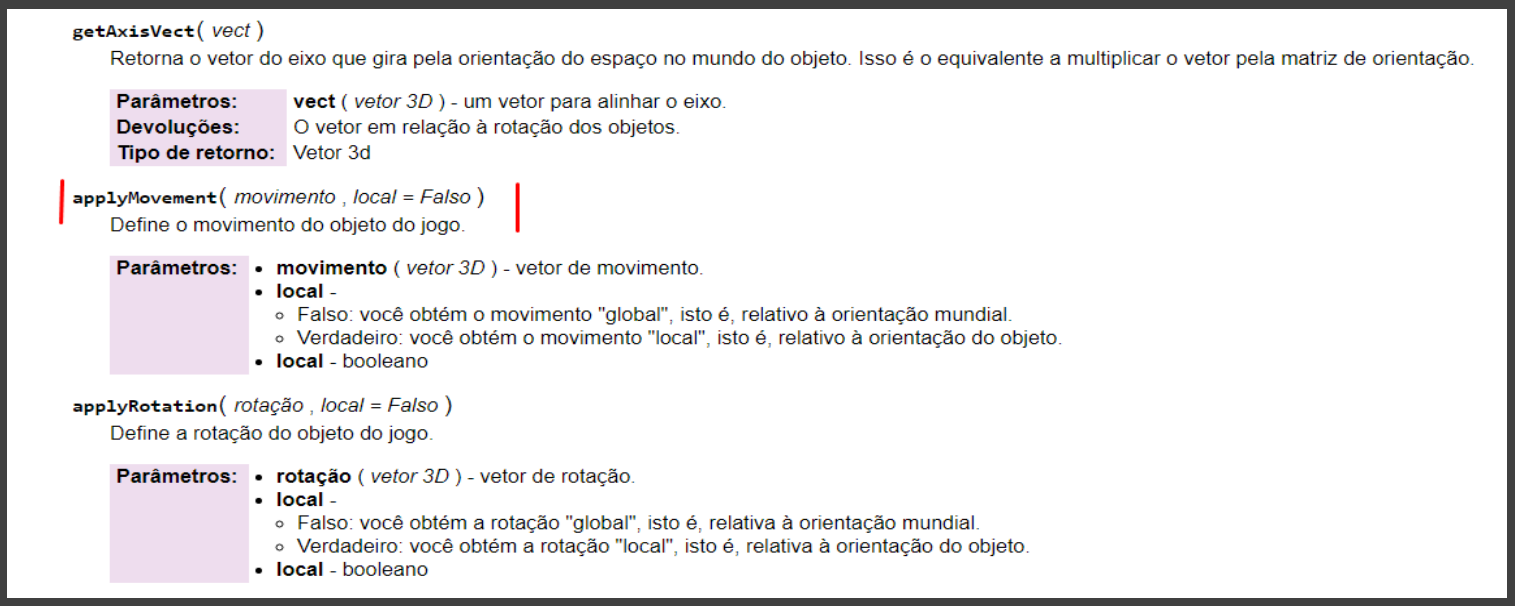
classe bge.types.**KX_GameObject**(*SCA_IObject*)

Todos os objetos do jogo são derivados dessa classe.

As propriedades atribuídas aos objetos do jogo são acessíveis como atributos desta classe.

Nota: Chamar QUALQUER método ou atributo em um objeto que foi removido de uma cena gerará um SystemError, se um objeto puder ter sido removido desde o último acesso a ele, use o `invalid`atributo para verificar.

https://docs.blender.org/api/2.79/bge.types.KX_GameObject.html#bge.types.KX_GameObject



getAxisVect(*vect*)

Retorna o vetor do eixo que gira pela orientação do espaço no mundo do objeto. Isso é o equivalente a multiplicar o vetor pela matriz de orientação.

Parâmetros:	**vect** (*vetor 3D*) - um vetor para alinhar o eixo.
Devoluções:	O vetor em relação à rotação dos objetos.
Tipo de retorno:	Vetor 3d

applyMovement(*movimento* , *local = Falso*)

Define o movimento do objeto do jogo.

Parâmetros:
- **movimento** (*vetor 3D*) - vetor de movimento.
- **local** -
  - Falso: você obtém o movimento "global", isto é, relativo à orientação mundial.
  - Verdadeiro: você obtém o movimento "local", isto é, relativo à orientação do objeto.
- **local** - booleano

applyRotation(*rotação* , *local = Falso*)

Define a rotação do objeto do jogo.

Parâmetros:
- **rotação** (*vetor 3D*) - vetor de rotação.
- **local** -
  - Falso: você obtém a rotação "global", isto é, relativa à orientação mundial.
  - Verdadeiro: você obtém a rotação "local", isto é, relativa à orientação do objeto.
- **local** - booleano

Tento isso em mente você poderá perceber que o que estiver dentro da classe Kx_GameObjects poderá ser utilizado após o (owner.) .

Uma coisa importante,se lembre que para cada função que for feita para um objeto e for chamada em Modulo no objeto que desejas manipular o (owner) se passa a ser o próprio objeto certo, ou seja, cada função poderá ser usada em cada objeto porém variáveis que estiverem dentro do escopo de uma função não poderá ser acessada dentro de outra !.

Como Movimentar e Rotacionar um obj e o que é Vetor2 E Vetor3:

Para movimentar ou rotacionar um objeto no universo do blender existe duas classes Vetoriais:

Para Movimentar:

1 = owner.applyMovent([vetor3],Boo)

owner.applyMovent([X,Y,Z],Boo)

Para rotacionar:

2 = owner.applyRotation([vetor3],Bool)

owner.applyRotation([X,Y,Z] ,Bool)

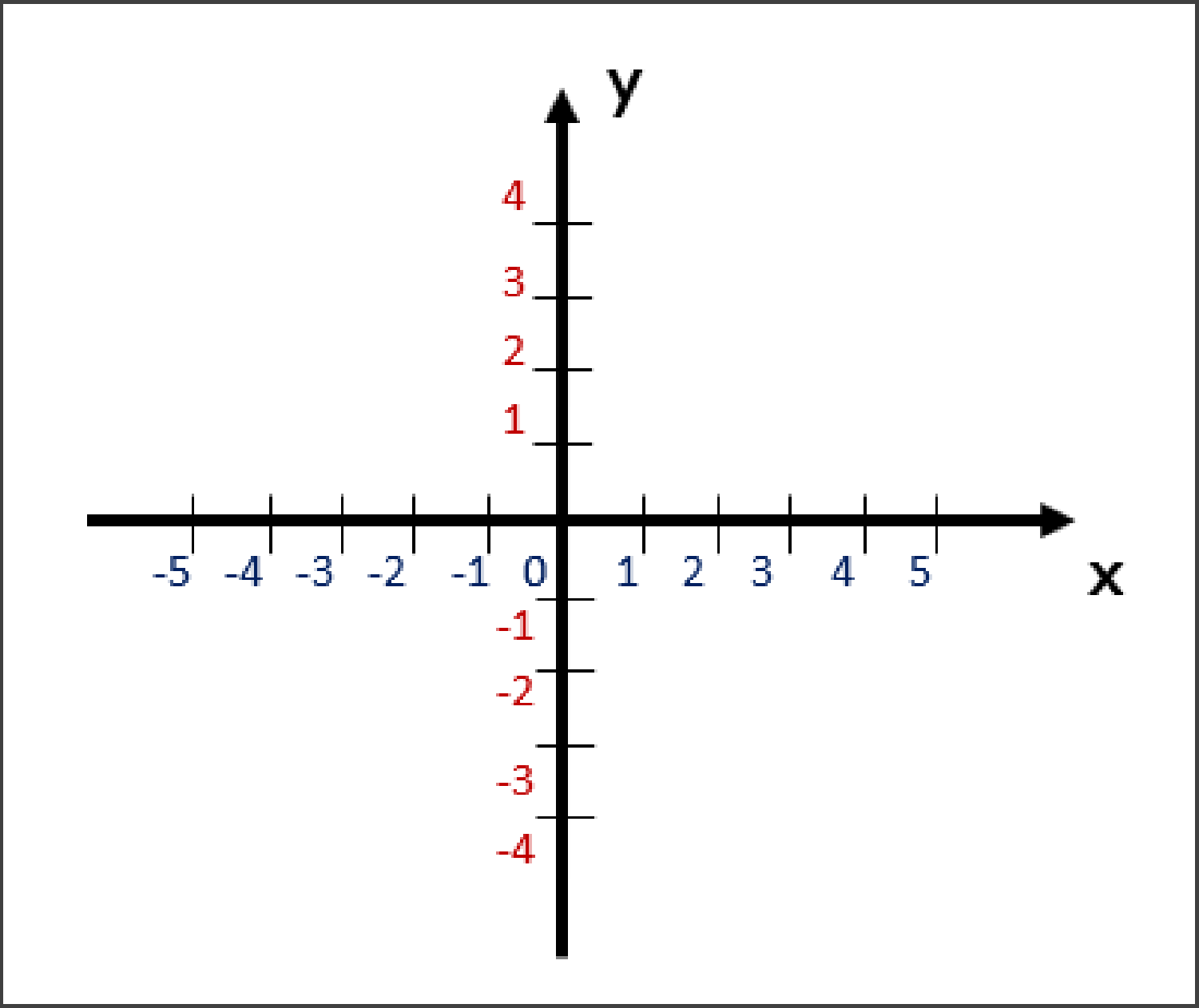
Vetor2 ou Vetor3 nada mais é na realidade um tipo de variavel que guarda posições geográficas, vector3 (X,Y,Z) se for vector2 seria (X,Y)

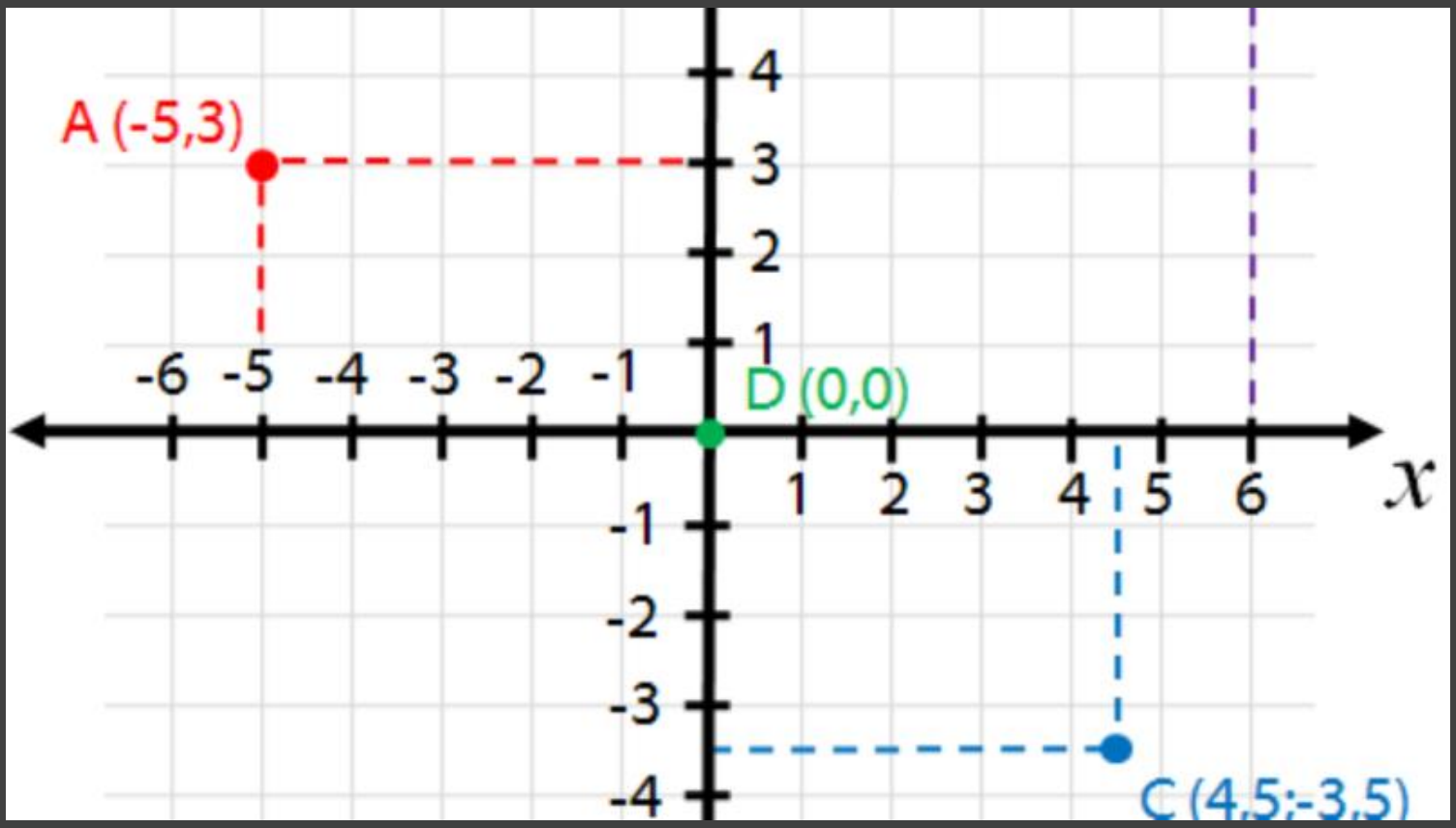
Com isso own.applyMovement([x,y,z],True) no lugar dos ([X,Y,Z]) coloque o valor que desejas que o objeto ande ou gire, tenha em mente que se você colocar (- ou +) na frente do valor desejado isso fará o objeto mudar de direção pois isso está baseado nos parâmetros geométricos do mundo 3D ou seja positivo para frente, negativo para trás e os valores tende á ser do tipo (Float) números quebrados ou numero seguidos de virgulas certo!, Pode sim usar números do tipo (Int) mais saiba que números inteiros são números exatos e corresponde a mais espaços para o objeto se deslocar pois o objeto não andar realmente ,claro! , mais sim se deslocar de acordo com o valor digitado, lembrou as aulas de física muito bem!

X = Largura , Y = Altura, Z = Profundidade

Vector2 pra Games 2d.

Vector3 pra Games 3d.





Como usar Inputs de Teclado e Mouse:

Para acessar teclas do teclado se deve chamar Chaves de Constantes, para isso chame todas as teclas do teclado:

Antes de chamar certifique-se de ter importado as bibliotecas de Logicas e eventos do blender assim:

From bge import logic, events

Feito isso agora crie uma variável com o nome que quiser mais que se torne fácil de lembrar claro e faça desse jeito:

Teclado = logic.keyboard.events

Com isso pronto vamos para usar cada tecla, e para isso vamos para a API com todas as teclas para usar: https://docs.blender.org/api/2.79/bge.events.html

E para usar a tecla escolhida basta criar uma condição de Input:

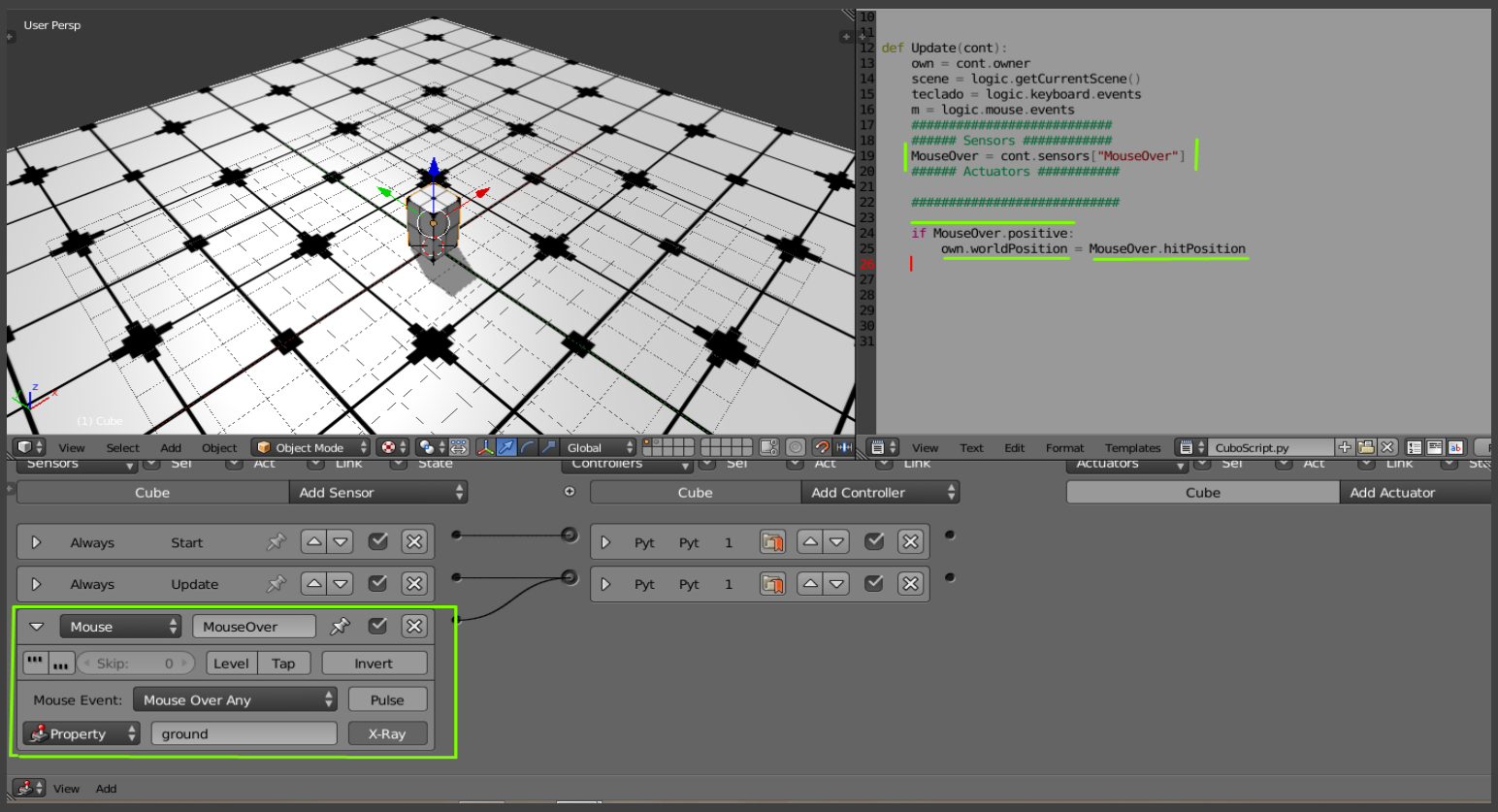
```
def Update(cont):
    own = cont.owner
    scene = logic.getCurrentScene()
    teclado = logic.keyboard.events
    m = logic.mouse.events
    ###################################
    ######## Sensors ##################
    ######## Actuators ################
    ###################################

    if teclado[events.WKEY]:
        print("Tecla W precionada !!")
```

(If teclado[biblioteca de eventos do BGE.Tecla escolhida terminando com KEY no final])

Com isso temos uma condição que quando a tecla W for apertada apois os dois pontos vc aperta inter e entrará dentro do escopo da condição que será abaixo com isso vc poderár fazer ações com o objeto colocando até o próprio own.applyMovement([x,y,z],True) e no lugar dos ([X,Y,Z]) coloque o valor que desejas que o objeto ande ou rotacione!

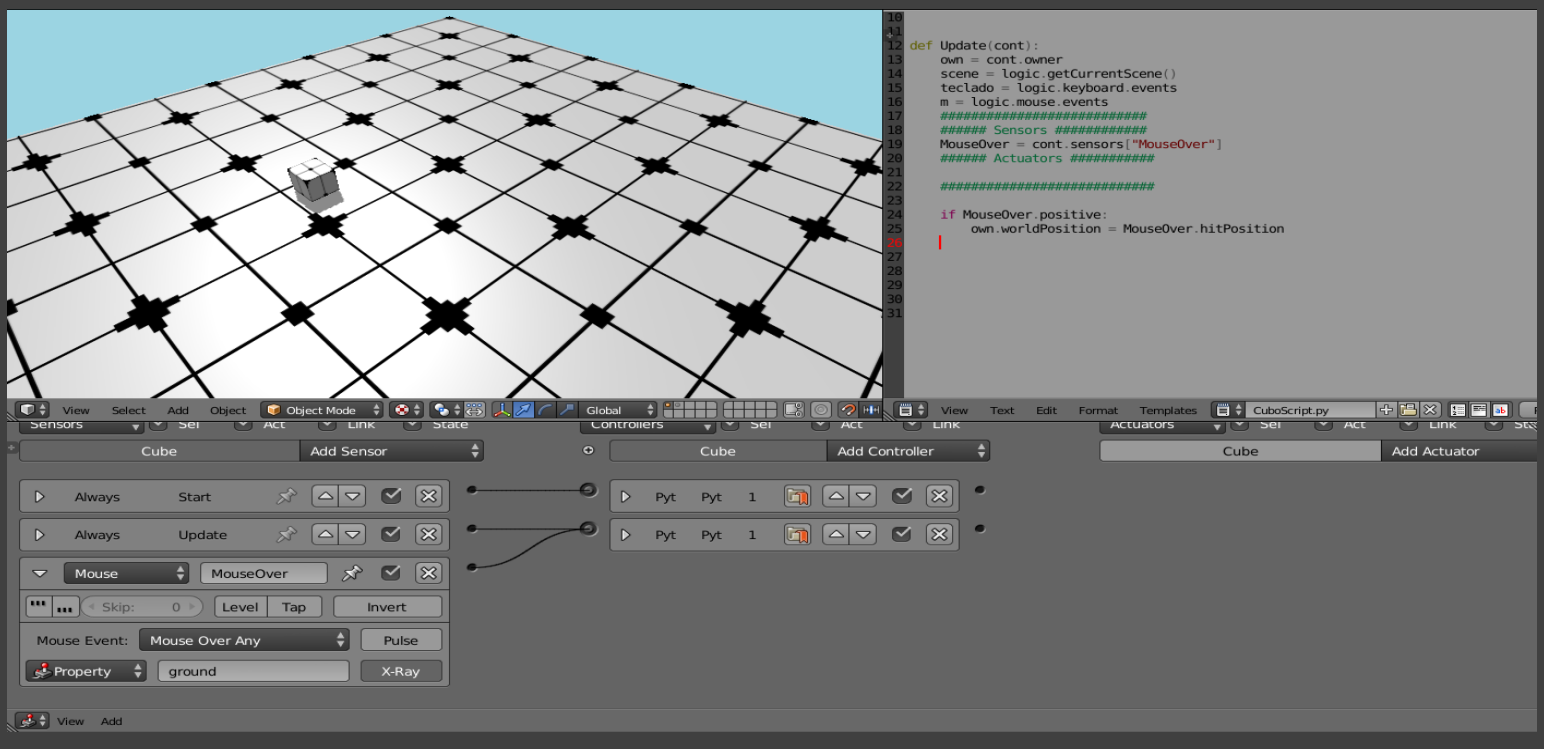
Como fazer um objeto seguir a posição do mouse:

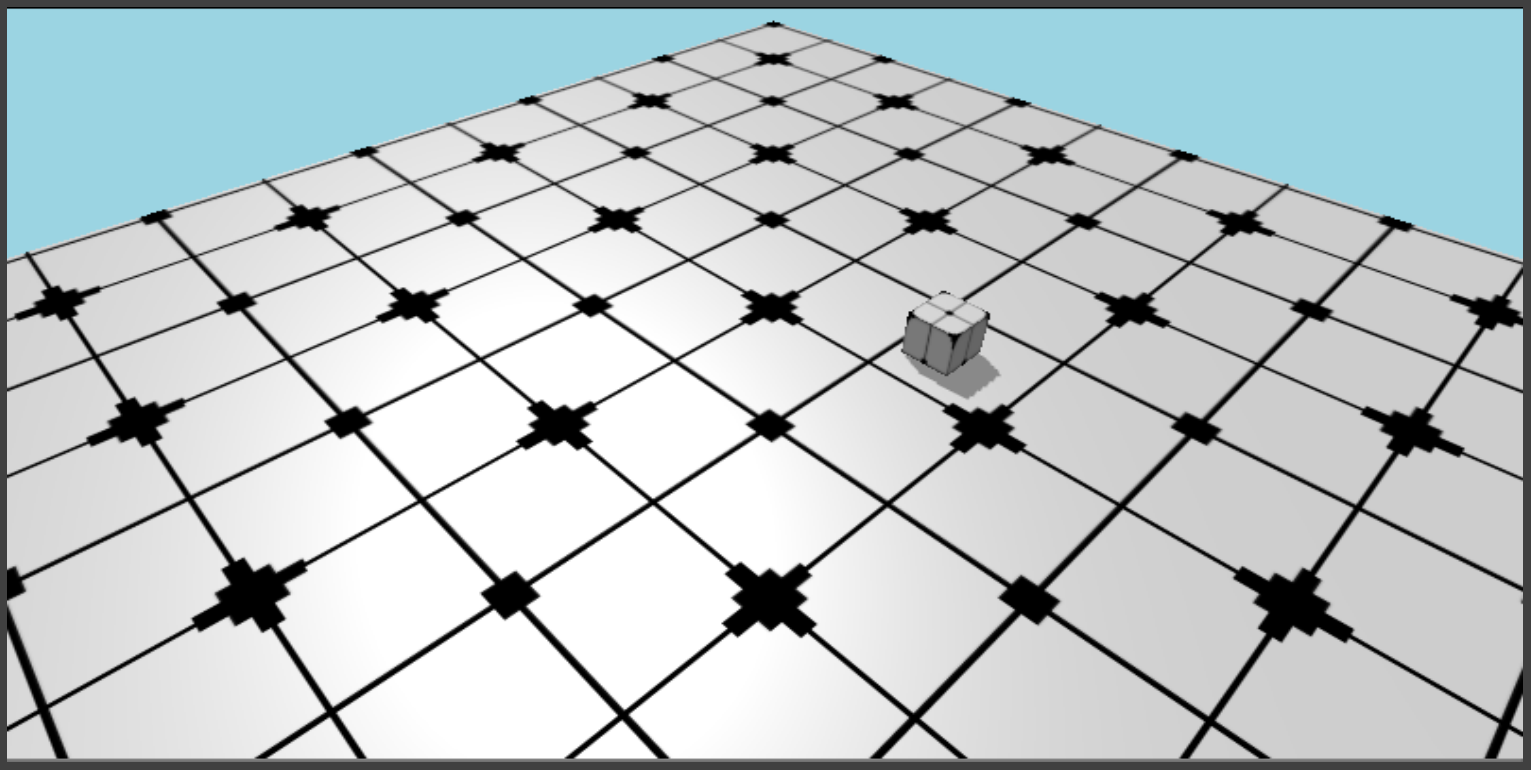
A propriedade “ground” foi colocada como exemplo mais pode ser qualquer uma.



Para um objeto seguir a posição do mouse é preciso ter um sensor do tipo (Mouse Over Any), com alguma propriedade que seja um plano ou o chão onde o personagem vai pisar isso é necessário para o objeto não vim em direção á câmera sem ordem.

Sabendo disso agora vamos precisar de duas como posso dizer classes de posição geométricas que também trabalham com vetor3 que é (worldPosition) essa variável contém todas as posições que o objeto pode ter em relação ao universo 3D do blender/(x,y,z) e a outras é uma das funções do sensor (MouseOverAny) que é (MouseOverAny.hitPoisition) e contem todos os hit que que o mouse detecta em sua posição e isso é bom pois tem base de vertor3 também /(x,y,z). Veja o exemplo com o sistema pronto:



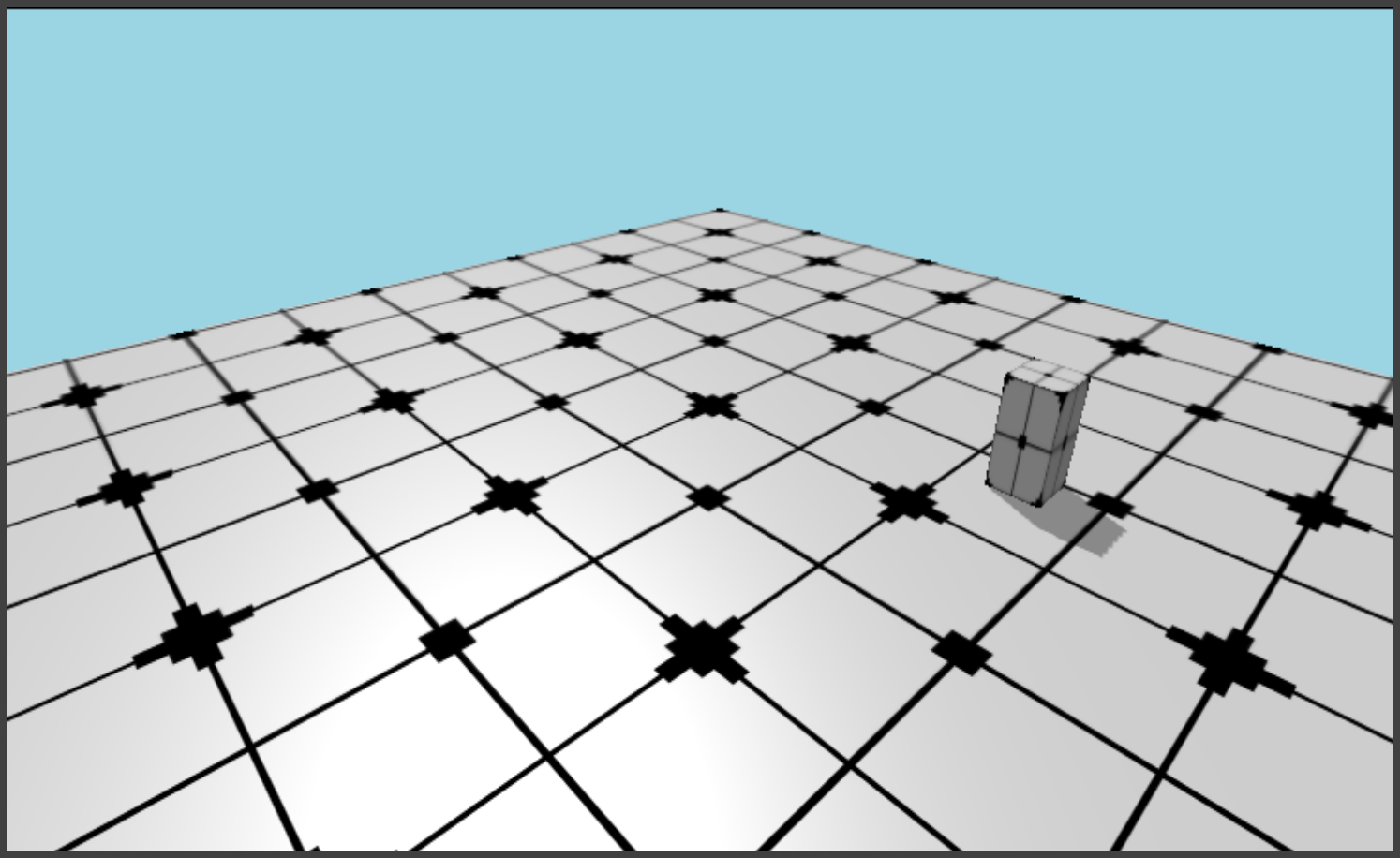


Mais note que o objeto está pela metade, pois, o resto está entrando no plano para resolver isso é muito simples você só precisar usar dois vetores em vez dos três, ou seja, (X, Y) e como fazer isso é simples:

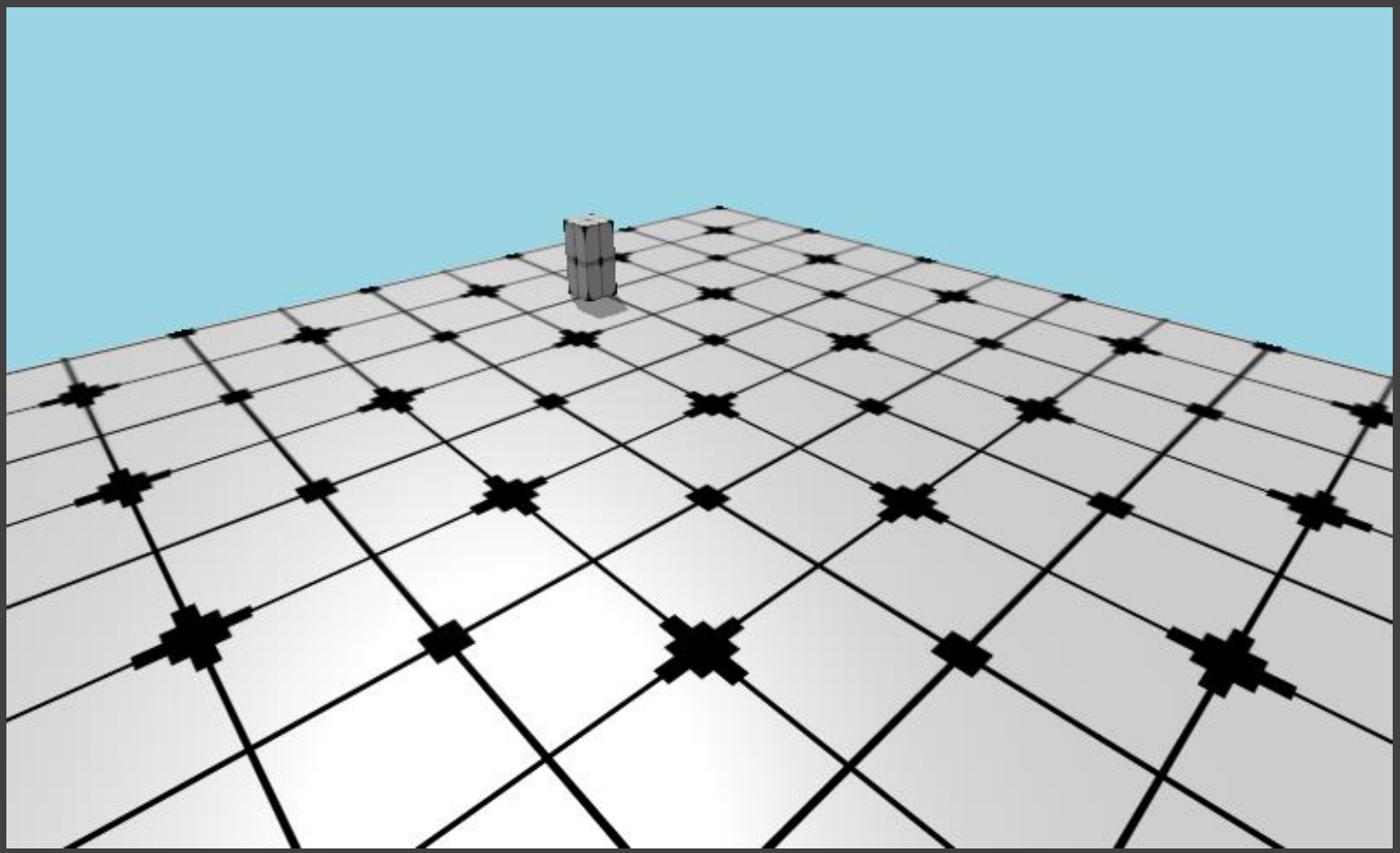
```
10
11
12 def Update(cont):
13     own = cont.owner
14     scene = logic.getCurrentScene()
15     teclado = logic.keyboard.events
16     m = logic.mouse.events
17     ############################
18     ###### Sensors ############
19     MouseOver = cont.sensors["MouseOver"]
20     ###### Actuators ############
21
22     ############################
23
24     if MouseOver.positive:
25         own.worldPosition.x = MouseOver.hitPosition.x
26         own.worldPosition.y = MouseOver.hitPosition.y
27
28
29
30
31
32
```

Ao colocar (.x , .y) em seus respectivos lugares você está a usar de forma direta do vetor desejado tanto no (worldPosition como no hitPosition) como o exemplo acima .

Posição 1:



Posição 2:



Se por acaso esse você está a usar e método em um personagem jogável mais não sabe como poder fazer par o personagem voltar a se movimentar com as teclas que você definiu no teclado basta criar uma condição (else:) logo abaixo do (if) pois é assim que funciona as estruturas de condição (if , elif e else:).

Como usar funções dos sensors no python:

Ao criar um sensor seja qual ele for você pode acessar quase todas as funcionalidades do próprio para usar ao seu favor como o exemplo acima que usamos o sensor MouseOver no tipo MouseOverAny e isso é sim possível com quase todos os sensores por que quase pois temos que ver o que sim podemos acessar na verdade para isso procure na API o sensor que deseja e o abra e veja o que se pode usar:

```
# Para conectar um sensor a este controlador.
# "sensorname" é o nome do sensor conforme definido na interface do Blender.
# + -------------------- + + -------- +
# | Sensor "nome do sensor" + - + Python +
# + -------------------- + + -------- +
sens  =  cont . sensores [ "sensorname" ]

# Para obter uma sequência de todos os sensores:
sensores  =  co . sensores
```

Consulte a referência do sensor para obter os métodos disponíveis:

- KX_MouseFocusSensor
- KX_NearSensor
- KX_NetworkMessageSensor
- KX_RadarSensor
- KX_RaySensor
- KX_TouchSensor
- SCA_DelaySensor
- SCA_JoystickSensor
- SCA_KeyboardSensor
- SCA_MouseSensor
- SCA_PropertySensor
- SCA_RandomSensor

No caso do MouserOver mostrado acima está localizado em (KX_MouseFocusSensor) ao abria ele no nome azulado você verá tudo o que pode usar.

KX_MouseFocusSensor (SCA_MouseSensor) ¶

classe base - SCA_MouseSensor

classe bge.types.**KX_MouseFocusSensor**(*SCA_MouseSensor*)

O sensor de foco do mouse detecta quando o mouse está sobre o objeto de jogo atual.

O sensor de foco do mouse funciona transformando as coordenadas do mouse do espaço do dispositivo 2D para o espaço 3D e depois transmitindo o raio para longe da câmera.

raySource
A fonte espacial do raio no mundo (a posição da vista).

Tipo: lista (vetor de 3 carros alegóricos)

rayTarget
O alvo do raio no espaço mundial.

Tipo: lista (vetor de 3 carros alegóricos)

rayDirection
O rayTarget- raySourcenormalizado.

Tipo: lista (vetor normalizado de 3 carros alegóricos)

hitObject
o último objeto em que o mouse acabou.

Tipo: KX_GameObject ou nenhum

hitPosition
A posição do espaço no mundo do raio intersecton.

Tipo: lista (vetor de 3 carros alegóricos)

hitNormal

o espaço do mundo normal da face no ponto de interseção.

Tipo: lista (vetor normalizado de 3 carros alegóricos)

hitUV

as coordenadas UV no ponto de interseção.

Tipo: lista (vetor de 2 carros alegóricos)

Se o objeto não tiver mapeamento UV, ele retornará [0, 0].

As coordenadas UV não são normalizadas, elas podem ser <0 ou> 1, dependendo do mapeamento UV.

usePulseFocus

Quando ativado, mover o mouse sobre um objeto diferente gera um pulso. (usado apenas quando a opção do sensor 'Passe o mouse sobre qualquer' está definida).

Tipo: boleano

useXRay

Se ativado, permite que o sensor veja objetos do jogo que não possuem a propriedade ou o material selecionado.

Tipo: boleano

propName

A propriedade ou material que o sensor está procurando.

Tipo: corda

useMaterial

Determina se o sensor está procurando uma propriedade ou material. KX_True = Localizar material; KX_False = Localizar propriedade.

Tipo: boleano

Como pode ver acima todas essas funções você pode usar, mais fica a pergunta como? É simples um sensor praticamente é uma classe, tendo isso em mente use essas funcionalidades como uma função que literalmente é o processo é simples:

Ou chamar um sensor como uma variável (MouseOver = cont.sensors[“MouseOver”]) você só basta agora usar o nome da variável que você criou seguido do ponto exemplo:

MouseOver = cont.sensors[“MouseOver”]

MouseOver.”função desejada”

Exemplos :

MouseOver.hitObject

MouseOver.hitPosition

MouseOver.hitNormal

E outros, porém deves saber o que a função retorna para teres o resultado que desejas para isso consulte a API do sensor que desejas usar e logo abaixo do nome da função terá algumas informação que mostras o que a função retorna para você usar. Exemplo:

hitPosition

A posição do espaço no mundo do raio intersecton.

Tipo: lista (vetor de 3 carros alegóricos)