

# World of Stairs

## API Documentation

January 12, 2011

(also available on Botanik-Guerilla.org)

## Contents

<b>Contents</b>	<b>1</b>
<b>1 Module MyConfigConnector</b>	<b>4</b>
1.1 Variables . . . . .	4
1.2 Class MY_SingleConfigConnector . . . . .	4
1.2.1 Methods . . . . .	4
1.2.2 Properties . . . . .	6
1.2.3 Instance Variables . . . . .	6
<b>2 Module MyConstants</b>	<b>7</b>
2.1 Variables . . . . .	7
2.2 Class MY_Constants . . . . .	7
2.2.1 Methods . . . . .	7
2.2.2 Properties . . . . .	7
2.2.3 Class Variables . . . . .	7
<b>3 Module MyControlKeys</b>	<b>8</b>
3.1 Variables . . . . .	8
3.2 Class MY_SingleControlKeys . . . . .	8
3.2.1 Methods . . . . .	8
3.2.2 Properties . . . . .	9
3.2.3 Instance Variables . . . . .	9
<b>4 Module MyGameActiveHidden</b>	<b>11</b>
4.1 Variables . . . . .	11
4.2 Class MY_SkyLight . . . . .	11
4.2.1 Methods . . . . .	11
4.2.2 Instance Variables . . . . .	12
4.3 Class MY_GameItem . . . . .	12
4.3.1 Methods . . . . .	13
4.3.2 Instance Variables . . . . .	14
4.4 Class MY_GameMerchItem . . . . .	14
4.4.1 Methods . . . . .	15
4.4.2 Instance Variables . . . . .	15
4.5 Class MY_GameHurdle . . . . .	15

4.5.1	Methods . . . . .	16
4.5.2	Instance Variables . . . . .	17
<b>5</b>	<b>Module MyGameActiveParts</b>	<b>18</b>
5.1	Variables . . . . .	18
5.2	Class MY_GameCharacter . . . . .	18
5.2.1	Methods . . . . .	18
5.2.2	Instance Variables . . . . .	22
5.3	Class MY_SkySphere . . . . .	24
5.3.1	Methods . . . . .	24
5.3.2	Instance Variables . . . . .	25
5.4	Class MY_GameGauge . . . . .	25
5.4.1	Methods . . . . .	25
5.4.2	Instance Variables . . . . .	26
<b>6</b>	<b>Module MyGameObjects</b>	<b>27</b>
6.1	Variables . . . . .	27
6.2	Class MY_GameObjectActive . . . . .	27
6.2.1	Methods . . . . .	27
6.2.2	Instance Variables . . . . .	29
6.3	Class MY_GameObjectActiveHidden . . . . .	29
6.3.1	Methods . . . . .	29
6.3.2	Instance Variables . . . . .	30
6.4	Class MY_GameObjectPassive . . . . .	31
6.4.1	Methods . . . . .	31
6.4.2	Instance Variables . . . . .	33
6.5	Class MY_GameCamera . . . . .	35
6.5.1	Methods . . . . .	35
6.5.2	Instance Variables . . . . .	37
6.6	Class MY_GameLight . . . . .	37
6.6.1	Methods . . . . .	37
6.6.2	Instance Variables . . . . .	38
<b>7</b>	<b>Module MyGamePassiveParts</b>	<b>40</b>
7.1	Variables . . . . .	40
7.2	Class MY_GamePlatform . . . . .	40
7.2.1	Methods . . . . .	40
7.2.2	Instance Variables . . . . .	42
7.3	Class MY_GameStair . . . . .	43
7.3.1	Methods . . . . .	43
7.3.2	Instance Variables . . . . .	45
<b>8</b>	<b>Module MyGameTimer</b>	<b>46</b>
8.1	Variables . . . . .	46
8.2	Class MY_SingleTimer . . . . .	46
8.2.1	Methods . . . . .	46
8.2.2	Properties . . . . .	47
8.2.3	Class Variables . . . . .	47
<b>9</b>	<b>Module MyGameUtils</b>	<b>48</b>
9.1	Functions . . . . .	48
9.2	Variables . . . . .	52

<b>10 Module MyGameWorld</b>	<b>53</b>
10.1 Variables . . . . .	53
10.2 Class MY_SingleGameWorld . . . . .	53
10.2.1 Methods . . . . .	53
10.2.2 Properties . . . . .	54
10.2.3 Instance Variables . . . . .	54
<b>11 Module MyScene</b>	<b>56</b>
11.1 Variables . . . . .	56
11.2 Class MY_SingleScene . . . . .	56
11.2.1 Methods . . . . .	56
11.2.2 Properties . . . . .	57
11.2.3 Instance Variables . . . . .	58
<b>12 Module MySpecialLists</b>	<b>59</b>
12.1 Variables . . . . .	59
12.2 Class MY_SingleIdList . . . . .	59
12.2.1 Methods . . . . .	59
12.2.2 Properties . . . . .	60
12.2.3 Instance Variables . . . . .	60
12.3 Class MY_SingleElementList . . . . .	60
12.3.1 Methods . . . . .	60
12.3.2 Properties . . . . .	61
12.3.3 Instance Variables . . . . .	61
12.4 Class MY_InventoryList . . . . .	62
12.4.1 Methods . . . . .	62
12.4.2 Instance Variables . . . . .	63

# 1 Module MyConfigConnector

## 1.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

## 1.2 Class MY\_SingleConfigConnector

object —  
MyConfigConnector.MY\_SingleConfigConnector

singleton class to parse the config file

**Author:** Niner

**Contact:** Admiral.Niner@gmail.com

### 1.2.1 Methods

**\_\_new\_\_**(*self*, \*args, \*\*kwargs)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.\_\_new\_\_ extit(inherited documentation)

**\_\_init\_\_**(*self*)

x.\_\_init\_\_(...) initializes x; see x.\_\_class\_\_.\_\_doc\_\_ for signature

Overrides: object.\_\_init\_\_ extit(inherited documentation)

**getConfig**(*self*)

checks if file exists and is writable otherwise create it with default values or throw exception when user has wrong privileges

**Return Value**

True if config was loaded, otherwise False

(*type=Boolean*)

**setDefault**(*self*)

set input keys to default (awsd) and fills properties with random values

**setConfig**(*self*)

writes config values to file

**setControlKeyValue**(*self, key, value*)

fills specific options in category 'keycontrol' with new values (dependent on the players orientation)

**Parameters**

**key:** represents the key in the in the control key section of the config file  
*(type=String)*

**value:** the value of the given key  
*(type=Integer)*

**getControlKeyValue**(*self, key*)

returns the value to the given key in the list of user's input keys

key = String

**Parameters**

**key:** the key out of the control key section  
*(type=String)*

**Return Value**

returns the value of the given key  
*(type=Integer)*

**setPowerUps**(*self, key, value*)

fills specific options in category 'powerups' with new values (collected player properties)

**Parameters**

**key:** represents the key in the in the powerups section of the config file  
*(type=String)*

**value:** the value of the the given key  
*(type=Integer)*

**getPowerUpValue**(*self, key*)

returns the value to the given key in the list of user's powerups

**Parameters**

**key:** the name of the powerup  
*(type=String)*

**Return Value**

returns the number of the given key  
*(type=Integer)*

**setMerch**(self, merch\_name)

fills specific options in category 'collecteditems' with new values (collected player merchandise)

**Parameters**

**merch\_name:** the name of the merchandise object  
(*type=String*)

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

**1.2.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**1.2.3 Instance Variables**

Name	Description
collectionList	a list of the collected merhcandise objects ( <i>type=List</i> )
keyList	a list of the ascii values for the specific keystrokes ( <i>type=List</i> )
loaded	True if config file was loaded and set ( <i>type=Boolean</i> )
powerupsList	a list of the number of collected powerups ( <i>type=List</i> )
ram	True if config file is not writable ( <i>type=Boolean</i> )

## 2 Module MyConstants

### 2.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 2.2 Class MY\_Constants



#### 2.2.1 Methods

##### *Inherited from object*

`--delattr--()`, `--format--()`, `--getattr--()`, `--hash--()`, `--init--()`, `--new--()`, `--reduce--()`,  
`--reduce_ex--()`, `--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

#### 2.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

#### 2.2.3 Class Variables


Name	Description
UP	<b>Value:</b> 1
DOWN	<b>Value:</b> 2
RIGHT	<b>Value:</b> 3

### 3 Module MyControlKeys

#### 3.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

#### 3.2 Class MY\_SingleControlKeys

object  **MyControlKeys.MY\_SingleControlKeys**  
 singleton class to control the keyboard input

##### 3.2.1 Methods

**--new--**(*self*, \**args*, \*\**kargs*)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.--new-- `exitit`(inherited documentation)

**--init--**(*self*)

`x.--init--(...)` initializes x; see `x.--class--.--doc--` for signature

Overrides: object.--init-- `exitit`(inherited documentation)

**initControlKeys**(*self*, *controller*)

set sensors to the right variables

**Parameters**

**controller:** the controller on which the keyboard sensors are connected to

(*type=SCA\_IController*)

**setControlKeys**(*self*)

reads ascii key values from the config list and sets it to the right memeber variables



**setControlKeysEffect**(*self*, *plane*=0, *opposite*=False)

change key assignments dependent on which plane is in use; normal use without any effect just call this method without any parameters

**Parameters**

**plane:** the plane number on which the user is right now  
(*type*=Integer [0,1,2])

**opposite:** whether it is the up or down on the specific plane  
(*type*=Boolean)

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

**3.2.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**3.2.3 Instance Variables**

Name	Description
BWD	keyboard sensor which listens to a specific keystroke ( <i>type</i> =SCA_KeyboardSensor)
BWD_key	ascii code of the keystroke for backward movement ( <i>type</i> =Integer)
DWD	keyboard sensor which listens to a specific keystroke ( <i>type</i> =SCA_KeyboardSensor)
DWD_key	ascii code of the keystroke for downward movement ( <i>type</i> =Integer)
DXR	keyboard sensor which listens to a specific keystroke ( <i>type</i> =SCA_KeyboardSensor)
DXR_key	ascii code of the keystroke for dexter movement ( <i>type</i> =Integer)

*continued on next page*

Name	Description
FWD	keyboard sensor which listens to a specific keystroke ( <i>type=SCA_KeyboardSensor</i> )
FWD_key	ascii code of the keystroke for forward movement ( <i>type=Integer</i> )
SNL	keyboard sensor which listens to a specific keystroke ( <i>type=SCA_KeyboardSensor</i> )
SNL_key	ascii code of the keystroke for sinsitral movement ( <i>type=Integer</i> )
UWD	keyboard sensor which listens to a specific keystroke ( <i>type=SCA_KeyboardSensor</i> )
UWD_key	ascii code of the keystroke for upward movement ( <i>type=Integer</i> )
init	True if controlkeys were initialized ( <i>type=Boolean</i> )

## 4 Module MyGameActiveHidden

### 4.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 4.2 Class MY\_SkyLight

MyGameObjects.MY\_GameLight —  
 MyGameActiveHidden.MY\_SkyLight  
 class of the sky lights to simulate a sun movement

#### 4.2.1 Methods

**`--init--`**(*self, scene, element, name*)

##### Parameters

**scene:** the blender scene object which holds all objects  
*(type=KX\_Scene)*

**element:** the name of the light object which shall included into the MY\_SingleScene class  
*(type=String)*

**name:** the name the object shall have in MY\_SingleScene  
*(type=String)*

Overrides: MyGameObjects.MY\_GameLight.`--init--`

**`setEnergyFade`**(*self, max\_amount, duration*)

fades the energy of the light object to a given amount in a given time

##### Parameters

**max\_amount:** amount to fade to  
*(type=Float)*

**duration:** duration which shall be used to fade the light's energy  
*(type=Float)*

**setRotation**(*self, plane, radius, pivot*)

set an infinty rotation for day night effect rotation axis depends on plane

**Parameters**

**plane:** the plane on which the light shall rotate  
*(type=Integer [0,1,2])*

**radius:** the radius for the rotation  
*(type=Float)*

**pivot:** the middle of the circle movement  
*(type=Vector)*

***Inherited from MyGameObjects.MY\_GameLight(Section 6.6)***

deleteObject(), getPosition(), setEnergy(), setOrientation(), setParent()

#### 4.2.2 Instance Variables

Name	Description
fadeStart	the start time for fading lights (on/off) <i>(type=Float)</i>
isFading	whether the light is fading or not <i>(type=Boolean)</i>
oldEnergy	the starting energy of the light object <i>(type=Float)</i>
<i>Inherited from MyGameObjects.MY_GameLight (Section 6.6)</i> gObject, name, type	

### 4.3 Class MY\_GameItem

MyGameObjects.MY\_GameObjectActiveHidden —  
 MyGameActiveHidden.MY\_GameItem

class of the game items that can be collected by the player and are positioned onm  
 top/bottom of platforms

## 4.3.1 Methods

---

**\_\_init\_\_**(*self*, *scene*, *element\_name*, *parent\_object*)

---

**Parameters**

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**element\_name:** the name of the light object which shall included  
into the MY\_SingleScene class  
(*type=String*)

**parent\_object:** the game object the item shall parented to  
(*type=MY\_GameObject*)

---

 Overrides: MyGameObjects.MY\_GameObjectActiveHidden.\_\_init\_\_

---



---

**getPosition**(*self*)

---

get the world position of the item

**Return Value**

the vector of the current world position  
(*type=Vector*)

---

 Overrides: MyGameObjects.MY\_GameObjectActiveHidden.getPosition

---



---

**setPosition**(*self*)

---

set an item at top or bottom of a parented platform

---



---

**setRotation**(*self*)

---

sets the smooth infinite rotation in the right plane of the item object

---



---

**setParentStatus**(*self*, *has\_item=False*, *item\_object=None*)

---

sets the status of the parent object, if it has an item or not and when 'yes',  
assign the item to it

**Parameters**

**has\_item:** True if the parent object will have an assigned item  
(*type=Boolean*)

**item\_object:** normally this item (self)  
(*type=MY\_GameObject*)

<b>deleteObject</b> ( <i>self</i> )
-------------------------------------

deletes object, removes it from the scene and frees its id
--

Overrides: MyGameObjects.MY_GameObjectActiveHidden.deleteObject
---

**Inherited from MyGameObjects.MY\_GameObjectActiveHidden(Section 6.3)**

getAlive(), setLinearVelocity()

#### 4.3.2 Instance Variables

Name	Description
parentObject	the items parent object to know which object can be relieved of the item ( <i>type=MY_GameObject</i> )
positionRotationX	specifies the tilt of the item dependent on its ground plane ( <i>type=Float</i> )
positionRotationY	specifies the tilt of the item dependent on its ground plane ( <i>type=Float</i> )
rotationDuration	how long shall it take to make a full circle rotation ( <i>type=Float</i> )
rotationStart	start time of the circle rotation ( <i>type=Float</i> )
worldPlane	the number of the its current plane ( <i>type=Integer (0,1,2)</i> )
<i>Inherited from MyGameObjects.MY_GameObjectActiveHidden (Section 6.3)</i> elementId, gObject, lifeTime, name, type	

#### 4.4 Class MY\_GameMerchItem

MyGameObjects.MY\_GameObjectActiveHidden — **MyGameActiveHidden.MY\_GameMerchItem**  
 class of the game merchandise items that can be collected by the player

#### 4.4.1 Methods

**\_\_init\_\_**(*self*, *scene*, *spawn\_point*)

**Parameters**

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**spawn\_point:** the object where the merch item shall be spawned  
(*type=KX\_GameObject*)

Overrides: MyGameObjects.MY\_GameObjectActiveHidden.\_\_init\_\_

**setPosition**(*self*, *position*)

set an item at top or bottom of a parented platform

**Parameters**

**position:** the vector of the new world position  
(*type=Vector*)

**deleteObject**(*self*)

deletes object, removes it from the scene and frees its id

Overrides: MyGameObjects.MY\_GameObjectActiveHidden.deleteObject

**Inherited from MyGameObjects.MY\_GameObjectActiveHidden (Section 6.3)**

getAlive(), getPosition(), setLinearVelocity()

#### 4.4.2 Instance Variables

Name	Description
<i>Inherited from MyGameObjects.MY_GameObjectActiveHidden (Section 6.3)</i>	
elementId, gObject, lifeTime, name, type, worldPlane	

### 4.5 Class MY\_GameHurdle

MyGameObjects.MY\_GameObjectActiveHidden

MyGameActiveHidden.MY\_GameHurdle

class of the enemy (hurdle) with its artificial intelligence or something like that ;)

## 4.5.1 Methods

---

**\_\_init\_\_**(*self, scene, element\_name, spawn\_point*)

---

**Parameters**

- scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)
- element\_name:** the name of the hurdle object which shall included into the MY\_SingleScene class  
(*type=String*)
- spawn\_point:** the object where the merch item shall be spawned  
(*type=KX\_GameObject*)

---

 Overrides: MyGameObjects.MY\_GameObjectActiveHidden.\_\_init\_\_

---



---

**setPosition**(*self, position*)

---

sets the world position of the hurdle

**Parameters**

- position:** the new position of the hurdle  
(*type=Vector*)

---

**setOrientation**(*self, prota\_pos*)

---

sets the hurdle z-axis to the world gravity orientation, so the hurdle looks right

**Parameters**

- protas\_pos:** the position of the player, where the hurdle looks at  
(*type=Vector*)
-



**setHurdle**(*self*, *prota\_pos*)

sets the hurdle onto the right way to the player

1. collision sphere (touch) detects items to eat and bullet hits

**Parameters**

**prota\_pos**: the position of the object the hurdle is after  
(*type=Vector*)

**Return Value**

1. 'nothing', if hurdle is up in the air and the 'bullet ohysic engine' takes over
2. 'item', if hurdle hits an item
3. 'ground', if hurdle hits the ground
4. 'dead', if hurdle was hit by a bullet

(*type=String*)

**setSpeedToPoint**(*self*)

sets the vectored speed of the hurdle

**deleteObject**(*self*)

deletes object, removes it from the scene and frees its id

Overrides: MyGameObjects.MY\_GameObjectActiveHidden.deleteObject

**Inherited from MyGameObjects.MY\_GameObjectActiveHidden (Section 6.3)**

getAlive(), getPosition(), setLinearVelocity()

#### 4.5.2 Instance Variables

Name	Description
groundObject	the current ground object of the hurdle ( <i>type=MY_GameObject</i> )
speed	the speed of the hurdle ( <i>type=Float</i> )
targetPoint	the point the hurdle has to go to ( <i>type=Vector</i> )
Inherited from MyGameObjects.MY_GameObjectActiveHidden (Section 6.3) elementId, gObject, lifeTime, name, type, worldPlane	

## 5 Module MyGameActiveParts

### 5.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 5.2 Class MY\_GameCharacter

MyGameObjects.MY\_GameObjectActive —  
MyGameActiveParts.MY\_GameCharacter

class of the main game character, which the user has to play

#### 5.2.1 Methods

**`--init--`**(*self, scene, element*)

**Parameters**

**scene:** the blender scene object which holds all objects

**element:** the name of the light object which shall included into the MY\_SingleScene class

Overrides: MyGameObjects.MY\_GameObjectActive.`--init--` exitit(inherited documentation)

**`setView`**(*self, window\_metric*)

sets the players view. to switch between first person and third person view

**Parameters**

**window\_metric:** a list of thw width and height of the game window

(*type=List [Integer, Integer]*)

**jump**(*self, speed, force*)

sets the jump digit of the speed vector with the given force

**Parameters**

**speed:** the speed vector of the player  
(*type=Vector*)

**force:** the power of the jump  
(*type=Integer*)

**Return Value**

the prepared speed vector  
(*type=Vector*)

**marioJump**(*self, hurdle\_pos*)

sets the mario jump, to get out of the way of an enemy after get hurt once

**Parameters**

**hurdle\_pos:** the global position of the hurdle  
(*type=Vector*)

**move**(*self, fwd, back, left, right, force*)

calculates the movement of the object, dependent on given input parameters (direction and force)

**Parameters**

**fwd:** the power of forward movement  
(*type=Boolean*)

**back:** the power of backward movement  
(*type=Boolean*)

**left:** the power of left movement  
(*type=Boolean*)

**right:** the power of right movement  
(*type=Boolean*)

**force:** the speed of the player's total movement  
(*type=Integer*)

**Return Value**

the prepared speed vector for the player's character  
(*type=Vector*)

**look**(*self*, *mouse\_screen\_pos*, *dtime*, *sensitivity*)

sets the players actual camera view dependent on user's mouse input, elapsed time and sensitivity

**Parameters**

**mouse\_screen\_pos:** the position of the mouse on the screen related to the window metric  
*(type=List [Integer, Integer])*

**dtime:** the delta of time between two logical ticks  
*(type=Float)*

**sensitivity:** the speed of turning your eyes around  
*(type=Float)*

**initOneighty**(*self*, *turn\_duration*)

initialize the 'Oneighty' with the right parameters and given duration it sets the used member variables with the right values to know how to rotate and the local rotation of the player afterwards and parents the player to the rotating ground object

**Parameters**

**turn\_duration:** how fast/slow the turn shall be  
*(type=Integer)*

**initWallRun**(*self*, *global\_collision\_side*, *hit\_plane*, *turn\_duration*)

initialize the wall turn. it sets the used member variables with the right values

**Parameters**

**global\_collision\_side:** where is the wall globally? (at least to know rotation axis)  
*(type=Integer (0,1,2,3))*

**hit\_plane:** the plane of the hit object, to know the player's world plane after rotation  
*(type=Integer (0,1,2))*

**turn\_duration:** time to rotate  
*(type=Integer)*

**planeTurn**(*self*)

do the Oneighty

---

**setInventoryItem**(*self*, *item\_name*, *add*)

---

 sets or removes given item into the inventory

**Parameters**

**item\_name:** the name of the item that was used/collected  
*(type=String)*

**add:** True if the item was collected, False if it was used  
*(type=Boolean)*

**Return Value**

'True' if item is in inventory, else 'False'  
*(type=Boolean)*

---

**setInventoryGauge**(*self*)

---

 refresh the inventory gauge on the screen and in the config parser

---

**getItemLifeTime**(*self*, *name*)

---

 sets the lifetime of an used item

**Parameters**

**name:** the name of the used item to get its lifetime  
*(type=String)*

**Return Value**

'True' if item is still alive, else 'False'  
*(type=Boolean)*

---

**setMerchItem**(*self*, *merch\_name*)

---

 adds a given merchandise item into the inventory

**Parameters**

**merch\_name:** the name of the merch item that was collected  
*(type=String)*

---

**setWarp**(*self*)

---

 do a space warp

---

**reloadAmmo**(*self*)

---

 reloads your ammo

*Inherited from MyGameObjects.MY\_GameObjectActive(Section 6.2)*

deleteObject(), getLinearVelocity(), getPosition(), getWorldOrientation(), setLinearVelocity(), setLocalOrientation(), setPosition()

### 5.2.2 Instance Variables

Name	Description
aim	True if player aims at a target, otherwise False ( <i>type=Boolean</i> )
ammo	the amount of bullets the player has in his gun ( <i>type=Integer</i> )
angleX	just a temporary variable to leave the angleXTotal variable untouched ( <i>type=Float</i> )
angleXTotal	the global rotation about the x-axis of the player (player rotates finally just about x and y) ( <i>type=Integer</i> )
angleY	just a temporary variable to leave the angleYTotal variable untouched ( <i>type=Float</i> )
angleYTotal	the global rotation about the y-axis of the player (player rotates finally just about x and y) ( <i>type=Integer</i> )
cosX	the cosinus amount of looking up and down ( <i>type=Float</i> )
flyTime	the time when you began to leave the ground ( <i>type=Float</i> )
globalRotationX	the rotation amount about x-axis when turning ( <i>type=Integer</i> )
globalRotationY	the rotation amount about y-axis when turning ( <i>type=Integer</i> )
godMode	True if player is in godmode; normally after a hurdle hit to stay alive and not to lose all your health ( <i>type=Boolean</i> )
godStartTime	the time when the player godmode started ( <i>type=Float</i> )
groundObject	the object that is right under the player's feet ( <i>type=MY_GameObject</i> )
hasVisor	True if player uses visor, else False ( <i>type=Boolean</i> )
inventoryList	a list that contains all amounts of your inventory items ( <i>type=MY_InventoryList</i> )

*continued on next page*

Name	Description
isCrouching	True if player is crouching ( <i>type=Boolean</i> )
isJumping	True if player is jumping ( <i>type=Boolean</i> )
lookCamera	the camera object the player uses for looking around (for example: first person or third person) ( <i>type=KX_Camera</i> )
mag	the total amount of the player's mag ( <i>type=Integer</i> )
oppositePlane	True if player is on one of the planes which gravity vector goes in th direction of z, -x or -y ( <i>type=Boolean</i> )
rotationPointObject	the object to set at the center of a rotation object while doing a oneighty ( <i>type=KX_GameObject</i> )
shotInterval	the interval of shots ( <i>type=Float</i> )
shotTime	the time when you shot a bullet ( <i>type=Float</i> )
sinX	the sinus amount of looking up and down ( <i>type=Float</i> )
stepDown	Amount of the time to get the player back to ground ( <i>type=Integer</i> )
stepUp	True if payer wants to go upstairs ( <i>type=Boolean</i> )
superGun	the player has collected the rare super gun in the game ( <i>type=Boolean</i> )
turnDuration	the duration of a turn ( <i>type=Integer</i> )
useSuperGun	True if the player is currently using the super gun, else False ( <i>type=Boolean</i> )
visorAnimStart	the time when player clicked the v-key to get his visor ( <i>type=Float</i> )
visualInput	the object to set at the screen to let the player know what orientation he/she/it has ( <i>type=KX_GameObject</i> )

continued on next page

Name	Description
visualInputObject	the actual object the payer will see as visual feedback for his/her orientation ( <i>type=KX_GameObject</i> )
wallStartTime	the start time of a wallturn ( <i>type=Float</i> )
wallTurn	True if player makes a ninety or oneighty turn ( <i>type=Boolean</i> )
wallTurnEnd	True if a turn just ended ( <i>type=Boolean</i> )
worldPlane	the players current plane ( <i>type=Integer (0,1,2)</i> )
zLook	amount of the rotation angle around the z-axis ( <i>type=Float</i> )
zTurn	the rotation about z-axis; just for calculating the wallturn while turning ( <i>type=Float</i> )
<i>Inherited from MyGameObjects.MY_GameObjectActive (Section 6.2)</i> gObject, name, type	

### 5.3 Class MY\_SkySphere

MyGameObjects.MY\_GameObjectActive —  
**MyGameActiveParts.MY\_SkySphere**  
 class of the unversum sphere which centers on the player

#### 5.3.1 Methods

**\_\_init\_\_(self, scene, element)**

**Parameters**

**scene:** the blender scene object which holds all objects

(*type=KX\_Scene*)

**element:** the name of the object which shall included into the MY\_SingleScene class

(*type=String*)

Overrides: MyGameObjects.MY\_GameObjectActive.\_\_init\_\_



<b>setAction</b> ( <i>self</i> , <i>axis</i> , <i>location</i> )
--

sets the sky dome rotation
----------------------------

<b>Parameters</b>
-------------------

<b>axis:</b>	the rotation axis of the sky sphere ( <i>type=Integer (0,1,2)</i> )
--------------	--

<b>location:</b>	the new position of the sphere's centre ( <i>type=Vector</i> )
------------------	---

***Inherited from MyGameObjects.MY\_GameObjectActive (Section 6.2)***

deleteObject(), getLinearVelocity(), getPosition(), getWorldOrientation(), setLinearVelocity(), setLocalOrientation(), setPosition()

### 5.3.2 Instance Variables

Name	Description
<i>Inherited from MyGameObjects.MY_GameObjectActive (Section 6.2)</i> gObject, name, type	

## 5.4 Class MY\_GameGauge

MyGameObjects.MY\_GameObjectActiveHidden —  
MyGameActiveParts.MY\_GameGauge

class of the gauge elements, like inventory, health etc.

### 5.4.1 Methods

<b>__init__</b> ( <i>self</i> , <i>scene</i> , <i>element</i> , <i>text=False</i> )
---

<b>Parameters</b>
-------------------

<b>scene:</b>	the blender scene object which holds all objects ( <i>type=KX_Scene</i> )
---------------	--

<b>text:</b>	True if it is a text otherwise it is a number ( <i>type=Boolean</i> )
--------------	--

Overrides: MyGameObjects.MY_GameObjectActiveHidden.__init__
---

<b>setText</b> ( <i>self</i> )
--------------------------------

sets the text to be displayed in the player's gauge
---

<b>setVisible</b> ( <i>self</i> , <i>visible</i> )
--

sets the visibility of a gauge item
-------------------------------------

<b>Parameters</b>
-------------------

<b>visible:</b> sets the visibility of the text (True = visible; False = not visible)
---

( <i>type=Boolean</i> )
-------------------------

<b>setPosition</b> ( <i>self</i> , <i>position</i> )
--

sets the world position
-------------------------

<b>Parameters</b>
-------------------

<b>position:</b> the new world position of the text
---

( <i>type=Vector</i> )
------------------------

**Inherited from MyGameObjects.MY\_GameObjectActiveHidden (Section 6.3)**

deleteObject(), getAlive(), getPosition(), setLinearVelocity()

#### 5.4.2 Instance Variables

Name	Description
name	the name that represents the object in MY_SingleScene ( <i>type=String</i> )
theText	the text the gauge shall display (Integer for gauge_num and String for gauge_text) ( <i>type=String or Integer</i> )
Inherited from MyGameObjects.MY_GameObjectActiveHidden (Section 6.3) elementId, gObject, lifeTime, type, worldPlane	

## 6 Module MyGameObjects

### 6.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 6.2 Class MY\_GameObjectActive

**Known Subclasses:** MyGameActiveParts.MY\_GameCharacter,  
MyGameActiveParts.MY\_SkySphere

base class of all the objects that are dynamic and NOT temporary

#### 6.2.1 Methods

**`--init--(self, scene, element)`**

**Parameters**

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**element:** the name of the light object which shall included into the  
MY\_SingleScene class  
(*type=String*)

**`setLocalOrientation(self, orientation)`**

sets the local orientation

**Parameters**

**orientation:** the new local orientation matrix  
(*type=Matrix*)

**`getWorldOrientation(self)`**

gets the world orientation

**Return Value**

a 3x3 matrix of the current orientation  
(*type=Matrix*)

**getPosition(self)**

gets the world position

**Return Value**

a vector of the current world position

(*type=Vector*)

**setPosition(self, position)**

sets the world position

**Parameters**

**position:** the new world position of the object

(*type=Vector*)

**getLinearVelocity(self, local)**

if 'local = True' it gets the local linear velocity otherwise it gets the global linear velocity

**Parameters**

**local:** True if you want the local linear velocity, otherwise it is global

(*type=Boolean*)

**Return Value**

the current linear velocity in all three directions

(*type=List [vx, vy, vz]*)

**setLinearVelocity(self, speed, local=0)**

if 'local = True' it sets the local linear velocity otherwise it sets the global linear velocity

**Parameters**

**speed:** the max speed for all three directions

(*type=Vector*)

**local:** True if the local velocity shall be set, otherwise global

(*type=Boolean*)

**deleteObject(self)**

deletes object and removes it from the scene

### 6.2.2 Instance Variables

Name	Description
gObject	a blender game object ( <i>type=KX_GameObject</i> )
name	the name that represents the object in MY_SingleScene ( <i>type=String</i> )
type	a name which identifies the object as a specific class in MY_SingleScene ( <i>type=String</i> )

## 6.3 Class MY\_GameObjectActiveHidden

**Known Subclasses:** MyGameActiveHidden.MY\_GameHurdle, MyGameActiveHidden.MY\_GameItem, MyGameActiveHidden.MY\_GameMerchItem, MyGameActiveParts.MY\_GameGauge

base class of all the objects that are dynamic but have a lifetime, so they are temporary ingame

### 6.3.1 Methods

**\_\_init\_\_**(*self, scene, element, spawn\_point=0*)

#### Parameters

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**element:** String  
(*type=the name of the object*)

**spawn\_point:** the object where the object shall be spawned  
(*type=KX\_GameObject*)

**getAlive**(*self, prota\_pos*)

checks if object's lifetime is expired or object is out of game view

#### Return Value

True if the item is still alive, otherwise False  
(*type=Boolean*)

**getPosition(*self*)**

gets the world position

**Return Value**

the current world position of the object

(*type=Vector*)

**setLinearVelocity(*self*, *speed*, *local=True*)**

just a forwarding of the blender setLinearVelocity

**Parameters**

**speed:** a vector with the speed of every single direction

(*type=Vector*)

**local:** True if velocity has effect just locally, else globally

(*type=Boolean*)

**deleteObject(*self*)**

deletes object, removes it from the scene and frees its id

**6.3.2 Instance Variables**

Name	Description
elementId	the unique identifier for this object ( <i>type=Integer</i> )
gObject	a blender game object ( <i>type=KX_GameObject</i> )
lifeTime	specifies the lifetime of the spawned item (negative = infinite) ( <i>type=Integer</i> )
name	the name that represents the object in MY_SingleScene ( <i>type=String</i> )
type	a name which identifies the object as a specific class in MY_SingleScene ( <i>type=String</i> )
worldPlane	the number of the its current plane ( <i>type=Integer (0,1,2)</i> )

## 6.4 Class *MY\_GameObjectPassive*

**Known Subclasses:** *MyGamePassiveParts.MY\_GamePlatform*,  
*MyGamePassiveParts.MY\_GameStair*

base class of all the objects in the game that are part of the environment, like platforms and stairs

### 6.4.1 Methods

***\_\_init\_\_***(*self*, *scene*, *element*)

**Parameters**

**scene:** the blender scene object which holds all objects  
*(type=KX\_Scene)*

**element:** the name of the object  
*(type=String)*

***getPlane***(*self*)

gets the world plane the object is in

**Return Value**

- returns '0' for xy plane
- returns '1' for xz plane
- returns '2' for yz plane

*(type=Integer (0,1,2))*

***setNewPlaneOnRotation***(*self*, *prev\_plane*, *connected\_dir*)

sets new plane after orthogonal rotation dependent on the plane of connected object and its connected direction

**Parameters**

**prev\_plane:** the plane of the connected object after ninety degree turn (dont know how to explain)  
*(type=Integer (0,1,2))*

**connected\_dir:** on which side is the connection (0:+y, 1:+x, 2:-y, 3:-x)  
*(type=Integer (0,1,2,3))*

**getPosition(*self*)**

gets the world position

**Return Value**

the vector of the current world position

(*type=Vector*)

**setConnectedSide(*self*, *connected\_side*, *connected\_object*)**

sets list for connected sides with the right id at the right position in the array

**Parameters**

**connected\_side:** on which side is the connection (0:+y, 1:+x, 2:-y, 3:-x)

(*type=Integer (0,1,2,3)*)

**connected\_object:** the object which is connected to THIS object

(*type=MY\_GameObject*)

**delConnectedSide(*self*, *connected\_object*)**

deletes entry out of the connected side list

**Parameters**

**connected\_object:** the connected object that was deleted

(*type=MY\_GameObject*)

**initPop(*self*, *pop\_steps*, *pop\_start*, *duration*)**

initialize popping when object is spawned

**Parameters**

**pop\_steps:** how many steps are needed to pop up/down the object

(*type=Integer (> 0)*)

**pop\_start:** start time of popping

(*type=Float*)

**duration:** for how long shall the object pop

(*type=Float*)



**pop**(*self*, *do\_pop*, *natal*=True)

pops the object with its initialized parameters

**Parameters**

**do\_pop:** do the pop (True) or not (False)  
(*type=Boolean*)

**natal:** whether it is the objects birth (True) or dead (False)  
(*type=Boolean*)

**getChildren**(*self*)

gets the direct children which are connected to the object

**Return Value**

the list of the objects children  
(*type=List*)

**setAlive**(*self*, *alive*)

sets the object and its children to dead if necessary or the object to alive

**Parameters**

**alive:** True if the object will live further more  
(*type=Boolean*)

**deleteObject**(*self*, *now*=False, *recursive\_delete*=True)

deletes the object. if a delay is given, the object will be deleted after the delay. with 'now' deletion can be forced all children can be deleted by setting the recursive\_delete variable

**Parameters**

**now:** False if the deletion will be delayed, otherwise delete it NOW  
(*type=Boolean*)

**recursive\_delete:** True if children will also been deleted  
(*type=Boolean*)

**Return Value**

returns 'True' if object was deleted, otherwise 'False'  
(*type=Boolean*)

#### 6.4.2 Instance Variables

Name	Description
alive	True if object is destined for deletion (in the future) ( <i>type=Boolean</i> )
boundingBox	contains all points of the bounding box which delimitates the object ( <i>type=List</i> )
connectedSides	a dictionary (in form of a quadrupel) which contains the connected objects (otherwise: None) in specific way {north=+y, east=+x, south=-y, west=-x} ( <i>type=Dict</i> )
curPopStep	the current pop step ( <i>type=Integer</i> )
curScale	the current scale factor ( <i>type=Vector</i> )
deleteDelay	the amount of the delay for deleting the object; set to '<0' for no deletion ( <i>type=Integer</i> )
elementId	a unique identifier ( <i>type=Integer</i> )
gObject	a blender game object ( <i>type=KX_GameObject</i> )
hasItem	True if the object has an item to administrate while deletion, otherwise False ( <i>type=Boolean</i> )
hurdleObject	the hurdle object that is on THIS object (for oneighty turns), if no hurdle is near this variable is 'None' ( <i>type=MY_GameObject</i> )
isRotated	three states: 0 = up; 1 = down; 2 = normal (in the xy-plane) ( <i>type=Integer (0,1,2)</i> )
item	the item object that is on top/bottom of tis object ( <i>type=MY_GameObject</i> )
name	the name of the object ( <i>type=String</i> )
origScale	the original scale factor ( <i>type=Vector</i> )
popDuration	the duration of popping ( <i>type=Integer</i> )

continued on next page

Name	Description
popStart	the start time of popping ( <i>type=Float</i> )
popSteps	the number of steps for popping ( <i>type=Integer</i> )
rootConnectObject	the connected object that leads to the players ground object ( <i>type=MY_GameObject</i> )
type	a name which identifies the object as a specific class in MY_SingleScene ( <i>type=String</i> )
worldDirection	the global adjustment of the object (plane 0: 0=+y; 1=+x; 2=-y; 3=+x / plane 1: 0=+z; 1=+x; 2=-z; 3=+x / plane 2: 0=+z; 1=+y; 2=-z; 3=+y) ( <i>type=Integer (0,1,2,3)</i> )
worldPlane	the objects plane information (0=xy; 1=xz; 2=yz) ( <i>type=Integer (0,1,2)</i> )

## 6.5 Class MY\_GameCamera

class for all the cameras used in the game

### 6.5.1 Methods

**\_\_init\_\_**(*self, scene, element*)

#### Parameters

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**element:** the name of the object  
(*type=String*)

**pointInsideFrustum**(*self*, *point*)

checks if a given point is inside the cameras frustum

**Parameters**

**point:** the point that will be checked if it is inside the camera frustum  
(*type=Vector*)

**Return Value**

- returns OUTSIDE, if point is outside
- returns INSIDE, if point is inside

(*type=String*)

**boxInsideFrustum**(*self*, *box*)

checks if a given box is inside/intersect/outside the cameras frustum

**Parameters**

**box:** the box parameters that will be checked  
(*type=List of Vectors*)

**Return Value**

- returns 0, if box is outside
- returns 2, if box intersects
- returns 1, if box is inside

(*type=Integer*)

**setLocalOrientation**(*self*, *orientation*)

sets the local orientation

**Parameters**

**orientation:** the new 3x3 orientation matrix  
(*type=Matrix*)

**getPosition**(*self*)

gets the local position

**Return Value**

the vector of the current local position

(*type=Vector*)

<b>deleteObject</b> ( <i>self</i> )
-------------------------------------

deletes the object and removes it from the scene
--

### 6.5.2 Instance Variables

Name	Description
gObject	a blender game object ( <i>type=KX_GameObject</i> )
name	the name that represents the object in MY_SingleScene ( <i>type=String</i> )
type	a name which identifies the object as a specific class in MY_SingleScene ( <i>type=String</i> )

## 6.6 Class MY\_GameLight

**Known Subclasses:** MyGameActiveHidden.MY\_SkyLight

base class of all the light objects in the game

### 6.6.1 Methods

<b>__init__</b> ( <i>self, scene, element, name</i> )
---

#### Parameters

**scene:** the blender scene object which holds all objects  
(*type=KX\_Scene*)

**element:** the name of the object  
(*type=String*)

**name:** the name that represents the object in MY\_SingleScene  
(*type=String*)

---

**setEnergy**(*self*, *energy*)

---

 sets the energy of the lamp

**Parameters**

**energy:** the energy of the light object  
*(type=Float)*

---

**setOrientation**(*self*, *angle*, *axis*)

---

 sets the local orientation

**Parameters**

**angle:** the new rotation angle  
*(type=Float)*

**axis:** the rotation axis  
*(type=String (x,y,z))*

---

**setParent**(*self*, *parent\_object*)

---

 sets objects as children of given parent object

**Parameters**

**parent\_object:** *(type=MY\_GameObject)*

---

**getPosition**(*self*, *local=1*)

---

 gets the local position, if 'local' is True, otherwise it gets the global position

**Parameters**

**local:** True if you want to get the local position of the object,  
 otherwise global  
*(type=Boolean)*

**Return Value**

the vector of the current global/local position  
*(type=Vector)*

---

**deleteObject**(*self*)

---

 deletes the object and removes it from the scene

### 6.6.2 Instance Variables

Name	Description
gObject	a blender game object ( <i>type=KX_GameObject</i> )
name	the name that represents the object in MY_SingleScene ( <i>type=String</i> )
type	a name which identifies the object as a specific class in MY_SingleScene ( <i>type=String</i> )

## 7 Module MyGamePassiveParts

### 7.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 7.2 Class MY\_GamePlatform

MyGameObjects.MY\_GameObjectPassive — **MyGamePassiveParts.MY\_GamePlatform**

class of the platform objects in the game for right rotation and translation etc.

#### 7.2.1 Methods

<b><code>--init--</code></b> ( <i>self</i> , <i>scene</i> )
<b>Parameters</b>
<b>scene</b> : the blender scene object where all other objects are connected to ( <i>type=KX_Scene</i> )
Overrides: MyGameObjects.MY_GameObjectPassive. <code>--init--</code>



---

**setElement**(*self, side\_location, prev\_plane, rotated, direct=0*)

---

initiates the calculation of position, orientation, midpoint, boundingbox and worldplane

**Parameters**

- side\_location:** to know exactly where the connection point is;  
that means what direction and at what position  
(*type=List[Integer (0,1,2,3), Vector]*)
- prev\_plane:** you have to know the previous plane (the plane of  
the object that this object is connected to) because  
of ninety degree turns  
(*type=Integer (0,1,2)*)
- rotated:** whether th object is rotated (0=up, 1=down) or it  
is just connected in the same plane (2=normal)  
(*type=Integer (0,1,2)*)
- direct:** True if the object shall be spawned even ther is no  
other object to connect to (for example at the  
beginning)  
(*type=Boolean*)

---

**setLocation**(*self, connected\_dir, location, prev\_plane, direct=0*)

---

sets the location by use of the MyGameUtils interface

**Parameters**

- connected\_dir:** one of the four directions an object can be  
connected to  
(*type=Integer (0,1,2,3)*)
- prev\_plane:** you have to know the previous plane (the plane of  
the object that this object is connected to) because  
of ninety degree turns  
(*type=Integer (0,1,2)*)
- location:** the position where the object shall be spawned  
(*type=Vector*)
- direct:** True if the object shall be spawned even ther is no  
other object to connect to (for example at the  
beginning)  
(*type=Boolean*)

**setOrientation**(*self*, *connected\_dir*, *location*, *prev\_plane*)

sets the orientation by use of the MyGameUtils interface

**Parameters**

**connected\_dir:** one of the four directions an object can be connected to  
(*type=Integer (0,1,2,3)*)

**prev\_plane:** you have to know the previous plane (the plane of the object that this object is connected to) because of ninety degree turns  
(*type=Integer (0,1,2)*)

**location:** the position where the object shall be spawned  
(*type=Vector*)

**getFreeSidesPositions**(*self*)

gets the positions of the not already connected sides of the object

**Return Value**

returns a list of all the free sides (direction and position)  
(*type=List[Integer (0,1,2,3), Vector]*)

**getBoundingBox**(*self*)

gets the object's bounding box (the cubic hull of the object)

**Return Value**

returns a list of vector points that represents the bounding box  
(*type=List*)

**Inherited from MyGameObjects.MY\_GameObjectPassive(Section 6.4)**

delConnectedSide(), deleteObject(), getChildren(), getPlane(), getPosition(), initPop(), pop(), setAlive(), setConnectedSide(), setNewPlaneOnRotation()

### 7.2.2 Instance Variables

Name	Description
freeSidesPosition	a list element that consists of the connected side and the position where the new object can be connected to ( <i>type=List [Integer (0,1,2,3), Vector]; not very attractive :()</i> )

*continued on next page*

Name	Description
midPoint	the center of the object (at least the right pivot, because at creation point of some elements the pivot is wrong. so you have to use this one to get the right rotations) ( <i>type=Vector</i> )
origScale	the original scale factor ( <i>type=Vector</i> )
<i>Inherited from MyGameObjects.MY_GameObjectPassive (Section 6.4)</i> alive, boundingBox, connectedSides, curPopStep, curScale, deleteDelay, elementId, gObject, hasItem, hurdleObject, isRotated, item, name, popDuration, popStart, popSteps, rootConnectObject, type, worldDirection, worldPlane	

### 7.3 Class MY\_GameStair

MyGameObjects.MY\_GameObjectPassive — **MyGamePassiveParts.MY\_GameStair**

class of all stair-like objects to set the right location and orientation etc.

#### 7.3.1 Methods

**\_\_init\_\_**(*self, scene, name, scale*)

##### Parameters

**scene**: the blender scene object where all other objects are connected to

(*type=KX\_Scene*)

**name**: the name the object shall have in MY\_SingleScene

(*type=String*)

**scale**: the new scaling factor to increase the variance of the stair objects

(*type=Integer*)

Overrides: MyGameObjects.MY\_GameObjectPassive.\_\_init\_\_

**setElement**(*self*, *side\_location*, *prev\_plane*)

initiates the calculation of position, orientation, midpoint, boundingbox and worldplane

**Parameters**

**side\_location:** to know exactly where the connection point is;  
that means what direction and at what position

(*type=List[Integer (0,1,2,3), Vector]*)

**prev\_plane:** you have to know the previous plane (the plane of  
the object that this object is connected to) because  
of ninety degree turns

(*type=Integer (0,1,2)*)

**getFreeSidesPositions**(*self*)

gets the positions of the not already connected sides of the object

**Return Value**

returns the free sides (direction and position)

(*type=List[Integer (0,1,2,3), Vector]*)

**getRightPlanePoint**(*self*, *point*)

gets the right point for a connection after all the creepy rotations of stairs and single steps :)

**Parameters**

**point:** the point that is calculated in the xy-plane before any  
rotation and translation of the object

(*type=Vector*)

**Return Value**

returns the right free side point

(*type=Vector*)

**getBoundingBox**(*self*)

gets the bounding box

**Return Value**

returns a list of all the vector points that represents the bounding  
box

(*type=List*)

<b>setMidPoint</b> ( <i>self</i> )
------------------------------------

sets th mid point for rotation (stair objects rotation point is at the first face!)
---

***Inherited from MyGameObjects.MY\_GameObjectPassive(Section 6.4)***

delConnectedSide(), deleteObject(), getChildren(), getPlane(), getPosition(), initPop(), pop(), setAlive(), setConnectedSide(), setNewPlaneOnRotation()

### 7.3.2 Instance Variables


Name	Description
isUp	True if the stair object goes upwards (will be initialized by a random for the lazy ones) ( <i>type=Boolean</i> )
midPoint	the center of the object (at least the right pivot, because at creation point of some elements the pivot is wrong. so you have to use this one to get the right rotations) ( <i>type=Vector</i> )
origScale	the original scale factor ( <i>type=Vector</i> )
<i>Inherited from MyGameObjects.MY_GameObjectPassive (Section 6.4)</i> alive, boundingBox, connectedSides, curPopStep, curScale, deleteDelay, elementId, gObject, hasItem, hurdleObject, isRotated, item, name, popDuration, popStart, popSteps, rootConnectObject, type, worldDirection, worldPlane	

## 8 Module MyGameTimer

### 8.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 8.2 Class MY\_SingleTimer


  
**MyGameTimer.MY\_SingleTimer**
  
 singleton class of the ingame timer; to know what time it is ;)

#### 8.2.1 Methods

**--new--**(*self*, \*args, \*\*kargs)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.--new-- `exitit`(inherited documentation)

**--init--**(*self*)

`x.--init--(...)` initializes x; see `x.--class--.--doc--` for signature

Overrides: object.--init-- `exitit`(inherited documentation)

**setCurrentTime**(*self*, *scene*)

sets the current game time

**Parameters**

**scene**: the blender game scene where all objects are bind to

(*type=KX\_Scene*)

<b>setPassedTime</b> ( <i>self</i> , <i>scene</i> )
---

sets the iassed ingame time
-----------------------------

<b>Parameters</b>
-------------------

<b>scene</b> : the blender game scene where all objects are bind to <i>(type=KX_Scene)</i>
---

***Inherited from object***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_reduce\_\_(), \_\_reduce\_ex\_\_(),  
 \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(), \_\_str\_\_(), \_\_subclasshook\_\_()

### 8.2.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 8.2.3 Class Variables

Name	Description
NOW	the current time in the game <b>Value:</b> 0 ( <i>type=Float</i> )
AGO	the last set current game time <b>Value:</b> 0 ( <i>type=Float</i> )

## 9 Module MyGameUtils

### 9.1 Functions

#### **GetBoxCollision**(*obj1, obj2*)

calculates the collision of game objects, like platforms and stairs

#### **Return Value**

'False' if there is a collision, else 'True'

(*type=Boolean*)

#### **SetReverseRootConnection**(*root\_ground*)

sets the right id as the root connected id

#### **Parameters**

**root\_ground:** the object that represents the current ground object of the player

(*type=MY\_GameObject*)

#### **GetGlobalOrientationToMe**(*source\_object, target\_object*)

gets the global direction of an object to me (at top, bottom, left, right, front or back) question is: where is an object from my point of view (globally)?

#### **Parameters**

**source\_object:** the source object to whom you want the orientation to (dont know how to explain!)

(*type=MY\_GameObject*)

**target\_object:** the object you want the orientation to (dont know how to explain!)

(*type=MY\_GameObject*)

#### **Return Value**

returns a string that represents the direction where to find the target object (top, bottom, left, right, front, back or none)

(*type=String*)



**SetPassivePartsConnection**(*connected\_side, source, target*)

sets the connected list of source and target object, so both know they are connected

**Parameters**

- connected\_side:** the connected side of source (passive) object  
where the target (passive) object is connected to  
(*type=Integer (0,1,2,3)*)
- source:** the object where the target is connected to  
(*type=MY\_GameObject*)
- target:** the object which is connected to source  
(*type=MY\_GameObject*)

**GetNormRotation**(*direction, plane, up=1*)

gets a normal rotation for an object dependent on the three planes and four direction; in the same plane as the connected object (mostly for stair objects)

**Parameters**

- direction:** the direction where an object is connected to  
(*type=Integer (0,1,2,3)*)
- plane:** the plane of the object that makes a normal rotation  
(*type=Integer (0,1,2)*)
- up:** True if object is turned upwards  
(*type=Boolean*)

**Return Value**

- returns the rotation matrix of a 'normal' rotation  
(*type=Matrix*)

**GetPlatfNormTransDelta**(*cur\_pos*, *direction*, *plane*, *o\_width*, *direct*=0)

gets the position of a platform by a normal translation with an offset (because of creepy center points)

**Parameters**

- cur\_pos:** the object's current position  
(*type*=*Vector*)
- direction:** the direction of the object where it is connected to  
(*type*=*Integer* (0,1,2,3))
- plane:** the plane of the object that makes the TransDelta  
(*type*=*Integer* (0,1,2))
- o\_width:** the width of an object in the xy-plane; only for platforms!!!  
(*type*=*Integer*)
- direct:** True if the object dont have to make a translation (for example: only one platform will be spawned in the game)  
(*type*=*Boolean*)

**Return Value**

- the new position after a normal translation with a delta  
(*type*=*Vector*)

**GetPlatfVertTransDelta**(*cur\_pos, direction, plane, o\_width, o\_height, up*)

gets the new position of a platform, when it's rotated

**Parameters**

- cur\_pos:** the object's current position  
(*type=Vector*)
- direction:** the direction of the object where it is connected to  
(*type=Integer (0,1,2,3)*)
- plane:** the plane of the object that makes the TransDelta  
(*type=Integer (0,1,2)*)
- o\_width:** the width of an object in the xy-plane; only for platforms!!!  
(*type=Integer*)
- o\_height:** the height of an object in the xy-plane; only for platforms!!!  
(*type=Integer*)
- up:** True if object is turned upward, False when downward (this 'def' is for platforms that are already defined as rotated)  
(*type=Boolean*)

**Return Value**

returns the new position after a translation with delta of an already rotated object  
(*type=Vector*)

**GetVertRotation**(*direction, plane*)

gets the rotation for an object if it is rotated (and changed to a new plane)

**Parameters**

- direction:** the direction of the object where it is connected to  
(*type=Integer (0,1,2,3)*)
- plane:** the plane of the object that makes this rotation  
(*type=Integer (0,1,2)*)

**Return Value**

returns the rotation matrix of a rotation when an object was already defined as rotated  
(*type=Matrix*)

## 9.2 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

## 10 Module MyGameWorld

### 10.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 10.2 Class MY\_SingleGameWorld

object —  
**MyGameWorld.MY\_SingleGameWorld**  
 singleton class of the game's world to set all its laws

#### 10.2.1 Methods

**--new--**(*self*, \*args, \*\*kargs)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.--new-- `exitit`(inherited documentation)

**--init--**(*self*)

`x.--init--(...)` initializes x; see `x.--class--.--doc--` for signature

Overrides: object.--init-- `exitit`(inherited documentation)

**setDefaultWorld**(*self*)

resets the whole game world to default values

**setGravityDirection**(*self*, *character\_orientation*)

sets the direction of gravity by *character\_orientation*

**Parameters**

**character\_orientation:** the current orientation of the player  
 character

(*type=Matrix*)

**setFeedBack**(*self*, *controller*, *feedback\_item*, *anim=False*)

sets the feedback pictures in the middle of the screen

**Parameters**

**controller:** SCA\_PythonController  
*(type=the controler where the ipo actuator is connected to (only useful with anim=True))*

**feedback\_item:** the name of the feedback that will appear  
*(type=String)*

**anim:** True if there will be an animation (use skill items),  
 False for not feasible actions and game options like sound  
*(type=Boolean)*

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

**10.2.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**10.2.3 Instance Variables**

Name	Description
<code>boxIntro</code>	True if player has to use the power switch to start the game <i>(type=Boolean)</i>
<code>deleteDelay</code>	the amount of the delay for deleting objects <i>(type=Integer)</i>
<code>gravityAxis</code>	the axis of the gravity direction <i>(type=Integer (0,1,2))</i>
<code>gravityForce</code>	the power of the gravity (default 9.81) <i>(type=Float)</i>
<code>gravityVector</code>	the standardized vector for gravity direction <i>(type=Vector)</i>
<code>isPaused</code>	True if the whole world was stopped by an event <i>(type=Boolean)</i>

*continued on next page*

Name	Description
isUnusual	True if you want the pure stair insanity ( <i>type=Boolean</i> )
nextHurdle	True if it is time for a new hurdle ( <i>type=Boolean</i> )
nextItemTime	this is the counter variable for setting an item in the game ( <i>type=Integer</i> )
numberOfHurdle	the number of all hurdles in the game ( <i>type=Integer</i> )
planeForceDirection	the gravity direction in the current plane (up=1 or down=-1) ( <i>type=Integer (1 or -1)</i> )
planeRotationMatrix	the rotation matrix that holds the information for every object to know how to get the head up in the sky (world coords) ( <i>type=Matrix</i> )
restartGame	True if the game shall be restarted ( <i>type=Boolean</i> )
soundOn	True if the sound in the background is switched on, False for off ( <i>type=Boolean</i> )
specialKeys	True if the player has to use special key input dependent on the current plane of the player ( <i>type=Boolean</i> )
switchOn	True is player has used the power switch ( <i>type=Boolean</i> )
tpcOn	True if the third person camera shall appear in the lower left corner of the window, False if you wish no visible third person camera ( <i>type=Boolean</i> )

## 11 Module MyScene

### 11.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 11.2 Class MY\_SingleScene

object  **MyScene.MY\_SingleScene**

singleton class of the used game scene to hold all the created objects

#### 11.2.1 Methods

**--new--**(*self*, \**args*, \*\**kargs*)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.--new-- extit(inherited documentation)

**--init--**(*self*)

x.--init--(...) initializes x; see x.--class--.--doc-- for signature

Overrides: object.--init-- extit(inherited documentation)

**addObject**(*self*, *my\_object*=None)

adds a given object to his predefined list

**Parameters**

**my\_object**: the object that shall be added to the scene

(*type*=*MY\_GameObject*)



---

**removeObject**(*self*, *my\_object=None*)

---

removes an object from his scene list

**Parameters**

**my\_object:** the object that shall be removed from the scene  
*(type=MY\_GameObject)*

---

**\_\_getitem\_\_**(*self*, *item*)

---

only for non passive elements, passive elements can be searched by id  
(.getMyPassiveObjectById())

---

**getMyPassiveObjectById**(*self*, *object\_id*)

---

gets a passive object by it's given id, like platforms and stairs

**Parameters**

**object\_id:** the id of a passive object  
*(type=Integer)*

**Return Value**

returns the passive object that goes with the given id  
*(type=MY\_GameObject)*

---

**getMyActiveTempObjectById**(*self*, *item\_id*)

---

gets a temporary active object by it's given id, like items

**Parameters**

**item\_id:** the id of a temporary active object  
*(type=Integer)*

**Return Value**

returns the temporary active object that goes with the given id  
*(type=MY\_GameObject)*

---

**freeLists**(*self*)

---

free the lists and destroy all game objects (when game is over for example)

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

### 11.2.2 Properties

Name	Description
<i>Inherited from object</i> __class__	

### 11.2.3 Instance Variables

Name	Description
objectsActive	all objects that are active like player and the universe sphere ( <i>type=List</i> )
objectsActiveTemp	all objects that are temporary available in the game scene like items, hurdle, gauge ( <i>type=List</i> )
objectsCamera	all cameras in the game scene ( <i>type=List</i> )
objectsLight	all the lights in the game scene ( <i>type=List</i> )
objectsMerch	all objects that are classified as xItem and were spawned into game scene ( <i>type=List</i> )
objectsPassive	all stairs and platforms in the game scene ( <i>type=List</i> )
scene	holds the blender game scene ( <i>type=KX_Scene</i> )

## 12 Module MySpecialLists

### 12.1 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> None

### 12.2 Class MY\_SingleIdList

object  **MySpecialLists.MY\_SingleIdList**  
 singleton class to hold unique ids for game objects

#### 12.2.1 Methods

**--new--**(*self*, \*args, \*\*kargs)

**Return Value**

a new object with type S, a subtype of T

Overrides: object.--new-- extit(inherited documentation)

**--init--**(*self*)

x.--init--(...) initializes x; see x.--class--.--doc-- for signature

Overrides: object.--init-- extit(inherited documentation)

**getElementId**(*self*)

gets a free id for any game object

**Return Value**

returns a free id that can be used for a unique game element

(*type=Integer*)

<b>setElementId</b> ( <i>self</i> , <i>free_id</i> )
--

sets an id free for further use
---------------------------------

<b>Parameters</b>
-------------------

<b>free_id</b> : the id that can be freed out of the idList <i>(type=Integer)</i>
--

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

### 12.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 12.2.3 Instance Variables

Name	Description
<code>idList</code>	the list of the used id's in the game <i>(type=List of Integer)</i>

## 12.3 Class MY\_SingleElementList



singleton class which holds lists for all ingame stairs, items, hurdles and bullets

### 12.3.1 Methods

<b>__new__</b> ( <i>self</i> , * <i>args</i> , ** <i>kargs</i> )
--

<b>Return Value</b>
---------------------

a new object with type S, a subtype of T
--

Overrides: <code>object.__new__</code> <code>exitit</code> (inherited documentation)
--

**`--init--(self)`**

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` `extit`(inherited documentation)

**`setStairIndexList(self, scene)`**

sets the stair list filled with their indices, to get a random/specific stair later

**Parameters**

**`scene`:** the blender scene object which holds all objects

(*type=KX\_Scene*)

**`setItemIndexList(self, scene)`**

sets the item list, filled with their indices, to get a random/specific item later

**Parameters**

**`scene`:** the blender scene object which holds all objects

(*type=KX\_Scene*)

**`setMerchList(self, scene)`**

creates a list of all merchandise items

**Parameters**

**`scene`:** the blender scene object which holds all objects

(*type=KX\_Scene*)

***Inherited from object***

`--delattr--()`, `--format--()`, `--getattr--()`, `--hash--()`, `--reduce--()`, `--reduce_ex--()`,  
`--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

**12.3.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

**12.3.3 Instance Variables**

Name	Description
itemList	the list that conatins all items that exists for the game (NOT in the game) ( <i>type=List [Blender scene indices, KX_GameObject]</i> )
merchList	the list that conatins all merch items that exists for the game (NOT in the game, so there is a static length per game of this list ) ( <i>type=List of KX_GameObject</i> )
merchObjectList	the list that conatins all merch objects that are used in the game (because linked objects appear twice, so this was a simple workaround to get a faultless game, sorry) ( <i>type=List of KX_GameObject</i> )
stairList	the list that conatins all stairs and patforms that exists for the game (NOT in the game) ( <i>type=List of Blender scene indices</i> )

## 12.4 Class MY\_InventoryList

class for the player's inventory with all needed methods

### 12.4.1 Methods

<b><code>__init__(self)</code></b>
<b><code>__getitem__(self, name)</code></b>
<b><code>fillWithConfigValues(self)</code></b> sets the inventory with values from the config parser

**useItem**(*self*, *item\_name*, *add*)

adds or remove an item from its list and the config parser

**Parameters**

**item\_name:** the name of the item that shall be used  
(*type=String*)

**add:** True if an item was collected, False if item was used  
(*type=Boolean*)

**Return Value**

'True' if the item exists, else 'False'  
(*type=Boolean*)

**addMerchItem**(*self*, *merch\_name*)

adds a merchandise item to the config parser

**Parameters**

**merch\_name:** the name of the merchandise item that was collected  
(*type=String*)

#### 12.4.2 Instance Variables

Name	Description
inventoryList	the list that contains the players skill levels (health etc.). one skill => [number of dose, item name, lifetime] ( <i>type=List of List [[Integer, String, Integer], ...]</i> )