Tutorial

Third Person Control

S2ENGINE HD ver. 1.4.5

In this tutorial I'll explain the basis for implementing a **Third Person Camera** control system.

At the end of this tutorial you will be able to create a new camera object from scratch and write a script that control camera movements around the player character and a script that make player character walking, runing and correctly facing camera just like in games such as Batman, Hitman or SplinterCell.

First we have to create a new scene in which make our experiments. We will create it inside the project provided with the engine installation package.

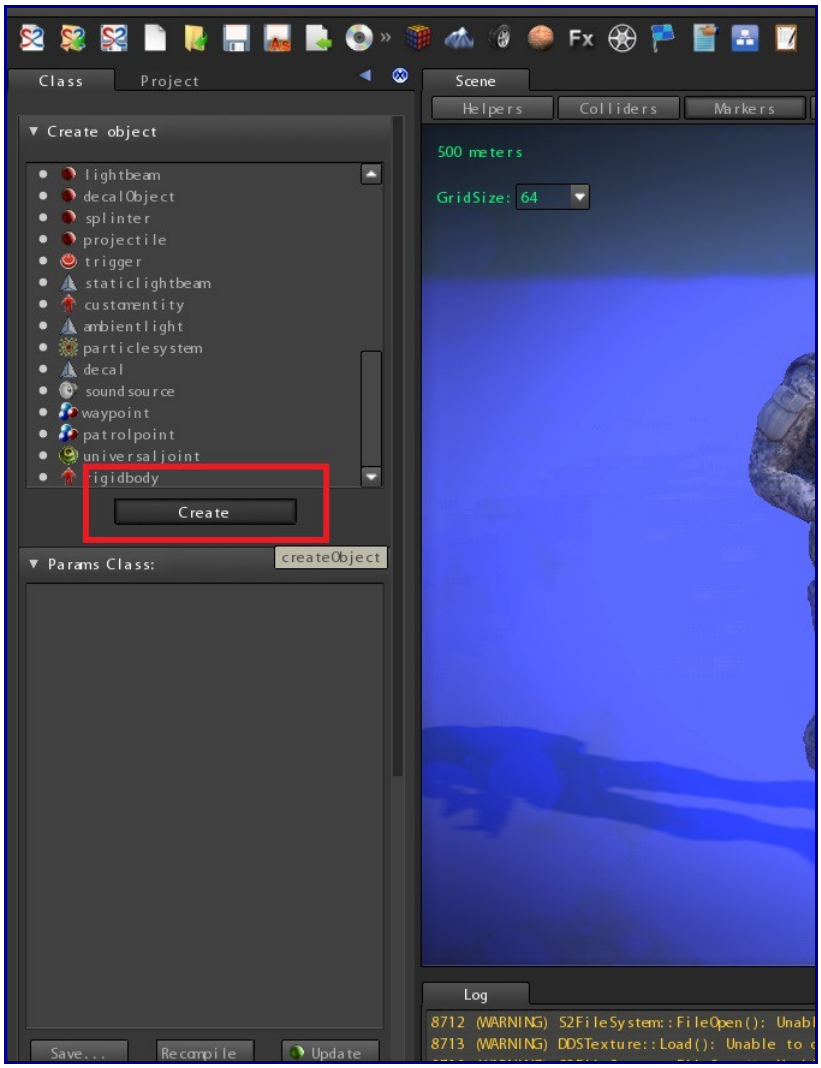
So open the **SampleProject03** project without loading any scene. Click on New icon to create a completely new scene inside the project. We will call it simply myScene.

Now Click on newTerrain icon to create a new terrain in the created scene, give it a name and click on create.

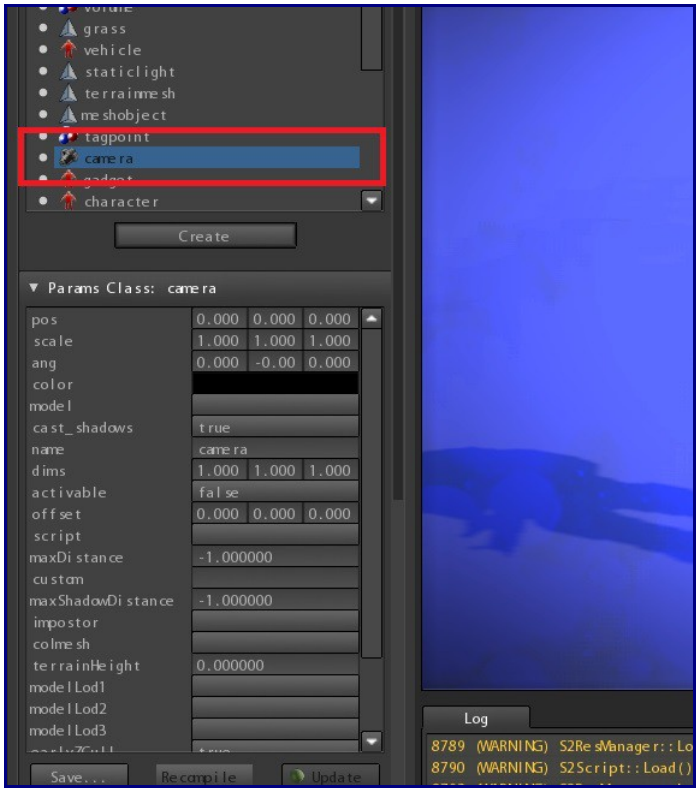
Now we will use the enemy marine character of the FPS demo for our experiments. In the project objects browser goto the characters folder and drag the marine into the scene.

NOTE that due to a know terrain BUG when inserting the character into the scene the terrain will disappear: save the scene, close the editor and reload all.

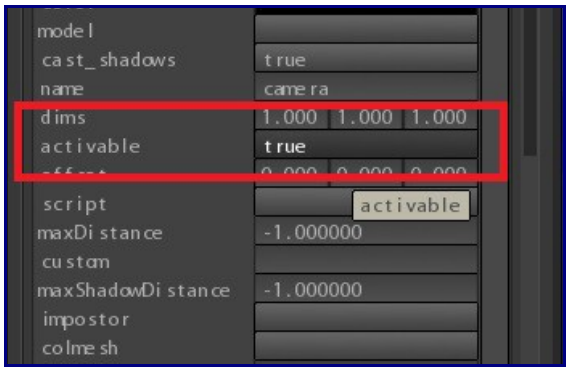
Now we create a new camera object. Click on create button into class tab, it will remain pressed until you click on it again. When the create button is pressed This mean that editor is in "***Create mode***".



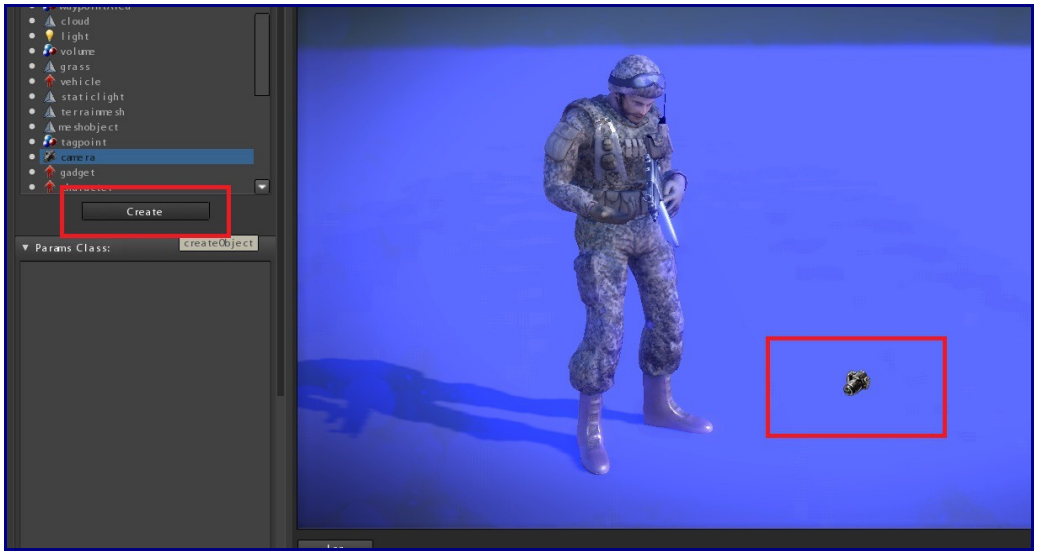
Click on the camera class in the class list of the BasePack.dll plugin, the camera class parameters will appear:



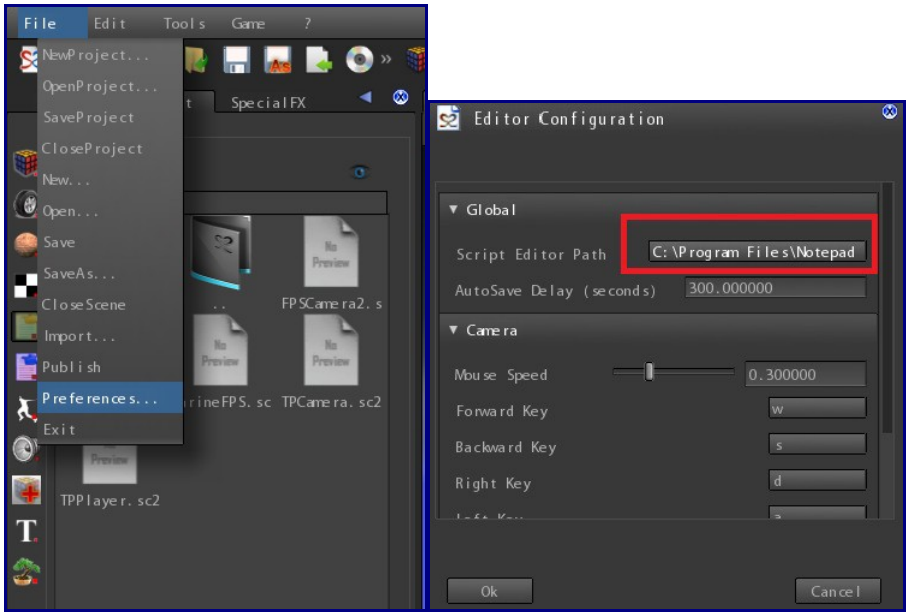
click on activable parameter value in order to make parameter value **true.**



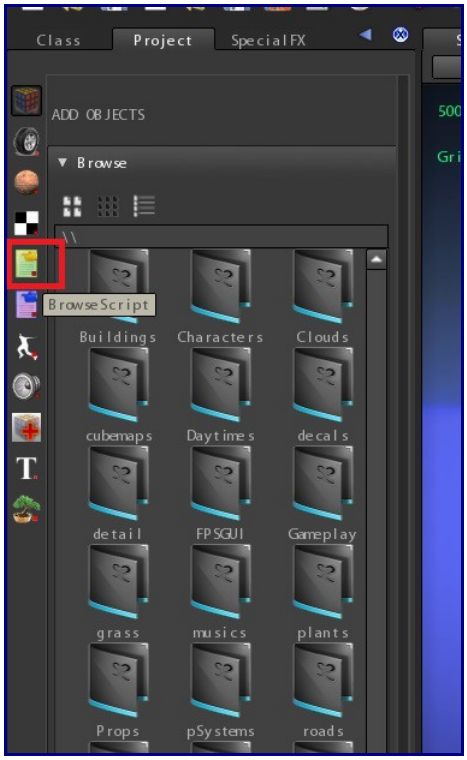
Now click on the terrain inside the scene viewport, you will see the camera icon appearing. Click on Create again to exit from "***Create Mode***".



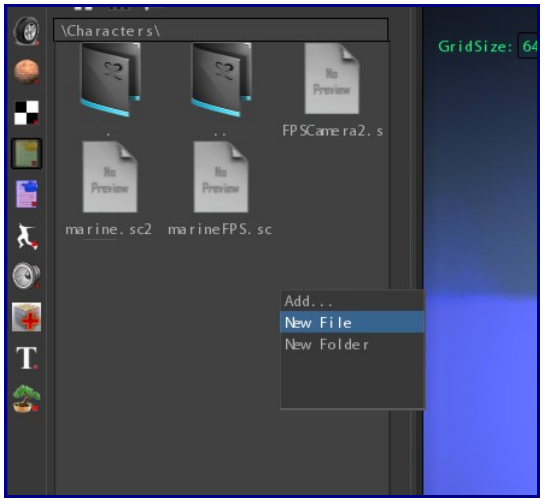
NOTE: If you haven't associated a Text editor for scripting open Preferences anc click on the Script editor combobox to choose the executable file the engine must use as script editor. I use NotePad++.



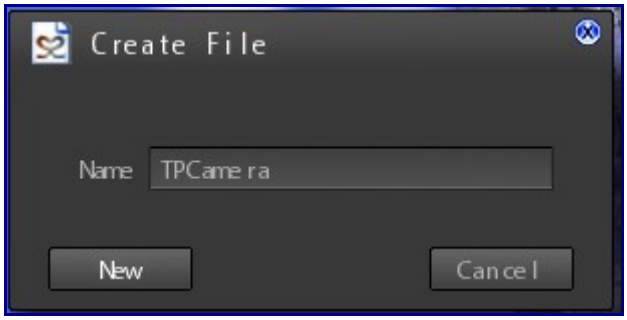
Now we will create a new script file.Select the script browser inside the project browser and choose the **characters** folder.



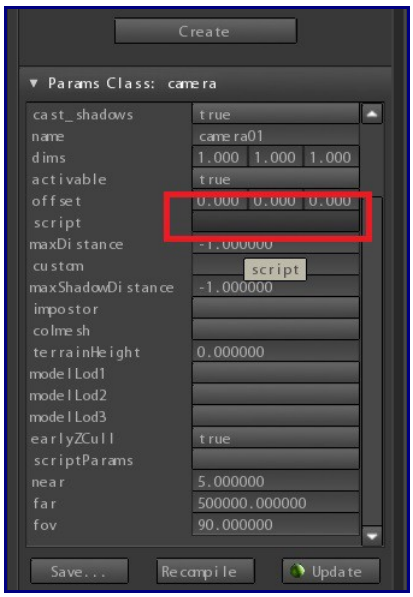
Press Right mouse button on an empty space inside the browser and click on new in the popup menu that will appear.



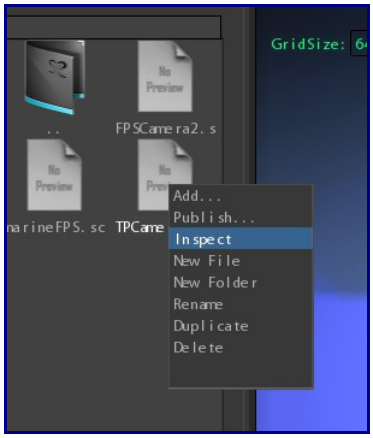
Name the new script as ***TPCamera*** and press ok: a new script file will be created.



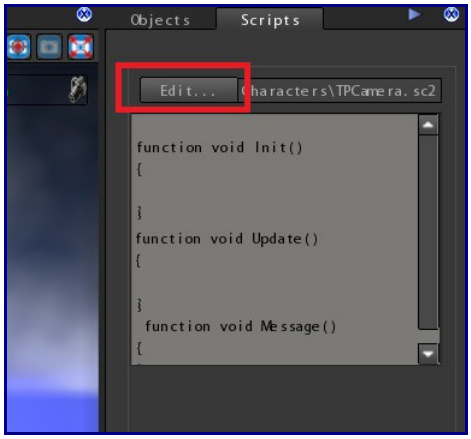
Now we have to associate that script to the camera. Select the camera and click on **class** tab. Click on the Script parameter and choose the ***TPCamera.sc2*** script.



Now return to the Script browser and again press right mouse button on the script icon and click on **Inspect**, the script inspector will appear with the new script start code inside.



Click on **edit…** to edit the script: at this point if you have configured well the script editor path, the editor window will appear with script loaded.



Lets start with scripting!

first we want camera rotate around Y axis when moving the mouse from left to right and viceversa, in the ***Update()*** entry function we write:

```
/* first rotate camera around (0,1,0) axis basing on MOUSEX value */
Rotate(vec3(0.0,1.0,0.0),"MOUSEX",0.1);
```

(see WIKI for more info about this function)

MOUSEX is a string that will be parsed by the scripting engine and means “get Mouse movement on X axis”.

Now we want camera rotating around Its right axis basing on the Up-Down mouse movement:

```
/* then rotate camera around its right axis basing on MOUSEY value */
Rotate("rgt","MOUSEY",0.1);
```

Rgt is a string that is parsed by scripting engine and it means “get this object right axis”.

Now that we have camera rotating basing on the mouse movements we can place it at a certain distance from the target character.

Since we want our script can be used indipendently from the target character we can add a parameter to the script that contains the name of the target character. At the beginning of the script code write:

```
#param string target
```

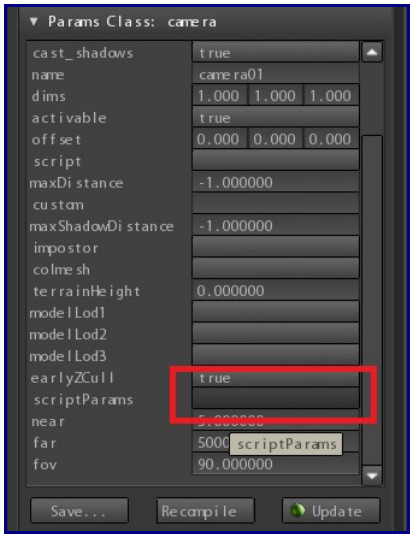
This will tell the script engine that the script has a parameter called ***target***. If you click on the script params object parameter (class tab) you will see a dialog containing the list of the script parameters

Now we can place the camera respect the target character by first placing it at character coords and then traslate it away from character along its forward axis. Again in the ***Update()*** entry function:

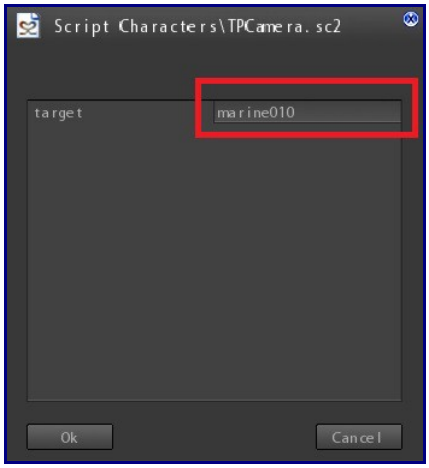
```
/* Now Set camera position from target object position */
SetPositionFromObject(target);

/* place camera 150 points far from target */
Traslate2("fwd",-150.0);
```

(see SetPositionFromObject() spec on the WIKI, see Traslate2() spec on the WIKI)

Now we set the target parameter with the name of the marine character. Click on **scriptParams** camera object parameter.



Now set the target parameter as ***marine010*** (the name of the marine character in the scene).



Now if you click on ***Play*** you will see the camera pointing the center of the character and, moving the mouse, the camera will move around it.

Now we can adjust the camera position basing on our needs. In this case we place the camera on the top-right corner respect to the player character (just like seen in Batman: Arkham Asylum):

```
/* Traslate camera 80 units on its right */
Traslate2("rgt",80.0);
/* traslate camera 50 units on Y axis*/
Traslate(0.0,50.0,0.0);
```

(see Traslate() spec on the WIKI)

Now we put it all together. The complete script will look like this:

```
#param string target
function void Init()
{
}

function void Update()
{
	/* first rotate camera around (0,1,0) axis basing on MOUSEX value */
	Rotate(vec3(0.0,1.0,0.0),"MOUSEX",0.1);

	/* then rotate camera around its right axis basing on MOUSEY value */
	Rotate("rgt","MOUSEY",0.1);

	/* Now Set camera position from target object position */
	SetPositionFromObject(target);

	/* place camera 150 points far from target */
	Traslate2("fwd",-150.0);

	/* and then adjust its position as needed */
	Traslate2("rgt",80.0);
	Traslate(0.0,50.0,0.0);
}

function void Message()
{
}
```

A last touch to our code: Lets make camera do not penetrating obstacles. In this case we use a function called ClipCamera().

The final code will be just like this:

```
#param string target


function void Init()
{
}

function void Update()
{
	/* first rotate camera around (0,1,0) axis basing on MOUSEX value */
	Rotate(vec3(0.0,1.0,0.0),"MOUSEX",0.1);

	/* then rotate camera around its right axis basing on MOUSEY value */
	Rotate("rgt","MOUSEY",0.1);

	/* Now Set camera position from target object position */
	SetPositionFromObject(target);

	/* place camera 150 points far from target */
	Traslate2("fwd",-150.0);

	/* and then adjust its position as needed */
	Traslate2("rgt",80.0);
	Traslate(0.0,50.0,0.0);

	/* clip camera movements against obstacles */
	ClipCamera(target, 50.0, 50.0);
}

function void Message()
{
}
```

Lets start with character now.

First we have to create a new script. We do it in the same folder as Camera script. We name it **TPPlayer.**
As made with Camera we now assign the new script to the marine character and repeating the same procedure done with **TPCamera** script we inspect the **TPPlayer** script and edit it.

What we have mainly to do with player character is making it to always be controlled respect to the camera orientation. In other words if, for example, player character is facing the camera and you press “forward” button the character turns itself in the direction of the camera “forward” and moves, the same thing with right, left and backward directions.

To do this we will use a powerful script function called ***setDirection*** (see setDirection() on the WIKI).

This function lets you to set the facing direction of an object in many ways. One way to do this is set the direction of the object basing on the current active camera direction, this is what we need to make our character control system.

Lets see the code:

```
function void Update()
{
	/* set direction forward respect the current camera*/
	if( IsKeyPressed("w",0) )
	{
		SetDirection("camfwd",1.0,0.25);
	}


	/* set direction backward respect the current camera*/
	if( IsKeyPressed("s",0) )
	{
		SetDirection("camfwd",-1.0,0.25);
	}

	/* set direction right respect the current camera*/
	if( IsKeyPressed("d",0) )
	{
		SetDirection("camrgt",1.0,0.25);
	}

	/* set direction left respect the current camera*/
	if( IsKeyPressed("a",0) )
	{
		SetDirection("camrgt",-1.0,0.25);
	}
}
```

Press play to see this script in action: rotate the camera around the character and then press “w”, you will see marine turn in the same direction of the camera.(see IsKeyPressed() function on the WIKI)

Now we have to animate and move it in order to make it walking and running.

We will use the animation functions of the s2script foundation library. See Animation chapter and Animation Functions on the WIKI.

First, in the **update()** function, we define a boolean variable that store if the character is in idle or walking state:

```
var bool walk;
walk=false; /* if nothing happens character is in idle state */
```

Now we modify the previous code in order to set the walk state if a key is pressed:

```
function void Update()
{
	var bool walk;
	walk=false;

	/* set direction forward respect the current camera*/
	if( IsKeyPressed("w",0) )
	{
		SetDirection("camfwd",1.0,0.25);
		walk=true;
	}


	/* set direction backward respect the current camera*/
	if( IsKeyPressed("s",0) )
	{
		SetDirection("camfwd",-1.0,0.25);
		walk=true;
	}

	/* set direction right respect the current camera*/
	if( IsKeyPressed("d",0) )
	{
		SetDirection("camrgt",1.0,0.25);
		walk=true;
	}

	/* set direction left respect the current camera*/
	if( IsKeyPressed("a",0) )
	{
		SetDirection("camrgt",-1.0,0.25);
		walk=true;
	}
}
```

Now we tell the engine to play a looping animation on the channel 0, if walk state is set we make “walk” animation blend weight be 1.0 and “standby” animation weight be 0.0, otherwise we make viceversa. Also, if walk state is set we move, using physics, the character in its **forward** direction :

```
/* play loop animation on channel 0 */
PlayAnimation(0,true);

/* get character direction */
if(walk==true)
{
	BlendCurrentAnimation("walk",1.0,0);
	BlendCurrentAnimation("standby",0.0,0);
	var vec3 dir;
	dir = GetAxis("fwd");
	physicsCharacterMove(dir,150.0);
}
else
{
	BlendCurrentAnimation("standby",1.0,0);
	BlendCurrentAnimation("walk",0.0,0);
}
```

(see PlayAnimation() , BlendCurrentAnimation(), physicsCharacterMove() and GetAxis() functions spec)

Using the above code we will see that when we press “w” the character turn and move along “camera forward” direction. We can note that the character animation go from **standby** to **walk** instantly, without any type of blend. We don’t want this so we can replace the BlendCurrentAnimation function calls with one call to **AnimationcrossFade()** function.

This function smoothly assigns maximum weigth (1.0) to a specified animation and minimum weight (0.0) to all the others of the same specified channel, you can specify how much time must be spent for the transition.

So the previous code become:

```
/* play loop animation on channel 0 */
PlayAnimation(0,true);

/* get character direction */
if(walk==true)
{
	AnimationcrossFade("walk",0,100.0);
	var vec3 dir;
	dir = GetAxis("fwd");
	physicsCharacterMove(dir,150.0);
}
else
{
	AnimationcrossFade("standby",0,100.0);
}
```

Now we can use another boolean varaible to specify if the character is in **running** or **walking** state and we can make character go to **running** state if ***shift*** key is pressed. We also cross fade *run animation* and accellerate movement if in **running** state.

Putting it all together the code will look like this:

```
function void Update()
{
	var bool walk;
	var bool run;
	walk=false;
	run=false;

	if(IsKeyPressed("@lshift",0))
	{
		run=true;
	}

	/* set direction forward respect the current camera*/
	if( IsKeyPressed("w",0) )
	{
		SetDirection("camfwd",1.0,0.25);
		walk=true;
	}
	/* set direction backward respect the current camera*/
	if( IsKeyPressed("s",0) )
	{
		SetDirection("camfwd",-1.0,0.25);
		walk=true;
	}

	/* set direction right respect the current camera*/
	if( IsKeyPressed("d",0) )
	{
		SetDirection("camrgt",1.0,0.25);
		walk=true;
	}

	/* set direction left respect the current camera*/
	if( IsKeyPressed("a",0) )
	{
		SetDirection("camrgt",-1.0,0.25);
		walk=true;
	}


	/* play loop animation on channel 0 */
	PlayAnimation(0,true);
	/* get character direction */
	if(walk==true)
	{
		var vec3 dir;
		dir = GetAxis("fwd");
```

```
            if(run)
            {
                  AnimationcrossFade("run",0,100.0);
                  physicsCharacterMove(dir,300.0);
            }

            else
            {
                  AnimationcrossFade("walk",0,100.0);
                  physicsCharacterMove(dir,150.0);
            }
      }
      else
      {
            AnimationcrossFade("standby",0,100.0);
      }
}
```