



Havok Xtra Behaviours

1 Character Controller



1.1 Intro

This behaviour provides familiar control over a character within a physically modelled environment. It provides all the standard walking, running and jumping functionality. Along with these come the inherent extras in the Havok Xtra by such as sliding along obstacles, falling of ledges and pushing other simulated objects.

1.2 WYSIWYG?

A visually complex character can be very expensive to simulate physically. Also the human body does a remarkable job of keeping itself upright - most of the time. We can't afford to do the calculations needed to keep a hugely complex representation physically simulated and upright, so we wouldn't.



Spheres on the other hand are cheap and they look the same from any direction. Hence when using the character behaviour we use a spherical physics proxy.

1.3 How things work

The behaviour takes the position of a spherical physics proxy and applies its positional information to a displayed model. It ignores the spheres rotation.

In order to provide control, the behaviour sets the linear velocity of the physical proxy based on various keystrokes. The actual velocities are determined from a number of factors.

Upon a key press:

1. The character velocity direction is accumulated based upon the user key presses.
2. Next the resulting vector is then multiplied by the required speed.
3. Finally if the character is in the air then the velocity will be damped.

When no keys are pressed:

- A damping of velocity is applied to stop the character's movement more abruptly.



1.4 Properties

Property	Description	Default
Display Model	Visual representation of the character	1 st Model in Scene
Havok Model	Spherical proxy representation	1 st Model in Scene
Proxy Visibility	Blend percentage of spherical proxy representation. Can be used for debug purposes. (Or a characters force shield?)	25
Local Forward	Direction the character is initially facing. This is usually aligned along one of the major axis. For example, the default values implies the visual character was modelled facing along the positive x-axis.	Vector(1, 0, 0)
Local Right	Direction the characters right arm would point if held straight out.	Vector(0, 1, 0)
Local Up	Direction the character would go if it jumped straight up. Usually this is the opposite to direction in which gravity acts.	Vector(0, 0, 1)
Mass	Mass of physics proxy. While this value is set when a rigid body is created it is so fundamental to the tweakability of a character controller it has been added here for convience.	25
Height	This value represents a percentage of the spherical proxies radius. This value effectively describes how steep an incline will the character be able to get over. 0 means that the character could effectively go up almost vertical walls (ladders?). 1 means that the character will only be able to manage relatively minor slopes	0.95
Walk Speed	The maximum speed the character can go when walking. Remember the dreaded scaling!	50
Run Speed	The maximum speed the character can go when running. Remember the dreaded scaling!	100
Turn Speed	The amount the character turns in degrees per step.	2



Jump Strength	Signifies not high a character can jump in direction of local up vector	100
Air Speed Damper	Amount of control which is lost when character is in the air (e.g. jumping or falling). 0 means no movement control, 1 means full movement control. N.B. full control does not stop the character falling under gravity though.	0.1
Velocity Damper	Represents how much velocity remains when a player is no longer controlling a character. 0 means that all of the current velocity is loss when no key is pressed (i.e. instant stop). 1 means that current velocity remains at 100 percentage.	0.1
Gravity Multiplier	Characters often feel very 'floaty'. Hence an extra gravity force is often applied. NB This value could also be negative (e.g. for a character on the moon, swimming or flying)	2
KeyCodes - XXXX	Key codes assigning key strokes to various character control actions	

1.5 Assumptions

- A bounding sphere representation must be used for the Havok proxy.
- Gravity is acting along one of the major axis.
- Display and Havok models have initial transforms aligned.
- Sphere friction is fixed to 0
- Sphere restitution is fixed to 0

1.6 Current Limitations

- Easy modifications ☺
 - Currently ignores friction of surface
 - Ignores linear velocity of surface (i.e. no moving platforms)
 - Not currently time independent
- Not so easy modifications ☹



-
- Spherical physical presentations only





Appendix - Behaviour Script

Most importantly: The code is given below with the understanding that it will be cut, pasted and modified to suit your own requirements. Have fun.

```
-- assumes that the following behaviors have been applied
--     Havok Physics

property pHavok
property pSprite
property pCrb, pCmdl
property pLocalForward, pLocalRight, pLocalUp
property pProxyVisibility
property pRunSpeed
property pWalkSpeed
property pTurnSpeed
property pAirSpeedDamper
property pJumpStrength
property pHeight
property pMass
property pGravityMultiplier
property pVelocityDamper
property pKUp, pKDown, pKLeft, pKRight
property pKStrafeLeft, pKStrafeRight, pKJump
property pOnGround
property pBaseVelocity
property pInAirBaseVelocity
property pInAirUp, pInAirDown, pInAirLeft, pInAirRight
property pLastTime
property pGravityAxis

on beginSprite me

    pSprite = sprite(me.spriteNum)
    pHavok = pSprite.pHavok

    if voidP(pCmdl) then pCmdl = pSprite.member.model[1].name
    if voidP(pCrb) then pCrb = pSprite.member.model[1].name

    pCmdl = pSprite.member.model( pCmdl )
    pCrb = pHavok.rigidBody( pCrb )
    pCrb.friction = 0
    pCrb.restitution = 0
    pCrb.mass = pMass
    mdl = pSprite.member.model( pCrb.name )
    -- mdl.shader.renderStyle = #wire
    -- mdl.visibility = #none
    mdl.shader.blend = pProxyVisibility

    pHeight = mdl.boundingSphere[2] * pHeight

    pHavok.registerInterest( pCrb.name, #all, 0, 0, #collisionHandler, me )
    pHavok.registerStepCallback( #controlCharacter, me )

    pBaseVelocity = vector(0,0,0)
    pInAirBaseVelocity = vector(0,0,0)
    pLastTime = 0

    if( not 0 = pLocalUp.x ) then pGravityAxis = 1
```



```
if( not 0 = pLocalUp.y ) then pGravityAxis = 2
if( not 0 = pLocalUp.z ) then pGravityAxis = 3

end

on collisionHandler me, details

    rbA = pHavok.rigidBody( details[1] )
    rbB = pHavok.rigidBody( details[2] )

    -- are we getting a collision with the characters feet
    d = (details[3]-pCrb.position)[pGravityAxis]
    if d < -pHeight then
        pOnGround = true
    end if

end

on controlCharacter me, newTime

    pCrb.active = true

    delta = newTime - pLastTime

    -- may want to use something other than shift
    if( the shiftDown ) then
        ls = pRunSpeed
    else
        ls = pWalkSpeed
    end if

    av = pCmdl.transform.rotation
    if keyPressed( pKLeft ) then av = av + (pLocalUp * pTurnSpeed)
    if keyPressed( pKRight ) then av = av - (pLocalUp * pTurnSpeed)
    -- set rotations directly to the display model
    -- we are ignoring rotations from the physics engine
    pCmdl.transform.rotation = av
    pCmdl.transform.position = pCrb.position

    if( pOnGround ) then
        -- handle control of character on the ground

        lv = vector(0,0,0)
        if keyPressed( pKUp ) then lv = lv + pLocalForward
        if keyPressed( pKDown ) then lv = lv - pLocalForward
        if keyPressed( pKStrafeLeft ) then lv = lv - pLocalRight
        if keyPressed( pKStrafeRight ) then lv = lv + pLocalRight

        if keyPressed( pKJump ) then
            pCrb.linearVelocity[pGravityAxis] = pJumpStrength
        end if

        if( 0.001 < lv.length ) then
            -- set controlling velocity
            t = transform()
            t.rotation = pCmdl.transform.rotation
            lv.normalize()
            lv = t * lv

            tV = lv * ls
        end if
    end if
end
```



```
tV[pGravityAxis] = pCrb.linearVelocity[pGravityAxis]
pCrb.linearVelocity = tV
else
    -- damp velocity if not moving
    tV = pCrb.linearVelocity
    tV = tV * pVelocityDamper
    tV[pGravityAxis] = pCrb.linearVelocity[pGravityAxis]
    pCrb.linearVelocity = tV
end if

pInAirUp = false
pInAirDown = false
pInAirLeft = false
pInAirRight = false
else
    -- handle control of character in the air

    lv = vector(0,0,0)

    if keyPressed( pKUp ) then
        if( not pInAirUp ) then
            lv = lv + pLocalForward
            pInAirUp = true
        end if
    else
        if( pInAirUp ) then
            lv = lv - pLocalForward
            pInAirUp = false
        end if
    end if

    if keyPressed( pKDown ) then
        if( not pInAirDown ) then
            lv = lv - pLocalForward
            pInAirDown = true
        end if
    else
        if( pInAirDown ) then
            lv = lv + pLocalForward
            pInAirDown = false
        end if
    end if

    if keyPressed( pKStrafeLeft ) then
        if( not pInAirLeft ) then
            lv = lv - pLocalRight
            pInAirLeft = true
        end if
    else
        if( pInAirLeft ) then
            lv = lv + pLocalRight
            pInAirLeft = false
        end if
    end if

    if keyPressed( pKStrafeRight ) then
        if( not pInAirRight ) then
            lv = lv + pLocalRight
            pInAirRight = true
        else
            lv = lv - pLocalRight
            pInAirRight = false
        end if
    end if
end if
```




```
end if

if( 0.001 < lv.length ) then
    -- set controlling velocity
    t = transform()
    t.rotation = pCmdl.transform.rotation
    lv.normalize()
    lv = t * lv

    tV = lv * ls * pAirSpeedDamper
    tV = tV + pCrb.linearVelocity
    pCrb.linearVelocity = tV
end if

-- apply extra gravity
f = pCrb.mass * pHavok.gravity * pGravityMultiplier
pCrb.applyForce( f )
end if

-- kill spin on sphere
pCrb.angularVelocity = vector(0,0,0)

pCollisionDetails = []
pOnGround = false

pLastTime = newTime

end

on getModels(me, aMember, aList)

    repeat with j = 1 to aMember.model.count
        if string(aMember.model[j]) contains "model" then
            aList.add(aMember.model[j].name)
        end if
    end repeat

    return(aList)

end getModels

on isOKtoAttach(aScript, aSpriteType, aSpriteNum)

    case aSpriteType of
        #Graphic:
            case sprite(aSpriteNum).member.type of
                #shockwave3d:
                    if aScript.getModels(sprite(aSpriteNum).member, []).count > 1 then
                        --There exist models to choose from.
                        return(TRUE)
                    else
                        --This is a 3D member that has no models.
                        return(FALSE)
                    end if
                #text:
                    if sprite(aSpriteNum).member.displayMode = #mode3D then
                        return(TRUE)
                    else
                        return(FALSE)
                    end if
            end case
    end case
```



```
        end case

        #script:
        return(FALSE)
    end case

    return(FALSE)
end isOKtoAttach

on getPropertyDescriptionList(aScript)

    if the currentSpriteNum > 0 then

        tGPDLList = [:]

        tList = []
        tList = aScript.getModels(sprite(the currentSpriteNum).member, tList)

        tGPDLList[#pCmd1] = \
        [\
        #comment:"Display Model",\
        #format: #string,\
        #range: tList,\
        #default: tList[1]\
        ]

        tGPDLList[#pCrb] = \
        [\
        #comment:"Havok Model",\
        #format: #string,\
        #range: tList,\
        #default: tList[1]\
        ]

        tGPDLList[#pProxyVisibility] = \
        [\
        #comment:"Proxy Visibility",\
        #range: [#min:0, #max:100],\
        #format: #float,\
        #default: 25\
        ]

        tGPDLList[#pLocalForward] = \
        [\
        #comment:"Local Forward",\
        #format: #vector,\
        #default: vector( 0, 1, 0 ) \
        ]

        tGPDLList[#pLocalRight] = \
        [\
        #comment:"Local Right",\
        #format: #vector,\
        #default: vector( 1, 0, 0 ) \
        ]

        tGPDLList[#pLocalUp] = \
        [\
        #comment:"Local Up",\
        #format: #vector,\
```



```
#default: vector( 0, 0, 1 ) \
]

tGPDList[#pMass] = \
[\
#comment:"Mass",\
#range: [#min:0.1, #max:1000],\
#format: #float,\
#default: 100\
]

tGPDList[#pHeight] = \
[\
#comment:"Height",\
#range: [#min:0, #max:1],\
#format: #float,\
#default: 0.95\
]

tGPDList[#pWalkSpeed] = \
[\
#comment:"Walk Speed",\
#range: [#min:0, #max:1000],\
#format: #float,\
#default: 50\
]

tGPDList[#pRunSpeed] = \
[\
#comment:"Run Speed",\
#range: [#min:0, #max:1000],\
#format: #float,\
#default: 100\
]

tGPDList[#pTurnSpeed] = \
[\
#comment:"Turn Speed",\
#range: [#min:0, #max:90],\
#format: #float,\
#default: 2\
]

tGPDList[#pJumpStrength] = \
[\
#comment:"Jump Strength",\
#range: [#min:0, #max:1000],\
#format: #float,\
#default: 100\
]

tGPDList[#pAirSpeedDamper] = \
[\
#comment:"Air Speed Damper",\
#range: [#min:0, #max:1],\
#format: #float,\
#default: 0.1\
]

tGPDList[#pVelocityDamper] = \
[\
#comment:"Velocity Damper",\
#range: [#min:0, #max:1],\
```



```
#format: #float,\
#default: 0.1\
]

tGPDList[#pGravityMultiplier] = \
[\
#comment:"Gravity Multiplier",\
#range: [#min:-10, #max:10],\
#format: #float,\
#default: 2\
]

-- 0: 'a'
-- 1: 's'
-- 2: 'd'
-- 12: 'q'
-- 13: 'w'
-- 14: 'e'
-- 49: ' '
-- 123: Left
-- 124: Right
-- 125: Down
-- 126: Up
tGPDList[#pKUp] = \
[\
#comment:"KeyCode - Up",\
#format: #integer,\
#default: 126\
]

tGPDList[#pKDown] = \
[\
#comment:"KeyCode - Down",\
#format: #integer,\
#default: 125\
]

tGPDList[#pKLeft] = \
[\
#comment:"KeyCode - Left",\
#format: #integer,\
#default: 123\
]

tGPDList[#pKRight] = \
[\
#comment:"KeyCode - Right",\
#format: #integer,\
#default: 124\
]

tGPDList[#pKStrafeLeft] = \
[\
#comment:"KeyCode - Strafe Left",\
#format: #integer,\
#default: 0\
]

tGPDList[#pKStrafeRight] = \
[\
#comment:"KeyCode - Strafe Right",\
#format: #integer,\
#default: 2\
]
1
```



For no reason whatsoever, this page is intentionally left blank.

(except for the words and other stuff ☺)