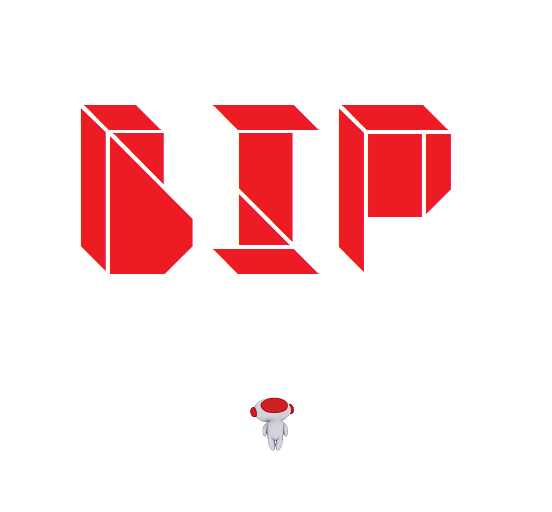
Camera Control Design Document

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Invisibit Studios**  invisibitstudios.com | info@invisibitstudios.com |  |
|  |  |  |



Contents

[Camera Overview 1](#_Toc437004215)

[The Camera Manager 1](#_Toc437004216)

[Camera Setup and Checks 2](#_Toc437004217)

[Camera Manager Information 2](#_Toc437004218)

[Waypoint Information 2](#_Toc437004219)

[Main Camera Information 3](#_Toc437004220)

[Camera Rules Outlined 4](#_Toc437004221)

[Camera Movement Script 5](#_Toc437004222)

[Variable Overview 5](#_Toc437004223)

[Main Script Loop 5](#_Toc437004224)

[Camera Sections Explained 6](#_Toc437004225)

[Pathing Routes Script 9](#_Toc437004226)

[Modifying the Camera Class 10](#_Toc437004227)

[Bip Flip Position 10](#_Toc437004228)

[Player Controlled Rotations 10](#_Toc437004229)

[Future Features 11](#_Toc437004230)

[Route Paths 11](#_Toc437004231)

[Known Issues and Bugs 12](#_Toc437004232)

[Flipping 12](#_Toc437004233)

[Camera Collision 12](#_Toc437004234)

# Camera Overview

As taken from the GameDesignDocument (GDD) the camera includes the following features:

* Always look at Bip from an elevated position.
* Orbit Bip so the player can see in all directions (player controlled).
* Collide with walls instead of flying through them.

Each of the criteria discussed below will go into detail about each section including other scenarios the camera could be found in. Amongst this, variables of each script and methods it implements have been given an overview. Currently the camera class has been split into separate scripts:

* **CameraMovement** – The main movement script of the camera. Defines how the camera rotates and pans towards and around the player whilst handling collisions and other events.
* **PathingRoutes** – A feature used like a cut-scene to pan the camera through a bunch of waypoints. This is used at the beginning of each scene and checkpoint triggers to help make the player aware of actions they take.

At the end there is a section to go over possible changes and corrections towards the components inside CameraManager in an attempt to better handle the layout and effect use of the cameras features.

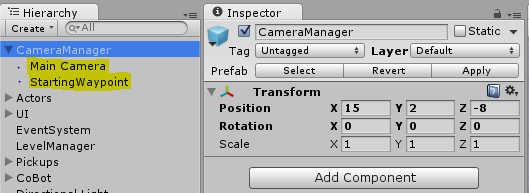
## The Camera Manager

Each scene has a Camera Manager game object which is an empty placeholder for information that the camera will need to have access to or use. This information includes the *Main Camera* game object and the camera potential path routes for the scene.

The *Main Camera* game object has two scripts attached to it; *CameraMovement* and *PathingRoutes*. The second script is responsible for disabling and enabling player controlled movement and rotations during camera paths. The camera object is tagged as “Main Camera”.

The waypoints used by *PathRoutes* are empty game objects which define where the camera is translated to and what rotation it ends up at. Tagged as “Waypoints” and are completely optional. To be active they must be selected and dragged into the “Way Points” array of the *PathRoutes* script.

# Camera Setup and Checks



## Camera Manager Information

There are currently two objects attached to the parent Camera Manager. One is the compulsory Main Camera of the scene. The second is an empty game object whose transform indicates a position and rotation for a camera to travel to. This is an optional game object to include in the scene and you may have multiple waypioints.

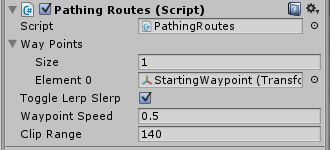
## Waypoint Information

This particular path has only one waypoint marker. It’s an empty game object under the tag of “Waypoint”. Its position is the direction the camera object is translated towards. The rotation of the waypoint is the angle the camera will be rotating towards. This particular waypoints rotation is defined in the “PathingRoutes” script to LookAt(player.transform) – Bip.

## Main Camera Information

The Main Camera is always needed in each scene and given the tag of “MainCamera”. Highlighted above the position of the camera object is important as this will be the starting position of the camera during run time if no waypoint (mentioned above) are defined for the camera. Currently the camera moves towards a fixed distance away from Bip which is calculated at the start of the scene: *so position accordingly!*

The second part is the two attached scripts which will be gone into more detail in the Camera Script section of the documentation. Each of these two scripts must be attached to the Main Camera object. It isn’t necessary that “Camera Movement” is enabled, but “Pathing Routes” ***must*** be enabled as it controls when the player and camera can be moved.

**Pathing Script –**Way Points variable is an array of Transforms. The current path holds 1 waypoint; *StartingWaypoint*.  
Toggle Lerp Slerp makes the pathing Lerp or Slerp to waypoints.  
Waypoint Speed is how fast the camera travels to each point. Finally Clip Range is the point at which it stops slowing down.

# Camera Rules Outlined

The following is a current list of implemented rules that the camera abides by.

**Collision**

* Camera moves itself forward by 1 unit vector when it collides with anything.

**Position**

* Camera stays at a set distance from the player calculated after the initial starting scene path.
* Camera moves towards this set distance if *smoothly* is set to true.
* Camera snaps to this set distance if *smoothly* is set to false.
* Camera travels set paths via defined waypoints during a triggered event; starting the scene or pushing a button. Player controls are disabled during this time.

**Player Controlled Rotation**

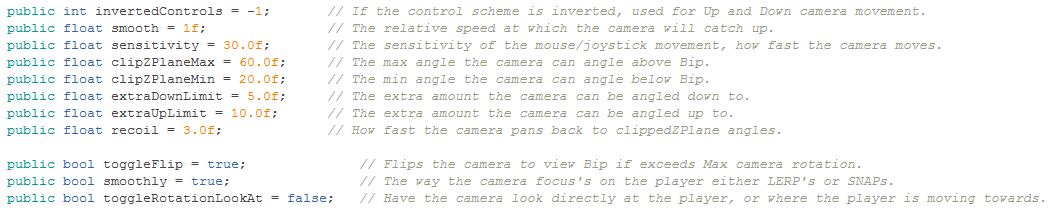
* Camera has full 360 degree rotation around the y-axis.
* Camera can be manually rotated from 20-60 degrees along the z-axis. These values are modified via the *clipZPlaneMin*, *clipZPlaneMax* variables.

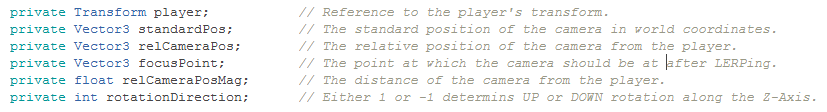
**Camera Controlled Rotation**

* Camera locks onto the players position if *smoothly* is set to false.
* Camera translates and rotates towards the players position if *smoothly* is set to true.
* Camera rotation is based on the current camera position towards the player’s position if *toggleRotationLookAt* is true.
* Camera rotation is based on where the cameras position will end at, towards the player if *toggleRotationLookAt* is false. Only during when *smoothly* is set to true.
* Camera checks checks the directions above Bip and towards the camera at the lowest point from Bip when Bip goes out of sight, and rotates only if a position is free. Rotating *upwards* takes priority over rotating *downwards*.
* Camera flips to a new position 180 around the Y axis if *toggleFlip* is set to true, when the upward rotation exceeds 60 + 10 degrees; *clipZPlaneMax* + *extraUpLimit*.
* Camera stops rotating completely when the downward rotation exceeds 20 – 5 degrees; *clipZPlaneMin* – *extraDownLimit*.
* Camera moves towards the rotation limits at a *recoil* speed value if they are exceeded. If the lower limit is exceeded the camera won’t start to recoil unless there is a free position above Bip.

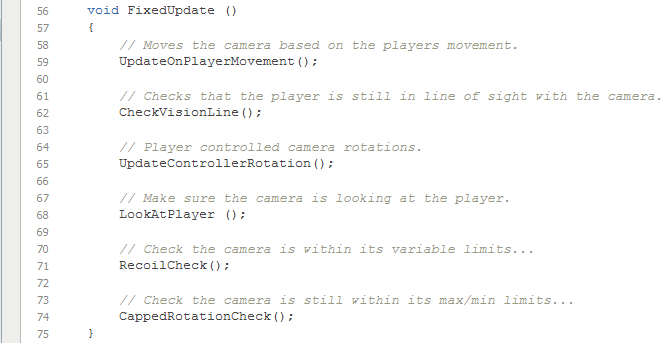
# Camera Movement Script

## Variable Overview

**Public**

**Private**

## Main Script Loop



## Camera Sections Explained

Overall the whole dynamic of the camera is split up into 3 categories. Each part is responsible for making sure the camera follows the outlined rules.

**Player Controlled Movement**The “UpdateControllerRotation()” function is the part of the script that handles all the input from the controller allowing free rotation around the y-axis and restricts rotations around the z-axis.

**Pseudo Code – UpdateControllerRotation()**

* Stores values from Input.GetAxis() for how much the control is rotating in Y and Z axis.
* Checks if any input was detected from the controller.
* If the camera is being rotated in the Z axis checks that the *clipZPlane* limits haven’t been passed.
* Rotates the camera in the Y axis around Bip.
* Rotates the camera in the Z axis around Bip.
* Save the cameras new relative position from Bip.

**Scene Related Movement**Before the “UpdateControllerRotation” is called, two functions handle the initial positioning and rotation of the camera. This makes sure Bip is always in a viewable position for the camera to focus on. The first function is a simple positioning function to keep updated with Bip’s movements.

**Pseudo Code – UpdateOnPlayerMovement()**

* Store a position based on the current players position and the distance away from Bip that was saved at the start.
* If the *smoothly* variable is set to true, make the camera transform Lerp towards the new position.
* Otherwise if the *smoothly* variable is false, set the cameras transform equal to the new position.
* Afterwards store the new LookAt position of where the camera will be focused upon.

After the new position of the camera has translated to a “CheckVisionLine” function is made to ensure that the camera still has Bip in sight. This is done via ray casting the cameras position towards the player to detect for anything in between.

**Pseudo Code – CheckVisionLine() + ViewingPosCheck()**

* Send a raycast to check if anything from the camera towards Bip is in the way.  
  Run the “ViewingPosCheck” function if the raycast returns true.
* Send two more raycasts from the highest and lowest position limits towards Bip.  
  If the highest place isfree set the rotation variable to 1 and return.
* Otherwise if the lowest place is free set the rotation variable to -1 and return.  
  Otherwise no free positions are found and return false.
* If “ViewPosCheck” returned false no rotations are made and the script continues.
* Otherwise the camera transform is rotated in the direction of the rotation variable
* If the new rotation exceeds the *clipZPlaneMax* + *extraUpLimit* the camera rotates 180 around the y axis.
* The new position of the camera relative to Bip is then saved.

**Camera Controlled Movement**This last section is all to do with handling variable adjustments to the camera object in relation to everything else that has been happening. There are 3 functions checks included in this section which are run in the order; “LookAtPlayer”, “RecoilCheck” and lastly “CappedRotationCheck”. The “LookAtPlayer” function simply handles how the camera rotates towards its focus point and is what keeps Bip in the camera range at all times. This function is run after player controlled rotations and scene related rotations have been made.

**Pseudo Code – LookAtPlayer()**

* Store the current relative position of the player from the camera.
* If *toggleRotationLookAt* is set to true create a rotation towards the relative position of the player.
* Otherwise if *toggleRotationLookAt* is false, create a rotation towards where the cameras focus position is. (It’s Lerp destination).
* If *smoothly* is set to true, Lerp the cameras rotation by the rotation angle created.
* Otherwise if *smoothly* is false, set the cameras rotation to LookAt rotation.

The “RecoilCheck” rotates the camera back towards its *clipZPlane* limits when they are exceeded. A check is done to the lower and upper limit and while the upper limit will recoil down freely, the lower limit will only recoil upwards up if there is nothing above Bip. The recoil speed can be adjusted.

**Pseudo Code – RecoilCheck()**

* If the current rotated angle around the z axis is less than *clipZPlaneMin* set rotation direction to 1.
* Otherwise if the current rotation of the camera is greater than *clipZPlaneMax* set the rotation direction to -1.
* If the direction of rotation is -1 rotate the camera in a downward direction at the *recoil* speed.
* If the rotation direction is 1 check that there is nothing above Bip and rotate upwards at the *recoil* speed.
* If the new position has made the camera lose sight of Bip rotate back.
* Update the new relative position of the camera in relation to Bip.

The “CappedRotationCheck” makes sure there is hard rotation limit cap on the z axis so the camera can’t rotate further than this setting.The player controlled rotations won’t allow the camera to fall beyond the *clipZPlane* limits, but scene related rotations will continuously rotate in a given direction.

**Pseudo Code – CappedRotationCheck()**

* Stores the lowest angle needed to rotate back to *clipZPlaneLimit +/- extraUpDown* values.
* If the current camera angle is lower than *clipZPlaneMin* – *extraDown* the direction stored is 1.
* Otherwise if the current camera angle is higher than *clipZPlaneMax + extraUp* the direction stored is -1.
* If the direction stored is not 0 set the cameras angle to directed limit.

# Pathing Routes Script

**TBC~**

# Modifying the Camera Class

## Bip Flip Position

During the upwards rotation when BIP goes out of sight the camera checks to see if the angle exceeds a maximum limit; *clipZPlaneMax* + *extraUpLimit*. If it does and *toggleFlip* is true, the camera snaps to 180 degrees around BIP.

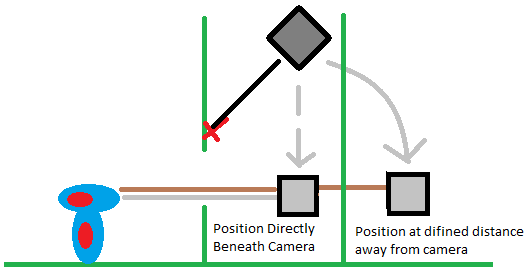
The following are some other ideas on how it could work better or differently:

* LERPing over or around BIP instead of snapping.

## Player Controlled Rotations Clipping

Inside “UpdateControllerRotation()” when the player rotates the camera via mouse or joystick the camera is tested to see if the Z axis has exceeded the defined limits and doesn’t allow the player to rotate the camera any more. Instead of just disabling rotation after the limits have been passed and allowing *recoil* to occur, should the camera test after the rotation and clamp it to the limit value? This way *recoil* is only specifically related to Scene Related Rotations when panning to keep in Bip’s view.

## Vision Checking Lowest Point Check

When a position is being tested for a free position for the camera to travel towards at the lowest point its calculated by taking the direction towards the camera and moving the point a defined position away in that direction. Instead the position to check could be directly underneath the camera. This could help with potential issues when the player is close to a wall and turning the camera.

# Future Features

Some ideas that could be worked or need to be worked on.

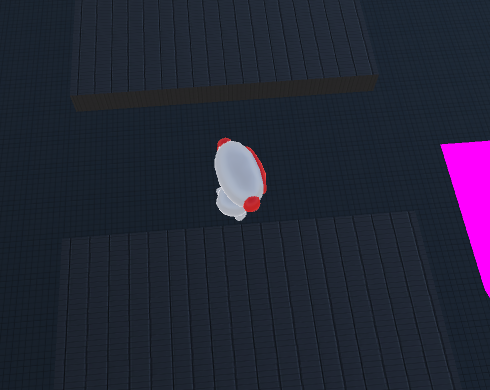
## Route Paths

Already mentioned briefly but when the scene hits a triggered event such as pulling a lever, or pressing a button the camera could then “cut-scene” and path towards a desired position. At the moment the only path available is the starting waypoint position; optional.

# Known Issues and Bugs

## Flipping

**Problem**When the camera exceeds its defined maximum angle of rotation and toggleFlip is true, if the camera flips and BIP is in a narrow space the camera may instantly flip back again.

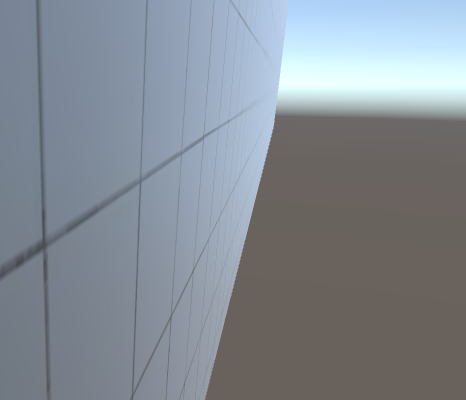
**Evidence  
**

**Solutions**

* Prior to the flip check that the destination is a free position.
* After the flip disable flipping again until the camera has rotated under its maximum rotation.

## Camera Collision

**Problem**When the camera is rotated to fast or the set distance to Bip is through an object the camera may end up going through it. This depends *sensitivity* of the rotation speed and if Bip is walking backwards into a wall, or rotating the after moving towards a wall. The Scene Related Rotation section in the camera eventually fixes this by rotating upwards, however stopping it before it happens is preferable.

**Evidence  
**

**Solutions**

* Move the camera along the collided objects normal via its collider component instead of always along the cameras forward trajectory.