

Technical Design Document

Third-Person Camera System

Version 0.1

|  |  |  |
| --- | --- | --- |
| **Position Title** | **Name** | **Signature** |
| Programmer / System Engineer | Neesarg Banglawala |  |

**Instructions:**

* **Delete all instructional information and the template watermark before turning in final version.**
* Replace the image above with an image containing the game name as it appears in logo form.
* Replace Team Name with the name of the team.
* Update the Version. (0.1 is the first draft, each delivery increments this number, final is 1.0)
* Replace Specialization in the Position Title column with the appropriate information and add the correct names in the Name column. Add additional rows as needed.
* Update the Header and Footer with the correct information. Lock the date to when the version is completed, not the date of its opening.

Each team member reviews and signs the completed TDD. This is a contract. This states that the team member will participate to the level asked of them, and be held responsible to all requirements stated here, and be subject to testing of its content. With each version submitted, the team should also submit one physical copy of the signed cover page.

**Document Revisions Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description** | **Requestor** | **Date** |
| *0.1* | *Initial Document* | *Professor* | *Date of Rev* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Instruction:

The document revisions table provides a location for revision requests and updates. An active role in maintaining this table and meeting the requests of faculty is required. Fill in the Description column with the details of any added, changed or deleted information with a reference to the page number it occurs on in the document. Fill out the Version, Requestor and Date columns appropriately. Do not forget to update the Version # on the Cover Page to correspond to the Document Revisions Table.

Table of Contents

[Table of Figures 5](#_Toc5801163)

[*Table of Videos* 5](#_Toc5801164)

[*Table of Tables* 5](#_Toc5801165)

[Project and Motivation 6](#_Toc5801166)

[Scope 6](#_Toc5801167)

[End Product 6](#_Toc5801168)

[Deliverables 7](#_Toc5801169)

[System Requirements 8](#_Toc5801170)

[Recommended 8](#_Toc5801171)

[Development System 8](#_Toc5801172)

[Resource Budget 9](#_Toc5801173)

[CPU Execution Time Estimate 9](#_Toc5801174)

[Memory Utilization Estimate 9](#_Toc5801175)

[Technology Sources 10](#_Toc5801176)

[Art Creation 10](#_Toc5801177)

[Software Engineering 10](#_Toc5801178)

[Miscellaneous 10](#_Toc5801179)

[Components 11](#_Toc5801180)

[3D Scene 11](#_Toc5801181)

[Controllable Third-Person Character – GameObject 11](#_Toc5801182)

[Perspective Camera 11](#_Toc5801183)

[Camera Manager 11](#_Toc5801184)

[Camera Behavior – Follow Camera 12](#_Toc5801185)

[Camera Behavior – Free-look Camera 12](#_Toc5801186)

[Camera Behavior – Shoulder View 12](#_Toc5801187)

[Camera Constraints – Camera Collision 12](#_Toc5801188)

[Camera Constraints – Line-of-sight 12](#_Toc5801189)

[Camera Constraints – Modified Cone Raycast 12](#_Toc5801190)

[Camera Motion Controller – Modified Proportional Controller 13](#_Toc5801191)

[Debug System 13](#_Toc5801192)

[Profiler 13](#_Toc5801193)

[System Architecture 14](#_Toc5801194)

[Camera Context 14](#_Toc5801195)

[Camera State 14](#_Toc5801196)

[Camera State History 15](#_Toc5801197)

[Camera Behavior 15](#_Toc5801198)

[Camera Constraint 15](#_Toc5801199)

[Camera Motion Controller 16](#_Toc5801200)

[Camera Manager 17](#_Toc5801201)

[Tutorials 18](#_Toc5801202)

[Camera Auto-Control 18](#_Toc5801203)

[Collision Avoidance 20](#_Toc5801204)

[Behavior Handover 22](#_Toc5801205)

[On-Demand Camera Reorientation 23](#_Toc5801206)

[Look Ahead 25](#_Toc5801207)

# Table of Figures

[1 Class Diagram - Camera Context, Camera State 14](#_Toc5797597)

[2 Class Diagram - Camera Manager 15](#_Toc5797598)

[3 Update Method of Camera Manager 17](#_Toc5797599)

[4 Modified Cone Raycast - Debug Circle 18](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797600)

[5 Modified Cone Raycast - Debug View 18](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797601)

[6 Top-down sketch showcasing radius reduction 20](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797602)

[7 Radius Reduction by Modified Cone Raycast 20](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797603)

[8 Weights assigned to each rays 20](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797604)

[9 Raycast Weights - Debug Circle 21](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797605)

[10 Two-dimensional Cone Raycast 21](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797606)

# Table of Videos

[Video 1 Camera Auto Control - No Debug 19](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797644)

[Video 2 Camera Auto Control - Debug 19](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797645)

[Video 3 Debug View - Weighted Raycasts 21](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797646)

[Video 4 Comparison - 2D vs 3D Cone Raycast 21](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797647)

[Video 5 Behavior Handover - Disabled vs Enabled 22](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797648)

[Video 6 Behavior Handover - No Debug 22](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797649)

[Video 7 Behavior Hanover - Debug 22](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797650)

[Video 8 Debug Compass - No Input Interpolation 23](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797651)

[Video 9 Problem Case - Debug Video 23](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797652)

[Video 10 Debug Compass with Input Interpolation 24](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797653)

[Video 11 Comparison - Before vs After the Suggested Changes 24](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797654)

[Video 12 Look Ahead - No Debug 25](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797655)

[Video 13 Look Ahead offset - Debug 25](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797656)

[Video 14 Comparison - Initial vs Later Attempts 26](file:///D:\Documents%20-%20Guildhall\Semester%203\Thesis\Sem%205\TDD%20Third-Person%20Camera%20System.docx#_Toc5797657)

# Table of Tables

[Table 1 Tools & Resources - Art Creation 11](#_Toc5800953)

[Table 2 Tools & Resources - Software Engineering 11](#_Toc5800954)

[Table 3 Tools & Resources – Miscellaneous 11](#_Toc5800955)

[Table 4 Camera Constraint for Follow Behavior 17](#_Toc5800956)

# Project and Motivation

In third-person games, the camera affects the player experience deeply at many levels. As example,

* Did Lara Croft just got shot by a bullet, the camera provides the feedback to the player.
* Is Lara running on low stamina, the camera lures the player to develop empathy.

Especially when a camera does its job well, players do not notice it. It's as if the camera isn't even there! That’s what motivates me to dive into the problem space of third-person cameras.

The goal of my thesis is to architect a camera system which enables smart and aesthetically pleasing camera movement.

## Scope

Scope of this project is to architect a camera system which can support scripted camera behaviors, camera constraints, as well as gives the designer a control over how the camera moves geared towards third-person game. On top of that, creating a game like environment where all the features of the camera system can be tested.

## End Product

The end product is a 3D scene where the player controls a third-person character. There are some solid geometry objects like Buildings, Roof, House and a Terrain which can be considered as obstacle while implementing custom features of the third-person camera. These custom features can be quantified in three categories:

1. Camera Behaviors
   1. Follow Camera
   2. Shoulder View
   3. Free-look
2. Camera Constraints
   1. Camera Collision
   2. Line-of-sight
   3. Modified Cone Raycast
   4. Handover to Shoulder View
   5. Handover to Follow
3. Camera Motion Controller
   1. Modified Proportional Controller

# Deliverables

|  |  |
| --- | --- |
| Asset | Description |
| *Game/Engine Visual Studio Project* | Visual Studio project source code for both the engine and the game. |
| *Third-Person Camera System.exe* | Executable that runs all the features of my camera system. |

# System Requirements

## Recommended

Processor: Intel® Core™ i7-7700HQ CPU @ 2.80GHz

Memory: 32 GB

Video Card: NVIDIA GeForce GTX 1070 (8192 MB)

Operating System: Windows 10 Home

Software: Visual Studio 2017

Peripheral: Keyboard, Mouse & Xbox Controller

## Development System

Processor: Intel® Core™ i7-7700HQ CPU @ 2.80GHz

Memory: 32 GB

Video Card: NVIDIA GeForce GTX 1070 (8192 MB)

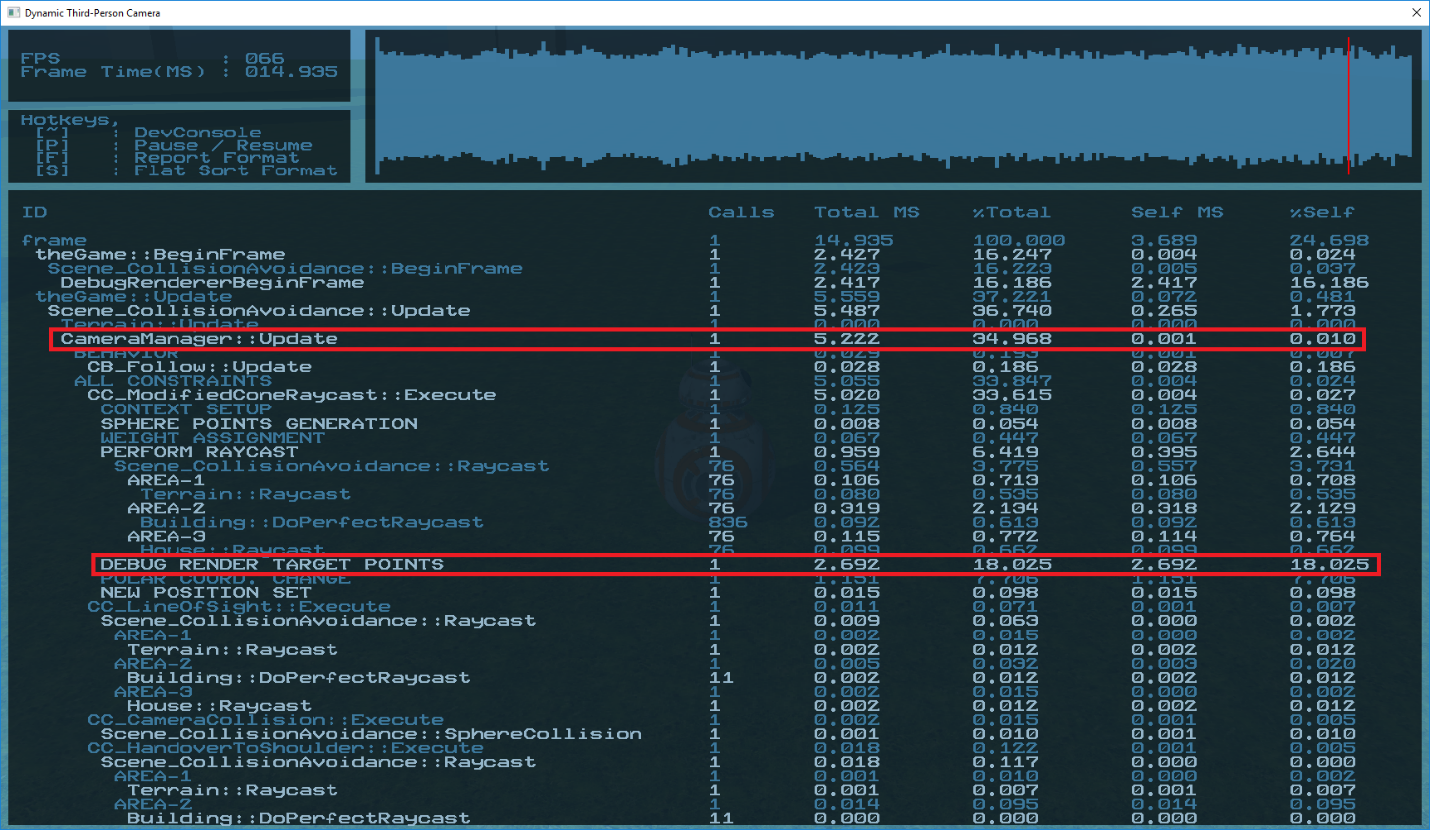
Operating System: Windows 10 Home

Software: Visual Studio 2017

Peripheral: Keyboard, Mouse & Xbox Controller

# Resource Budget

## CPU Execution Time Estimate

The camera system uses 2.53 milliseconds when compared to the total frame time of 15 milliseconds. This is when all the features are enabled: Camera Auto Look, Collision Avoidance, Behavior Handover, Camera Reorientation, and Look Ahead.

*1* Profiler - Camera System

## Memory Utilization Estimate

The executable (.exe) takes 460 MB to run. This involves all the components mentioned later in the document, and not just the Camera System.

# Technology Sources

As external tools and resources look at tables provided below.

### Art Creation

|  |  |  |
| --- | --- | --- |
| Tool / Asset Name | License | Link |
| BB8 3D Model | CC Attribution | <https://sketchfab.com/3d-models/bb8-6aff787c459a4e00a26ed11ac8f148a1> |

Table Tools & Resources - Art Creation

### Software Engineering

|  |  |  |
| --- | --- | --- |
| Tool / Asset Name | License | Link |
| Visual Studio 2017 | Community License | <https://visualstudio.microsoft.com/license-terms/mlt553321/> |

Table Tools & Resources - Software Engineering

### Miscellaneous

|  |  |  |
| --- | --- | --- |
| Tools / Asset Name | License | Link |
|  |  |  |
|  |  |  |

Table Tools & Resources – Miscellaneous

# Components

## 3D Scene

**Description:** This is the scene which provides the testing ground for the camera system. It includes a third-person player-controlled character, terrain, and different sorts of obstacles – house, buildings, roof. 3D Scene also includes lighting, camera. All the meshes will get rendered using a Forward Rendering Pipeline.

**Failure Impact:** Without a scene the camera system can’t be tested.

**Current Status:** Implemented.

## Controllable Third-Person Character – GameObject

**Description:** Since the camera system is supposed to be geared towards third-person games, such third-person controllable character is essential to test and implement all the features of the system.

**Failure Impact:** Without a third-person character, features like follow camera, collision avoidance, camera auto-control can’t be implemented.

**Current Status:** Implemented.

## Perspective Camera

**Description:** Provides color target and depth targets using which the scene is rendered. It contains Transform defining its position and rotation. The camera’s perspective projection can be changed with different values of near z, far z, field of view, and aspect ratio. The Camera Manager manipulates these properties exposed by a perspective camera.

**Failure Impact:** Without a perspective camera, the scene cannot be rendered.

**Current Status:** Implemented.

## Camera Manager

**Description:** Provides the framework to run customizable features of the system, which are given below.

* Camera Behavior: It defines how the camera behaves assuming ideal conditions.
* Camera Constraints: These are set of rules which a camera behavior needs to follow.
* Camera Motion Controller: It moves the camera from its current state to the target state.
* Change Active Camera Behavior: As the name suggests, it changes the active behavior and its constraints as well as suggested motion controller.

**Failure Impact:** The camera cannot be scripted for any sort of customizable characteristics. None of the following game-side features would work without it.

**Current Status:** Implemented.

## Camera Behavior – Follow Camera

**Description:** Game side scripted camera behavior which operates based on camera’s polar coordinates such that it follows the third-person anchor character in the scene. It serves as the main camera behavior during the gameplay experience.

**Failure Impact:** Without it the camera can only provide a static view of the scene.

**Current State:** Implemented.

## Camera Behavior – Free-look Camera

**Description:** It serves as an observation camera, which can freely fly around the scene. Such camera behavior is especially useful as a Debug Camera – to observer ongoing situation from a different perspective or position.

**Failure Impact:** Debugging will be much harder without it.

**Current State:** Implemented.

## Camera Behavior – Shoulder View

**Description:** As the name suggests, it provides the shoulder view from the third-person character. A behavior like this can serve as an aim camera, or it can provide more visibility when there is not enough space for the follow camera to position itself around the player.

**Failure Impact:** When player is in congested areas the visibility might be compromised.

**Current State:** Implemented.

## Camera Constraints – Camera Collision

**Description:** It makes sure that the camera is not colliding with a geometry.

**Failure Impact:** Camera might go into a geometry which can affect its depth buffer. A sudden change in depth buffer because of the invalid position of the camera might affect gameplay code which depends on the depth buffer.

**Current State:** Implemented.

## Camera Constraints – Line-of-sight

**Description:** It makes sure that the camera maintains the line-of-sight to the third-person character.

**Failure Impact:** Camera does not provide guarantee for maintaining line-of-sight.

**Current State:** Implemented.

## Camera Constraints – Modified Cone Raycast

**Description:** This constraint provides two main features of the camera system: Auto-Control, Collision Avoidance. When the player is not controlling the camera, the Auto-Control feature positions the camera such that visibility in the movement direction is good. When the player is in control of the camera the Collision Avoidance positions the camera to avoid future collisions.

**Failure Impact:** Compromised visibility in the cases of player controlling the camera, and not controlling the camera.

**Current State:** Implemented.

## Camera Motion Controller – Modified Proportional Controller

**Description:** A custom motion controller which is responsible for moving the camera to its goal state based on the player velocity, and the camera velocity. As of now, the Look Ahead feature is implemented in this component of the system.

**Failure Impact:** Smooth movement of the camera cannot be guaranteed. Look Ahead feature won’t work.

**Current State:** Implemented.

## Debug System

**Description:** Engine side library of Debug Render draw calls. It is helpful to debug the camera properties, raycasts, collision, weights, velocities or any vectors, 2D graphs, etc. The debug system can also work while the game is pause. It supports a debug camera, picture-in-picture and full overlay debug windows.

**Failure Impact:** Severe, unless you implement all the components of the camera system correctly on first attempt. Debug System is an invaluable tool which helped me during each stage of the development whether it while architecting the system or tuning any of its components.

**Current State:** Implemented.

## Profiler

**Description:** Primary engine-side tool to profile the camera system. It shows the frame execution time, CPU cost of any camera system component, and the call count of the methods. Useful to determine the CPU cost of the project.

**Failure Impact:** Any detailed data about the performance of the camera system cannot be provided.

**Current State:** Implemented.

# System Architecture

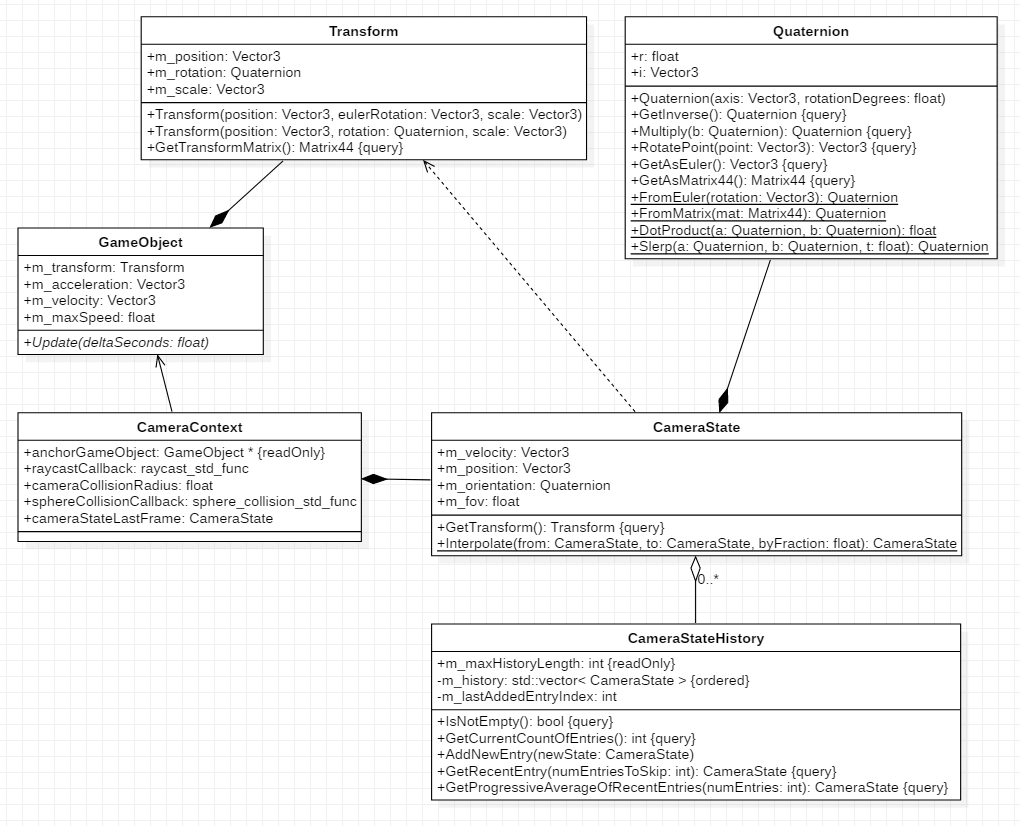
Camera Context and Camera State are two fundamental concepts that forms the basis of communication between camera system's other components.

## Camera Context

As the name suggests, the Camera Context has all the contextual information about the camera which can be essential to the camera system:

* Anchor game object: a pointer to the character (here BB8) which the camera is following
* Raycast callback: a pointer to the raycast method, implemented on the game side
* Camera collision radius: the radius of the camera's collider
* Sphere collision callback: pointer to the collision check method, implemented on the game side
* Camera state last frame: camera's state on the previous frame

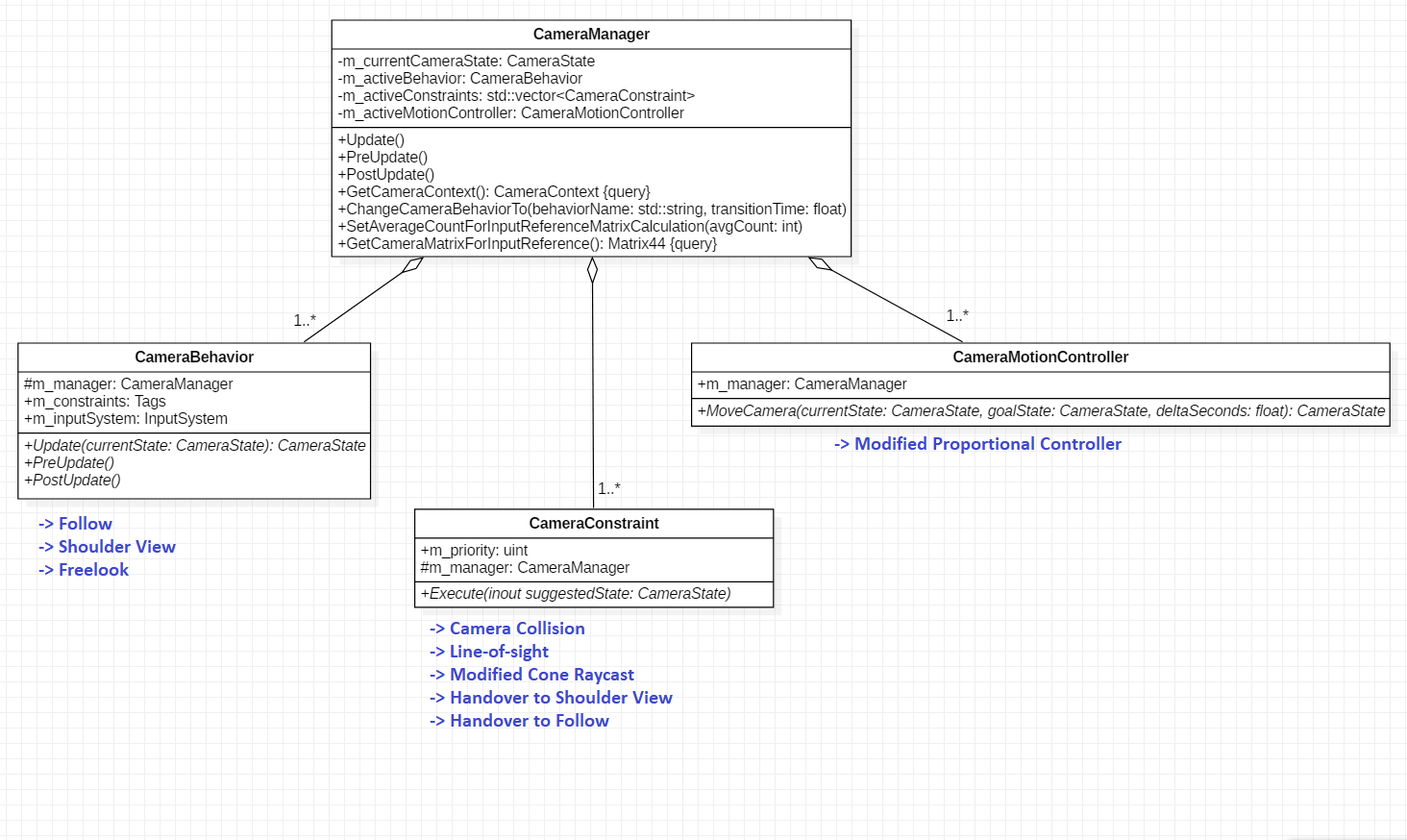
## Camera State

Camera State contains essentially every property which can be tweaked by the camera system. The system can smoothly interpolate between two camera states if needed too. Take a look at the class diagram given below.

Class Diagram - Camera Context, Camera State

## Camera State History

The purpose of the Camera State History is to keep a record of previous camera states, and the camera manager uses it. By getting an average of these camera states the Input Interpolation feature works.

Here is the class diagram of the Camera Manager:

Class Diagram - Camera Manager

## Camera Behavior

A Camera Behavior defines how the camera will behave in the most ideal conditions. These are some behaviors I scripted:

* A Follow camera is good at following a third-person character.
* A Shoulder View camera is good at showing what's ahead when there isn't enough room for the camera around the character. Another common use case of shoulder view camera is as an aim camera in third-person combat.
* A Freelook camera lets an observer roam freely around the map. My debug camera is a Freelook.

## Camera Constraint

Camera Constraint(s) are a set of rules which can be applied on a camera state returned by the active camera behavior. If the current state does not satisfy a given constraint, the constraint will update the state such that it is acceptable. All constraints are applied according to the painter's algorithm: from lowest to highest priority.

Each camera behavior has tags indicating names of the constraints it has to satisfy. As an example, the Freelook behavior doesn't need to satisfy any constraints, but the Follow behavior has to satisfy all of the constraints given below:

|  |  |  |
| --- | --- | --- |
| Constraint Name | Priority | Function |
| Camera Collision | 4 (highest) | Pushes the camera out of geometry |
| Line-of-sight | 3 | Makes sure that the camera has line-of-sight to the character |
| Modified Cone Raycast | 2 | Works to avoid future-collisions based on raycasts |
| Handover to Shoulder View | 1 (lowest) | If in a situation where Shoulder View can serve better, changes the active behavior to it. |

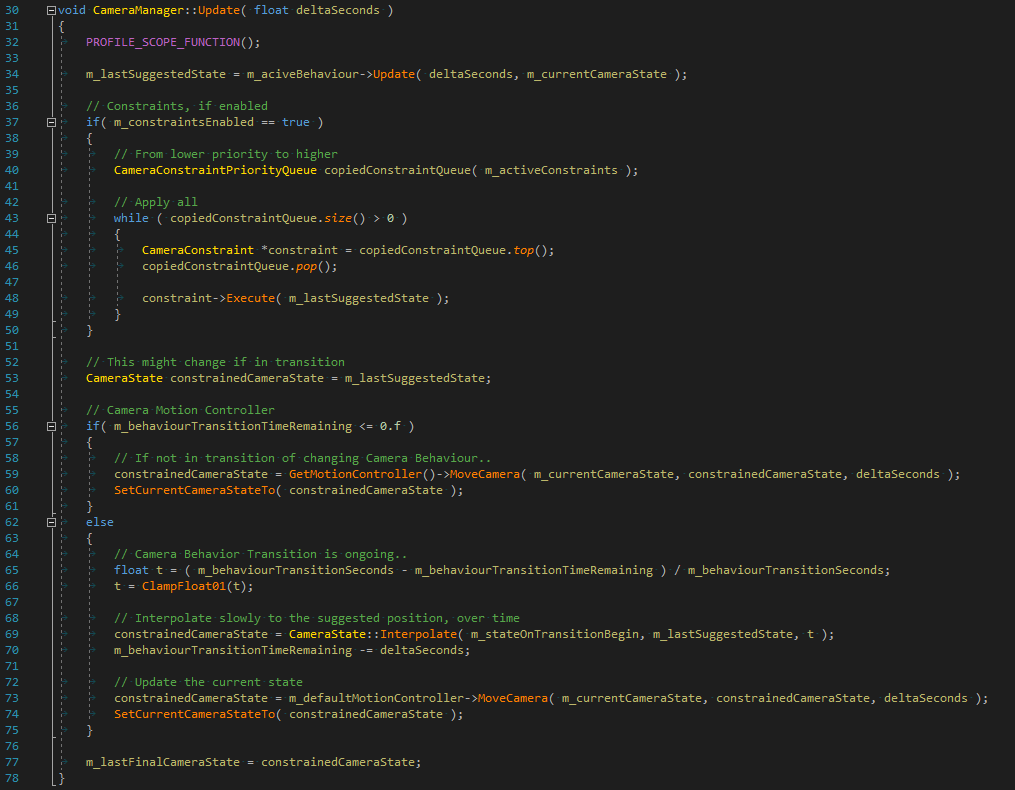
Table Camera Constraint for Follow Behavior

## Camera Motion Controller

After all the constraints are executed, we end up with a camera state which is considered as the goal state. Here the Camera Motion Controller comes in the picture, which is in charge of moving the camera to this goal state. A motion controller could be as simple as snapping the camera's position to the goal, which is the default motion controller provided by my system. But a designer can code any type of complex motion controllers that fits the game.

An example of such a custom motion controller is the Modified Proportional Controller (MPC) which works based on player and camera's velocities. The Look Ahead feature is implemented based on MPC.

## Camera Manager

Thus, the camera gets moved after going through all three components of the system: Camera Behavior, Camera Constraints, and Camera Motion Controller. Here is the code of Update( ) method of class Camera Manager:​

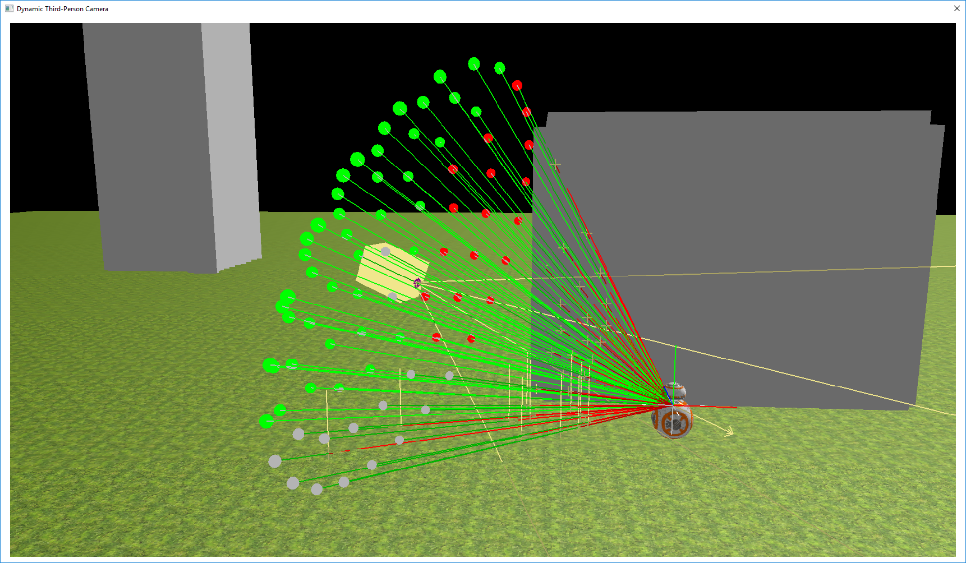
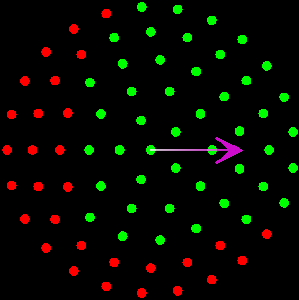
Update Method of Camera Manager

# Tutorials

## Camera Auto-Control

In a moderate to fast paced third-person game, the player is usually engaged with the gameplay so much that controlling the camera on top of it can easily feel like a burden and break the immersion. Saying that, should player want to take control over the camera, they always can!

The intention behind auto-control feature was to enable the camera to position itself automatically such that it complements the player movement. I call my solution Modified Cone Raycast. Let's dive into how it works.

As you can see in the left debug image, a handful of raycasts in the shape of a three-dimensional cone are fired to sample the surrounding environment. The right debug image shows these raycasts' target points, as if they are projected on the camera's front/near plane. A red dot indicates an impact with a geometry and the green dots indicates no impact. Based on which part of the circle has more “complaints” about impacting with surrounding geometry, the camera is then smoothly rotated towards the opposite direction. This is indicated by white-magenta arrow.

5 Modified Cone Raycast - Debug Circle

6 Modified Cone Raycast - Debug View

For simplicity, the camera uses polar coordinates (radius, rotation, altitude), and its anchor is the BB8. One thing to keep in mind is that the Auto-Control changes only the rotation of the camera's polar coordinates.

These videos are here to help you visualize how it works in real time:

Video Camera Auto Control - No Debug

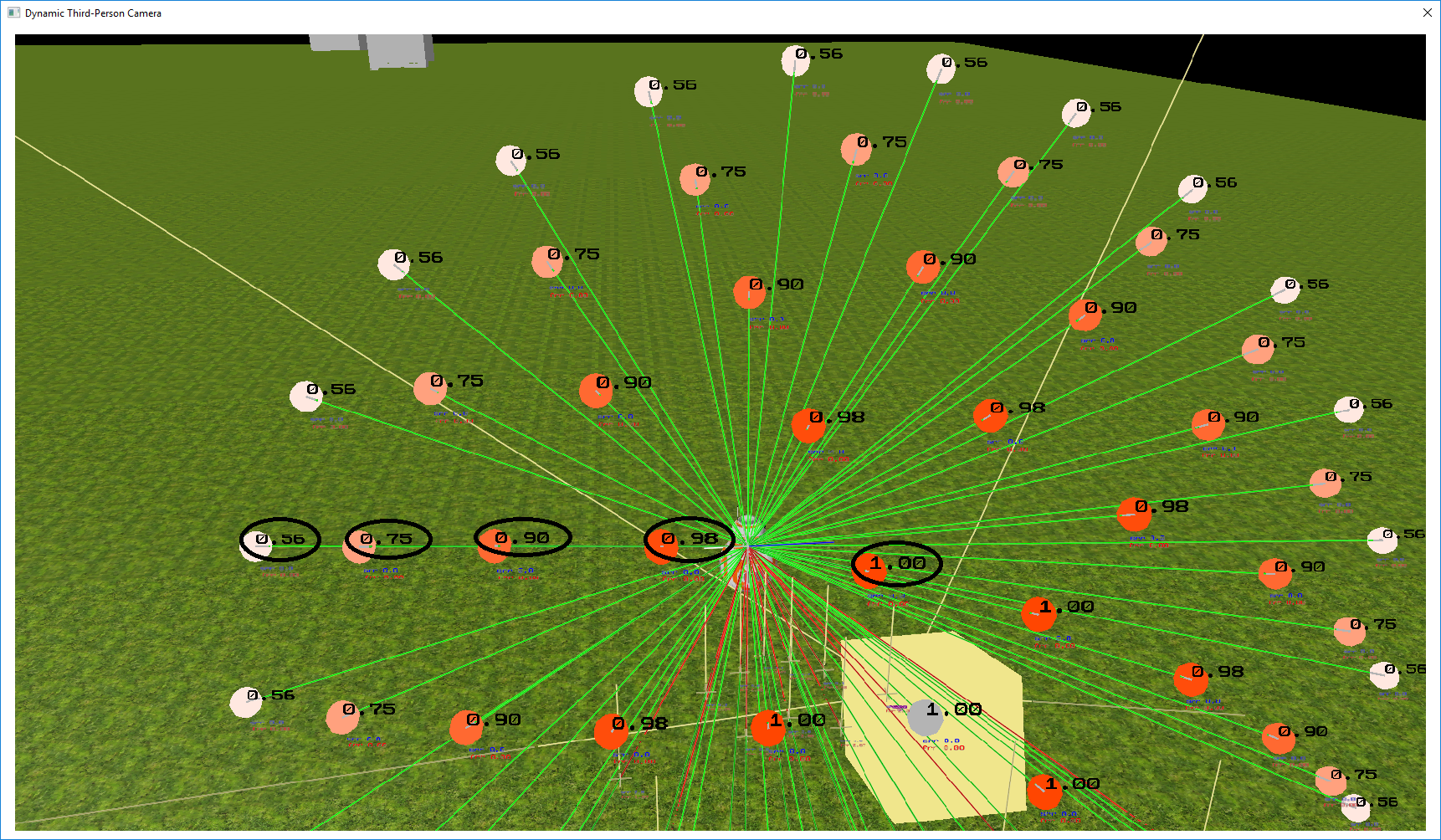
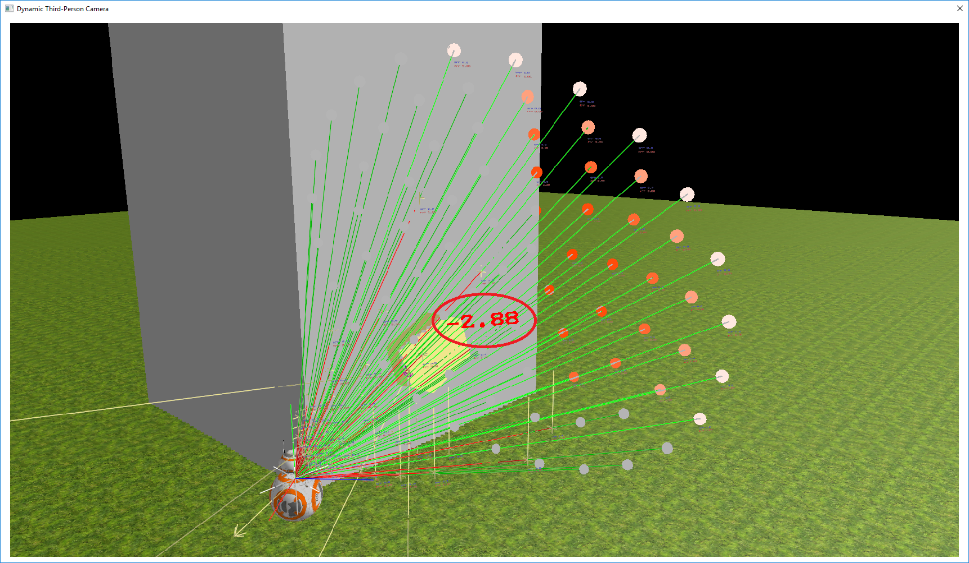
Video Camera Auto Control - Debug

## Collision Avoidance

7 Top-down sketch showcasing radius reduction

Camera Auto-Control does a fine job in avoiding collisions by changing just the rotation of camera’s polar coordinates. However, when the player is in control of the camera, a different take on the collision avoidance becomes essential.

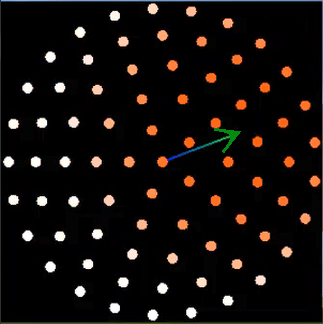
To visualize the problem, take a look at the top-down sketch on the right. Let’s say the player wants to rotate the camera counterclockwise from a start position, but there is a building in the way. In this scenario, reducing the radius of the camera’s polar coordinates will be a good way to avoid the building.

I use the same Modified Cone Raycast to determine how much reduction in the radius (of camera's polar coordinate) should happen. Each ray has a weight assigned to it. Each ray votes for the amount of radius reduction that should happen to prevent the collision reported on their end. Later these votes are resolved in a form of a weighted average to calculate the final reduction in the radius.

8 Radius Reduction by Modified Cone Raycast

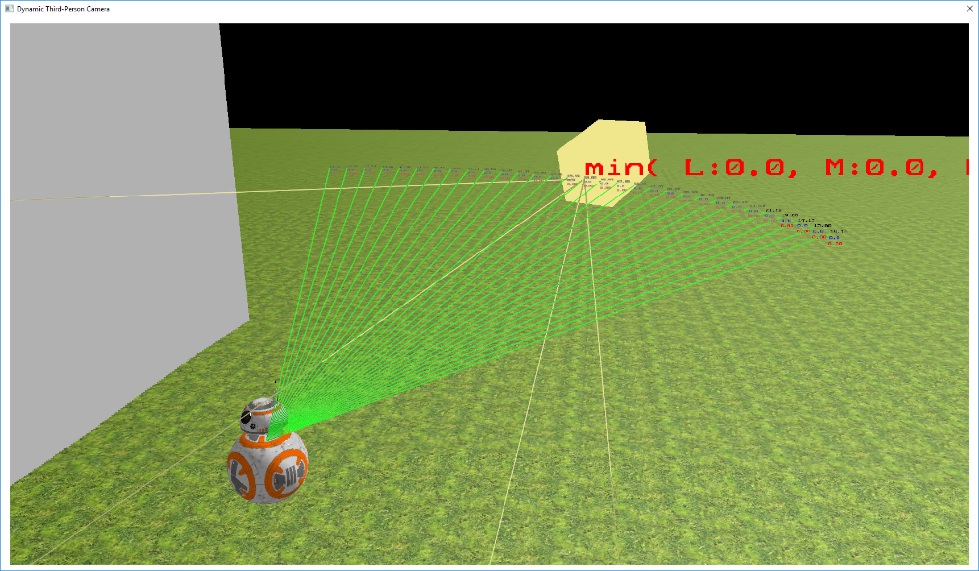
9 Weights assigned to each ray

Because the total number of rays is very high compared to the rays that actually reports an impact, the weighted average ends up being too small; which is it not effective. To mitigate this problem, the assigned weights reacts based on the velocity of the camera, which you'll see in the following video. For reference while watching the video, keep an eye on the debug circle similar to the right image provided below. Each dot represents the target point of a raycast projected on the camera's front/near plane. The blue-green arrow represents camera's current velocity. White dot means that the weight assigned to the raycast is the low, and the red dot represents a higher weight.

Since the chance of camera-collision is much higher in the direction where the camera is going (i.e. direction of the camera velocity), the respective part of the Modified Cone Raycast ends up with much higher weights. Thus, they contribute more towards the final reduction in the radius. This approach produced much more effective results.

Video Debug View - Weighted Raycasts

10 Raycast Weights - Debug Circle

You might ask, why even bother with too many raycasts if getting the weighted average is tricky? I tried raycasts in a simpler 2D cone formation, which works almost flawlessly except in one case: where the potential collision might happen because of the geometry situated directly above/below the raycasts. As a result, the reduction in radius happens very abruptly with 2D Cone Raycast. You can see the comparison in the following video.

Video Comparison - 2D vs 3D Cone Raycast

11 Two-dimensional Cone Raycast

## Behavior Handover

Consider a scenario where the BB8 is standing next to a building, and the player tries to position the camera towards the building. To avoid the collision in such a scenario, the camera will position itself very close to the BB8, and the body of the BB8 ends up covering the majority of the screen. If the camera can switch its behavior from follow camera to shoulder view, then the shoulder view will let the player see what’s ahead much more clearly. The following video shows the comparison.

Video Behavior Handover - Disabled vs Enabled

Each Camera Behavior (like Follow, Shoulder View, or Freelook) has a different set of rules it needs to satisfy. These set of rules are called Camera Constraints - you’ll know more about it in the System Architecture section. Now, when the active behavior gets changed to a different one, the active set of rules also needs to be changed. As you can see in the left debug video below, when the Follow behavior is active the Modified Cone Raycast camera constraint is enabled. After the behavior handover to the shoulder view, the cone raycast gets disabled in favor of enabling constraints of the shoulder view. My system also makes sure that the behavior handover doesn’t happen in middle of a frame. Otherwise it can produce side effects.

Video Behavior Handover - No Debug

Video Behavior Hanover - Debug

## On-Demand Camera Reorientation

This feature lets the player reorient the camera to the back side of BB8 by press of a button. Sounds easy, right? But there is some subtlety to it.

The problem arises because the movement input (left-stick) is relative to the camera. If the camera is facing north and you tilt the left-stick forward then the BB8 will move in the north direction, and when you tilt the stick towards right then the BB8 starts moving towards east (again, relative to the camera).

Keep that in mind while taking a look into the debug video below. You’ll realize that because of the camera relative controls, even though the left-stick is held at the same place, the BB8 changes its movement direction when the camera reorientation happens. This could be frustrating to the player because they never intended to change the movement direction of the BB8, they only intended to change the camera's orientation.

Video Debug Compass - No Input Interpolation

Video Problem Case - Debug Video

In the right debug video, the yellow-red line shows the left-stick input's effective movement direction mapped on the compass (which represents in-game north, east, south, and west). We can see it clearly how the movement/input direction based on left-stick is getting changed in between the START and END of the camera reorientation. I'm going to refer this yellow-red line as camera relative input direction for the lack of a better term.

Our goal is to find a way by which the camera relative input direction maintains its position on the compass during the whole reorientation unless the player tries to change it.

Work on this feature is still in progress, but these are two techniques by which I'm solving it:

1. Input interpolation
2. Retain the input direction

**Input Interpolation:** The name comes from an idea of gently interpolating from a previous camera relative input direction towards the current/actual camera relative input direction. I know it sounds a little confusing, but if you take a look at the debug compass video on right, you can notice how the yellow-red line is following the gray line. The gray line represents input relative to the camera's actual/current orientation. The yellow-red line represents input interpolation that is happening from START to the GRAY line.

Once the player starts adjusting the left-stick to match the camera's orientation, the input interpolation gives them a little more flexibility in terms of time and accuracy.

Video Debug Compass with Input Interpolation

**Retain the input direction:** Ideally, the input interpolation will get activated once the player decides to change the movement direction by moving the left-stick. Unless the player wishes to do so, the BB8 should keep moving in the same direction even while the camera is reorienting. Thus, retaining the input direction on the world compass comes in the picture. The following debug video showcases what I mean.

Video Comparison - Before vs After the Suggested Changes

When the discussed techniques are combined, the operation flow will look like this:

* The player presses a button, and the camera reorientation begins.
* Game retains the input direction. i.e. the player keeps moving in the same direction.
* Once the player moves the left-stick, the input interpolation kicks in. This allows the input direction to smoothly catch up with the camera.

## Look Ahead

The purpose of the Look Ahead feature is to complement the player movement. As the name suggests, the camera tries to look ahead according to where the player is headed. You can see in the video below that if the BB8 is moving forward, the camera moves a little more towards the forward direction. If the BB8 is moving in the right direction, the camera tries to show what’s ahead in the right direction.

Video Look Ahead - No Debug

I implemented a Camera Motion Controller which is responsible for moving the camera and handling this characteristic. Look into the System Architecture section to know more about the Camera Motion Controller.

The motion controller knows the player’s current velocity, and it also knows the current position of the camera and its goal position. So, by adding some offset to the goal position according to the player velocity, the look ahead characteristics can be achieved. In following debug video, the white cross-point showcases the added offset relative to the player's position.

Video Look Ahead offset - Debug

In my initial attempts, I ended up directly adding the player velocity to the camera position, which works well only if the acceleration of BB8 is low. But as I increased the acceleration to higher values for better feeling player movement, the change in offset became too abrupt such that it can easily make the player feel motion sick. Thus, I decided to smoothly interpolate towards the new position as an alternative.

Video Comparison - Initial vs Later Attempts