

Paulo Renato Conceição Mendes

Matrícula: 1921162

# **Projeto Final de Programação: Player de Vídeo 360 graus Interativo**

Rio de Janeiro, Brasil

2020

Paulo Renato Conceição Mendes  
Matrícula: 1921162

## **Projeto Final de Programação: Player de Vídeo 360 graus Interativo**

Trabalho apresentado ao coordenador do programa de pós-graduação em informática da PUC-Rio como requisito para obtenção de nota na disciplina INF2102-Projeto Final de Programação

Pontifícia Universidade Católica do Rio de Janeiro

Departamento de Informática

Programa de Pós-Graduação em Informática

Orientador: Sérgio Colcher

Rio de Janeiro, Brasil

2020

# Sumário

<b>Sumário</b>	.....	2
<b>1</b>	<b>ESPECIFICAÇÃO DO PROGRAMA</b>	3
1.1	Objetivo	3
1.2	Escopo	3
1.3	Requisitos	3
1.3.1	Requisitos Funcionais	3
1.3.2	Requisitos Não-Funcionais	6
1.4	Descrição de Cena 360°	6
1.4.1	Estrutura e Conteúdo	7
1.4.2	Especificação Espacial	8
1.4.3	Especificação Temporal	8
1.4.4	Navegação e Guia de Atenção	9
<b>2</b>	<b>ARQUITETURA</b>	10
2.1	Carregamento de Apresentação	11
2.1.1	Vídeos 360°	11
2.1.2	Objetos de Mídia	12
2.1.3	Multimedia Controller	13
2.2	Visualização de Apresentação	13
2.2.1	Visualização em Realidade Virtual	13
2.2.2	Controles Imersivos	14
<b>3</b>	<b>TESTES</b>	16
3.1	Testes Unitários	16
3.1.1	Resultados	16
3.2	Testes de Estudos de Caso	17
<b>4</b>	<b>DOCUMENTAÇÃO PARA O USUÁRIO</b>	20
<b>5</b>	<b>CÓDIGO FONTE</b>	23

# 1 Especificação do Programa

## 1.1 Objetivo

Com a recente popularidade de câmeras 360° e Head Mounted Displays (HMDS), o uso de vídeos 360° vem crescendo. Esses vídeos são sinais visuais esféricos que possibilitam que espectadores olhem ao redor de uma cena em 360° a partir de um ponto. Essa interação de olhar ao redor é limitada, mas pode ser estendida como elementos interativos como os seguintes: informações 2D/3D sobrepostas, hiperlinks, elementos de entrada adicionais, navegação entre vídeos e outros tipos de interação. Exemplos desse tipo de vídeo 360° interativos são museus virtuais, narrativas interativas imersivas e conteúdo educacional interativo. O objetivo deste projeto de programação final é o desenvolvimento de um player para vídeos 360° interativos que permita a interação do usuário a partir de controles imersivos. Esse player deve também suportar uma descrição de cena 360° com abstrações para facilitar seu desenvolvimento.

## 1.2 Escopo

Este projeto de programação final tem como escopo o desenvolvimento de um player de vídeos 360° interativos, estendidos com conteúdo 2D/3D adicional, elementos de entrada, navegação entre vídeos e conteúdo educacional interativo. Para isso, esse player deve dar suporte à especificação desses elementos com abstrações que facilitem o seu desenvolvimento.

## 1.3 Requisitos

### 1.3.1 Requisitos Funcionais

Com o objetivo de especificar os requisitos funcionais que devem ser suportados pelo player, foram escolhidos três cenários que utilizem vídeos 360° interativos.

**Hipervídeo 360.** Esse cenário é caracterizado pela navegação entre vídeos 360, e pode ser utilizado, por exemplo, para fins educativos ou de entretenimento (ver Figura 1). O usuário pode navegar do vídeo 360° atual para outro, sendo que este último é mostrado em formato de *preview* e pode ser selecionado através de controles imersivos. Além do suporte à navegação, também deve ser possível a adição de conteúdo sobreposto ao vídeo 360 (e.g., imagens, vídeos, texto) para incrementar a experiência do usuário. Um exemplo deste cenário é um tour virtual em um museu, onde o usuário pode navegar por diferentes

salas. Cada sala podendo ter informações adicionais sobre a obra de arte apresentada. Outro exemplo é um ambiente de aprendizagem virtual, onde o usuário pode navegar por diferentes tópicos e aprender mais sobre eles através de informações adicionais sobrepostas no vídeo.

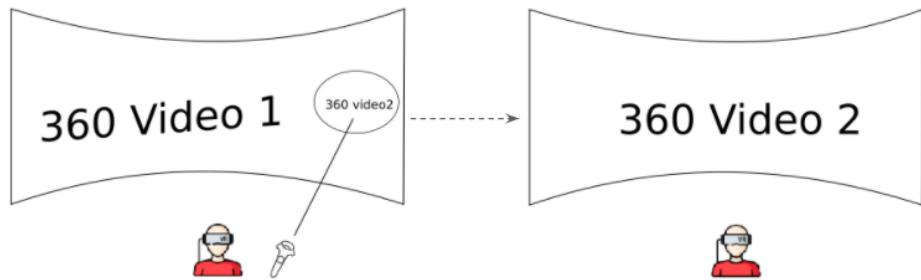


Figura 1 – Cenário de hipervídeo 360º

**Video 360º Acessível.** Nesse cenário, um vídeo 360º é mostrado em conjunto com sua tradução em linguagem de sinais (Libras, por exemplo) ou legendas (ver Figura 2). Para o caso de linguagens de sinais, um vídeo 2D comum pode ser mostrado em modo Picture-in-Picture (PiP). Para o caso das legendas, o texto destas pode ser mostrado em conjunto com o vídeo. Tanto o vídeo 2D quanto as legendas podem (ou não) se moverem de acordo com o movimento da visão do usuário.

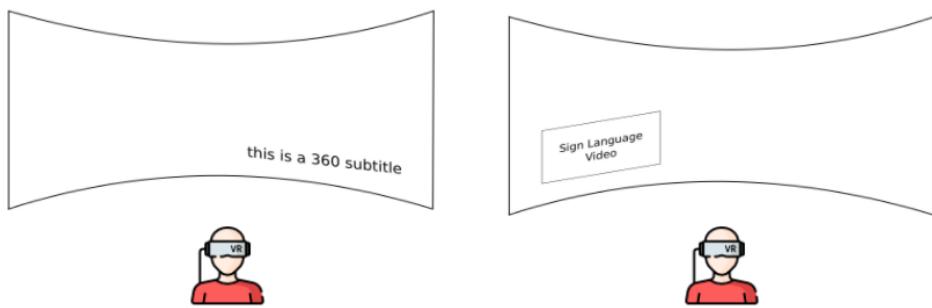


Figura 2 – Cenário de vídeo 360º acessível

**Vídeo 360º com atenção guiada.** Nesse cenário, um vídeo 360º possui uma região recomendada para se olhar. Sempre que o usuário não estiver olhando para a região recomendada, a sua atenção deve ser guiada para onde o usuário deveria estar olhando (ver Figura 3). Além disso, o conteúdo dessa região deve ser mostrado em modo PiP em parte do campo de visão. Dessa forma, o usuário pode sempre acompanhar o conteúdo da região recomendada, mesmo que não esteja olhando diretamente para ela. Além desses guias visuais, guias utilizando áudio espacial podem ser usadas para guiar a atenção do usuário. Por essa razão, o player deve suportar áudio espacial. Um exemplo desse cenário é uma palestra, em que a região recomendada para se olhar é a posição onde o palestrante está posicionado, o palco.

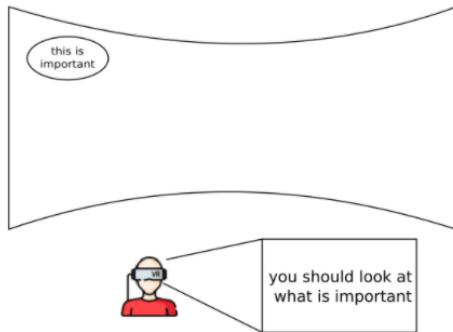


Figura 3 – Cenário de vídeo 360° atenção guiada

Com base nos cenários descritos acima, foram extraídos requisitos de apresentação (RA) e de interação (RI) que devem ser suportados pelo player de vídeos 360° interativos. Tais requisitos são detalhados a seguir.

- **RA.1 Informação sobreposta.** Objetos de mídia adicionais (vídeos, imagens, sons e textos) podem ser posicionados sobre vídeos 360°.
- **RA.2 Reúso.** O player deve dar suporte à reutilização de posições e características de apresentação na autoria de vídeos 360° interativos. Assim, o usuário pode reutilizar o mesmo estilo em diferentes elementos.
- **RA.3 Áudio 3D.** Áudio 3D manipula a entrega de informação auditiva através de caixas de som estéreo ou fones de ouvido, criando a impressão de que o áudio está vindo de uma direção específica.
- **RA.4 Legendas.** Suporte a legendas tanto fixas no ambiente quanto fixas para o usuário.
- **RA.5 Layout Espacial.** Os objetos de mídia devem usar um sistema baseado em coordenadas esféricas, em que o usuário é o centro.
- **RA.6 Tempo de Apresentação.** Os objetos de mídia devem ser apresentados sincronizados com o vídeo 360 da cena.
- **RI.1 Navegação.** Deve ser possível a navegação entre vídeos 360°. Isso inclui a apresentação para o usuário uma prévia dos vídeos que ele pode visitar.
- **RI.2 Hotspot.** Deve ser possível a especificação de uma região do vídeo 360° que dispare ações caso esteja sendo olhada.
- **RI.3 Espelhamento de Viewport.** Deve ser possível o espelhamento de uma região do vídeo 360° definida por um *hotspot*, sendo mostrada em modo PiP

### 1.3.2 Requisitos Não-Funcionais

Dentre os requisitos não-funcionais do sistema, estão os seguintes:

- **RN.1 Usabilidade.** Associada à facilidade no uso do sistema.
- **RN.2 Portabilidade.** O sistema poderá ser executado em múltiplas plataformas.
- **RN.3 Confiabilidade.** Deve-se buscar baixa frequência de falhas e robustez do sistema na recuperação destas falhas.
- **RN.4 Performance.** Relacionada ao tempo de resposta do sistema durante o uso dos recursos disponibilizados.
- **RN.5 Padrões.** Conformidade com os padrões e normas a serem seguidas no desenvolvimento do sistema.

## 1.4 Descrição de Cena 360°

Com o objetivo de atender aos requisitos funcionais definidos na Seção 1.3.1, foi desenvolvida uma representação em Extensible Markup Language (XML) para descrever cenas 360° baseadas em vídeos 360° que atende tais requisitos. A decisão pela criação dessa descrição foi baseada na pouca exploração desse tema pela literatura. Essa descrição, será então interpretada pelo player desenvolvido neste trabalho. Os elementos presentes no modelo de descrição desenvolvido estão presentes na Tabela 1. As Subseções 1.4.1, 1.4.2, 1.4.3, e 1.4.4 descrevem detalhadamente o modelo de descrição de cena utilizado.

element	children	attributes
<i>presentation360</i>	<i>head</i> , <i>body</i>	
<i>head</i>	<i>style</i>	
<i>style</i>	*	
<i>body</i>	<i>scene360+</i>	entry
<i>scene360</i>	<i>text*</i> , <i>image*</i> , <i>video*</i> , <i>subtitle*</i> , <i>preview*</i> , <i>hotspot*</i> , <i>mirror*</i>	src, volume?
<i>text</i> , <i>image</i>		id, src, r?, phi?, theta?, style?, begin?, dur?, followCamera?, onselect?
<i>audio</i> , <i>video</i> , <i>subtitle</i> , <i>preview</i>		id, src, r?, phi?, theta?, style?, begin?, dur?, followCamera?, onselect?, clipBegin?, clipEnd?
<i>mirror</i>		id, src, r?, phi?, theta?, style?, begin?, dur?, followCamera?
<i>hotspot</i>		id, src, r?, phi?, theta?, style?, begin?, dur?, onLookAt?, duringNotLookingAt?, onSelect?

Tabela 1 – BNF dos elementos do modelo de descrição de cena

#### 1.4.1 Estrutura e Conteúdo

O elemento `<presentation360>` é o elemento raiz do modelo. Ele possui dois elementos-filho: `<head>` e `<body>`.

No elemento `<head>`, configurações de apresentação reusáveis podem ser agrupadas pelo elemento `<style>` (RA.2). Assim, qualquer elemento pode reutilizar os atributos referenciando o *id* (atributo identificador) de um elemento `<style>` pré-definido.

O elemento `<body>` é composto por diferentes elementos `<scene360>`. O atributo *entry* define quais dessas cenas será iniciada com a aplicação. O elemento `<scene360>` é definido por seu *id*, *src* (caminho para o vídeo 360° da cena) e *volume*. Cada cena é composta por um conjunto de objetos de mídia e o sincronismo temporal desses objetos (RA.1). Tais objetos podem ser imagens, vídeos, sons ou textos, cada um com seu respectivo elemento XML. Os atributos a seguir são compartilhados por todos os objetos de mídia:

- *id*: atributo identificador pelo qual o objeto de mídia pode ser referenciado por outros elementos do modelo.
- *r, phi, theta*: definem o posicionamento espacial do objeto de mídia. O funcionamento desses atributos são discutidos na Subseção [1.4.2](#).
- *begin*: define o instante de tempo, em segundos, em que o elemento é iniciado relativamente ao tempo do vídeo 360° de que ele é filho.
- *dur*: define a duração, em segundos, do elemento
- *clipBegin, clipEnd*: definem a porção do elemento que será apresentada. Pode-se, por exemplo, definir que apenas o trecho de 5s a 10s de um áudio será tocado.
- *followCamera*: um atributo booleano que, caso seja verdadeiro, faz com que o elemento se move com a visão do usuário. Esse movimento cria a impressão de que o elemento é fixo para o usuário.
- *onSelect*: aponta para o *id* de uma outra cena (`<scene360>`) para a qual o usuário será transportado caso o elemento seja selecionado.

Além dos atributos descritos acima, objetos de mídia específicos possuem atributos adicionais:

- `<image>`, `<audio>`, `<video>` e `<subtitle>`(RA.4) possuem o atributo *src* para especificar o caminho do arquivo referenciado.
- `<audio>` e `<video>` possuem o atributo *volume* para especificar o volume em que são tocados, variando de 0 a 1. Esses dois elementos possuem a característica de áudio espacial (RA.3)

Os elementos `<preview>` e `<mirror>` são objetos de mídia não tradicionais e são utilizados para navegação e direcionamento de atenção (detalhados na Subseção 1.4.4).

### 1.4.2 Especificação Espacial

Para especificar o posicionamento dos objetos de mídia presentes em uma cena 360°, foi utilizado um sistema de coordenadas polares (RA.5). A Figura 4a mostra da definição das coordenadas do ponto  $C$ , que representa o centro de um objeto de mídia.  $C1$  é a projeção de  $C$  no plano  $xz$ , enquanto  $C2$  é a projeção no plano  $yz$ . O atributo  $r$  define o raio da circunferência imaginária onde o objeto de mídia é posicionado. Todas essas esferas imaginárias são concêntricas. Os ângulos são definidos utilizando o segmento que vai da origem (centro da esfera) para a borda como referência. O atributo  $phi$  ( $\phi$ ) especifica o ângulo horizontal (em graus) desse segmento para o segmento que vai da origem para  $C1$ . De modo similar, o atributo  $theta$  ( $\theta$ ) define o ângulo vertical, mas usando  $C2$ .

A rotação de um objeto de mídia é feita de modo que o vetor normal de sua superfície aponta para a origem do sistema de coordenadas. Dessa forma, em qualquer posição que esse objeto de mídia seja posicionado, ele está sempre de frente para o usuário. A Figura 4b mostra essa rotação, em que o ponto  $A$  é o centro de um objeto de mídia.

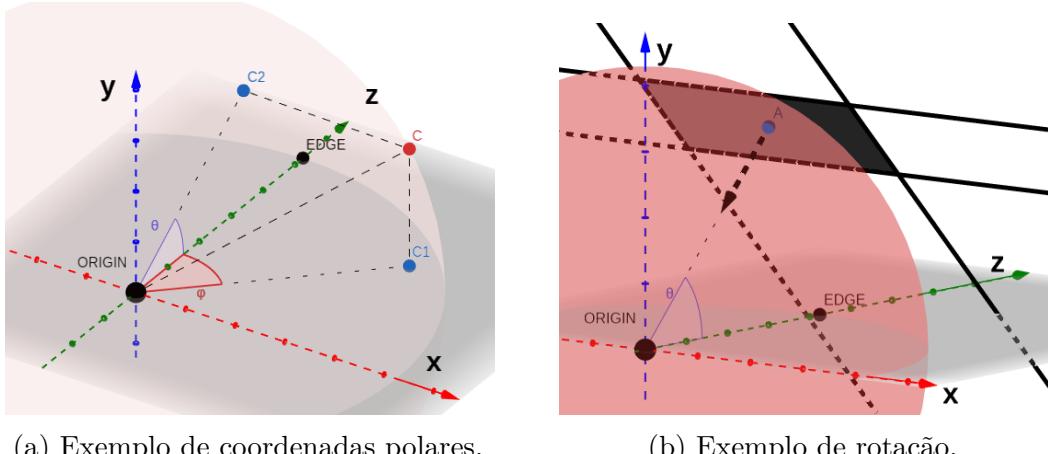


Figura 4 – Sistema de coordenadas polares

### 1.4.3 Especificação Temporal

Cada `<scene360>` possui sempre um vídeo 360. A cena inicial, em que a aplicação começa, é definida pelo atributo `entry` do elemento `<body>`. Esse atributo referencia o `id` da `<scene360>` inicial. Objetos de mídia dentro de uma cena 360 são sincronizados com o vídeo 360 principal (RA.6) através dos atributos `begin` e `dur`. Tais atributos definem, respectivamente, o tempo de início e duração da apresentação do objeto de mídia sendo especificado.

#### 1.4.4 Navegação e Guia de Atenção

O atributo *onSelect* é usado para permitir a navegação entre cenas 360 (RI.1). Esse atributo pode ser definido em qualquer objeto de mídia, referenciando o *id* da cena-alvo para a qual deseja-se navegar. Assim, esse objeto de mídia transforma-se em um objeto que pode ser selecionado. Além disso, o elemento *<preview>* define um objeto que replica o vídeo 360 da cena-alvo como uma miniatura selecionável que direciona o usuário para essa cena.

O elemento *<hotspot>* é utilizado para definir regiões de interesse em um vídeo 360º. Por exemplo, o palco em um show. Esse elemento possui os atributos *onLookAt* e *duringNotLookingAt*. O primeiro especifica um elemento que é iniciado quando o usuário olha para o *<hotspot>*. O segundo especifica um elemento que é mostrado sempre que o usuário não está olhando para o *<hotspot>*.

O elemento *<mirror>* faz o espelhamento de uma região da *<scene360>* (RI.3) e a apresenta em modo PiP. Essa região é especificada definindo o atributo *src* do *<mirror>* como o *id* de um elemento *<hotspot>*. Assim, o autor da aplicação pode definir que, quando o usuário esteja olhando para uma região que não é de interesse, o elemento *<mirror>* é iniciado em PiP, atrairindo atenção do usuário para a região de interesse.

## 2 Arquitetura

O player foi implementado utilizando a *engine* de jogos Unity 3D, por ela fornecer suporte ao uso de *Head Mounted Displays* (HMDs) e facilitar o desenvolvimento de aplicações tridimensionais. Dentre as diversas versões disponíveis, optou-se pela versão Unity 2019 *Long-Term Support (LTS)* por esta ser mais estável e recente. O elemento mais básico de um projeto desenvolvido no Unity é o *gameObject*. Ele possui um componente *transform*, que define a posição, rotação e escala do *gameObject* associado. Além do *transform*, podem ser adicionados outros componentes nativos do Unity a *gameObjects* como: texturas, imagens, sons, *scripts*, etc. Os *scripts* controlam o comportamento dos *gameObjects*, bem como dos demais componentes a eles adicionados. O *player* foi desenvolvido como uma Unity *scene*, sendo assim executado dentro do ambiente da Unity utilizando *gameObjects* e relacionamentos entre eles usando *scripts*.

A pasta raiz do projeto está dividida em 5 pastas: *.vs*, *Assets*, *Logs*, *Packages* e *ProjectSettings*. A pasta *Assets* contém os artefatos desenvolvidos neste projeto, as demais pastas possuem arquivos gerados automaticamente pelo Unity. A pasta *Assets* possui as seguintes pastas:

- *Fonts*: possui as fontes tipográficas utilizadas no projeto.
- *Materials*: possui os materiais utilizados no projeto. Os materiais defininem do que "são feitos" os *gameObjects*. Poderíamos ter, por exemplo, madeira como um material.
- *Textures*: possui as texturas/imagens utilizadas no projeto.
- *Scenes*: possui a cena do player desenvolvido no projeto. Essa cena possui o *gameObject presentation360*, que controla toda a execução da apresentação.
- *Scripts*: possui os códigos-fonte do projeto. Alguns códigos são adicionados a *gameObjects* e outros são utilizados como classes.
- *Prefabs*: possui *gameObjects* que são carregados em tempo de execução. Foram criados *prefabs* para cada um dos tipos de objeto de mídia suportados pelo modelo de descrição.
- *Examples* possui alguns exemplos de aplicações que podem ser executadas pelo player.

Além dessas pastas, também estão presentes as pastas iniciadas com o prefixo “SteamVR”. Essas pastas e o conteúdo destas fazem parte do pacote *SteamVR*, utilizado para suportar a utilização do HMD HTC Vive.

A Figura 5 mostra os módulos do player. Tais módulos estão divididos em duas partes: *carregamento de apresentação* e *visualização de apresentação*, descritas a seguir.

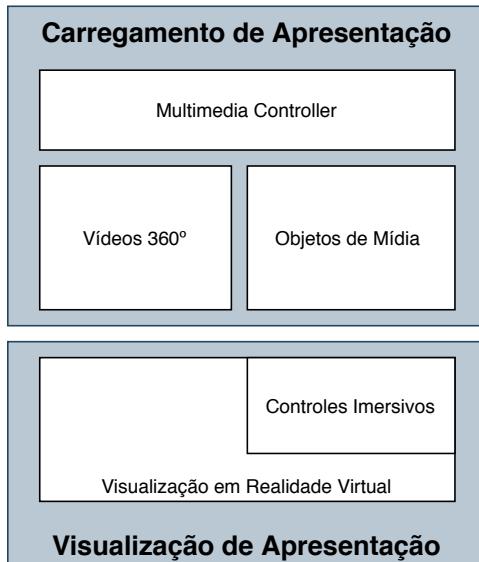


Figura 5 – Módulos do player de vídeos 360° interativos

## 2.1 Carregamento de Apresentação

Esta Seção detalha o carregamento e preparação de um vídeo 360° interativo de acordo com uma cena especificada em XML utilizando o modelo de descrição de cena definido na Seção 1.4.

### 2.1.1 Vídeos 360°

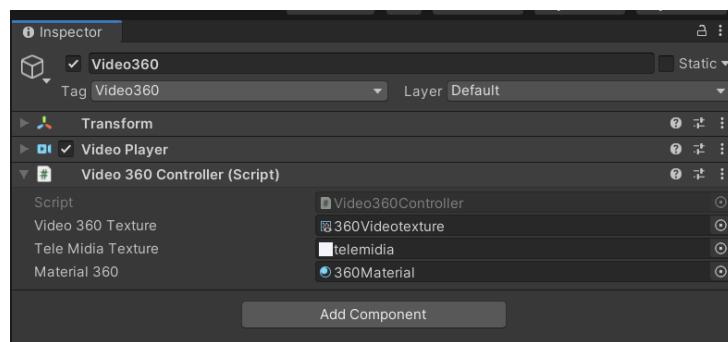


Figura 6 – Prefab para Vídeos 360°

Para cada vídeo 360° definido na cena, é carregado um *prefab* que possui um componente *Video Player* do Unity e o script *Video 360 Controller*. Esse script controla as ações do vídeo (tocar e parar), e também armazena os objetos de mídia adicionais que estão associados a esse vídeo. Nesse script, é definido que, quando o vídeo está tocando, o conteúdo visual do vídeo é renderizado na *skybox* do ambiente Unity através

da renderização do *Video Player* para um textura. Tal textura é atribuída ao material da *skybox*, fazendo com que cada quadro seja mostrado nela sucessivamente. Quando o vídeo não está tocando, o *skybox* mostra o logo do laboratório TeleMídia. A Figura 6 mostra os componentes do prefab de vídeo 360°.

### 2.1.2 Objetos de Mídia

Cada *prefab* de objeto de mídia possui um *script* controlador (terminando com o sufixo *Controller*) correspondente. Os controladores dos objetos de mídia adicionais estendem a classe *MediaControllerAbstract* (ver Figura 7). Essa classe possui sete métodos concretos que fazem o controle dos componentes de qualquer objeto de mídia, em relação ao tempo de início e duração, relação com o vídeo principal, posicionamento de acordo com coordenadas polares (convertidas para cartesianas), etc. Ela também possui três métodos abstratos: *Load*, *PlayMedia* e *StopMedia*. Esses três métodos precisam ser implementados por qualquer objeto de mídia adicional. O método *Load* deve carregar os elementos necessários para o funcionamento do objeto mídia a qual ele está associado. Por exemplo, o *prefab* de vídeo precisa dos componentes *videoPlayer* e *audioSource* do Unity, além de renderizadores e detectores de colisão (para a seleção na navegação). Já os métodos *PlayMedia* e *StopMedia* devem utilizar os componentes específicos de cada objeto de mídia para criar os ações de tocar e parar a sua reprodução. A utilização da herança, facilita a adição e suporte de possíveis novos objetos de mídia.

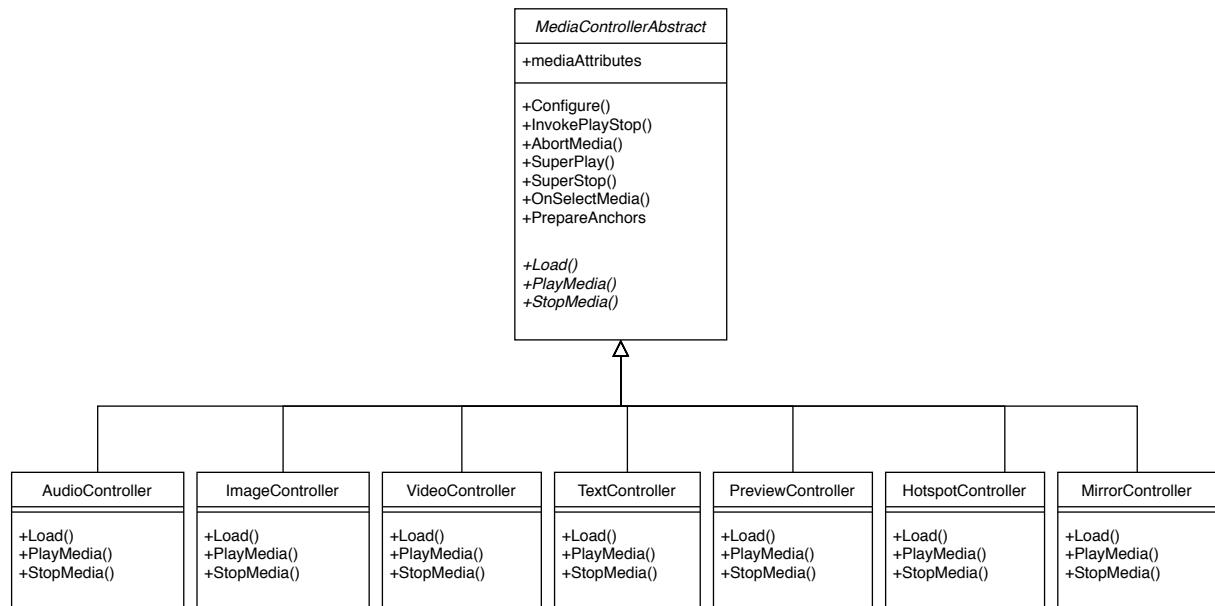


Figura 7 – Diagrama de classes dos controladores de objetos de mídia

### 2.1.3 Multimedia Controller

O *Multimedia Controller* é definido por um *gameObject* que possui como componente o script *MultimediaControllerScript*. Esse script é responsável por ler um arquivo XML especificado pelo usuário, carregar todos os vídeos 360º definidos nele e adicionar os objetos de mídia relacionados a cada um deles, bem como seus comportamentos. Por essa razão, ele recebe os prefabs para cada tipo de mídia para que os possa instanciar em tempo de execução. Além de fazer o carregamento, esse script também controla o estado da aplicação, a iniciando e parando. A Figura 8 mostra o *gameObject* *Multimedia Controller*.

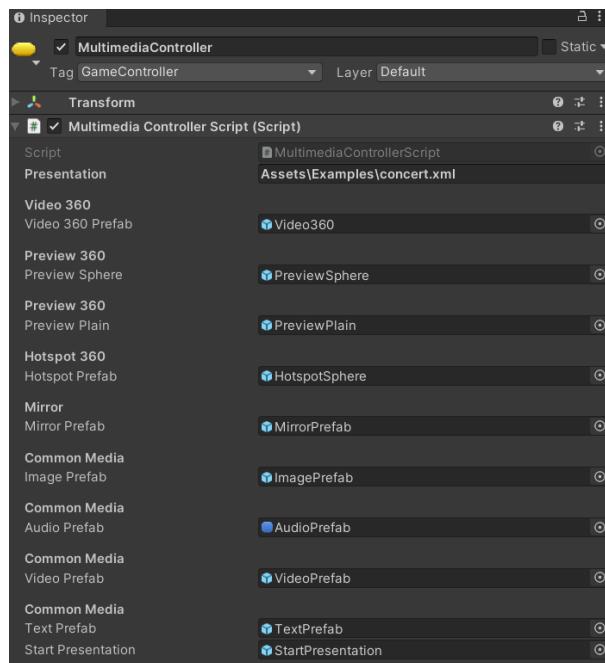


Figura 8 – Módulo Multimedia Controller

## 2.2 Visualização de Apresentação

Esta Seção descreve a visualização da apresentação, uma vez que esta foi carregada e preparada pelos módulos descritos na Seção anterior.

### 2.2.1 Visualização em Realidade Virtual

A câmera (posição do usuário) é posicionada no centro do sistemas de coordenadas polares (0,0,0) e possui um *field of view* (fov) de 60 graus. Para a utilização do HMD HTC Vive, é utilizado o pacote *Steam VR* que possui uma câmera que se move de acordo com a posição e rotação do Vive. Assim, o usuário é posicionado no centro, e os objetos de mídia são mostrados ao seu redor, bem como os vídeos 360º (no plano mais externo).

A posição desta câmera é utilizada para verificar quando o usuário está ou não olhando para um *hotspot*, através do script *LookHotspotScript*. A Figura 9 mostra o

posicionamento de um *hotspot*, da câmera e de sua visualização. Um *hotspot* é configurado como uma esfera invisível (mostrado na imagem apenas para efeitos ilustrativos). No caso da imagem, o usuário estaria olhando parcialmente para o *hotspot*.

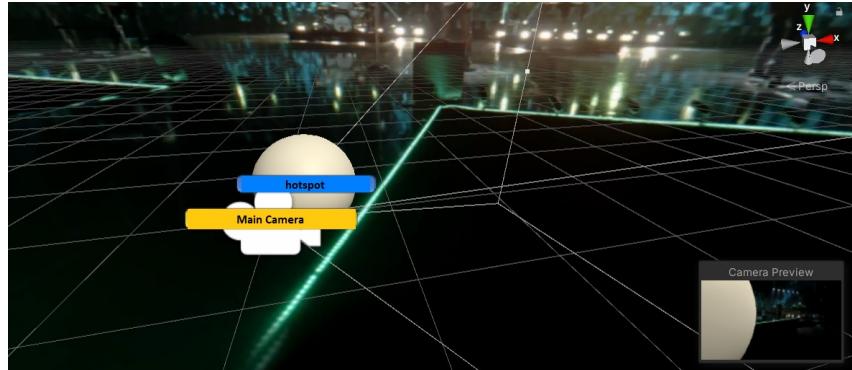


Figura 9 – Verificação de visualização de hotspot

Como a câmera se move de acordo com o movimento do usuário, também foi colocado nela o componente *AudioListener* do Unity. Esse componente capta os sons emitidos pelos demais *gameObjects* da cena, criando, assim, a sensação de áudio espacial.

Também foi pensado o caso de um HTC Vive ou outro dispositivo de realidade virtual não estar conectado ao player. Por isso, foi criado o script *RotateScript*, que possibilita o movimento da câmera com o arrastar do mouse. A Figura 10 mostra a câmera de realidade virtual com todos os seus componentes.

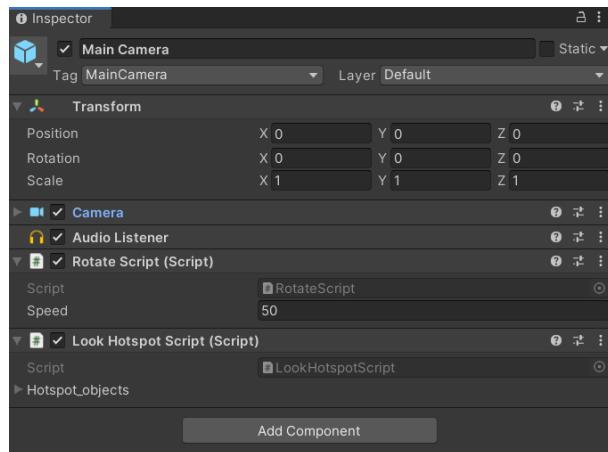


Figura 10 – Componentes da câmera de realidade virtual

### 2.2.2 Controles Imersivos

Os controles imersivos são utilizados para a seleção e navegação. Para isso são utilizados os controles do HTC Vive, que são detectados e rastreados utilizando o pacote *SteamVR*. Para seleção e navegação, foi criado o script *RaySelect* que lança um raio a partir da posição e rotação do controle (ver Figura 11). Caso o botão de gatilho do controle

seja pressionado e o raio esteja colidindo com algum objeto da cena, é iniciado o objeto correspondente ao atributo *onSelect* do objeto selecionado. Foi utilizado o componente do Unity *Line Renderer* para desenhar o raio que sai do controle, facilitando a utilização deste pelo usuário. Também foi feito um *gameObject* auxiliar, chamado *Controller Detector*, com o script *Controller Checker*, que é responsável por esconder o objeto virtual do controle caso ele esteja desativado. Esses comportamentos são compartilhados por ambos os controles: direito e esquerdo.

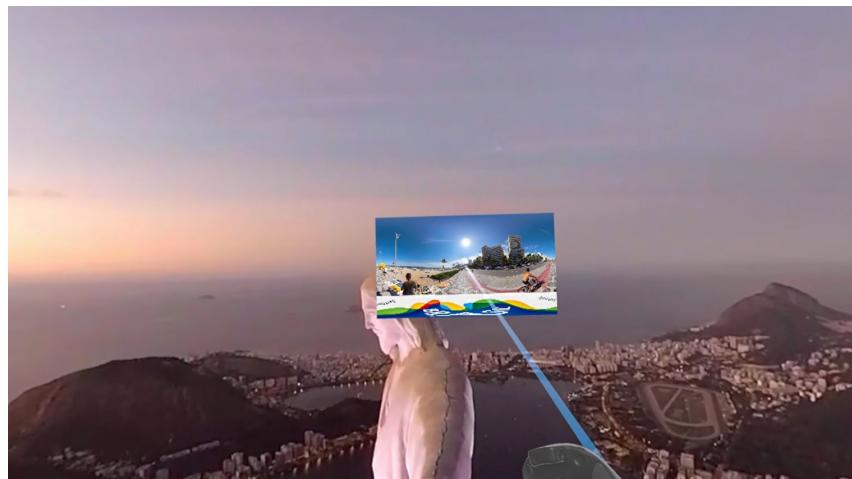


Figura 11 – Controle imersivo com raio seletor.

# 3 Testes

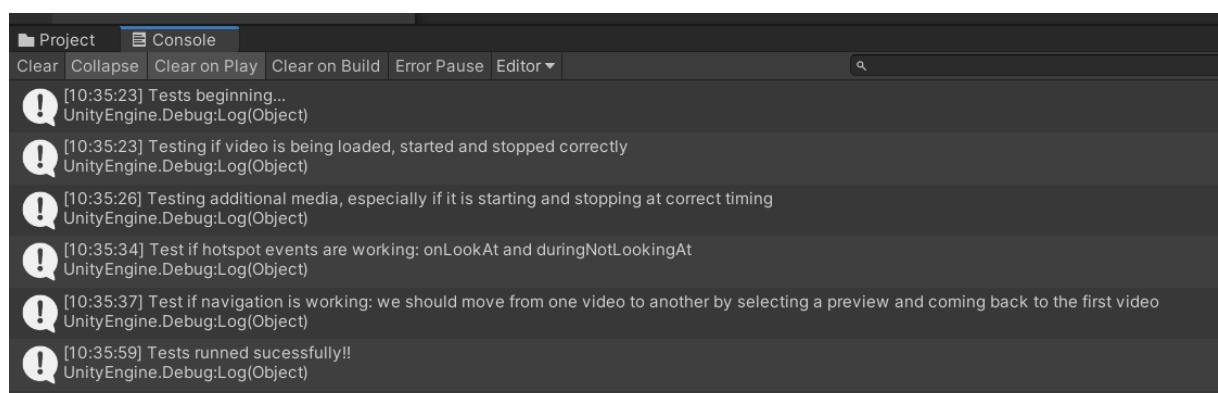
Com o objetivo de verificar as funcionalidades do sistema, foram feitos dois tipos de testes: testes unitários e testes de estudos de caso. Esses testes estão descritos respectivamente nas Seções 3.1 e 3.2.

## 3.1 Testes Unitários

Para os testes unitários, foi criado o arquivo *TestScript.cs*. Nesse arquivo estão definidos quatro testes unitários, correspondentes às principais funcionalidades do *player* desenvolvido neste projeto final.

- *TestLoadStartStop360Video*. Nesse teste, é verificado com assertivas (Debug.Assert) se um vídeo 360 é carregado, iniciado e parado corretamente.
- *TestAdditionalMedia360Video*. Nesse teste, é verificado com assertivas (Debug.Assert) se uma mídia associada a um vídeo 360 é iniciada corretamente de acordo com seu tempo de início (*begin*) e parada corretamente de acordo com sua duração (*duration*).
- *TestHotSpot\_OnFocus\_DuringOutOfFocus*. Nesse teste, é verificado com assertivas (Debug.Assert) se as mídias associadas a um *hotspot* são iniciadas e paradas corretamente de acordo com as definições de *onLookAt* e *duringNotLookingAt*.
- *TestNavigation*. Nesse teste, é verificado com assertivas (Debug.Assert) se a navegação (mudar de um vídeo 360 para outro) é realizada corretamente ao selecionar um elemento *preview* que possui o atributo *onSelect*.

### 3.1.1 Resultados



The screenshot shows the Unity Editor's Console tab. The tab title is "Console". Below the title, there are several buttons: "Project", "Clear", "Collapse", "Clear on Play", "Clear on Build", "Error Pause", and "Editor ▾". A search bar is located at the top right. The main area displays a list of log messages, each preceded by a blue exclamation mark icon. The messages are:

- [10:35:23] Tests beginning...  
UnityEngine.Debug:Log(Object)
- [10:35:23] Testing if video is being loaded, started and stopped correctly  
UnityEngine.Debug:Log(Object)
- [10:35:26] Testing additional media, especially if it is starting and stopping at correct timing  
UnityEngine.Debug:Log(Object)
- [10:35:34] Test if hotspot events are working: onLookAt and duringNotLookingAt  
UnityEngine.Debug:Log(Object)
- [10:35:37] Test if navigation is working: we should move from one video to another by selecting a preview and coming back to the first video  
UnityEngine.Debug:Log(Object)
- [10:35:59] Tests runned sucessfully!!  
UnityEngine.Debug:Log(Object)

Figura 12 – Logs dos testes unitários do Player.

Os testes foram realizados com sucesso. O player realizou com sucesso todas as funcionalidades definidas nos testes. Os logs do teste podem ser vistos na Figura 12. Além disso, um vídeo da execução dos testes está disponível no YouTube.<sup>1</sup>

## 3.2 Testes de Estudos de Caso

Como estudo de caso para validar nosso modelo e player propostos nesse projeto de programação final, foram implementadas duas aplicações que utilizam suas funcionalidades.

A primeira aplicação consiste em um tour virtual na cidade do Rio de Janeiro (ver Figura 13). Nesse tour, o usuário começa em uma visão aérea da cidade. Então, o usuário pode escolher navegar para outras duas localidades: praia de Ipanema ou Pão de Açúcar. Essa navegação ocorre através da seleção dos elementos *preview* com controles imersivos. Uma vez nestas localidades, o usuário pode voltar para a cena inicial. Em cada uma dessas cenas foi adicionado uma música relacionada à cidade do Rio com configurações de áudio 3D.



Figura 13 – Tour Virtual no Rio de Janeiro

O código dessa aplicação pode ser visto no Código 1. O player tocou a aplicação com sucesso e um vídeo que mostra essa aplicação sendo tocada pode ser visto no YouTube.<sup>2</sup>

Código 1 – Exemplo Tour Virtual no Rio de Janeiro

```
<?xml version="1.0" encoding="utf-8" ?>

<presentation360>
<head>
  <style id="beachPos" r="3" phi="-180" theta="0"/>
  <style id="center" r="10" phi="0" theta="0"
    followCamera="true"/>
```

<sup>1</sup> <https://youtu.be/GPXYgihgAI>

<sup>2</sup> <https://youtu.be/6RKXTC438cY>

```

<style id="songPos" r="0" phi="0" theta="0"/>
<style id="aerialViewSt" clipBegin="5s" clipEnd="10s" r="8"
      phi="-80" theta="0" begin="2s" dur="20s"/>
</head>
<body entry="aerialView">
<scene360 id="aerialView" src="D:/Movies/aerialView.mp4">
  <audio begin="0s" src="D:/Songs/mas_que_nada.mp3"
         style="songPos" volume="1"/>
  <preview clipBegin="5s" clipEnd="10s" src="ipanema" r="4"
            phi="0" theta="0" begin="7s" dur="20s" shape="sphere"/>
  <preview clipBegin="5s" clipEnd="10s" src="sugarloaf" r="8"
            phi="-100" theta="0" begin="7s" dur="20s"/>
</scene360>
<scene360 id="ipanema" src="D:/Movies/ipanema.mp4">
  <audio begin="0s" src="D:/Songs/garota_de_ipanema.mp3"
         style="songPos" volume="1"/>
  <preview src="aerialView" style="aerialViewSt"
           shape="sphere"/>
</scene360>
<scene360 id="sugarloaf" src="D:/Movies/sugarloaf.mp4">
  <audio begin="0s" src="D:/Songs/aquele_abraco.mp3"
         style="songPos" volume="1"/>
  <preview src="aerialView" style="aerialViewSt"/>
</scene360>
</body>
</presentation360>

```

Na segunda aplicação, o usuário é apresentado à um concerto musical (ver Figura 14). O usuário pode mover sua cabeça livremente, e quando não está olhando para o palco é mostrada uma transmissão do que está ocorrendo no palco em PiP.



Figura 14 – Concerto musical com *hotspot* no palco.

O código dessa aplicação pode ser visto no Código 2. Nesse código é definido uma

*scene360* que possui um texto de copyright, um áudio da música, e os elementos *hotspot* e *mirrror*. O *hotspot* é posicionado no local do palco, e é definido que quando o usuário não está olhando para ele, é mostrado em PiP o que acontece no palco através do elemento *mirror*. O áudio também é posicionado no local do palco, de modo que o usuário tem a impressão de que o som está vindo do local onde o palco está posicionado. O player tocou a aplicação com sucesso e um vídeo que mostra essa aplicação sendo tocada pode ser visto no YouTube.<sup>3</sup>

Código 2 – Exemplo do Concerto Musical

```
<?xml version="1.0" encoding="utf-8" ?>

<presentation360>

<head>
    <style id="palco_pos" r="0,7" phi="20" theta="0"/>
</head>

<body entry = "concert">
    <scene360 id="concert" src="D:/Movies/video_aha.mp4"
        volume="1">
        <text id="copyright" r="14" theta="20" text="The rights of
            this video belongs to 'Nobel Peace Prize Concert'
            begin="5s" duration="20s" followCamera="true"/>
        <audio id="audio_video" begin="0s"
            src="C:/Users/paulo/Videos/Video360/audio_aha.wav"
            style="palco_pos" volume="1"/>
        <hotspot id="hotspot_palco" style ="palco_pos" begin="1s"
            duringNotLookingAt="mirror_palco"/>
        <mirror id="mirror_palco" src="hotspot_palco"
            followCamera="true" r="4" phi="0" theta="20"/>
    </scene360>
</body>
</presentation360>
```

---

<sup>3</sup> <https://youtu.be/BPVGBCFifP0>

# 4 Documentação para o Usuário

A utilização do player é bem simples. O primeiro passo é fazer o download do projeto, que pode ser feito no GitHub.<sup>1</sup> Uma vez que o projeto esteja baixado, o usuário deve abrir a cena Player360 com o Unity 2019 Long-Term Support (LTS). Essa cena é encontrada no caminho Assets>Scenes>Player360.unity. Ao abrir a cena, o usuário verá a tela mostrada na Figura 15.

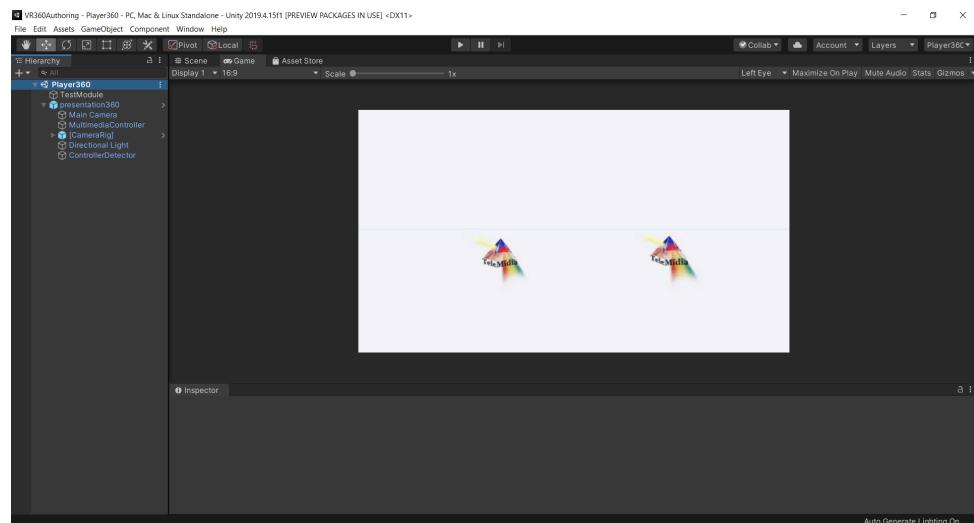


Figura 15 – Tela Inicial

Uma vez na tela inicial, o usuário deve selecionar na barra lateral esquerda o objeto *MultimediaController*, como mostrado na Figura 16.

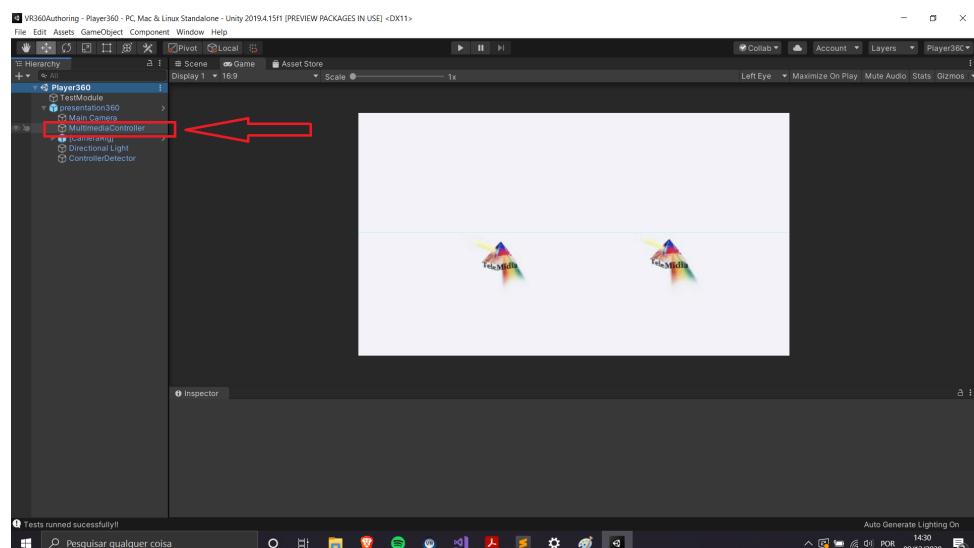


Figura 16 – Selecionando o objeto MultimediaController

<sup>1</sup> <https://github.com/TeleMidia/VR360Authoring>

Após selecionar o objeto *MultimediaController*, abrirá um menu na parte inferior da tela. O usuário deverá colocar no campo *Presentation* o path para o XML do vídeo 360 interativo que deseja abrir. Esse processo é mostrado na Figura 17. Para a edição do XML, o usuário pode usar qualquer editor de texto de sua preferência. O Unity possui integração com o *Visual Studio*, o que facilita a edição.

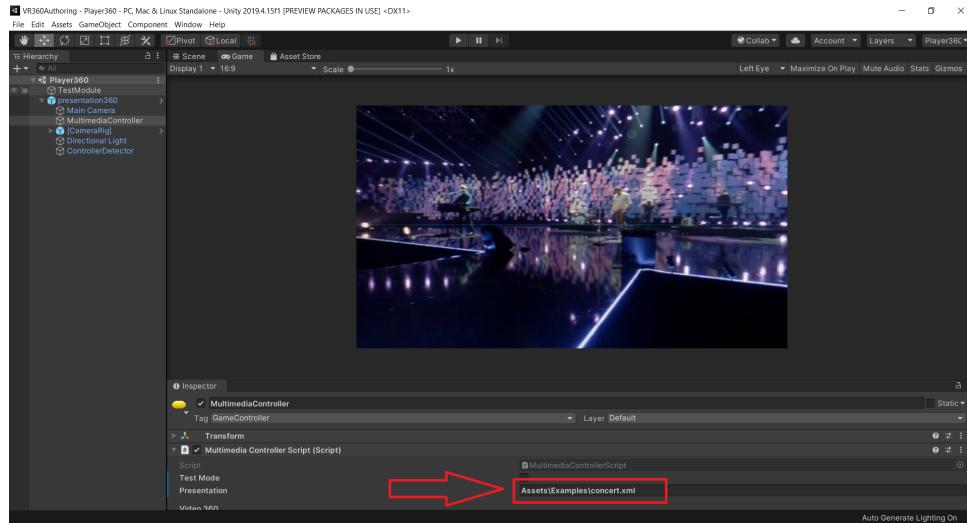


Figura 17 – Inserindo o path da apresentação a ser tocada

Após inserir o path para a aplicação desejada, basta clicar no botão de play (ver Figura 18) para que a apresentação inicie.

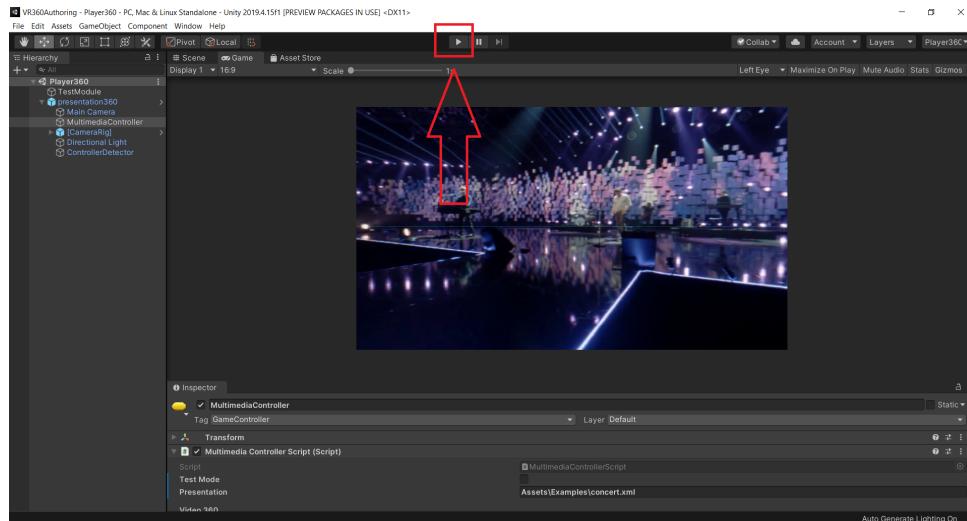


Figura 18 – Iniciando a apresentação desejada

Caso deseje executar os testes, após clicar no objeto *MultimediaController*, o usuário deve marcar a opção *TestMode* e em seguida apertar o botão de play. Esse processo é mostrado na Figura 19. O player pode ser utilizado com ou sem um óculos de realidade virtual. A utilização sem um óculos impossibilita a navegação entre cenas, uma vez que esta é feita a partir da seleção com controles imersivos. Para os testes, não é necessária

a utilização de óculos de realidade virtual ou controles imersivos, pois essas ações são simuladas.

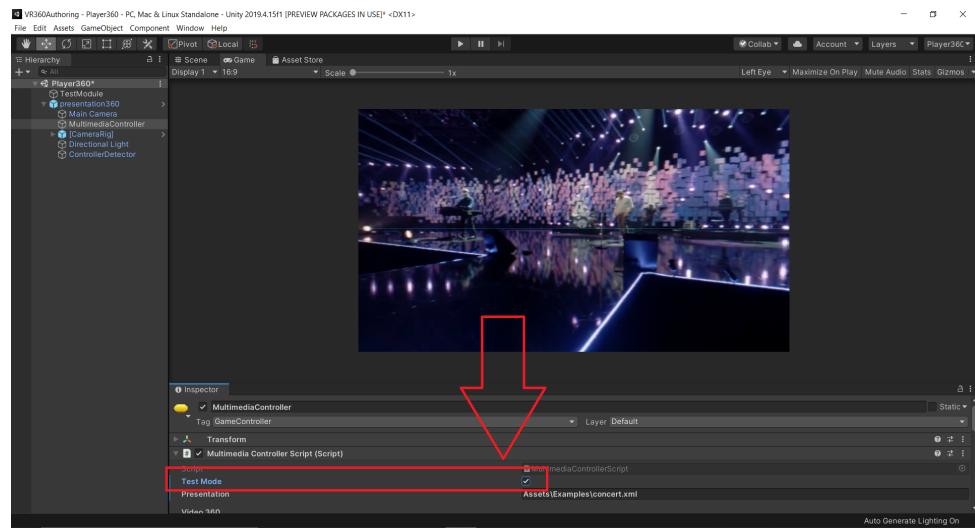


Figura 19 – Iniciando o modo de testes.

## 5 Código Fonte

Todo o código fonte deste projeto de programação final está disponível no GitHub.<sup>1</sup> Além disso, foi gerada uma documentação de todo o código fonte, presente no fim deste documento.

---

<sup>1</sup> <https://github.com/TeleMidia/VR360Authoring>

## Player de Vídeo 360 graus Interativo

Generated by Doxygen 1.8.20



---

<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 AudioController Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Member Function Documentation	6
3.1.2.1 Load()	6
3.1.2.2 LoadAudio()	7
3.1.2.3 PlayMedia()	7
3.1.2.4 StopMedia()	7
3.2 ControllerChecker Class Reference	8
3.2.1 Detailed Description	8
3.2.2 Member Function Documentation	8
3.2.2.1 Change()	8
3.3 HotspotController Class Reference	9
3.3.1 Detailed Description	9
3.3.2 Member Function Documentation	10
3.3.2.1 Load()	10
3.3.2.2 PlayMedia()	10
3.3.2.3 StopMedia()	10
3.4 ImageController Class Reference	11
3.4.1 Detailed Description	11
3.4.2 Member Function Documentation	11
3.4.2.1 Load()	12
3.4.2.2 PlayMedia()	12
3.4.2.3 StopMedia()	12
3.5 LookHotspotScript Class Reference	12
3.5.1 Detailed Description	13
3.5.2 Member Function Documentation	13
3.5.2.1 Update()	13
3.6 MediaControllerAbstract Class Reference	13
3.6.1 Detailed Description	16
3.6.2 Member Function Documentation	16
3.6.2.1 AbortMedia()	16
3.6.2.2 Configure()	17
3.6.2.3 InvokePlayStop()	18
3.6.2.4 Load()	19
3.6.2.5 OnSelectMedia()	19
3.6.2.6 PlayMedia()	19

---

3.6.2.7 PrepareAnchors()	20
3.6.2.8 StopMedia()	20
3.6.2.9 SuperPlay()	20
3.6.2.10 SuperStop()	21
3.6.3 Property Documentation	22
3.6.3.1 IsPlaying	22
3.7 MirrorController Class Reference	22
3.7.1 Detailed Description	23
3.7.2 Member Function Documentation	23
3.7.2.1 Load()	23
3.7.2.2 PlayMedia()	23
3.7.2.3 StopMedia()	23
3.7.2.4 Update()	24
3.8 MultimediaControllerScript Class Reference	24
3.8.1 Detailed Description	26
3.8.2 Member Function Documentation	26
3.8.2.1 AddAdditionalMedia()	26
3.8.2.2 AddVideo360()	26
3.8.2.3 LoadXmlFile()	27
3.8.2.4 SetAsInitial()	28
3.8.2.5 Start()	28
3.8.2.6 StartPresentation()	29
3.8.2.7 StopPresentation()	29
3.9 PreviewController Class Reference	30
3.9.1 Detailed Description	31
3.9.2 Member Function Documentation	31
3.9.2.1 Load()	31
3.9.2.2 PauseVideoPreview()	31
3.9.2.3 PlayMedia()	32
3.9.2.4 PlayVideoPreview()	32
3.9.2.5 StopMedia()	32
3.10 RaySelect Class Reference	33
3.10.1 Detailed Description	34
3.10.2 Member Function Documentation	34
3.10.2.1 ShootLaserFromTargetPosition()	34
3.10.2.2 Start()	34
3.10.2.3 TriggerDown()	35
3.10.2.4 Update()	35
3.11 RotateScript Class Reference	36
3.11.1 Detailed Description	37
3.11.2 Member Function Documentation	37
3.11.2.1 Update()	37

---

---

3.11.3 Member Data Documentation . . . . .	37
3.11.3.1 speed . . . . .	37
3.12 SubtitleFragment Class Reference . . . . .	37
3.12.1 Detailed Description . . . . .	38
3.12.2 Constructor & Destructor Documentation . . . . .	38
3.12.2.1 SubtitleFragment() . . . . .	38
3.12.3 Member Function Documentation . . . . .	38
3.12.3.1 ToString() . . . . .	38
3.12.4 Member Data Documentation . . . . .	39
3.12.4.1 begin . . . . .	39
3.12.4.2 duration . . . . .	39
3.12.4.3 text . . . . .	39
3.13 SubtitleReader Class Reference . . . . .	39
3.13.1 Detailed Description . . . . .	39
3.13.2 Member Function Documentation . . . . .	39
3.13.2.1 ReadSubtitles() . . . . .	39
3.14 TestScript Class Reference . . . . .	40
3.14.1 Detailed Description . . . . .	41
3.14.2 Member Function Documentation . . . . .	41
3.14.2.1 RunTests() . . . . .	41
3.14.2.2 TestAdditionalMedia360Video() . . . . .	42
3.14.2.3 TestHotSpot_OnLookAt_DuringNotLookingAt() . . . . .	43
3.14.2.4 TestLoadStartStop360Video() . . . . .	44
3.14.2.5 TestNavigation() . . . . .	45
3.14.3 Member Data Documentation . . . . .	46
3.14.3.1 controller . . . . .	46
3.15 TextController Class Reference . . . . .	46
3.15.1 Detailed Description . . . . .	47
3.15.2 Member Function Documentation . . . . .	47
3.15.2.1 Load() . . . . .	47
3.15.2.2 PlayMedia() . . . . .	47
3.15.2.3 StopMedia() . . . . .	47
3.16 Utils Class Reference . . . . .	48
3.16.1 Detailed Description . . . . .	48
3.16.2 Member Function Documentation . . . . .	48
3.16.2.1 PolarToCartesian() . . . . .	48
3.17 Video360Controller Class Reference . . . . .	49
3.17.1 Detailed Description . . . . .	50
3.17.2 Member Function Documentation . . . . .	50
3.17.2.1 AbortOtherMedia() . . . . .	50
3.17.2.2 AddMedia() . . . . .	51
3.17.2.3 AddSubtitle() . . . . .	52

---

3.17.2.4 EndVideo360() . . . . .	53
3.17.2.5 LoadVideo360() . . . . .	53
3.17.2.6 StartOtherMedia() . . . . .	54
3.17.2.7 StartVideo360() . . . . .	54
3.17.2.8 StopVideo360() . . . . .	55
3.18 VideoController Class Reference . . . . .	55
3.18.1 Detailed Description . . . . .	56
3.18.2 Member Function Documentation . . . . .	56
3.18.2.1 Ended() . . . . .	56
3.18.2.2 Load() . . . . .	57
3.18.2.3 LoadVideo() . . . . .	57
3.18.2.4 PlayMedia() . . . . .	58
3.18.2.5 StopMedia() . . . . .	58
Index . . . . .	59

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

MonoBehaviour	
ControllerChecker	8
LookHotspotScript	12
MediaControllerAbstract	13
AudioController	5
HotspotController	9
ImageController	11
MirrorController	22
PreviewController	30
TextController	46
VideoController	55
MultimediaControllerScript	24
RaySelect	33
RotateScript	36
TestScript	40
Video360Controller	49
SubtitleFragment	37
SubtitleReader	39
Utils	48



# Chapter 2

## Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">AudioController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits	<a href="#">MediaControllerAbstract</a> .
	This class is the script that controls the audio media object . . . . .
	<a href="#">5</a>
<a href="#">ControllerChecker</a>	
Author:	Paulo Renato Conceição Mendes.
Responsible for checking wether the immersives controllers are connected . . . . .	<a href="#">8</a>
<a href="#">HotspotController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits	<a href="#">MediaControllerAbstract</a> .
	This class is the script that controls the hotspot media object . . . . .
	<a href="#">9</a>
<a href="#">ImageController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits	<a href="#">MediaControllerAbstract</a> .
	This class is the script that controls the image media object . . . . .
	<a href="#">11</a>
<a href="#">LookHotspotScript</a>	
Author:	Paulo Renato Conceição Mendes.
Responsible for controlling when the camera is looking or not at the defined hotspots . . . . .	<a href="#">12</a>
<a href="#">MediaControllerAbstract</a>	
Author:	Paulo Renato Conceição Mendes.
Basic controller for all media objects that are added to 360 videos . . . . .	<a href="#">13</a>
<a href="#">MirrorController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits	<a href="#">MediaControllerAbstract</a> .
	This class is the script that controls the mirror media object . . . . .
	<a href="#">22</a>
<a href="#">MultimediaControllerScript</a>	
Author:	Paulo Renato Conceição Mendes.
Main controller of the project. It reads presentations and controls them . . . . .	<a href="#">24</a>
<a href="#">PreviewController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits	<a href="#">MediaControllerAbstract</a> .
	This class is the script that controls the preview media object . . . . .
	<a href="#">30</a>
<a href="#">RaySelect</a>	
Author:	Paulo Renato Conceição Mendes.
Responsible for casting a ray/laser from the controller and checking if a selection is performed . . . . .	<a href="#">33</a>
<a href="#">RotateScript</a>	
Author:	Paulo Renato Conceição Mendes.
This script is responsible for allowing the movement of the camera thorugh the mouse (when a HMD is not available) . . . . .	<a href="#">36</a>

<a href="#">SubtitleFragment</a>	
Author:	Paulo Renato Conceição Mendes.
Defines a subtitle fragment, which is a block of text that appears at a certain time and disappears	<a href="#">37</a>
<a href="#">SubtitleReader</a>	
Author:	Paulo Renato Conceição Mendes.
This class reads a srt file and converts it to Subtitles Fragments	<a href="#">39</a>
<a href="#">TestScript</a>	
Author:	Paulo Renato Conceição Mendes.
This class performs the unit tests on the player	<a href="#">40</a>
<a href="#">TextController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits <a href="#">MediaControllerAbstract</a> . This class is the script that controls the text media object	<a href="#">46</a>
<a href="#">Utils</a>	
Author:	Paulo Renato Conceição Mendes.
This class has static methods that can be used in situations related to different contexts	<a href="#">48</a>
<a href="#">Video360Controller</a>	
Author:	Paulo Renato Conceição Mendes.
This class is the script that controls the video360	<a href="#">49</a>
<a href="#">VideoController</a>	
Author:	Paulo Renato Conceição Mendes.
Inherits <a href="#">MediaControllerAbstract</a> . This class is the script that controls the video media object	<a href="#">55</a>

# Chapter 3

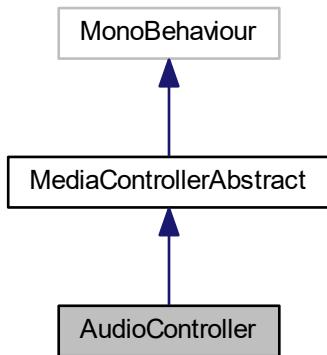
## Class Documentation

### 3.1 AudioController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the audio media object.

Inheritance diagram for AudioController:



### Public Member Functions

- `override void Load ()`  
*Inherited from [MediaControllerAbstract](#). Configures the audioContainer to load the audio with its configurations.*
- `override void PlayMedia ()`  
*Inherited from [MediaControllerAbstract](#). Playing for the audio is to play the audioContainer.*
- `override void StopMedia ()`  
*Inherited from [MediaControllerAbstract](#). Stopping for the audio is to stop the audioContainer.*

## Private Member Functions

- IEnumerator [LoadAudio \(\)](#)

*Coroutine that loads the audio from the file path.*

## Private Attributes

- AudioSource [audioContainer](#)  
*it is an audio source used to play the audio*

## Additional Inherited Members

### 3.1.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the audio media object.

### 3.1.2 Member Function Documentation

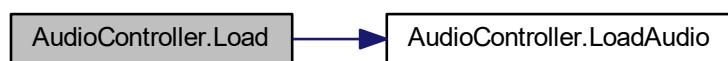
#### 3.1.2.1 Load()

```
override void AudioController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Configures the audioContainer to load the audio with its configurations.

Implements [MediaControllerAbstract](#).

Here is the call graph for this function:



### 3.1.2.2 LoadAudio()

```
IEnumerator AudioController.LoadAudio () [private]
```

Coroutine that loads the audio from the file path.

#### Returns

IEnumerator to coroutine

Here is the caller graph for this function:



### 3.1.2.3 PlayMedia()

```
override void AudioController.PlayMedia () [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the audio is to play the audioContainer.

Implements [MediaControllerAbstract](#).

### 3.1.2.4 StopMedia()

```
override void AudioController.StopMedia () [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the audio is to stop the audioContainer.

Implements [MediaControllerAbstract](#).

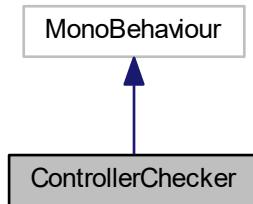
The documentation for this class was generated from the following file:

- Assets/Scripts/MediaControllers/AudioController.cs

## 3.2 ControllerChecker Class Reference

Author: Paulo Renato Conceição Mendes.  
 Responsible for checking whether the immersives controllers are connected.

Inheritance diagram for ControllerChecker:



### Public Member Functions

- void [Change](#) (SteamVR\_Behaviour\_Pose pose, SteamVR\_Input\_Sources sources, bool connected)  
*This function is called when the state of connection of a controller changes. If the controller is disconnected, the Ray Select and line renderers are deactivated. The reverse occurs otherwise.*

#### 3.2.1 Detailed Description

Author: Paulo Renato Conceição Mendes.  
 Responsible for checking whether the immersives controllers are connected.

#### 3.2.2 Member Function Documentation

##### 3.2.2.1 Change()

```
void ControllerChecker.Change (
    SteamVR_Behaviour_Pose pose,
    SteamVR_Input_Sources sources,
    bool connected )
```

This function is called when the state of connection of a controller changes. If the controller is disconnected, the Ray Select and line renderers are deactivated. The reverse occurs otherwise.

##### Parameters

<i>pose</i>	The controller
<i>sources</i>	The input sources of the controller
<i>connected</i>	If the controller was connected or disconnected

The documentation for this class was generated from the following file:

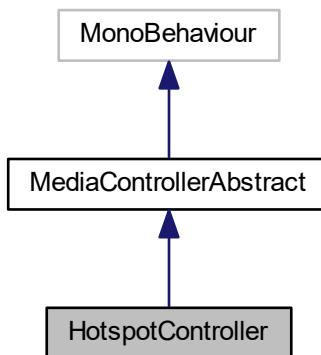
- Assets/Scripts/ControllerChecker.cs

### 3.3 HotspotController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the hotspot media object.

Inheritance diagram for HotspotController:



#### Public Member Functions

- override void [Load \(\)](#)  
*Inherited from [MediaControllerAbstract](#). This media object doesn't load anything in specific besides what is loaded in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.*
- override void [PlayMedia \(\)](#)  
*Inherited from [MediaControllerAbstract](#). This media object doesn't do anything additionaly to play besides what is done in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.*
- override void [StopMedia \(\)](#)  
*Inherited from [MediaControllerAbstract](#). This media object doesn't do anything additionaly to stop besides what is done in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.*

#### Additional Inherited Members

##### 3.3.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the hotspot media object.

### 3.3.2 Member Function Documentation

#### 3.3.2.1 Load()

```
override void HotspotController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). This media object doesnt load anything in specific besides what is loaded in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.

Implements [MediaControllerAbstract](#).

#### 3.3.2.2 PlayMedia()

```
override void HotspotController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). This media object doesnt do anything additionaly to play besides what is done in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.

Implements [MediaControllerAbstract](#).

#### 3.3.2.3 StopMedia()

```
override void HotspotController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). This media object doesnt do anything additionaly to stop besides what is done in the [MediaControllerAbstract](#) Class . It is empty because the method has to be implemented.

Implements [MediaControllerAbstract](#).

The documentation for this class was generated from the following file:

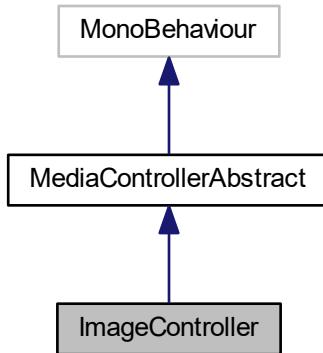
- Assets/Scripts/MediaControllers/HotspotController.cs

## 3.4 ImageController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the image media object.

Inheritance diagram for ImageController:



### Public Member Functions

- `override void Load ()`  
*Inherited from [MediaControllerAbstract](#). Loads the image creating a material from a texture and assigning that material to the object mesh renderer.*
- `override void PlayMedia ()`  
*Inherited from [MediaControllerAbstract](#). Playing for the image consists in enabling its MeshRenderer.*
- `override void StopMedia ()`  
*Inherited from [MediaControllerAbstract](#). Stopping for the image consists in disabling its MeshRenderer.*

### Additional Inherited Members

#### 3.4.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the image media object.

#### 3.4.2 Member Function Documentation

### 3.4.2.1 Load()

```
override void ImageController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Loads the image creating a material from a texture and assigning that material to the object mesh renderer.

Implements [MediaControllerAbstract](#).

### 3.4.2.2 PlayMedia()

```
override void ImageController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the image consists in enabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

### 3.4.2.3 StopMedia()

```
override void ImageController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the image consists in disabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

The documentation for this class was generated from the following file:

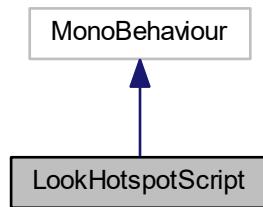
- Assets/Scripts/MediaControllers/ImageController.cs

## 3.5 LookHotspotScript Class Reference

Author: Paulo Renato Conceição Mendes.

Responsible for controlling when the camera is looking or not at the defined hotspots.

Inheritance diagram for LookHotspotScript:



## Public Attributes

- `GameObject[] hotspot_objects`

*list of hotspot objects*

## Private Member Functions

- `void Update ()`

*Called once per frame, inherited from MonoBehaviour.  
At each time it checks for each hotspot wether it is being seen or not.*

### 3.5.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Responsible for controlling when the camera is looking or not at the defined hotspots.

### 3.5.2 Member Function Documentation

#### 3.5.2.1 Update()

```
void LookHotspotScript.Update ( ) [private]
```

Called once per frame, inherited from MonoBehaviour.  
At each time it checks for each hotspot wether it is being seen or not.

The documentation for this class was generated from the following file:

- `Assets/Scripts/LookHotspotScript.cs`

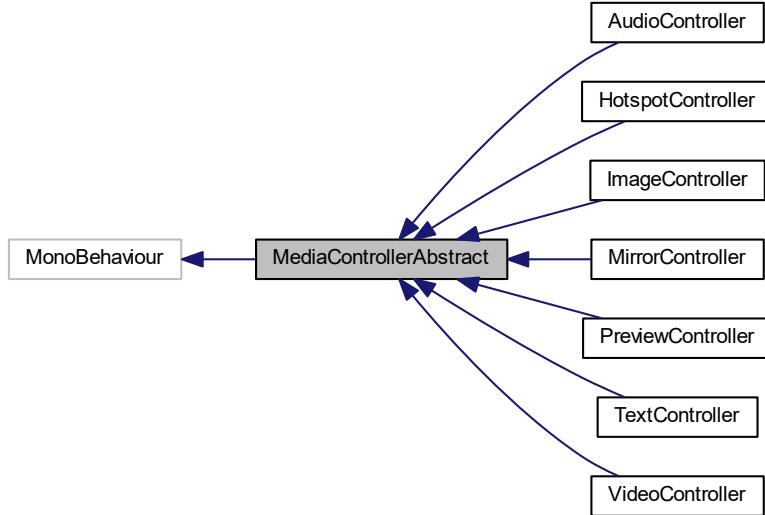
## 3.6 MediaControllerAbstract Class Reference

Author: Paulo Renato Conceição Mendes.

Basic controller for all media objects that are added to 360 videos.

---

Inheritance diagram for MediaControllerAbstract:



## Public Member Functions

- void [InvokePlayStop \(\)](#)  
*Calls the play and invokes the stop of the media object (given its duration).*
- void [OnSelectMedia \(\)](#)  
*This method is called when this media is selected.*
- void [PrepareAnchors \(\)](#)  
*Prepare the time that the media object will start given the start time.*
- virtual void [Configure \(string id, GameObject father, float start\\_time, float duration, string file\\_path, float r, float theta, float phi, float volume, bool loop, bool follow\\_camera, string text, string on\\_select\\_name, string on\\_focus\\_name, string during\\_out\\_of\\_focus\\_name, float clipBegin, float clipEnd\)](#)  
*Sets the initial configuration of the media object such as: its position and rotation, whether it follows the camera, etc.*
- void [AbortMedia \(\)](#)  
*Abort the media and the anchors that make it play, if any.*
- abstract void [Load \(\)](#)  
*Abstract method that should be implemented by any media object type. This method should configure the initial state of the components that are specific to the media object*
- abstract void [StopMedia \(\)](#)  
*Abstract method that should be implemented by any media object type. This method should configure how the components specific of the media object behave to make it stop*
- abstract void [PlayMedia \(\)](#)  
*Abstract method that should be implemented by any media object type. This method should configure how the components specific of the media object behave to make it play*

## Public Attributes

- float [start\\_time](#)  
*start time of the media object in seconds*

- float `duration`  
*duration time of the media object in seconds*
- float `volume`  
*volume of the media object*
- string `file_path`  
*file path of the media object*
- string `text`  
*text of the media object, if any*
- bool `loop`  
*true if the media object executes in loop*
- bool `follow_camera`  
*true if the media object follows the camera (fixed to the user)*
- GameObject `mainCamera`  
*main camera of the project*
- GameObject `presentation360`  
*presentation360 game object of the scene*
- string `id`  
*id of the current media object*
- string `on_select_name`  
*id the of media object that will be played when the current is selected*
- string `on_focus_name`  
*id the of media object that will be played when the current is on focus*
- string `during_out_of_focus_name`  
*id the of media object that will be played when the current is out of focus*
- GameObject `on_select_object`  
*game object the of media object that will be played when the current is selected*
- GameObject `on_focus_object`  
*game object the of media object that will be played when the current is on focus*
- GameObject `during_out_of_focus_object`  
*game object the of media object that will be played when the current is out of focus*
- GameObject `src_hotspot_object`  
*game object of the hotspot to which the current media object is pointed (in case of mirror)*
- GameObject `father`  
*game object that is father of the current game object*
- float `clipBegin`  
*initial time of the segment of the media that will be played*
- float `clipEnd`  
*final time of the segment of the media that will be played*

## Properties

- bool `IsPlaying` [get]  
*Returns wether the media object is playing.*

## Private Member Functions

- void `SuperPlay ()`  
*Activates the colliders and sets that the media object is playing.*
- void `SuperStop ()`  
*Deactivates the colliders and sets that the media object is not playing.*

## Private Attributes

- float `theta`  
*vertical angle of the media object in degrees*
- float `phi`  
*horizontal angle of the media object in degrees*
- float `r`  
*media object distance from the center*
- Vector3 `origin`  
*vector that defines the origin from which the rotations are calculated*
- bool `isPlaying`  
*true if the mediaObject is playing*
- Vector3 `start_pos`  
*initial position of the media object*

### 3.6.1 Detailed Description

Author: Paulo Renato Conceição Mendes.  
 Basic controller for all media objects that are added to 360 videos.

### 3.6.2 Member Function Documentation

#### 3.6.2.1 AbortMedia()

```
void MediaControllerAbstract.AbortMedia ( )
```

Abort the media and the anchors that make it play, if any.

Here is the call graph for this function:



### 3.6.2.2 Configure()

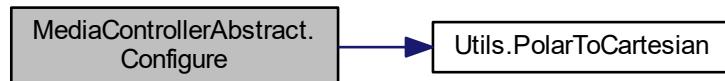
```
virtual void MediaControllerAbstract.Configure (
    string id,
    GameObject father,
    float start_time,
    float duration,
    string file_path,
    float r,
    float theta,
    float phi,
    float volume,
    bool loop,
    bool follow_camera,
    string text,
    string on_select_name,
    string on_focus_name,
    string during_out_of_focus_name,
    float clipBegin,
    float clipEnd ) [virtual]
```

Sets the initial configuration of the media object such as: its position and rotation, whether it follows the camera, etc.

#### Parameters

<i>id</i>	id of the media object
<i>father</i>	father of the media object
<i>start_time</i>	start time of the media object in seconds
<i>duration</i>	duration of the media object in seconds
<i>file_path</i>	source file url of the media object
<i>r</i>	media object distance from the center
<i>theta</i>	vertical angle of the media object in degrees
<i>phi</i>	horizontal angle of the media object in degrees
<i>volume</i>	volume of the media object
<i>loop</i>	whether the media object executes in loop
<i>follow_camera</i>	whether the media object follows the camera (is fixed to the user)
<i>text</i>	text of the media object
<i>on_select_name</i>	media object that will be played when the current is selected
<i>on_focus_name</i>	media object that will be played when the current is on focus
<i>during_out_of_focus_name</i>	media object that will be played when the current is out of focus
<i>clipBegin</i>	initial time of the segment of the media that will be played
<i>clipEnd</i>	final time of the segment of the media that will be played

Here is the call graph for this function:



### 3.6.2.3 InvokePlayStop()

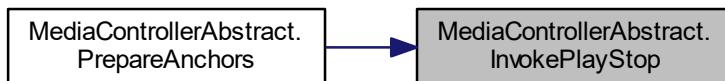
```
void MediaControllerAbstract.InvokePlayStop ( )
```

Calls the play and invokes the stop of the media object (given its duration).

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.6.2.4 Load()

```
abstract void MediaControllerAbstract.Load ( ) [pure virtual]
```

Abstract method that should be implemented by any media object type. This method should configure the initial state of the components that are specific to the media object

Implemented in [VideoController](#), [TextController](#), [PreviewController](#), [MirrorController](#), [ImageController](#), [HotspotController](#), and [AudioController](#).

Here is the caller graph for this function:



### 3.6.2.5 OnSelectMedia()

```
void MediaControllerAbstract.OnSelectMedia ( )
```

This method is called when this media is selected.

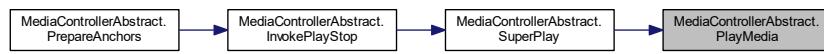
### 3.6.2.6 PlayMedia()

```
abstract void MediaControllerAbstract.PlayMedia ( ) [pure virtual]
```

Abstract method that should be implemented by any media object type. This method should configure how the components specific of the media object behave to make it play

Implemented in [VideoController](#), [TextController](#), [PreviewController](#), [MirrorController](#), [ImageController](#), [HotspotController](#), and [AudioController](#).

Here is the caller graph for this function:

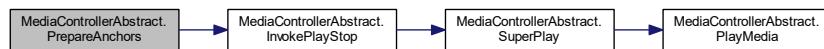


### 3.6.2.7 PrepareAnchors()

```
void MediaControllerAbstract.PrepareAnchors ( )
```

Prepare the time that the media object will start given the start time.

Here is the call graph for this function:



### 3.6.2.8 StopMedia()

```
abstract void MediaControllerAbstract.StopMedia ( ) [pure virtual]
```

Abstract method that should be implemented by any media object type. This method should configure how the components specific of the media object behave to make it stop

Implemented in [VideoController](#), [TextController](#), [PreviewController](#), [MirrorController](#), [ImageController](#), [HotspotController](#), and [AudioController](#).

Here is the caller graph for this function:

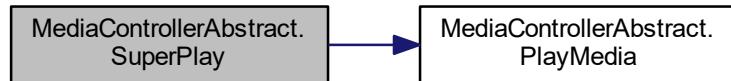


### 3.6.2.9 SuperPlay()

```
void MediaControllerAbstract.SuperPlay ( ) [private]
```

Activates the colliders and sets that the media object is playing.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.6.2.10 SuperStop()

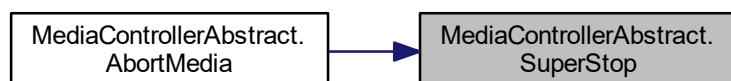
```
void MediaControllerAbstract.SuperStop ( ) [private]
```

Deactivates the colliders and sets that the media object is not playing.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.6.3 Property Documentation

#### 3.6.3.1 IsPlaying

```
bool MediaControllerAbstract.isPlaying [get]
```

Returns whether the media object is playing.

The documentation for this class was generated from the following file:

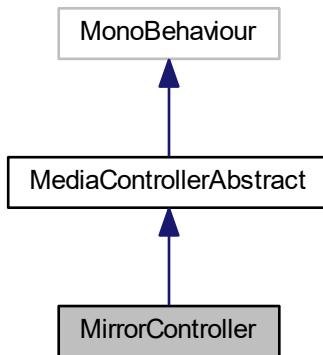
- Assets/Scripts/MediaControllers/MediaControllerAbstract.cs

## 3.7 MirrorController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the mirror media object.

Inheritance diagram for MirrorController:



### Public Member Functions

- override void [Load \(\)](#)  
*Inherited from [MediaControllerAbstract](#). This media object doesn't load anything in specific besides what is loaded in the [MediaControllerAbstract](#) Class. It is empty because the method has to be implemented*
- override void [PlayMedia \(\)](#)  
*Inherited from [MediaControllerAbstract](#). Playing for the mirror consists in enabling its MeshRenderer.*
- override void [StopMedia \(\)](#)  
*Inherited from [MediaControllerAbstract](#). Stopping for the mirror consists in disabling its MeshRenderer.*

## Private Member Functions

- void [Update \(\)](#)

*Called once per frame, inherited from MonoBehaviour. Makes the mirror always transmits the content of hotspot. It internally uses a camera that is always "filming" the hotspot's position.*

## Additional Inherited Members

### 3.7.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the mirror media object.

### 3.7.2 Member Function Documentation

#### 3.7.2.1 Load()

```
override void MirrorController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). This media object doesn't load anything in specific besides what is loaded in the [MediaControllerAbstract](#) Class. It is empty because the method has to be implemented

Implements [MediaControllerAbstract](#).

#### 3.7.2.2 PlayMedia()

```
override void MirrorController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the mirror consists in enabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

#### 3.7.2.3 StopMedia()

```
override void MirrorController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the mirror consists in disabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

### 3.7.2.4 Update()

```
void MirrorController.Update ( ) [private]
```

Called once per frame, inherited from MonoBehaviour. Makes the mirror always transmits the content of hotspot. It internally uses a camera that is always "filming" the hotspot's position.

The documentation for this class was generated from the following file:

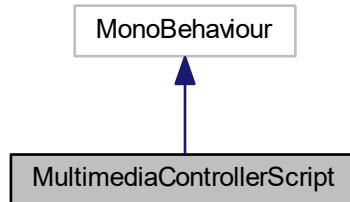
- Assets/Scripts/MediaControllers/MirrorController.cs

## 3.8 MultimediaControllerScript Class Reference

Author: Paulo Renato Conceição Mendes.

Main controller of the project. It reads presentations and controls them.

Inheritance diagram for MultimediaControllerScript:



### Public Member Functions

- delegate void [MyHandler \(\)](#)  
*handler used to handle events*
- GameObject [AddVideo360 \(string src, float volume=1f\)](#)  
*Loads a 360 video from a url.*
- void [StartPresentation \(\)](#)  
*Starts the presentation.*
- void [StopPresentation \(\)](#)  
*Stops the presentation.*
- void [LoadXmlFile \(string file\\_path\)](#)  
*Loads all the 360 videos and their media objects from a url to an xml file.*
- void [AddAdditionalMedia \(XmlNode scene360node, GameObject video360, Dictionary< string, GameObject > scene\\_objects, Dictionary< string, XmlNode > styles\)](#)  
*Add additional media objects to a 360 video.*

## Public Attributes

- bool `testMode`  
*controls if player is in test Mode. Only prefabs are loaded to be used in tests.*
- string `presentation`  
*url of the presentation*
- GameObject `video360Prefab`  
*prefab of 360 videos*
- GameObject `previewSphere`  
*prefab of preview sphere*
- GameObject `previewPlain`  
*prefab of preview plain*
- GameObject `hotspotPrefab`  
*prefab of hotspot*
- GameObject `mirrorPrefab`  
*prefab of mirror*
- GameObject `imagePrefab`  
*prefab of image*
- GameObject `audioPrefab`  
*prefab of audio*
- GameObject `videoPrefab`  
*prefab of video*
- GameObject `textPrefab`  
*prefab of text*
- GameObject `startPresentation`  
*game object the when selected starts the presentation*

## Events

- MyHandler Entry  
*event that calls initial 360 video*
- MyHandler End  
*event that stops videos 360 when the application ends*

## Private Member Functions

- void `Start ()`  
*Called when the gameobject starts, inherited from MonoBehaviour.*
- void `SetAsInitial (GameObject video)`  
*Sets a 360 video as initial.*

## Private Attributes

- GameObject `initialScene`  
*stores the initial 360 video*
- int `globalid = 0`  
*controls the ids that will be assigned to media objects to which the user haven't assigned any id.*

### 3.8.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Main controller of the project. It reads presentations and controls them.

### 3.8.2 Member Function Documentation

#### 3.8.2.1 AddAdditionalMedia()

```
void MultimediaControllerScript.AddAdditionalMedia (
    XmlNode scene360node,
    GameObject video360,
    Dictionary< string, GameObject > scene_objects,
    Dictionary< string, XmlNode > styles )
```

Add aditional media objects to a 360 video.

##### Parameters

<code>scene360node</code>	Current scene 360 being read
<code>video360</code>	360 video to which the media objects will be added
<code>scene_objects</code>	List of all scene 360 objects
<code>styles</code>	List of styles defined in the Head. Those styles may be reused by media objects

Here is the caller graph for this function:



#### 3.8.2.2 AddVideo360()

```
GameObject MultimediaControllerScript.AddVideo360 (
    string src,
    float volume = 1f )
```

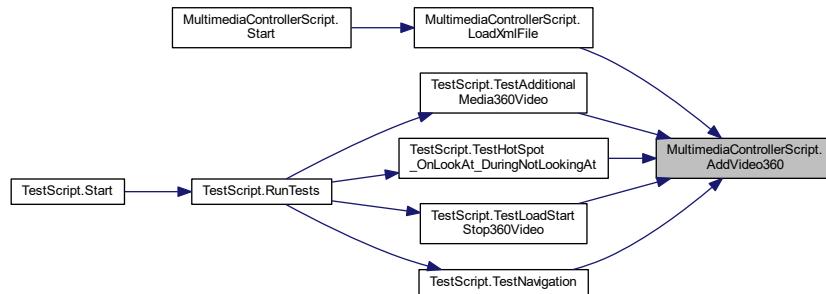
Loads a 360 video from a url.

##### Parameters

<code>src</code>	url of the 360 video
<code>volume</code>	volume that the 360 video will be played

## Returns

Here is the caller graph for this function:



### 3.8.2.3 LoadXmlFile()

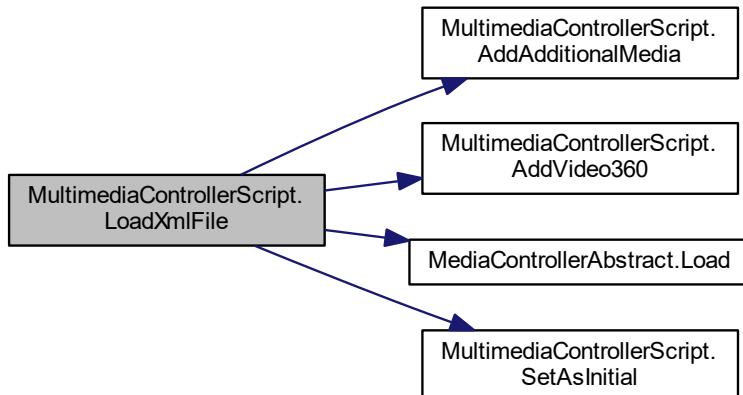
```
void MultimediaControllerScript.LoadXmlFile (
    string file_path )
```

Loads all the 360 videos and their media objects from a url to an xml file.

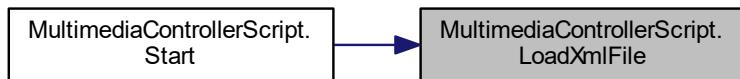
#### Parameters

<i>file_path</i>	url of the xml file
------------------	---------------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.8.2.4 SetAsInitial()

```
void MultimediaControllerScript.SetAsInitial (\n    GameObject video ) [private]
```

Sets a 360 video as initial.

#### Parameters

video	video that will be assigned as initial
-------	--

Here is the caller graph for this function:

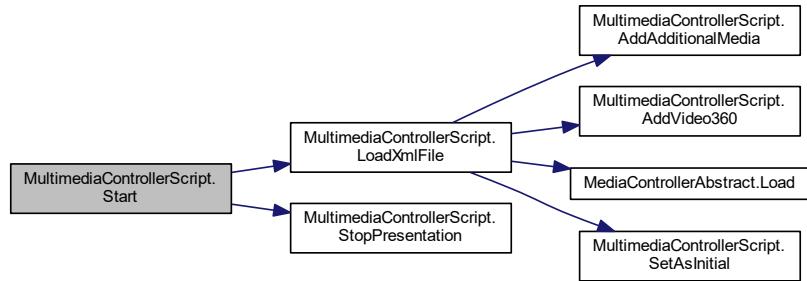


### 3.8.2.5 Start()

```
void MultimediaControllerScript.Start ( ) [private]
```

Called when the gameobject starts, inherited from MonoBehaviour.

Here is the call graph for this function:



### 3.8.2.6 StartPresentation()

```
void MultimediaControllerScript.StartPresentation( )
```

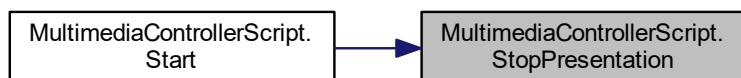
Starts the presentation.

### 3.8.2.7 StopPresentation()

```
void MultimediaControllerScript.StopPresentation( )
```

Stops the presentation.

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

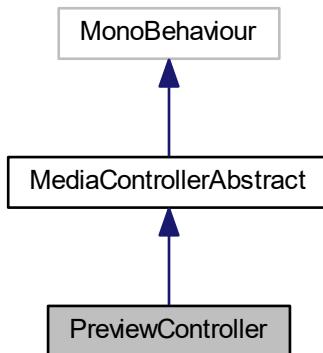
- Assets/Scripts/MultimediaControllerScript.cs

## 3.9 PreviewController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the preview media object.

Inheritance diagram for PreviewController:



### Public Member Functions

- `override void Load ()`  
*Inherited from [MediaControllerAbstract](#). It loads the video on the video player from the url, prepare it and seek it to the time of startPreview*
- `override void PlayMedia ()`  
*Inherited from [MediaControllerAbstract](#). Playing for the preview consists in starting the loop of the preview and enabling its MeshRenderer.*
- `override void StopMedia ()`  
*Inherited from [MediaControllerAbstract](#). Stopping for the preview consists in stopping the video player and disabling its MeshRenderer.*

### Private Member Functions

- `void PlayVideoPreview ()`  
*Starts the loop of playing the segment of the video.*
- `void PauseVideoPreview ()`  
*Ends the loop of playing the segment of the video and call the start again.*

### Private Attributes

- `float startpreview`  
*time of the clipBegin. The time that the segment that is played in the preview starts*
- `float stoppreview`  
*time of the clipEnd. The time that the segment that is played in the preview ends*

## Additional Inherited Members

### 3.9.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the preview media object.

### 3.9.2 Member Function Documentation

#### 3.9.2.1 Load()

```
override void PreviewController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). It loads the video on the video player from the url, prepare it and seek it to the time of startPreview

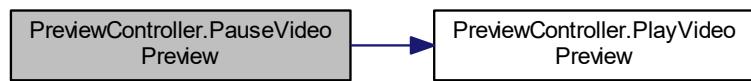
Implements [MediaControllerAbstract](#).

#### 3.9.2.2 PauseVideoPreview()

```
void PreviewController.PauseVideoPreview ( ) [private]
```

Ends the loop of playing the segment of the video and call the start again.

Here is the call graph for this function:



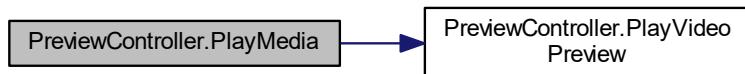
### 3.9.2.3 PlayMedia()

```
override void PreviewController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the preview consists in starting the loop of the preview and enabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

Here is the call graph for this function:

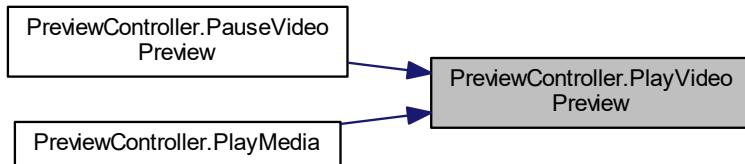


### 3.9.2.4 PlayVideoPreview()

```
void PreviewController.PlayVideoPreview ( ) [private]
```

Starts the loop of playing the segment of the video.

Here is the caller graph for this function:



### 3.9.2.5 StopMedia()

```
override void PreviewController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the preview consists in stopping the video player and disabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

The documentation for this class was generated from the following file:

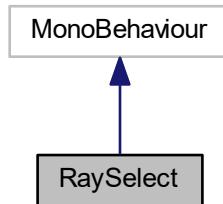
- Assets/Scripts/MediaControllers/PreviewController.cs

## 3.10 RaySelect Class Reference

Author: Paulo Renato Conceição Mendes.

Responsible for casting a ray/laser from the controller and checking if a selection is performed.

Inheritance diagram for RaySelect:



### Public Member Functions

- void [TriggerDown](#) (SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)  
*If the trigger is down, verifies if something is selected.*

### Public Attributes

- float [laserWidth](#) = 0.1f  
*width of the laser*
- float [laserMaxLength](#) = 10f  
*maximum size of the laser*
- SteamVR\_Action\_Boolean [SelectVideo](#)  
*a reference to the action*
- SteamVR\_Input\_Sources [handType](#)  
*a reference to the hand*

### Private Member Functions

- void [Start](#) ()  
*Start is called before the first frame update.*
- void [Update](#) ()  
*Called once per frame, inherited from MonoBehaviour.  
At each time, casts a ray with the line renderer.*
- void [ShootLaserFromTargetPosition](#) (Vector3 targetPosition, Vector3 direction, float length)  
*Shoots a laser from a position with a direction.*

## Private Attributes

- LineRenderer [laserLineRenderer](#)  
*line renderer that draws ray*

### 3.10.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Responsible for casting a ray/laser from the controller and checking if a selection is performed.

### 3.10.2 Member Function Documentation

#### 3.10.2.1 ShootLaserFromTargetPosition()

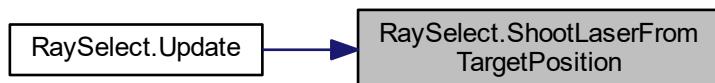
```
void RaySelect.ShootLaserFromTargetPosition (
    Vector3 targetPosition,
    Vector3 direction,
    float length ) [private]
```

Shoots a laser from a position with a direction.

##### Parameters

<i>targetPosition</i>	Initial position of the laser
<i>direction</i>	Direction of the laser
<i>length</i>	Length of the laser

Here is the caller graph for this function:

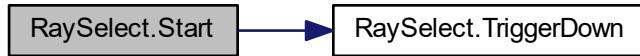


#### 3.10.2.2 Start()

```
void RaySelect.Start ( ) [private]
```

Start is called before the first frame update.

Called when the gameobject starts, inherited from MonoBehaviour. Here is the call graph for this function:



### 3.10.2.3 TriggerDown()

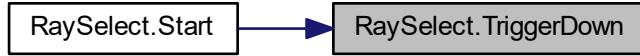
```
void RaySelect.TriggerDown (
    SteamVR_Action_Boolean fromAction,
    SteamVR_Input_Sources fromSource )
```

If the trigger is down, verifies if something is selected.

#### Parameters

<i>fromAction</i>	Action that called the method
<i>fromSource</i>	Controller that called the method

Here is the caller graph for this function:

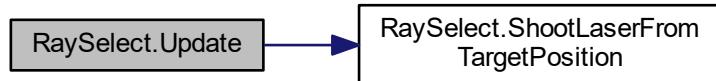


### 3.10.2.4 Update()

```
void RaySelect.Update ( ) [private]
```

Called once per frame, inherited from MonoBehaviour.  
At each time, casts a ray with the line renderer.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

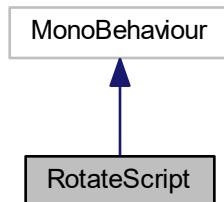
- Assets/Scripts/RaySelect.cs

## 3.11 RotateScript Class Reference

Author: Paulo Renato Conceição Mendes.

This script is responsible for allowing the movement of the camera through the mouse (when a HMD is not available).

Inheritance diagram for RotateScript:



### Public Attributes

- float `speed`  
*Speed of the rotation of the camera.*

### Private Member Functions

- void `Update ()`  
*Called once per frame, inherited from MonoBehaviour.*

### 3.11.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

This script is responsible for allowing the movement of the camera through the mouse (when a HMD is not available).

### 3.11.2 Member Function Documentation

#### 3.11.2.1 Update()

```
void RotateScript.Update ( ) [private]
```

Called once per frame, inherited from MonoBehaviour.

### 3.11.3 Member Data Documentation

#### 3.11.3.1 speed

```
float RotateScript.speed
```

Speed of the rotation of the camera.

The documentation for this class was generated from the following file:

- Assets/Scripts/RotateScript.cs

## 3.12 SubtitleFragment Class Reference

Author: Paulo Renato Conceição Mendes.

Defines a subtitle fragment, which is a block of text that appears at a certain time and disappears.

### Public Member Functions

- [SubtitleFragment \(float begin, float duration, string text\)](#)

*Constructor of the fragment.*

- [override string ToString \(\)](#)

*Readable information about the object.*

## Public Attributes

- float `begin`  
*Time at which the fragment starts in seconds.*
- float `duration`  
*Duration of the fragment in seconds.*
- string `text`  
*Text of the fragment.*

### 3.12.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Defines a subtitle fragment, which is a block of text that appears at a certain time and disappears.

### 3.12.2 Constructor & Destructor Documentation

#### 3.12.2.1 SubtitleFragment()

```
SubtitleFragment.SubtitleFragment (
    float begin,
    float duration,
    string text )
```

Constructor of the fragment.

##### Parameters

<code>begin</code>	time of begin in seconds
<code>duration</code>	duration in seconds
<code>text</code>	text of the fragment

### 3.12.3 Member Function Documentation

#### 3.12.3.1 ToString()

```
override string SubtitleFragment.ToString ( )
```

Readable information about the object.

##### Returns

string with information about the object

### 3.12.4 Member Data Documentation

#### 3.12.4.1 begin

```
float SubtitleFragment.begin
```

Time at which the fragment starts in seconds.

#### 3.12.4.2 duration

```
float SubtitleFragment.duration
```

Duration of the fragment in seconds.

#### 3.12.4.3 text

```
string SubtitleFragment.text
```

Text of the fragment.

The documentation for this class was generated from the following file:

- Assets/Scripts/Subtitle.cs

## 3.13 SubtitleReader Class Reference

Author: Paulo Renato Conceição Mendes.

This class reads a srt file and converts it to Subtitles Fragments.

### Static Public Member Functions

- static [SubtitleFragment\[\] ReadSubtitles](#) (string file\_path)  
*Reads a srt file and returns it as a list of subtitles fragments.*

### 3.13.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

This class reads a srt file and converts it to Subtitles Fragments.

### 3.13.2 Member Function Documentation

#### 3.13.2.1 ReadSubtitles()

```
static SubtitleFragment [ ] SubtitleReader.ReadSubtitles (
    string file_path ) [static]
```

Reads a srt file and returns it as a list of subtitles fragments.

**Parameters**

<code>file_path</code>	path of the srt file
------------------------	----------------------

**Returns**

list of subtitles fragments

Here is the caller graph for this function:



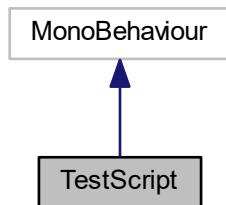
The documentation for this class was generated from the following file:

- Assets/Scripts/Subtitle.cs

## 3.14 TestScript Class Reference

Author: Paulo Renato Conceição Mendes.  
This class performs the unit tests on the player.

Inheritance diagram for TestScript:



## Public Member Functions

- IEnumerator [RunTests \(\)](#)  
*This method runs the test modules*
- IEnumerator [TestLoadStartStop360Video \(string video360\\_path\)](#)  
*Test if video is being loaded, started and stopped correctly*
- IEnumerator [TestAdditionalMedia360Video \(string video360\\_path, GameObject mediaPrefab, string mediaPath, float begin, float duration\)](#)  
*Test additional media, especially if it is starting and stopping at correct timing*
- IEnumerator [TestHotSpot\\_OnLookAt\\_DuringNotLookingAt \(string video360\\_path, GameObject mediaPrefab, string media1\\_path, float duration, string media2\\_path\)](#)  
*Test if hotspot events are working: onLookAt and duringNotLookingAt*
- IEnumerator [TestNavigation \(string video360\\_1\\_path, string video360\\_2\\_path, GameObject previewPrefab, float duration\)](#)  
*Test if navigation is working: we should move from one video to another by selecting a preview and coming back to the first video*

## Private Member Functions

- void [Start \(\)](#)  
*Start is called before the first frame update. It starts the tests if the player is on test mode.*

## Private Attributes

- [MultimediaControllerScript controller](#)  
*The multimedia controller script that contains the prefabs of media objects to be tested.*
- int [waitTime = 1](#)

### 3.14.1 Detailed Description

Author: Paulo Renato Conceição Mendes.  
This class performs the unit tests on the player.

### 3.14.2 Member Function Documentation

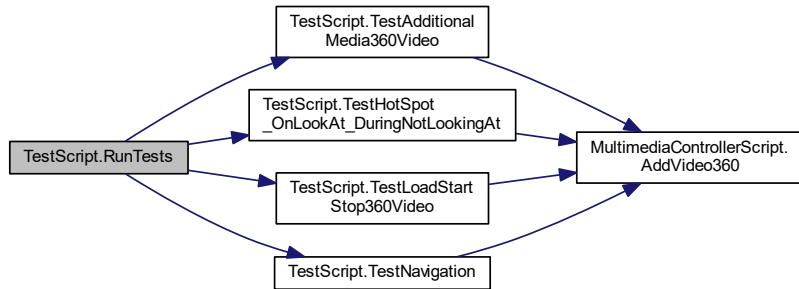
#### 3.14.2.1 RunTests()

```
IEnumerator TestScript.RunTests ( )
```

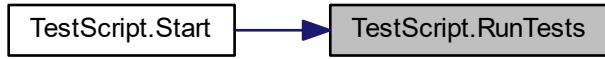
This method runs the test modules

**Returns**

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.2.2 TestAdditionalMedia360Video()

```
IEnumerator TestScript.TestAdditionalMedia360Video (
    string video360_path,
    GameObject mediaPrefab,
    string media_path,
    float begin,
    float duration )
```

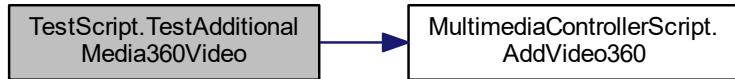
Test additional media, especially if it is starting and stopping at correct timing

**Parameters**

<i>video360_path</i>	Path to 360 video
<i>mediaPrefab</i>	Prefab of the media added to 360 video
<i>media_path</i>	Path to the media
<i>begin</i>	Begin time of the media
<i>duration</i>	Duration of the media

**Returns**

Here is the call graph for this function:



Here is the caller graph for this function:

**3.14.2.3 TestHotSpot\_OnLookAt\_DuringNotLookingAt()**

```
IEnumerator TestScript.TestHotSpot_OnLookAt_DuringNotLookingAt (
    string video360_path,
    GameObject mediaPrefab,
    string media1_path,
    float duration,
    string media2_path )
```

Test if hotspot events are working: onLookAt and duringNotLookingAt

**Parameters**

<i>video360_path</i>	Path to the 360 video
<i>mediaPrefab</i>	Prefab of the media
<i>media1_path</i>	Path to media1
<i>duration</i>	duration of media1
<i>media2_path</i>	Path to media2

**Returns**

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.2.4 TestLoadStartStop360Video()

```
IEnumerator TestScript.TestLoadStartStop360Video ( string video360_path )
```

Test if video is being loaded, started and stopped correctly

#### Parameters

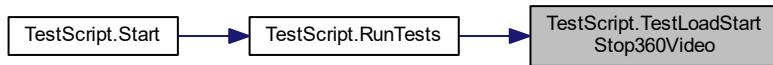
<i>url</i>	url of the video
<i>waitTime</i>	time waited to test

#### Returns

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.2.5 TestNavigation()

```
IEnumerator TestScript.TestNavigation (
    string video360_1_path,
    string video360_2_path,
    GameObject previewPrefab,
    float duration )
```

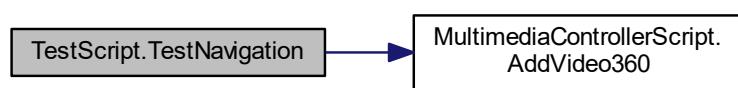
Test if navigation is working: we should move from one video to another by selecting a preview and coming back to the first video

#### Parameters

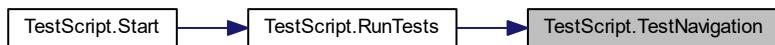
<i>video360_1_path</i>	Path to the first video
<i>video360_2_path</i>	Path to the second video
<i>previewPrefab</i>	Prefab of preview object
<i>duration</i>	duration of the previews

#### Returns

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.3 Member Data Documentation

#### 3.14.3.1 controller

```
MultimediaControllerScript TestScript.controller [private]
```

The multimedia controller script that contains the prefabs of media objects to be tested.

The documentation for this class was generated from the following file:

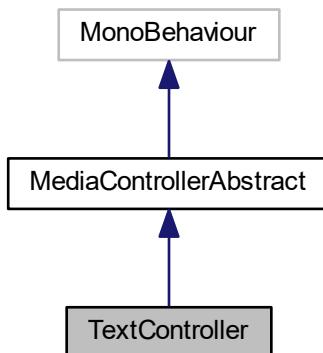
- Assets/Scripts/TestScript.cs

## 3.15 TextController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the text media object.

Inheritance diagram for TextController:



## Public Member Functions

- `override void Load ()`  
*Inherited from [MediaControllerAbstract](#). Loads the text to the TextMesh component.*
- `override void PlayMedia ()`  
*Inherited from [MediaControllerAbstract](#). Playing for the text consists in enabling its MeshRenderer.*
- `override void StopMedia ()`  
*Inherited from [MediaControllerAbstract](#). Stopping for the text consists in disabling its MeshRenderer.*

## Additional Inherited Members

### 3.15.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Implements [MediaControllerAbstract](#). This class is the script that controls the text media object.

### 3.15.2 Member Function Documentation

#### 3.15.2.1 Load()

```
override void TextController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Loads the text to the TextMesh component.

Implements [MediaControllerAbstract](#).

#### 3.15.2.2 PlayMedia()

```
override void TextController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the text consists in enabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

#### 3.15.2.3 StopMedia()

```
override void TextController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the text consists in disabling its MeshRenderer.

Implements [MediaControllerAbstract](#).

The documentation for this class was generated from the following file:

- Assets/Scripts/MediaControllers/TextController.cs

## 3.16 Utils Class Reference

Author: Paulo Renato Conceição Mendes.

This class has static methods that can be used in situations related to different contexts.

### Static Public Member Functions

- static Vector3 [PolarToCartesian](#) (Vector3 origin, float theta, float phi)

*This method converts a polar coordinate to cartesian.*

#### 3.16.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

This class has static methods that can be used in situations related to different contexts.

#### 3.16.2 Member Function Documentation

##### 3.16.2.1 PolarToCartesian()

```
static Vector3 Utils.PolarToCartesian (
    Vector3 origin,
    float theta,
    float phi ) [static]
```

This method converts a polar coordinate to cartesian.

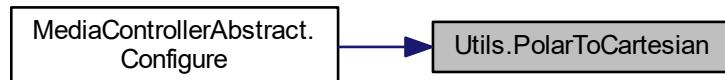
##### Parameters

<i>origin</i>	Vector origin
<i>theta</i>	Vertical angle in degrees from origin
<i>phi</i>	Horizontal angle in degrees from origin

**Returns**

Point in cartesian coordinate system

Here is the caller graph for this function:



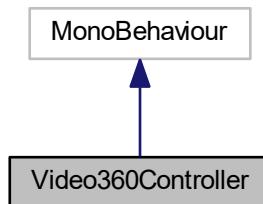
The documentation for this class was generated from the following file:

- Assets/Scripts/Utils.cs

## 3.17 Video360Controller Class Reference

Author: Paulo Renato Conceição Mendes.  
This class is the script that controls the video360.

Inheritance diagram for Video360Controller:



### Public Member Functions

- void [LoadVideo360](#) (string file\_path, float volume)  
*Loads and prepares to play the 360 video.*
- void [StartVideo360](#) ()  
*Starts the 360 video.*
- void [StopVideo360](#) ()  
*Stops the 360 video.*

- `GameObject AddMedia (string id, GameObject mediaPrefab, float begin=0, float r=0, float theta=0, float phi=0, string file_path="", float volume=0, bool loop=false, float duration=float.MaxValue, bool follow_camera=false, string text="", string on_select_name="", string during_out_of_focus_name="", string on_focus_name="", float clipBegin=0, float clipEnd=5)`  
*Adds additional media objects to this 360 video.*
- `void AddSubtitle (string id, GameObject mediaPrefab, string file_path, float r=0, float theta=0, float phi=0, string on_select_name=""")`  
*Add subtitles to the current 360 video.*

## Public Attributes

- Texture `Video360Texture`  
*texture for 360 videos*
- Texture `TeleMidiaTexture`  
*texture used when no 360 video is playing*
- Material `Material360`  
*material used for 360 videos*

## Private Member Functions

- `void EndVideo360 (VideoPlayer source)`  
*Stops the presentation the video finishes.*
- `void StartOtherMedia (VideoPlayer source)`  
*Starts the additional media objects that are added to this 360 video.*
- `void AbortOtherMedia (VideoPlayer source)`  
*Aborts the media objects when the video finishes.*

## Private Attributes

- `VideoPlayer video360`  
*videoPlayer component that plays the video*
- `List< GameObject > other_media`  
*list of the media objects that are added to this interactive 360 video*

### 3.17.1 Detailed Description

Author: Paulo Renato Conceição Mendes.  
 This class is the script that controls the video360.

### 3.17.2 Member Function Documentation

#### 3.17.2.1 AbortOtherMedia()

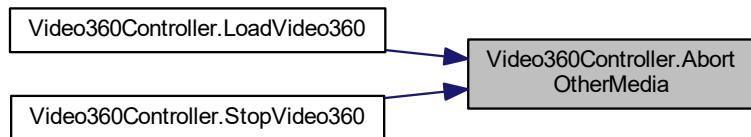
```
void Video360Controller.AbortOtherMedia (
    VideoPlayer source ) [private]
```

Aborts the media objects when the video finishes.

**Parameters**

<code>source</code>	The video player that is playing the 360 video
---------------------	--

Here is the caller graph for this function:

**3.17.2.2 AddMedia()**

```
GameObject Video360Controller.AddMedia (
    string id,
    GameObject mediaPrefab,
    float begin = 0,
    float r = 0,
    float theta = 0,
    float phi = 0,
    string file_path = "",
    float volume = 0,
    bool loop = false,
    float duration = float.MaxValue,
    bool follow_camera = false,
    string text = "",
    string on_select_name = "",
    string during_out_of_focus_name = "",
    string on_focus_name = "",
    float clipBegin = 0,
    float clipEnd = 5 )
```

Adds additional media objects to this 360 video.

**Parameters**

<code>id</code>	id of the media object
<code>mediaPrefab</code>	prefab of the media object that will be added
<code>begin</code>	start time of the media object in seconds
<code>r</code>	media object distance from the center
<code>theta</code>	vertical angle of the media object in degrees
<code>phi</code>	horizontal angle of the media object in degrees
<code>file_path</code>	source file url of the media object
<code>volume</code>	volume of the media object

**Parameters**

<i>loop</i>	wether the media object executes in loop
<i>duration</i>	duration of the media object in seconds
<i>follow_camera</i>	wether the media object follows the camera (is fixed to the user)
<i>text</i>	text of the media object
<i>on_select_name</i>	media object that will be played when the current is selected
<i>during_out_of_focus_name</i>	media object that will be played when the current is out of focus
<i>on_focus_name</i>	media object that will be played when the current is on focus
<i>clipBegin</i>	initial time of the segment of the media that will be played
<i>clipEnd</i>	final time of the segment of the media that will be played

**Returns**

The media object added

Here is the caller graph for this function:

**3.17.2.3 AddSubtitle()**

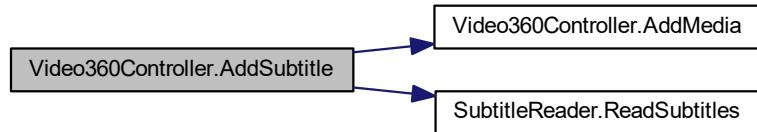
```
void Video360Controller.AddSubtitle (
    string id,
    GameObject mediaPrefab,
    string file_path,
    float r = 0,
    float theta = 0,
    float phi = 0,
    string on_select_name = "" )
```

Add subtitles to the current 360 video.

**Parameters**

<i>id</i>	id of the subtitles
<i>mediaPrefab</i>	Prefab of the text
<i>file_path</i>	path to the srt file
<i>r</i>	distance from the center
<i>theta</i>	vertical angle of the media object in degrees
<i>phi</i>	horizontal angle of the media object in degrees
<i>on_select_name</i>	media object that will be played when the current is selected

Here is the call graph for this function:



### 3.17.2.4 EndVideo360()

```
void Video360Controller.EndVideo360 (
    VideoPlayer source ) [private]
```

Stops the presentation the video finishes.

#### Parameters

<i>source</i>	The video player that is playing the 360 video
---------------	--

Here is the caller graph for this function:



### 3.17.2.5 LoadVideo360()

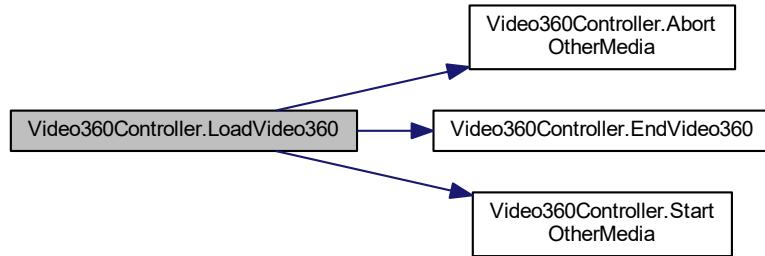
```
void Video360Controller.LoadVideo360 (
    string file_path,
    float volume )
```

Loads and prepares to play the 360 video.

#### Parameters

<i>file_path</i>	Path of the 360 video
<i>volume</i>	volume of the 360 video

Here is the call graph for this function:



### 3.17.2.6 StartOtherMedia()

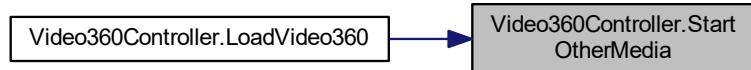
```
void Video360Controller.StartOtherMedia (
    VideoPlayer source ) [private]
```

Starts the additional media objects that are added to this 360 video.

#### Parameters

source	The video player that is playing the 360 video
--------	--

Here is the caller graph for this function:



### 3.17.2.7 StartVideo360()

```
void Video360Controller.StartVideo360 ( )
```

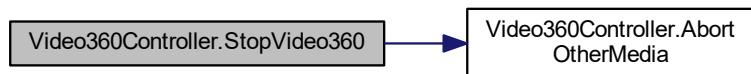
Starts the 360 video.

### 3.17.2.8 StopVideo360()

```
void Video360Controller.StopVideo360 ( )
```

Stops the 360 video.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

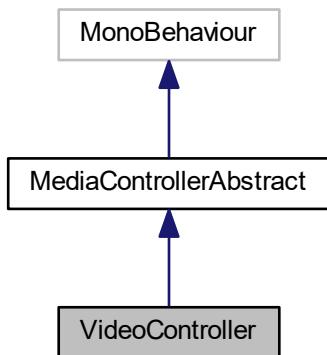
- Assets/Scripts/MediaControllers/Video360Controller.cs

## 3.18 VideoController Class Reference

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the video media object.

Inheritance diagram for VideoController:



## Public Member Functions

- `override void Load ()`  
*Inherited from [MediaControllerAbstract](#). It loads the video on the video player from the url, prepare it and call the LoadVideo routine when the video is prepared.*
- `void LoadVideo (VideoPlayer source)`  
*Called when the video file has been loaded and the video player is prepared. Additional configurations are made.*
- `override void PlayMedia ()`  
*Inherited from [MediaControllerAbstract](#). Playing for the video consists in playing the videoPlayer and enabling its mesh renderer*
- `override void StopMedia ()`  
*Inherited from [MediaControllerAbstract](#). Stopping for the video consists in stopping the videoPlayer, prepare it to play again, and disabling its mesh renderer*

## Private Member Functions

- `void Ended (VideoPlayer source)`  
*Defines that when the video reaches the loop point, the mesh renderer is disabled if the video is not to play in loop*

## Private Attributes

- `VideoPlayer videoPlayer`  
*Video Player component that is used to play the video.*
- `AudioSource audioSource`  
*Audio source that is used to stores and play the audio of the video.*

## Additional Inherited Members

### 3.18.1 Detailed Description

Author: Paulo Renato Conceição Mendes.

Inherits [MediaControllerAbstract](#). This class is the script that controls the video media object.

### 3.18.2 Member Function Documentation

#### 3.18.2.1 Ended()

```
void VideoController.Ended (
    VideoPlayer source ) [private]
```

Defines that when the video reaches the loop point, the mesh renderer is disabled if the video is not to play in loop

#### Parameters

<code>source</code>	Video Player
---------------------	--------------

Here is the caller graph for this function:



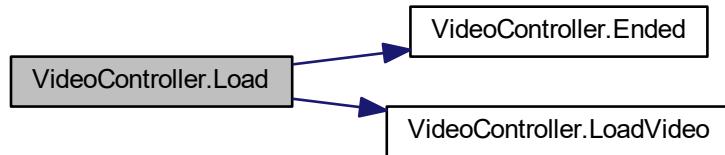
### 3.18.2.2 Load()

```
override void VideoController.Load ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). It loads the video on the video player from the url, prepare it and call the LoadVideo routine when the video is prepared.

Implements [MediaControllerAbstract](#).

Here is the call graph for this function:



### 3.18.2.3 LoadVideo()

```
void VideoController.LoadVideo (   
    VideoPlayer source )
```

Called when the video file has been loaded and the video player is prepared. Additional configurations are made.

#### Parameters

source	Video Player
--------	--------------

Here is the caller graph for this function:



### 3.18.2.4 PlayMedia()

```
override void VideoController.PlayMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Playing for the video consists in playing the videoPlayer and enabling its mesh renderer

Implements [MediaControllerAbstract](#).

### 3.18.2.5 StopMedia()

```
override void VideoController.StopMedia ( ) [virtual]
```

Inherited from [MediaControllerAbstract](#). Stopping for the video consists in stopping the videoPlayer, prepare it to play again, and disabling its mesh renderer

Implements [MediaControllerAbstract](#).

The documentation for this class was generated from the following file:

- Assets/Scripts/MediaControllers/VideoController.cs

# Index

AbortMedia  
    MediaControllerAbstract, 16

AbortOtherMedia  
    Video360Controller, 50

AddAdditionalMedia  
    MultimediaControllerScript, 26

AddMedia  
    Video360Controller, 51

AddSubtitle  
    Video360Controller, 52

AddVideo360  
    MultimediaControllerScript, 26

AudioController, 5  
    Load, 6  
    LoadAudio, 6  
    PlayMedia, 7  
    StopMedia, 7

begin  
    SubtitleFragment, 39

Change  
    ControllerChecker, 8

Configure  
    MediaControllerAbstract, 16

controller  
    TestScript, 46

ControllerChecker, 8  
    Change, 8

duration  
    SubtitleFragment, 39

Ended  
    VideoController, 56

EndVideo360  
    Video360Controller, 53

HotspotController, 9  
    Load, 10  
    PlayMedia, 10  
    StopMedia, 10

ImageController, 11  
    Load, 11  
    PlayMedia, 12  
    StopMedia, 12

InvokePlayStop  
    MediaControllerAbstract, 18

IsPlaying  
    MediaControllerAbstract, 22

Load  
    AudioController, 6  
    HotspotController, 10  
    ImageController, 11  
    MediaControllerAbstract, 18  
    MirrorController, 23  
    PreviewController, 31  
    TextController, 47  
    VideoController, 57

LoadAudio  
    AudioController, 6

LoadVideo  
    VideoController, 57

LoadVideo360  
    Video360Controller, 53

LoadXmlFile  
    MultimediaControllerScript, 27

LookHotspotScript, 12  
    Update, 13

MediaControllerAbstract, 13  
    AbortMedia, 16  
    Configure, 16  
    InvokePlayStop, 18  
    IsPlaying, 22  
    Load, 18  
    OnSelectMedia, 19  
    PlayMedia, 19  
    PrepareAnchors, 19  
    StopMedia, 20  
    SuperPlay, 20  
    SuperStop, 21

MirrorController, 22  
    Load, 23  
    PlayMedia, 23  
    StopMedia, 23  
    Update, 23

MultimediaControllerScript, 24  
    AddAdditionalMedia, 26  
    AddVideo360, 26  
    LoadXmlFile, 27  
    SetAsInitial, 28  
    Start, 28  
    StartPresentation, 29  
    StopPresentation, 29

OnSelectMedia  
    MediaControllerAbstract, 19

PauseVideoPreview

PreviewController, 31  
**PlayMedia**  
 AudioController, 7  
 HotspotController, 10  
 ImageController, 12  
 MediaControllerAbstract, 19  
 MirrorController, 23  
 PreviewController, 31  
 TextController, 47  
 VideoController, 58  
**PlayVideoPreview**  
 PreviewController, 32  
**PolarToCartesian**  
 Utils, 48  
**PrepareAnchors**  
 MediaControllerAbstract, 19  
**PreviewController**, 30  
 Load, 31  
 PauseVideoPreview, 31  
 PlayMedia, 31  
 PlayVideoPreview, 32  
 StopMedia, 32  
**RaySelect**, 33  
 ShootLaserFromTargetPosition, 34  
 Start, 34  
 TriggerDown, 35  
 Update, 35  
**ReadSubtitles**  
 SubtitleReader, 39  
**RotateScript**, 36  
 speed, 37  
 Update, 37  
**RunTests**  
 TestScript, 41  
**SetAsInitial**  
 MultimediaControllerScript, 28  
**ShootLaserFromTargetPosition**  
 RaySelect, 34  
**speed**  
 RotateScript, 37  
**Start**  
 MultimediaControllerScript, 28  
 RaySelect, 34  
**StartOtherMedia**  
 Video360Controller, 54  
**StartPresentation**  
 MultimediaControllerScript, 29  
**StartVideo360**  
 Video360Controller, 54  
**StopMedia**  
 AudioController, 7  
 HotspotController, 10  
 ImageController, 12  
 MediaControllerAbstract, 20  
 MirrorController, 23  
 PreviewController, 32  
 TextController, 47  
 VideoController, 58  
**StopPresentation**  
 MultimediaControllerScript, 29  
**StopVideo360**  
 Video360Controller, 54  
**SubtitleFragment**, 37  
 begin, 39  
 duration, 39  
 SubtitleFragment, 38  
 text, 39  
 ToString, 38  
**SubtitleReader**, 39  
 ReadSubtitles, 39  
**SuperPlay**  
 MediaControllerAbstract, 20  
**SuperStop**  
 MediaControllerAbstract, 21  
**TestAdditionalMedia360Video**  
 TestScript, 42  
**TestHotSpot\_OnLookAt\_DuringNotLookingAt**  
 TestScript, 43  
**TestLoadStartStop360Video**  
 TestScript, 44  
**TestNavigation**  
 TestScript, 45  
**TestScript**, 40  
 controller, 46  
 RunTests, 41  
 TestAdditionalMedia360Video, 42  
 TestHotSpot\_OnLookAt\_DuringNotLookingAt, 43  
 TestLoadStartStop360Video, 44  
 TestNavigation, 45  
**text**  
 SubtitleFragment, 39  
**TextController**, 46  
 Load, 47  
 PlayMedia, 47  
 StopMedia, 47  
**ToString**  
 SubtitleFragment, 38  
**TriggerDown**  
 RaySelect, 35  
**Update**  
 LookHotspotScript, 13  
 MirrorController, 23  
 RaySelect, 35  
 RotateScript, 37  
**Utils**, 48  
 PolarToCartesian, 48  
**Video360Controller**, 49  
 AbortOtherMedia, 50  
 AddMedia, 51  
 AddSubtitle, 52  
 EndVideo360, 53  
 LoadVideo360, 53  
 StartOtherMedia, 54

StartVideo360, [54](#)  
StopVideo360, [54](#)  
VideoController, [55](#)  
    Ended, [56](#)  
    Load, [57](#)  
    LoadVideo, [57](#)  
    PlayMedia, [58](#)  
    StopMedia, [58](#)