My Project

Generated by Doxygen 1.9.7

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

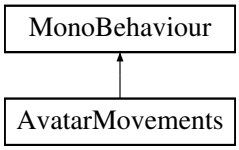
Chapter 3

Class Documentation

3.1 AvatarMovements Class Reference

The following script contains the logic to move an avatar to different positions.

Inheritance diagram for AvatarMovements:



Public Member Functions

- void **goToStraight** ()

  When this method is called, the avatar goes in the straight position.
- void **waveHandRight** ()

  When this method is called, the avatar waves his hand to the right.
- void **waveHandLeft** ()

  When this method is called, the avatar waves his hand to the left.
- void **walkRight** ()

  When this method is called, the avatar lifts the right leg.
- void **walkLeft** ()

  When this method is called, the avatar lifts the left leg.
- void **handsTalkingUp** ()

  When this method is called, the avatar's hands are moved up.
- void **handsTalkingDown** ()

  When this method is called, the avatar's hands are moved down.
- void **rock** ()

  When this method is called, the avatar's does a rock with his hand.
- void **paper** ()

  When this method is called, the avatar's does a paper with his hand.
- void **scissors** ()

  When this method is called, the avatar's does scissors with his hand.

Public Attributes

- float **desiredDuration** = 2.5f

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **moveGameObjects** ()

  This method moves the different part of the avatar when called.
- void goToPose (Vector3[ ] pos, Vector3[ ] angles, float desiredDuration)

  This method is used to setup the movement (the positions and angles of the different parts of the avatar) done in the ¨MoveGameObjects¨ method.

Private Attributes

- GameObject **headGO**
- GameObject **hipGO**
- GameObject **l_handGO**
- GameObject **r_handGO**
- GameObject **r_thumbGO**
- GameObject **r_indexGO**
- GameObject **r_middleGO**
- GameObject **r_ringGO**
- GameObject **r_littleGO**
- GameObject **l_thumbGO**
- GameObject **l_indexGO**
- GameObject **l_middleGO**
- GameObject **l_ringGO**
- GameObject **l_littleGO**
- GameObject **l_footGO**
- GameObject **r_footGO**
- Vector3[ ] **neutralStraightPose**
- Vector3[ ] **neutralStraightAngles**
- Vector3[ ] **waveHandRightPose**
- Vector3[ ] **waveHandRightAngles**
- Vector3[ ] **waveHandLeftPose**
- Vector3[ ] **waveHandLeftAngles**
- Vector3[ ] **rightWalkingPose**
- Vector3[ ] **rightWalkingAngles**
- Vector3[ ] **leftWalkingPose**
- Vector3[ ] **leftWalkingAngles**
- Vector3[ ] **handsTalkingUpPose**
- Vector3[ ] **handsTalkingUpAngles**
- Vector3[ ] **handsTalkingDownPose**
- Vector3[ ] **handsTalkingDownAngles**
- Vector3[ ] **rockPose**
- Vector3[ ] **rockAngles**
- Vector3[ ] **paperPose**
- Vector3[ ] **paperAngles**
- Vector3[ ] **scissorsPose**

- Vector3[ ] **scissorsAngles**
- AnimationCurve **curve**
- GameObject[ ] **gameObjects** = new GameObject[16]
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startPosTab** = new List<Vector3>()
- List< Vector3 > **endPosTab** = new List<Vector3>()
- List< Vector3 > **startAngleTab** = new List<Vector3>()
- List< Vector3 > **endAngleTab** = new List<Vector3>()
- List< GameObject > **GOTab** = new List<GameObject>()
- List< float > **desiredDurationTab** = new List<float>()

3.1.1 Detailed Description

The following script contains the logic to move an avatar to different positions.

3.1.2 Member Function Documentation

3.1.2.1 goToPose()

```
void AvatarMovements.goToPose (
            Vector3[] pos,
            Vector3[] angles,
            float desiredDuration )  [private]
```

This method is used to setup the movement (the positions and angles of the different parts of the avatar) done in the ¨MoveGameObjects¨ method.

Parameters

pos	The new positions of the different parts of the avatar.
angles	The new angles of the different parts of the avatar.

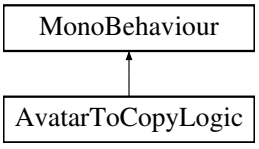
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/AvatarMovements.cs

3.2 AvatarToCopyLogic Class Reference

The following script contains the logic of the avatar doing the movements the player has to imitate in the Dance Game. All the movements of this avatar are done here. A movement is triggered by a boolean designating a specific movement.

Inheritance diagram for AvatarToCopyLogic:



Public Attributes

- bool **rightKneeUp** = false
- bool **leftKneeUp** = false
- bool **rightKneeMiddleUp** = false
- bool **leftKneeMiddleUp** = false
- bool **straight** = false
- bool **leftTargetPoint** = false
- bool **rightTargetPoint** = false

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **moveGameObjects** ()

  This method moves the different part of the avatar to copy when called.
- void goToPose (Vector3[ ] pos, Vector3[ ] angles)

  This method is used to setup the movement (the positions and angles of the different parts of the avatar to copy) done in the ¨MoveGameObjects¨ method.

Private Attributes

- Spawner **spawner**
- OpponentLogic **opponent**
- GameObject **headGO**
- GameObject **hipGO**
- GameObject **l_handGO**
- GameObject **r_handGO**
- GameObject **r_thumbGO**
- GameObject **r_indexGO**
- GameObject **r_middleGO**
- GameObject **r_ringGO**
- GameObject **r_littleGO**
- GameObject **l_thumbGO**
- GameObject **l_indexGO**
- GameObject **l_middleGO**
- GameObject **l_ringGO**
- GameObject **l_littleGO**
- GameObject **l_footGO**
- GameObject **r_footGO**
- Vector3[ ] **neutralStraightPose**
- Vector3[ ] **neutralStraightAngles**
- Vector3[ ] **leftKneeUpPose**
- Vector3[ ] **leftKneeUpAngles**
- Vector3[ ] **rightKneeUpPose**
- Vector3[ ] **rightKneeUpAngles**
- Vector3[ ] **leftMiddleKneeUpPose**
- Vector3[ ] **leftMiddleKneeUpAngles**
- Vector3[ ] **rightMiddleKneeUpPose**
- Vector3[ ] **rightMiddleKneeUpAngles**
- Vector3[ ] **leftTargetPointPose**

- Vector3[ ] **leftTargetPointAngles**
- Vector3[ ] **rightTargetPointPose**
- Vector3[ ] **rightTargetPointAngles**
- float **desiredDuration** = 2.5f
- AnimationCurve **curve**
- GameObject[ ] **gameObjects** = new GameObject[16]
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startPosTab** = new List<Vector3>()
- List< Vector3 > **endPosTab** = new List<Vector3>()
- List< Vector3 > **startAngleTab** = new List<Vector3>()
- List< Vector3 > **endAngleTab** = new List<Vector3>()
- List< GameObject > **GOTab** = new List<GameObject>()

3.2.1 Detailed Description

The following script contains the logic of the avatar doing the movements the player has to imitate in the Dance Game. All the movements of this avatar are done here. A movement is triggered by a boolean designating a specific movement.

3.2.2 Member Function Documentation

3.2.2.1 goToPose()

```
void AvatarToCopyLogic.goToPose (
            Vector3[] pos,
            Vector3[] angles )  [private]
```

This method is used to setup the movement (the positions and angles of the different parts of the avatar to copy) done in the ¨MoveGameObjects¨ method.

Parameters

pos	The new positions of the different parts of the avatar to copy.
angles	The new angles of the different parts of the avatar to copy.

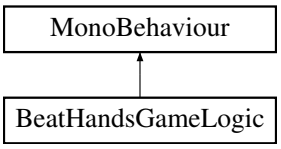
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Avatar↩ ToCopyLogic.cs

3.3 BeatHandsGameLogic Class Reference

This class is the logic of the BeatHands game.

Inheritance diagram for BeatHandsGameLogic:



Public Member Functions

- void SetDifficulty (int difficulty)

  This method sets the difficulty of the spawner.
- void **ToggleIsPlaying** ()

  This method starts or stops the game depending on the current state.

Public Attributes

- int **totalScore** = 0
- int **difficulty** = 0
- bool **finished** = false
- string **resultText** = ""

Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- UIPointer **pointer**
- BeatHandsSpawner **spawner**
- TextMeshProUGUI **timerText**
- TextMeshProUGUI **scoreText**
- TextMeshProUGUI **finalScoreText**
- TextMeshProUGUI **timerTextFR**
- TextMeshProUGUI **scoreTextFR**
- TextMeshProUGUI **finalScoreTextFR**
- MenuLogic **menu**
- float **inGameTime** = 0f
- bool **isPlaying** = false

3.3.1 Detailed Description

This class is the logic of the BeatHands game.

3.3.2 Member Function Documentation

3.3.2.1 SetDifficulty()

```
void BeatHandsGameLogic.SetDifficulty (
            int difficulty )
```

This method sets the difficulty of the spawner.

Parameters

difficulty	The difficulty set for the game.

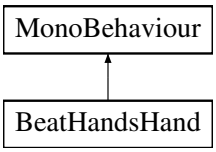
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BeatHandsScripts/BeatHandsGameLogic.cs

3.4 BeatHandsHand Class Reference

This class represent a player's hand in the BeatHands game.

Inheritance diagram for BeatHandsHand:



Private Member Functions

- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is a target and if it is the case, the score augments and the target is destroyed.

Private Attributes

- SoundManager **soundManager**
- string **color**
- BeatHandsGameLogic **gameLogic**

3.4.1 Detailed Description

This class represent a player's hand in the BeatHands game.

3.4.2 Member Function Documentation

3.4.2.1 OnTriggerEnter()

```
void BeatHandsHand.OnTriggerEnter (
            Collider other )  [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is a target and if it is the case, the score augments and the target is destroyed.

Parameters

other	The other collider that hit the collider attached to this object.

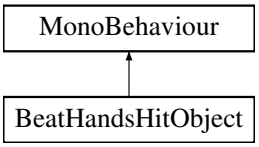
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BeatHandsScripts/BeatHandsHand.cs

3.5 BeatHandsHitObject Class Reference

This class represents a target for the BeatHands game.

Inheritance diagram for BeatHandsHitObject:



Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- AudioSource **source**
- bool **playSound**
- BeatHandsSpawner **spawner**

3.5.1 Detailed Description

This class represents a target for the BeatHands game.

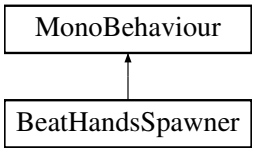
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BeatHandsScripts/BeatHandsHitObject.cs

3.6 BeatHandsSpawner Class Reference

This class represents the target spawner for the BeatHands game.

Inheritance diagram for BeatHandsSpawner:



Public Member Functions

- void SetDifficulty (int difficulty)

  This method is used to set the difficulty. An harder difficulty means more different spawn points and spawn points being further away from the player.

Public Attributes

- bool **isPlaying** = false

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **SpawnNewTarget** ()

  This method spawns a new target.

Private Attributes

- SoundManager **soundManager**
- GameObject[ ] **hitObjects**
- Transform[ ] **easySpawnPoints**
- Transform[ ] **normalSpawnPoints**
- Transform[ ] **hardSpawnPoints**
- Transform[ ] **superHardSpawnPoints**
- Transform **lookAtPoint**
- List< Transform > **possiblePositions** = new List<Transform>()

3.6.1 Detailed Description

This class represents the target spawner for the BeatHands game.

3.6.2 Member Function Documentation

3.6.2.1 SetDifficulty()

```
void BeatHandsSpawner.SetDifficulty (
            int difficulty )
```

This method is used to set the difficulty. An harder difficulty means more different spawn points and spawn points being further away from the player.

Parameters

difficulty	The difficulty to set to the game.

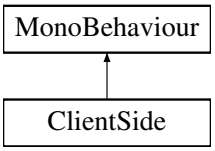
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BeatHandsScripts/BeatHandsSpawner.cs

3.7 ClientSide Class Reference

This script has mainly been done by Eugénie and has been reused and modified for this project. This script handles the connection with the exoskeleton Autonomyo.

Inheritance diagram for ClientSide:



Classes

- class TCP

Public Member Functions

- void **ConnectFromButton** ()
- void **TryConnectingAgain** ()
- void **TryStreamedVars** ()
- void **setToConnecting** ()
- void **setToConnected** ()
- void **setToDisconnected** ()
- void **FinishInit** ()
- void **LoadControllerParams** (string name1, string name2, string name3)
- SyncVar **GetSyncVar** (string name)
- bool **ConvertFloatToBool** (float value)
- void **SetBoolSyncVar** (string name, bool b)
- void **SetIntSyncVar** (string name, int value)
- void **SetFloatSyncVar** (string name, float value)
- void **SetStringSyncVar** (string name, string str)

Public Attributes

- string **serverAddr** = ¨192.168.200.102¨
- int **port** = 9255
- int **myId** = 0
- TCP **tcp**
- TextMeshProUGUI **connectText**
- TextMeshProUGUI **connectTextFR**
- int **connectionCounter** = 0
- bool **isConnected** = false
- bool **isConnecting** = false
- bool **updateText** = false
- bool **started** = false
- bool **svIWasCreated** = false
- bool **svIIsFull** = false
- bool **iHaveValue** = false
- bool **isStreaming** = false
- bool **receivedHeartbeat** = false
- List< SyncVar > **syncVarsList**
- List< SyncVar > **streamedVars**

Static Public Attributes

- const int **SYNCVAR_NAME_COMM_LENGTH** = 100

  Length of SyncVar names during remote listing.
- const int **SYNCVAR_UNIT_COMM_LENGTH** = 20

  Length of SyncVar units during remote listing.
- const int **SYNCVAR_LIST_ITEM_SIZE** = (SYNCVAR_NAME_COMM_LENGTH + SYNCVAR_UNIT_COMM_LENGTH + 1 + 1 + 4)

  SyncVar description bytes size during remote listing.
- const char **SYNCVAR_NAME_SEPARATOR** = '/'

  SyncVar prefix separators.
- static ClientSide **instance**
- static int **dataBufferSize** = 16384

Private Member Functions

- void **Start** ()
- void **Update** ()
- void **Awake** ()
- void **ConnectToServer** ()
- void **SendingThread** ()
- IEnumerator **SetupExoskeleton** ()
- short **GetSyncVarIndex** (string sv_name)

Private Attributes

- GameObject **connectionTab**
- Button **connectButton**
- GameObject **menu**
- MenuLogic **menuLogic**
- TextMeshProUGUI **batteryPairedText**
- GameObject **connectionPopupPanel**
- Button **connectionSelectedButton**
- List< byte > **byteList**

  Received recombined bytes (actual data bytes) list.
- ushort **nVars**

  Number of variables from the exoskeleton.
- MbToPcMessageType **rxCurrentMessageType**

  Current type of message being received.
- int **rxBytesCount**

  Number of bytes received for the current message.
- int **firstHalfByte**

  First half of the byte being recomposed from two received bytes.
- bool **tryAgain** = true
- int **streamPacketSize**

  Size of a streaming packet [B].
- byte **streamingRequest** = 0
- System.Timers.Timer **heartBeatTimer**
- System.Timers.Timer **connectionTimeoutTimer**
- System.Timers.Timer **sendTimer**
- DateTime **lastSentHeartbeatTime**
- DateTime **lastSentStreamPacketTime**
- Thread **clientReceiveThread**
- Thread **clientDecodeThread**

Static Private Attributes

- const int **TCP_MAX_TIME_WITHOUT_RX** = 10000

3.7.1 Detailed Description

This script has mainly been done by Eugénie and has been reused and modified for this project. This script handles the connection with the exoskeleton Autonomyo.

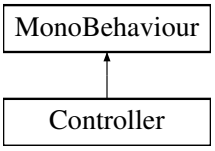
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/ClientSide.cs

3.8 Controller Class Reference

The following script is used to control the Exoskeleton Autonomyo.

Inheritance diagram for Controller:



Public Member Functions

- void **toggleArmMotors** ()

  When this method is called, the motors of the exoskeleton are armed or disarmed based on the current state.
- void **setAbductionZero** ()

  When this method is called, the abduction of the exo is set to zero. This is used to callibrate the exo for the app to work properly.
- void **setJointZero** ()

  When this method is called, the joints of the exo are set to zero. This is used to callibrate the exo for the app to work properly.
- void **setRightCellsZero** ()

  When this method is called, the right cells of the exo are set to zero. This is used to callibrate the exo for the app to work properly.
- void **setLeftCellsZero** ()

  When this method is called, the left cells of the exo are set to zero. This is used to callibrate the exo for the app to work properly.

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.

Private Attributes

- GameObject **theClient**
- GameObject **checkMark1**
- GameObject **checkMark2**
- GameObject **checkMark1FR**
- ClientSide **clientSide**
- bool **areMotorsArmed** = false

3.8.1 Detailed Description

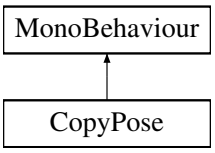
The following script is used to control the Exoskeleton Autonomyo.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/Controller.cs

3.9 CopyPose Class Reference

Inheritance diagram for CopyPose:



Protected Attributes

- QuickCopyPoseBase **_copyPose** = null

Private Member Functions

- void **Start** ()
- void **Update** ()

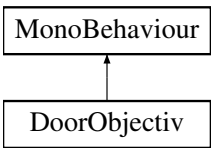
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Copy↩ Pose.cs

3.10 DoorObjectiv Class Reference

This class represents an objectiv for the door balance game. An objectiv is a checkpoint to reach with the green sphere. All the objectivs need to be reached in the correct order to open the door.

Inheritance diagram for DoorObjectiv:



Public Member Functions

- void **resetObjCount** ()

  This method sets the object count to 0.
- void **resetObj** ()

  This method resets the objective.

Public Attributes

- bool **succeed** = false

Private Member Functions

- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the objective is the next objective to validate and if it is the case, it validates the objective.

Private Attributes

- TextMeshProUGUI **text**
- int **objNb**

Static Private Attributes

- static int **objCount** = 0

3.10.1 Detailed Description

This class represents an objectiv for the door balance game. An objectiv is a checkpoint to reach with the green sphere. All the objectivs need to be reached in the correct order to open the door.

3.10.2 Member Function Documentation

3.10.2.1 OnTriggerEnter()

```
void DoorObjectiv.OnTriggerEnter (
            Collider other )  [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the objective is the next objective to validate and if it is the case, it validates the objective.

Parameters

other	The other collider that hit the collider attached to this object.

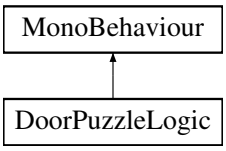
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BalanceGame↩ Scripts/DoorObjectiv.cs

3.11 DoorPuzzleLogic Class Reference

This class is the logic of the balance game.

Inheritance diagram for DoorPuzzleLogic:



Public Member Functions

- void **startGame** ()

  This method is called to start the game.

Public Attributes

- bool **test** = false
- string **resultText** = ""

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **resetGame** ()

  This method is used to reset the game.

Private Attributes

- StoryGameLogic **storyGameLogic**
- MenuLogic **menu**
- int **puzzleNb**
- List< DoorObjectiv > **objectivs1** = new List<DoorObjectiv>()
- List< DoorObjectiv > **objectivs2** = new List<DoorObjectiv>()
- List< DoorObjectiv > **objectivs3** = new List<DoorObjectiv>()
- GameObject **objectivs1GO**
- GameObject **objectivs2GO**
- GameObject **objectivs3GO**
- GameObject **door**
- GameObjectController **sphere**
- Vector3 **startAngle**
- Vector3 **endAngle**
- int **rotationSpeed**
- int **nbOfLevels**
- Transform **initSpherePos**
- bool **done** = false
- float **startTime**
- float **distanceBetweenRot**
- bool **finished** = false
- bool **doneFinish** = false
- int **levelNb** = 1
- List< DoorObjectiv > **currentObjectivs**
- float **startGameTime** = 0f

3.11.1 Detailed Description

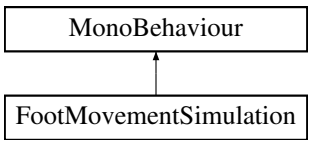
This class is the logic of the balance game.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/BalanceGame↩ Scripts/DoorPuzzleLogic.cs

3.12 FootMovementSimulation Class Reference

Inheritance diagram for FootMovementSimulation:



Private Member Functions

- void **Start** ()
- void **Update** ()
- void **startMovementListener** (string toggleString, ref bool isMovingForward, bool isMovingBackward, ref Vector3 initialPos, ref Vector3 finalPos, GameObject foot, Vector3 movementvector, bool isLeft)
- bool **executeForwardMovement** (bool isMovingForward, ref bool isMovingBackward, ref float elapsedTime, ref float percentageComplete, ref GameObject foot, ref Vector3 initialPos, ref Vector3 finalPos, Vector3 movementvector)
- bool **executeBackwardMovement** (bool isMovingBackward, ref float elapsedTime, ref float percentage↩ Complete, ref GameObject foot, Vector3 initialPos, Vector3 finalPos, bool isLeft)

Private Attributes

- GameObject **leftFoot**
- GameObject **rightFoot**
- float **desiredDuration**
- AnimationCurve **curve**
- Vector3 **upKneeMovementVector**
- Vector3 **abductionRightVector**
- Vector3 **abductionLeftVector**
- bool **isLeftMoving** = false
- bool **isRightMoving** = false
- float **elapsedTimeUpKneeRight** = 0
- float **elapsedTimeUpKneeLeft** = 0
- float **elapsedTimeAbductionLeft** = 0
- float **elapsedTimeAbductionRight** = 0
- Vector3 **initialPosUpKneeLeft**
- Vector3 **finalPosUpKneeLeft**
- Vector3 **initialPosUpKneeRight**
- Vector3 **finalPosUpKneeRight**

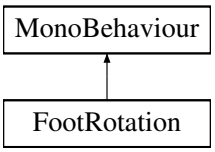
- Vector3 **initialPosAbductionLeft**
- Vector3 **finalPosAbductionLeft**
- Vector3 **initialPosAbductionRight**
- Vector3 **finalPosAbductionRight**
- bool **isMovingBackwardUpKneeLeft** = false
- bool **isMovingBackwardUpKneeRight** = false
- bool **isMovingForwardUpKneeRight** = false
- bool **isMovingForwardUpKneeLeft** = false
- bool **isMovingBackwardAbductionLeft** = false
- bool **isMovingBackwardAbductionRight** = false
- bool **isMovingForwardAbductionRight** = false
- bool **isMovingForwardAbductionLeft** = false
- float **percentageCompleteUpKneeLeft**
- float **percentageCompleteUpKneeRight**
- float **percentageCompleteAbductionLeft**
- float **percentageCompleteAbductionRight**

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Foot↩ MovementSimulation.cs

3.13 FootRotation Class Reference

Inheritance diagram for FootRotation:



Private Member Functions

- void **Start** ()
- void **Update** ()

Private Attributes

- Transform **toeTransform**
- Transform **shinTransform**
- Transform **footTransform**
- Transform **targetFootTransform**
- Vector3 **sideFootShin**
- Vector3 **sideFootToe**
- float **baseAngle**
- float **actualAngle**
- float **angleToModify**

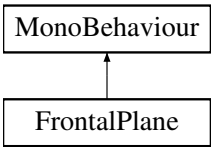
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Foot↩ Rotation.cs

3.14 FrontalPlane Class Reference

This script have been done by Euge and is reused as is in this project. This script creates the frontal plane of the exo based on the abduction, hips and knees angles of the exo.

Inheritance diagram for FrontalPlane:



Public Member Functions

- void **UpdateSegment** (RectTransform rectTransform, RectTransform object1, RectTransform object2)

Public Attributes

- float **l_knee_posy**
- float **l_knee_posz**
- float **r_knee_posy**
- float **r_knee_posz**
- float **l_foot_posy**
- float **l_foot_posz**
- float **r_foot_posy**
- float **r_foot_posz**

Private Member Functions

- void **Start** ()
- void **Update** ()

Private Attributes

- float **REDUCTION_FACTOR** = 10f
- float **SEGMENTWIDTH** = 3f
- GameObject **theClient**
- Vector3 **canvasPositionOffset**
- RectTransform **leftBackRT**
- RectTransform **rightBackRT**
- RectTransform **leftAbdRT**
- RectTransform **rightAbdRT**
- RectTransform **leftHipRT**
- RectTransform **rightHipRT**
- RectTransform **leftKneeRT**
- RectTransform **rightKneeRT**
- RectTransform **leftFootRT**
- RectTransform **rightFootRT**
- RectTransform **leftBA_rt**

- RectTransform **rightBA_rt**
- RectTransform **leftAH_rt**
- RectTransform **rightAH_rt**
- RectTransform **leftHK_rt**
- RectTransform **rightHK_rt**
- RectTransform **leftKF_rt**
- RectTransform **rightKF_rt**
- float **leftAbd**
- float **rightAbd**
- float **leftHip**
- float **rightHip**
- float **leftKnee**
- float **rightKnee**
- float **leftFoot**
- float **rightFoot**
- ClientSide **clientSide**
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- float **l0**
- float **l1**
- float **l3**
- float **l4**
- float **l_abd_angle** = -8.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_abd_angle** = 0
- float **l_hip_angle** = 40.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_hip_angle** = -30.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **l_knee_angle** = 60.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_knee_angle** = 10.0f / 360.0f ∗ 2 ∗ Mathf.PI
- bool **done** = false

3.14.1 Detailed Description

This script have been done by Euge and is reused as is in this project. This script creates the frontal plane of the exo based on the abduction, hips and knees angles of the exo.

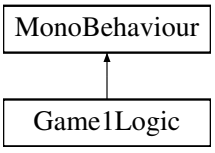
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/FrontalPlane.cs

3.15 Game1Logic Class Reference

The following script contains the logic of the Dance Game. The validation of the movements, the trigger of the movements of the avatar to copy as well as all the game gestion are done here.

Inheritance diagram for Game1Logic:



Public Member Functions

- void SetDifficulty (int difficulty)

  This method sets the difficulty of the game (modifying the margin of errors of the angles).

Public Attributes

- bool **inGame** = false
- bool **finished** = false
- bool **isBlinking** = false
- float **totalScore** = 0f
- int **difficulty** = 0
- string **resultText** = ""

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- bool checkPosewithAngles ()

  This method checks if the position of the player (depending on knee hip and abduction angles) is correct.
- float checkSpecificAngle (float targetAngle, float actualAngle, float marginOfError)

  This method checks if an angle is correct given the actual angle, the target angle and the margin of error.
- bool checkAngleCorrectness (float check_l_abdAngle, float check_r_abdAngle, float check_l_hipAngle, float check_r_hipAngle, float check_l_kneeAngle, float check_r_kneeAngle)

  This method checks if all angles (hips, knees, abductions) are correct given the results for each angle. This method also modifies the color of the helpers (green if correct, red otherwise).
- void **blink** ()

  This method is used to create a blink effect for the blue helpers of the position to immitate.
- void **setStartTime** ()

  This method is used to setup the start time of the game in order to get time data during the game.

Private Attributes

- OpponentLogic **opponent**
- AvatarToCopyLogic **avatarToCopyLogic**
- float **timeToValidate**
- FrontalPlane **frontalPlane**
- SagittalPlane **sagittalPlane**
- float **marginOfErrorPos**
- float **marginOfErrorAngles**
- float **marginOfErrorAnglesAbd**
- CircularProgressBar **progressBar**
- GameObject **theClient**
- GameObject **f_l_kneeGO**
- GameObject **f_r_kneeGO**
- GameObject **f_l_footGO**
- GameObject **f_r_footGO**
- GameObject **s_l_kneeGO**
- GameObject **s_r_kneeGO**

- GameObject **s_l_footGO**
- GameObject **s_r_footGO**
- TextMeshProUGUI **scoreText**
- TextMeshProUGUI **avrTimeText**
- TextMeshProUGUI **scoreTextFR**
- TextMeshProUGUI **avrTimeTextFR**
- int **maxNbOfSuccesses** = 10
- SoundManager **soundManager**
- Spawner **spawner**
- WindowGraph **graph**
- WindowGraph **graphFR**
- MenuLogic **menu**
- Material **correctAngleMat**
- Material **wrongAngleMat**
- GameObject **l_hipCorrectness**
- GameObject **l_kneeCorrectness**
- GameObject **r_hipCorrectness**
- GameObject **r_kneeCorrectness**
- GameObject **l_arrowExter**
- GameObject **l_arrowInter**
- GameObject **r_arrowExter**
- GameObject **r_arrowInter**
- float **validationCounter** = 0f
- Vector3[ ] **targetPositions** = null
- float[ ] **targetAngles** = null
- bool **newPose** = true
- ClientSide **clientSide**
- bool **done** = false
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- int **nbOfSuccesses** = 0
- string **poseName**
- bool **poseFinished** = false
- GameObject **blinkGO1**
- GameObject **blinkGO2**
- bool **blinkUp** = true
- float **startTime**
- float **totalTime**
- List< float > **poseTimes** = new List<float>()
- Image **scoreBarImg**
- bool **firstFrame** = true
- float **poseScore** = 0f
- TextMeshProUGUI **finalScoreText**
- TextMeshProUGUI **finalScoreTextFR**
- TextMeshProUGUI **playerScoreText**

3.15.1 Detailed Description

The following script contains the logic of the Dance Game. The validation of the movements, the trigger of the movements of the avatar to copy as well as all the game gestion are done here.

3.15.2 Member Function Documentation

3.15.2.1 checkAngleCorrectness()

```
bool Game1Logic.checkAngleCorrectness (
            float check_l_abdAngle,
            float check_r_abdAngle,
            float check_l_hipAngle,
            float check_r_hipAngle,
            float check_l_kneeAngle,
            float check_r_kneeAngle )  [private]
```

This method checks if all angles (hips, knees, abductions) are correct given the results for each angle. This method also modifies the color of the helpers (green if correct, red otherwise).

Parameters

check_l_abdAngle	The correctness result of the left abduction angle.
check_r_abdAngle	The correctness result of the right abduction angle.
check_l_hipAngle	The correctness result of the left hip angle.
check_r_hipAngle	The correctness result of the right hip angle.
check_l_kneeAngle	The correctness result of the left knee angle.
check_r_kneeAngle	The correctness result of the right knee angle.

Returns

true if all the angles are correct, false otherwise.

3.15.2.2 checkPosewithAngles()

```
bool Game1Logic.checkPosewithAngles ( )  [private]
```

This method checks if the position of the player (depending on knee hip and abduction angles) is correct.

Returns

true if the position of the player (depending on knee hip and abduction angles) is correct, return false otherwise.

3.15.2.3 checkSpecificAngle()

```
float Game1Logic.checkSpecificAngle (
            float targetAngle,
            float actualAngle,
            float marginOfError )  [private]
```

This method checks if an angle is correct given the actual angle, the target angle and the margin of error.

Parameters

targetAngle	The angle the player has to reach.
actualAngle	The angle of the actual position of the player.
marginOfError	The margin of error accepted between the actual angle and the target angle.

Returns

1 if the angle done by the player is correct, return -1 otherwise.

3.15.2.4 SetDifficulty()

```
void Game1Logic.SetDifficulty (
            int difficulty )
```

This method sets the difficulty of the game (modifying the margin of errors of the angles).

Parameters

difficulty	The difficulty set for the game..

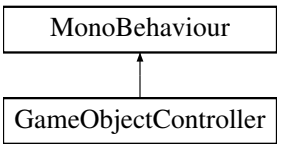
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Game1Logic.cs

3.16 GameObjectController Class Reference

The following script handles the controll of a gameObject using the balance of the player (thanks to the soles of the exoskeleton).

Inheritance diagram for GameObjectController:



Public Member Functions

- void **startGame** ()

  When this method is called, the control of the gameObject is activated.

Public Attributes

- bool **_move** = false

Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the game↩ Object hits a wall.
- void OnTriggerExit (Collider other)

  This method is triggered whenever another collider exits the collider attached to this object. It checks if the game↩ Object hits a wall.
- void **moveSphere** ()

  This is used to avoid the sphere to move through the walls.

Private Attributes

- LowerLimbsMovements **lowerLimbs**
- float **_maxRightLeftSpeed** = 1
- float **_maxUpDownSpeed** = 1
- GameObject **balanceSphere**
- float **sphereSpeed**
- DoorPuzzleLogic **puzzleLogic**
- bool **canGoPositivZ** = true
- bool **canGoNegativZ** = true
- bool **canGoPositivY** = true
- bool **canGoNegativY** = true
- float **diffY**
- float **diffZ**

3.16.1 Detailed Description

The following script handles the controll of a gameObject using the balance of the player (thanks to the soles of the exoskeleton).

3.16.2 Member Function Documentation

3.16.2.1 OnTriggerEnter()

```
void GameObjectController.OnTriggerEnter (
            Collider other )  [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the game↩ Object hits a wall.

Parameters

other	The other collider that hit the collider attached to this object.

3.16.2.2 OnTriggerExit()

```
void GameObjectController.OnTriggerExit (
            Collider other )  [private]
```

This method is triggered whenever another collider exits the collider attached to this object. It checks if the gameObject hits a wall.

Parameters

other	The other collider that hit the collider attached to this object.

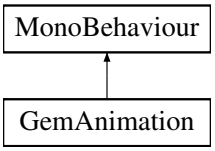
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/GameObjectController.cs

3.17 GemAnimation Class Reference

The following script handles the animation of the final gem of the story game.

Inheritance diagram for GemAnimation:



Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- float **maxHeight**
- float **minHeight**
- float **movementSpeed**
- float **rotationSpeed**
- bool **isGoingUp** = true

3.17.1 Detailed Description

The following script handles the animation of the final gem of the story game.

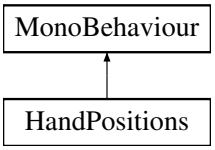
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/GemAnimation.cs

3.18 HandPositions Class Reference

This class handles the player's hands positions (to know if the player is doing rock, paper or scissors) as well as the shifumi game logic.

Inheritance diagram for HandPositions:



Public Member Functions

- void **botPlaysRock** ()

  This method is called when the bot plays rock.
- void **botPlaysPaper** ()

  This method is called when the bot plays paper.
- void **botPlaysScissors** ()

  This method is called when the bot plays scissors.
- void **ResetGame** ()

  This method resets the shifumi game parameters.
- void **pressUseRightHand** ()

  When the button to which this method is attached is pressed, the right hand becomes the hand checked when playing the shifumi game.
- void **pressUseLeftHand** ()

  When the button to which this method is attached is pressed, the left hand becomes the hand checked when playing the shifumi game.
- void **restart** ()

  This method resets the game and restarts the shifumi game.

Public Attributes

- bool **isAlone** = true
- bool **startTimer** = false
- string **resultText** = ""

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- bool CheckScissorsRight ()

  Checks if the right hand of the player is doing scissors.
- bool CheckScissorsLeft ()

  Checks if the left hand of the player is doing scissors.
- bool CheckRockRight ()

  Checks if the right hand of the player is doing rock.

- bool CheckRockLeft ()

  Checks if the left hand of the player is doing rock.
- bool CheckPaperRight ()

  Checks if the right hand of the player is doing paper.
- bool CheckPaperLeft ()

  Checks if the left hand of the player is doing paper.
- bool CheckAngle (float angle, float target, float marginOfAngle)

  Checks if an angle is close enough to a target by a given margin.
- int CheckResult (bool rockR, bool paperR, bool scissorsR, bool rockL, bool paperL, bool scissorsL, bool isRightHand)

  Checks if the player won the shifumi round.
- void **DesactiveAllResultImages** ()

  Desactivate all the result images displayed on the screen.
- void **pressCheckResult** ()

  Whenever this method is called, the hands positions and the result of the round are going to be checked.
- void UpdateScores (int result)

  This method updates the score of the shifumi game.
- void **removeImages** ()

  Desactivate all the position images displayed on the screen.
- void **showBotImage** ()

  Displays the image that corresponds to the hand position of the bot.
- void **StartTimer** ()

  This method starts the timer (going from 3 to 0) for the shifumi game.

Private Attributes

- SoundManager **soundManager**
- MenuLogic **menu**
- GameObject **r_thumbGO**
- GameObject **r_indexGO**
- GameObject **r_middleGO**
- GameObject **r_ringGO**
- GameObject **r_littleGO**
- GameObject **l_thumbGO**
- GameObject **l_indexGO**
- GameObject **l_middleGO**
- GameObject **l_ringGO**
- GameObject **l_littleGO**
- TextMeshProUGUI **t1**
- TextMeshProUGUI **t2**
- TextMeshProUGUI **t3**
- TextMeshProUGUI **t4**
- TextMeshProUGUI **t5**
- TextMeshProUGUI **t12**
- TextMeshProUGUI **t22**
- TextMeshProUGUI **t32**
- TextMeshProUGUI **t42**
- TextMeshProUGUI **t52**
- TextMeshProUGUI **tScissors**
- TextMeshProUGUI **tRock**
- TextMeshProUGUI **tPaper**
- GameObject **r_RockGO**

- GameObject **r_PaperGO**
- GameObject **r_ScissorsGO**
- GameObject **l_RockGO**
- GameObject **l_PaperGO**
- GameObject **l_ScissorsGO**
- GameObject **r_CrossGO**
- GameObject **l_CrossGO**
- GameObject **r_CheckGO**
- GameObject **l_CheckGO**
- GameObject **r_MinusGO**
- GameObject **l_MinusGO**
- TextMeshProUGUI **yourScoreText**
- TextMeshProUGUI **opponentScoreText**
- TextMeshProUGUI **timerText**
- Button **useRightHandButton**
- Button **useLeftHandButton**
- Color **colorUsed**
- Color **colorNotUsed**
- Color **colorHighlighted**
- AvatarMovements **avatarMovements**
- Button **startButton**
- GameObject **victoryGO**
- GameObject **defeatGO**
- GameObject **handPositionWarningGO**
- bool **botRock** = false
- bool **botPaper** = false
- bool **botScissors** = false
- int **botChoice** = -1
- bool **checkResult** = false
- int **result** = 0
- bool **isRightHand** = true
- float **timer** = 0f
- bool **waitBeforeCheck** = false
- float **waitTimer** = 0f
- bool **waitAfterCheck** = false
- float **waitAfterTimer** = 0f
- int **yourScore** = 0
- int **opponentScore** = 0

3.18.1 Detailed Description

This class handles the player's hands positions (to know if the player is doing rock, paper or scissors) as well as the shifumi game logic.

3.18.2 Member Function Documentation

3.18.2.1 CheckAngle()

```
bool HandPositions.CheckAngle (
            float angle,
            float target,
            float marginOfAngle )  [private]
```

Checks if an angle is close enough to a target by a given margin.

Parameters

angle	The angle to check.
target	The target angle.
marginOfAngle	The margin of error.

Returns

true if the angle is close enough to the target, false otherwise.

3.18.2.2 CheckPaperLeft()

```
bool HandPositions.CheckPaperLeft ( )  [private]
```

Checks if the left hand of the player is doing paper.

Returns

true if the player's left hand is doing paper and false otherwise.

3.18.2.3 CheckPaperRight()

```
bool HandPositions.CheckPaperRight ( )  [private]
```

Checks if the right hand of the player is doing paper.

Returns

true if the player's right hand is doing paper and false otherwise.

3.18.2.4 CheckResult()

```
int HandPositions.CheckResult (
            bool rockR,
            bool paperR,
            bool scissorsR,
            bool rockL,
            bool paperL,
            bool scissorsL,
            bool isRightHand )  [private]
```

Checks if the player won the shifumi round.

Parameters

rockR	true if rock with right hand was the position done, false otherwise.
paperR	true if paper with right hand was the position done, false otherwise.
scissorsR	true if scissors with right hand was the position done, false otherwise.
rockL	true if rock with left hand was the position done, false otherwise.
paperL	true if paper with left hand was the position done, false otherwise.
scissorsL	true if scissors with left hand was the position done, false otherwise.
isRightHand	true the hand used by the player is the right hand, false if it's the left hand.

Returns

-1 if the player lost, 1 if the player won.

3.18.2.5 CheckRockLeft()

```
bool HandPositions.CheckRockLeft ( )  [private]
```

Checks if the left hand of the player is doing rock.

Returns

true if the player's left hand is doing rock and false otherwise.

3.18.2.6 CheckRockRight()

```
bool HandPositions.CheckRockRight ( )  [private]
```

Checks if the right hand of the player is doing rock.

Returns

true if the player's right hand is doing rock and false otherwise.

3.18.2.7 CheckScissorsLeft()

```
bool HandPositions.CheckScissorsLeft ( )  [private]
```

Checks if the left hand of the player is doing scissors.

Returns

true if the player's left hand is doing scissors and false otherwise.

3.18.2.8 CheckScissorsRight()

```
bool HandPositions.CheckScissorsRight ( )  [private]
```

Checks if the right hand of the player is doing scissors.

Returns

true if the player's right hand is doing scissors and false otherwise.

3.18.2.9 UpdateScores()

```
void HandPositions.UpdateScores (
            int result )  [private]
```

This method updates the score of the shifumi game.

Parameters

result	The result of the round (-1 == Lose / 0 == Even / 1 == Win).

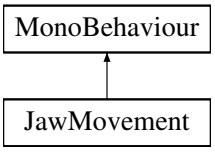
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/ShifumiGame/HandPositions.cs

3.19 JawMovement Class Reference

The following script sed to move the jaw of an avatar when he is talking.

Inheritance diagram for JawMovement:



Public Member Functions

- void setJawGO (GameObject newJaw)

  This method sets the new jaw to animate.
- void **toggleIsTalking** ()

  This method toggles if the jaw is animated or not.

Public Attributes

- bool **isTalking** = false

Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **jawMovement** ()

  This method handles the jaw movements.
- void **openJaw** ()

  When this method is called, the jaw of the avatar opens.
- void **closeJaw** ()

  When this method is called, the jaw of the avatar closes.

Private Attributes

- GameObject **jaw**
- Vector3 **closeJawPos**
- Vector3 **openJawPos**
- float **desiredDuration** = 0.5f
- AnimationCurve **curve**
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startPosTab** = new List<Vector3>()
- List< Vector3 > **endPosTab** = new List<Vector3>()
- bool **closed** = true
- bool **doneSetup** = false

3.19.1 Detailed Description

The following script sed to move the jaw of an avatar when he is talking.

3.19.2 Member Function Documentation

3.19.2.1 setJawGO()

```
void JawMovement.setJawGO (
            GameObject newJaw )
```

This method sets the new jaw to animate.

Parameters

newJaw	The new jaw to animate.

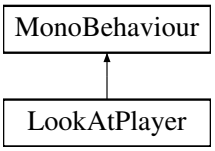
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/JawMovement.cs

3.20 LookAtPlayer Class Reference

The following script is used to make the gameObject look at the player.

Inheritance diagram for LookAtPlayer:



Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- GameObject **playerGO**

3.20.1 Detailed Description

The following script is used to make the gameObject look at the player.

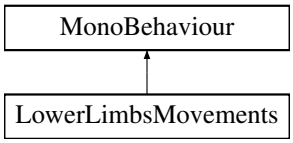
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/LookAtPlayer.cs

3.21 LowerLimbsMovements Class Reference

The following script moves the lower limbs (hips, knees, foot) of the player's avatar accordingly with the data received by the exoskeleton.

Inheritance diagram for LowerLimbsMovements:



Public Attributes

- bool **leftFootOnGround**
- bool **rightFootOnGround**
- float **dirAngle** = 0f

Private Member Functions

- void Start ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- float **UpdateCells** (int i, char letter)

  This method is used to update the soles cells values.

Private Attributes

- TextMeshProUGUI **reductionfactortext**
- TextMeshProUGUI **heightoffsettext**
- TextMeshProUGUI **forwardoffsettext**
- TextMeshProUGUI **rForceXText**
- TextMeshProUGUI **rForceYText**
- TextMeshProUGUI **lForceXText**
- TextMeshProUGUI **lForceYText**
- float **REDUCTION_FACTOR** = 1200f
- GameObject **theClient**
- float **heightOffset**
- float **forwardOffset**
- Transform **leftFootTransform**
- Transform **rightFootTransform**
- Transform **hipTransform**
- float **footOnGroundThreshold**
- GameObject **feetOnGroundColorGO**
- GameObject **balanceSphere**
- GameObject[ ] **leftCells**
- GameObject[ ] **rightCells**
- Vector3 **hipInitPos**
- Vector3 **rFootInitPos**
- Vector3 **lFootInitPos**
- float **leftAbd**
- float **rightAbd**
- float **leftHip**
- float **rightHip**
- float **leftKnee**
- float **rightKnee**
- float **leftFoot**
- float **rightFoot**
- ClientSide **clientSide**
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- SyncVar **activatedCells**
- SyncVar[ ] **svLeft** = new SyncVar[8]
- SyncVar[ ] **svRight** = new SyncVar[8]
- float **left_force_total** = 0
- float **right_force_total** = 0
- bool **lastFootOnGroundIsLeft** = false
- Vector3 **lastLeftPos** = new Vector3(0f, 0f, 0f)
- Vector3 **lastRightPos** = new Vector3(0f, 0f, 0f)
- bool **isMirror** = true
- float **l0**
- float **l1**
- float **l2**
- float **l3**
- float **l4**
- float **l_abd_angle** = -8.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_abd_angle** = 0

- float **l_hip_angle** = 40.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_hip_angle** = -30.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **l_knee_angle** = 60.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_knee_angle** = 10.0f / 360.0f ∗ 2 ∗ Mathf.PI
- bool **done** = false

Static Private Attributes

- const int **SATURATION_THRESHOLD** = 200

3.21.1 Detailed Description

The following script moves the lower limbs (hips, knees, foot) of the player's avatar accordingly with the data received by the exoskeleton.

3.21.2 Member Function Documentation

3.21.2.1 Start()

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

Start method is called before the first frame update and is used to setup what is needed at the start of the App.

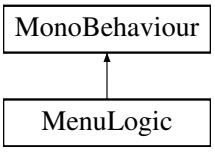
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/LowerLimbsMovements.↩ cs

3.22 MenuLogic Class Reference

This class contains the logic of the menu. All the buttons logic and all the transitions are handled here.

Inheritance diagram for MenuLogic:



Public Member Functions

- void **PressWelcomeContinueButton** ()

  This method is executed when the ¨Continue¨ button of the first menu tab is pressed. It then goes to the language selection tab.
- void PressLanguageButton (int language)

  This method is executed when a language is selected in the language tab of the menu. It then goes to the connection tab.
- void **PressGamesButton** ()

  This method is executed when the games button of the bottom menu is pressed. It then goes to the games tab.
- void **PressConnexionTabButton** ()

  When a button with this function attached is pressed, it goes to the connection tab of the menu. It has the same purpose as the ¨PressLanguageButton(int language)¨ method but it used when the language selection is skipped.
- void **PressControllerTabButton** ()

  This method goes to the controller tab of the menu. This method is not used anymore since the controller is now on a side screen.
- void **ConnectedToTheExo** ()

  This method is called when the connection with the exo is done. It then goes to the calibration tab.
- void **PressLeftCellsZero** ()

  This method is called when the Zero left cells calibration button of the calibration tab is pressed. It calibrates the left cells of the exo and then shows the right cells calibration tab.
- void **PressRightCellsZero** ()

  This method is called when the Zero right cells calibration button of the calibration tab is pressed. It calibrates the right cells of the exo and then goes to the next calibration tab.
- void **PressDoneAbdZero** ()

  This method is called when the other parts of the exo (motor armed/zero abduction/zero joints) are calibrated. It then goes to the avatar calibration tab.
- void **PressDoneCalibration** ()

  This method is called when the avatar is calibrated. It then goes to the games tab.
- void **PressDanceGameButton** ()

  This method is called when dance game button of the games tab is pressed. It then setup and goes to the dance game. This method is also used to reset the game and go back to the beginning of the game.
- void **PressBackToMenuButton** ()

  This method is called the Menu button is pressed It resets everything and go back to the games tab.
- void **PressPlayDanceGameButton** ()

  This method is called when the play button of the dance game is pressed. It starts the dance game.
- void **ToggleMirrorAvatarButton** ()

  This method is called when the mirror avatar toggle button is pressed. If the button is On, the avatar to copy is going to do the movements to copy in a mirrored way. If the button is Off, the avatar to copy is going to do the movements to copy in an anatomical way.
- void **SetGameFinishedTab** ()

  This method is called when the dance game is finished. It then shows the results tab.
- void **PressDanceGameSeeStatsButton** ()

  This method is called when See Statistics button is pressed. It then shows the statistics tab.
- void **PressDanceGamedifficultyButton** ()

  This method is called when the difficulty button is pressed. It then goes to the difficulty tab where the player can select the difficulty for the dance game.
- void **PressWIPButton** ()

  This method is called when the WIP button of the games tab is pressed. It then goes to the Walking in place landscape. This place is not a game but a place to try and test freely the different walking modes.
- void **PressBeatHandsGameButton** ()

  This method is called when the BeatHands button of the games tab is pressed. It then goes to the BeatHands game. This method is also used to reset the game and go back to the beginning of the game.
- void **PressBeatHandsGamedifficultyButton** ()

This method is called when the difficulty button is pressed. It then goes to the difficulty tab where the player can select the difficulty for the BeatHands game.

- void **PressPlayBeatHandsGameButton** ()

  This method is called when the play button of the BeatHands game is pressed. It starts the BeatHands game.

- void **BeatHandsSetGameFinishedTab** ()

  This method is called when the beatHands game is finished. It then shows the results tab.

- void **PressShifumiButton** ()

  This method is called when the shifumi game button of the games tab is pressed. It then goes to the shifumi game.

- void **PressBalanceGameButton** ()

  This method is called when the balance game button of the games tab is pressed. It then goes to the balance game.

- void **ToggleControllerTab** ()

  This method makes the controller tab visible or invisible based on the current state.

- void **BalanceGamePressPlay** (GameObjectController sphere)

  This method is called when the play button of the Balance game is pressed. It starts the Balance game.

Parameters

sphere	*The sphere used in the balance game.*

- void **PressStoryGameButton** ()

  This method is called when the story game button of the games tab is pressed. It then goes to the story game and starts the game.

- void **StoryGameGoToDanceGame** ()

  This method is called when the dance game is automatically launched in the story game. It then goes to the dance game..

- void **StoryGameGoToBeatHandsGame** ()

  This method is called when the BeatHands game is automatically launched in the story game. It then goes to the BeatHands game.

- void **StoryGameGoToBalanceGame** ()

  This method is called when the first Balance game is automatically launched in the story game. It then goes to the first Balance game.

- void **StoryGameBalanceGamePressPlay** (GameObjectController sphere)

  This method is called when the play button of the first Balance game is pressed inside the story game. It starts the first Balance game.

Parameters

sphere	*The sphere used in the first balance game.*

- void **StoryGameBalanceGamePressPlay2** (GameObjectController sphere)

  This method is called when the play button of the second Balance game is pressed inside the story game. It starts the second Balance game.

Parameters

sphere	*The sphere used in the second balance game.*

- void **StoryGameBalanceGamePressFinish** ()

  This method is called when the Balance game is finished in the story game.

- void **StoryGameGoToBalanceGame2** ()

  This method is called when the second Balance game is automatically launched in the story game. It then goes to the second Balance game.

- void **StoryGameGoToShifumiGame** ()

  This method is called when the shifumi game is automatically launched in the story game.

- void **StoryGameShifumiGamePressFinish** ()

  This method is called when the shifumi game is finished in the story game.

Public Attributes

- int **language** = 0
- bool **isConnected** = false

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **changeActualTab** (GameObject newTab)

  This method is used to change the tab of the menu.

Parameters

newTab	*The new tab to be showed in the menu.*

Private Attributes

- GameObject **player**
- Transform **initPlayerTransform**
- Spawner **spawner**
- SoundManager **soundManager**
- Material **danceGameSkyMat**
- Material **menuSkyMat**
- GameObject **menuTabGO**
- GameObject **controllerTabGO**
- GameObject **menuInitPosGO**
- GameObject **menuDanceGamePosGO**
- GameObject **controllerInitPosGO**
- GameObject **controllerDanceGamePosGO**
- GameObject **WIPlandscape**
- GameObject **danceGameLandscape**
- GameObject **menuLandscape**
- GameObject **BeatHandsLandscape**
- GameObject **storyGameLandscape**
- GameObject **languageTab**
- GameObject **connexionTab**
- GameObject **connexionTabFR**
- GameObject **controllerTab**
- GameObject **welcomeTab**
- GameObject **welcomeTabFR**
- GameObject **menuTab**
- GameObject **gamesTab**
- GameObject **gamesTabFR**
- GameObject **danceGameTab**

- GameObject **danceGameTabFR**
- GameObject **danceGameDifficultyTab**
- GameObject **danceGameDifficultyTabFR**
- GameObject **danceGameFinishedTab**
- GameObject **danceGameFinishedTabFR**
- GameObject **danceGameInGameTab**
- GameObject **danceGameInGameTabFR**
- GameObject **danceGameStatsTab**
- GameObject **danceGameStatsTabFR**
- GameObject **beatHandsGameTab**
- GameObject **beatHandsGameTabFR**
- GameObject **beatHandsGameDifficultyTab**
- GameObject **beatHandsGameDifficultyTabFR**
- GameObject **beatHandsGameInGameTab**
- GameObject **beatHandsGameInGameTabFR**
- GameObject **beatHandsGameFinishedTab**
- GameObject **beatHandsGameFinishedTabFR**
- GameObject **calibrationTab**
- GameObject **calibrationTabFR**
- GameObject **WIPTab**
- GameObject **shifumiTab**
- GameObject **shifumiTabFR**
- GameObject **backgroundTab**
- GameObject **batteriesTab**
- GameObject **storyGameTab**
- GameObject **storyGameTabFR**
- GameObject **storyGameDanceGameTab**
- GameObject **storyGameDanceGameTabFR**
- GameObject **storyGameBeatHandsGameTab**
- GameObject **storyGameBeatHandsGameTabFR**
- GameObject **storyGameBalanceGameTab**
- GameObject **storyGameBalanceGameTabFR**
- GameObject **storyGameBalanceGame2Tab**
- GameObject **storyGameBalanceGame2TabFR**
- GameObject **storyGameShifumiGame1Tab**
- GameObject **storyGameShifumiGame1TabFR**
- GameObject **balanceGameTab**
- GameObject **balanceGameTabFR**
- Button **firstButton**
- Button **connectionButton**
- Button **connectionButtonFR**
- Button **EngButton**
- Button **zeroLeftCellsButton**
- Button **zeroLeftCellsButtonFR**
- Button **zeroRightCellsButton**
- Button **zeroRightCellsButtonFR**
- GameObject **zeroLeftCellsTextGO**
- GameObject **zeroRightCellsTextGO**
- GameObject **zeroLeftCellsTextGOFR**
- GameObject **zeroRightCellsTextGOFR**
- Toggle **armMotorsToggle**
- Button **abdZeroButton**
- Button **jointZeroButton**
- Button **abdZeroButton2**
- Button **doneAbdZeroButton**

- Button **doneCalibrationButton**
- GameObject **howToCalibrateAvatarTextGO**
- Toggle **armMotorsToggleFR**
- Button **abdZeroButtonFR**
- Button **jointZeroButtonFR**
- Button **doneAbdZeroButtonFR**
- Button **doneCalibrationButtonFR**
- GameObject **howToCalibrateAvatarTextGOFR**
- Button **gamesButton**
- Button **danceGameDoneStatsButton**
- Button **danceGameDoneStatsButtonFR**
- Button **danceGameDifficultyButton**
- Button **danceGameDifficultyButtonFR**
- Button **danceGameBackToMenuButton**
- Button **danceGameSeeStatsButton**
- Button **danceGameSeeStatsButtonFR**
- Game1Logic **game1Logic**
- GameObject **danceGameGO**
- GameObject **mirrorAvatar**
- GameObject **avatarToCopy**
- Image **scoreBarImg**
- Image **scoreBarImgOpponent**
- ScoreBarLogic **scoreBarLogic**
- OpponentLogic **opponent**
- TextMeshProUGUI **opponentScoreText**
- TextMeshProUGUI **playerScoreText**
- Button **difficulty0Button**
- Button **difficulty0ButtonFR**
- GameObject **isMirrorCheckMark**
- GameObject **isMirrorCheckMarkFR**
- GameObject **danceGameInitPos**
- GameObject **danceGameStoryGamePos**
- Button **beatHandsGameDifficultyButton**
- Button **beatHandsGameDifficultyButtonFR**
- GameObject **beatHandsGameGO**
- GameObject **rightBracelet**
- GameObject **leftBracelet**
- Button **beatHandsDifficulty0Button**
- Button **beatHandsDifficulty0ButtonFR**
- BeatHandsSpawner **beatHandsSpawner**
- BeatHandsGameLogic **beatHandsGameLogic**
- Button **beatHandsFinishNextButton**
- Button **beatHandsFinishNextButtonFR**
- GameObject **beatHandsGameInitPos**
- GameObject **beatHandsGameStoryGamePos**
- Button **storyGameStartBalanceGameButton**
- Button **storyGameStartBalanceGameButtonFR**
- Button **storyGameStartBalanceGameButton2**
- Button **storyGameStartBalanceGameButton2FR**
- Button **startBalanceGameButton**
- Button **startBalanceGameButtonFR**
- GameObject **balanceGameGO**
- DoorPuzzleLogic **balance1Logic**
- DoorPuzzleLogic **balance2Logic**
- WalkingInPlace **Wip**

- Button **WIPBackToMenuButton**
- Button **shifumiBackToMenuButton**
- Button **shifumiBackToMenuButtonFR**
- Button **storyGameStartShifumiGameButton**
- Button **storyGameStartShifumiGameButtonFR**
- HandPositions **shifumiLogic**
- HandPositions **storyShifumiLogic**
- HandPositions **shifumiLogicFR**
- HandPositions **storyShifumiLogicFR**
- Button **storyGameBackToMenuButton**
- Button **storyGameBackToMenuButtonFR**
- GameObject **storyGameGO**
- Button **storyGameStartDanceGameButton**
- Button **storyGameStartDanceGameButtonFR**
- StoryGameLogic **storyGameLogic**
- Button **storyGameStartBeatHandsGameButton**
- Button **storyGameStartBeatHandsGameButtonFR**
- StoryGameResults **resultsTexts**
- bool **isMirror** = true
- bool **inStoryGame** = false
- GameObject **actualLandscape**
- GameObject **actualTab**

3.22.1 Detailed Description

This class contains the logic of the menu. All the buttons logic and all the transitions are handled here.

3.22.2 Member Function Documentation

3.22.2.1 PressLanguageButton()

```
void MenuLogic.PressLanguageButton (
            int language )
```

This method is executed when a language is selected in the language tab of the menu. It then goes to the connection tab.

Parameters

language	The language selected by the player.

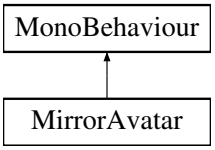
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/MenuLogic.cs

3.23 MirrorAvatar Class Reference

The following script is used to mirror an avatar.

Inheritance diagram for MirrorAvatar:



Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- Transform **parent**
- Vector3 **pos**
- Vector3 **fw**
- Vector3 **up**

3.23.1 Detailed Description

The following script is used to mirror an avatar.

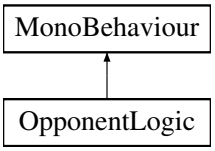
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Mirror↩ Avatar.cs

3.24 OpponentLogic Class Reference

The following script contains the logic of the opponent of the dance game. All the movements of this avatar are done here. A movement is triggered by a boolean designating a specific movement.

Inheritance diagram for OpponentLogic:



Public Member Functions

- void SetDifficulty (int difficulty)

  This method sets the difficulty of the game (the harder is the difficulty the higher the opponent score will be).

Public Attributes

- bool **rightKneeUp** = false
- bool **leftKneeUp** = false
- bool **rightKneeMiddleUp** = false
- bool **leftKneeMiddleUp** = false
- bool **straight** = false
- bool **leftTargetPoint** = false
- bool **rightTargetPoint** = false
- bool **canMove** = false
- bool **isReady** = true
- int **poseNb** = 0

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **moveGameObjects** ()

  This method moves the different part of the opponent avatar when called.
- void goToPose (Vector3[ ] pos, Vector3[ ] angles)

  This method is used to setup the movement (the positions and angles of the different parts of the opponent avatar) done in the ¨MoveGameObjects¨ method.

Private Attributes

- Spawner **spawner**
- GameObject **headGO**
- GameObject **hipGO**
- GameObject **l_handGO**
- GameObject **r_handGO**
- GameObject **r_thumbGO**
- GameObject **r_indexGO**
- GameObject **r_middleGO**
- GameObject **r_ringGO**
- GameObject **r_littleGO**
- GameObject **l_thumbGO**
- GameObject **l_indexGO**
- GameObject **l_middleGO**
- GameObject **l_ringGO**
- GameObject **l_littleGO**
- GameObject **l_footGO**
- GameObject **r_footGO**
- Vector3[ ] **neutralStraightPose**
- Vector3[ ] **neutralStraightAngles**
- Vector3[ ] **leftKneeUpPose**
- Vector3[ ] **leftKneeUpAngles**
- Vector3[ ] **rightKneeUpPose**
- Vector3[ ] **rightKneeUpAngles**
- Vector3[ ] **leftMiddleKneeUpPose**
- Vector3[ ] **leftMiddleKneeUpAngles**
- Vector3[ ] **rightMiddleKneeUpPose**

- Vector3[ ] **rightMiddleKneeUpAngles**
- Vector3[ ] **leftTargetPointPose**
- Vector3[ ] **leftTargetPointAngles**
- Vector3[ ] **rightTargetPointPose**
- Vector3[ ] **rightTargetPointAngles**
- float **desiredDuration** = 2.5f
- AnimationCurve **curve**
- TextMeshProUGUI **opponentScoreText**
- Image **scoreBarImg**
- GameObject[ ] **gameObjects** = new GameObject[16]
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startPosTab** = new List<Vector3>()
- List< Vector3 > **endPosTab** = new List<Vector3>()
- List< Vector3 > **startAngleTab** = new List<Vector3>()
- List< Vector3 > **endAngleTab** = new List<Vector3>()
- List< GameObject > **GOTab** = new List<GameObject>()
- bool **wasStraight** = true
- float **difficultyOffset** = 0f

3.24.1 Detailed Description

The following script contains the logic of the opponent of the dance game. All the movements of this avatar are done here. A movement is triggered by a boolean designating a specific movement.

3.24.2 Member Function Documentation

3.24.2.1 goToPose()

```
void OpponentLogic.goToPose (
            Vector3[] pos,
            Vector3[] angles )  [private]
```

This method is used to setup the movement (the positions and angles of the different parts of the opponent avatar) done in the ¨MoveGameObjects¨ method.

Parameters

pos	The new positions of the different parts of the opponent avatar.
angles	The new angles of the different parts of the opponent avatar.

3.24.2.2 SetDifficulty()

```
void OpponentLogic.SetDifficulty (
            int difficulty )
```

This method sets the difficulty of the game (the harder is the difficulty the higher the opponent score will be).

Parameters

difficulty	The difficulty set for the game.

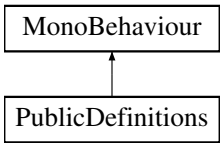
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Opponent↩ Logic.cs

3.25 PublicDefinitions Class Reference

The following script has been done by Euge and is used as is in this project. This script defines the public definitions to communicate with the exoskeleton Autonomyo.

Inheritance diagram for PublicDefinitions:



Public Types

- enum PcToMbMessageType {
  HEARTBEAT = 0 , **SET_DATE** = 1 , **GET_VARS_LIST** = 8 , **GET_VAR** = 9 ,
  SET_VAR = 10 , **SET_VAR_LOG** = 14 , **SET_STREAMING** = 11 , **LOG_MESSAGE** = 12 ,
  SYNC_LED = 13 }

  Remote computer to mainboard message types enumeration.
- enum MbToPcMessageType {
  HEARTBEAT = 0 , **STATUS** , **DEBUG_TEXT** , **VARS_LIST** ,
  VAR_VALUE , **STREAMING** }

  Mainboard to remote computer message types enumeration.
- enum VarType {
  BOOL = 0 , **UINT8** , **INT8** , **UINT16** ,
  INT16 , **UINT32** , **INT32** , **UINT64** ,
  INT64 , **FLOAT32** , **FLOAT64** , **TRIVIAL_COPY** ,
  STRING }

  SyncVar variable types enumeration.
- enum VarAccess { **NONE** = 0 , **READ** , **WRITE** , **READWRITE** }

  SyncVar variable accesses enumeration.
- enum **Modes** {
  MODE_TRANSPARENT , **MODE_UPRIGHT** , **MODE_WALK** , **MODE_STAIRS** ,
  MODE_LUNGES , **MODE_STAND_UP** , **MODE_SIT_DOWN** , **MODE_STS** ,
  MODE__COUNT }

3.25.1 Detailed Description

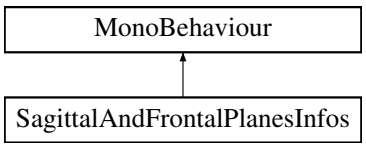
The following script has been done by Euge and is used as is in this project. This script defines the public definitions to communicate with the exoskeleton Autonomyo.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/PublicDefinitions.cs

3.26 SagittalAndFrontalPlanesInfos Class Reference

Inheritance diagram for SagittalAndFrontalPlanesInfos:



Public Member Functions

- void **UpdateTextsFrontal** ()
- void **UpdateTextsSagittal** ()

Private Member Functions

- void **Start** ()
- void **Update** ()
- void **UpdateTextsSoles** ()
- float **UpdateCells** (int i, char letter)

Private Attributes

- float **REDUCTION_FACTOR** = 1200f
- float **SEGMENTWIDTH** = 0.025f
- GameObject **theClient**
- TextMeshProUGUI **abdAngleText**
- TextMeshProUGUI **hipAngleText**
- TextMeshProUGUI **kneeAngleText**
- TextMeshProUGUI **f_backPosText**
- TextMeshProUGUI **f_abdPosText**
- TextMeshProUGUI **f_hipPosText**
- TextMeshProUGUI **f_kneePosText**
- TextMeshProUGUI **f_footPosText**
- TextMeshProUGUI **s_abdPosText**
- TextMeshProUGUI **s_hipPosText**
- TextMeshProUGUI **s_kneePosText**
- TextMeshProUGUI **s_footPosText**
- TextMeshProUGUI **l_soleForce**

- TextMeshProUGUI **r_soleForce**
- TextMeshProUGUI **l_footPosText**
- TextMeshProUGUI **r_footPosText**
- TextMeshProUGUI **hipPosText**
- Transform **leftFootTransform**
- Transform **rightFootTransform**
- Transform **hipTransform**
- float **leftAbd**
- float **rightAbd**
- float **leftHip**
- float **rightHip**
- float **leftKnee**
- float **rightKnee**
- float **leftFoot**
- float **rightFoot**
- SyncVar **activatedCells**
- SyncVar[ ] **svLeft** = new SyncVar[8]
- SyncVar[ ] **svRight** = new SyncVar[8]
- float **left_force_total** = 0
- float **right_force_total** = 0
- ClientSide **clientSide**
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- float **l0**
- float **l1**
- float **l2**
- float **l3**
- float **l4**
- float **l_abd_angle** = -8.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_abd_angle** = 0
- float **l_hip_angle** = 40.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_hip_angle** = -30.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **l_knee_angle** = 60.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_knee_angle** = 10.0f / 360.0f ∗ 2 ∗ Mathf.PI
- bool **done** = false

Static Private Attributes

- const int **SATURATION_THRESHOLD** = 200

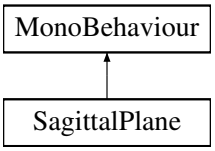
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Sagittal↩ AndFrontalPlanesInfos.cs

3.27 SagittalPlane Class Reference

This script have been done by Euge and is reused as is in this project. This script creates the sagittal plane of the exo based on the abduction, hips and knees angles of the exo.

Inheritance diagram for SagittalPlane:



Public Member Functions

- void **UpdateSegment** (RectTransform rectTransform, RectTransform object1, RectTransform object2)

Public Attributes

- float **l_knee_posx**
- float **l_knee_posz**
- float **r_knee_posx**
- float **r_knee_posz**
- float **l_foot_posx**
- float **l_foot_posz**
- float **r_foot_posx**
- float **r_foot_posz**

Private Member Functions

- void **Start** ()
- void **Update** ()

Private Attributes

- float **REDUCTION_FACTOR** = 10f
- float **SEGMENTWIDTH** = 3f
- GameObject **theClient**
- Vector3 **canvasPositionOffset**
- RectTransform **leftAbdRT**
- RectTransform **rightAbdRT**
- RectTransform **leftHipRT**
- RectTransform **rightHipRT**
- RectTransform **leftKneeRT**
- RectTransform **rightKneeRT**
- RectTransform **leftFootRT**
- RectTransform **rightFootRT**
- RectTransform **leftAH_rt**
- RectTransform **rightAH_rt**
- RectTransform **leftHK_rt**

- RectTransform **rightHK_rt**
- RectTransform **leftKF_rt**
- RectTransform **rightKF_rt**
- float **leftAbd**
- float **rightAbd**
- float **leftHip**
- float **rightHip**
- float **leftKnee**
- float **rightKnee**
- float **leftFoot**
- float **rightFoot**
- ClientSide **clientSide**
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- float **l1**
- float **l2**
- float **l3**
- float **l4**
- float **l_abd_angle** = -8.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_abd_angle** = 0
- float **l_hip_angle** = 40.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_hip_angle** = -30.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **l_knee_angle** = 60.0f / 360.0f ∗ 2 ∗ Mathf.PI
- float **r_knee_angle** = 10.0f / 360.0f ∗ 2 ∗ Mathf.PI
- bool **done** = false

3.27.1 Detailed Description

This script have been done by Euge and is reused as is in this project. This script creates the sagittal plane of the exo based on the abduction, hips and knees angles of the exo.

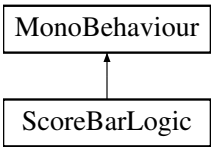
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/SagittalPlane.cs

3.28 ScoreBarLogic Class Reference

The following script contains the logic of the score bar of the player in the Dance Game. It's here that the stars of the score bar are activated as well as the sound design of the score bar.

Inheritance diagram for ScoreBarLogic:



Public Attributes

- bool **done1** = false
- bool **done2** = false
- bool **done3** = false

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

Private Attributes

- Image **fillImage**
- Image **star1**
- Image **star2**
- Image **star3**
- Image **star1Bis**
- Image **star2Bis**
- Image **star3Bis**
- Color **reachedColor**
- Color **notReachedColor**
- SoundManager **soundManager**

3.28.1 Detailed Description

The following script contains the logic of the score bar of the player in the Dance Game. It's here that the stars of the score bar are activated as well as the sound design of the score bar.

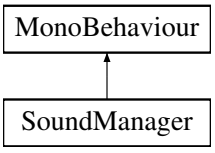
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Score↩ BarLogic.cs

3.29 SoundManager Class Reference

The following script manages all the musics and sounds in the game. Other classes need to call a function from this class to obtain the wanted sound/music.

Inheritance diagram for SoundManager:



Public Member Functions

- void **playHomeMusic** ()

  This method plays the home music.
- void **playGame1Music** ()

  This method plays the dance game music.
- void **playWIPMusic** ()

  This method plays the Story game music.
- void **stopActualMusic** ()

  This method stops the music which is running.
- void **playSuccessSound** ()

  This method plays the success sound.
- void **playCheeringSound** ()

  This method plays the cheering sound.
- void **playStarSound** ()

  This method plays the star sound.
- void **playPlopSound** ()

  This method plays the plop sound.
- void **playIntroAvatarVoice** ()

  This method plays the intro avatar voice.
- void **playStoryV1_1** ()

  This method plays the story 1_1 avatar voice.
- void **playStoryV2_1** ()

  This method plays the story 2_1 avatar voice.
- void **playStoryV2_2** ()

  This method plays the story 2_2 avatar voice.
- void **playStoryV3_1** ()

  This method plays the story 3_1 avatar voice.
- void **playStoryV3_2** ()

  This method plays the story 3_2 avatar voice.
- void **playStoryV4_1** ()

  This method plays the story 4_1 avatar voice.
- void **playStoryV4_2** ()

  This method plays the story 4_2 avatar voice.
- void **playStoryV5_1** ()

  This method plays the story 5_1 avatar voice.
- void **playStoryV5_2** ()

  This method plays the story 5_2 avatar voice.
- void **playStoryV6_1** ()

  This method plays the story 6_1 avatar voice.
- void **playStoryV6_2** ()

  This method plays the story 6_2 avatar voice.
- void **playStoryV7_1** ()

  This method plays the story 7_1 avatar voice.
- void **playStoryV7_2** ()

  This method plays the story 7_2 avatar voice.
- void **playIntroMenu1** ()

  This method plays the intro menu voice 1.
- void **playIntroMenu2** ()

  This method plays the intro menu voice 2.
- void **playIntroMenu3** ()

This method plays the intro menu voice 3.

- void **toggleWIPMusic** ()

  This method toggles the story music.

- void **repeatVoice** ()

  This method repeats last voice.

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.

Private Attributes

- MenuLogic **menu**
- AudioSource **homeMusic**
- AudioSource **game1Music**
- AudioSource **successSound**
- AudioSource **cheeringSound**
- AudioSource **starSound**
- AudioSource **wipMusic**
- AudioSource **natureSounds**
- AudioSource **plopSound**
- AudioSource **storyV1ENG1**
- AudioSource **storyV1FR1**
- AudioSource **storyV2ENG1**
- AudioSource **storyV2ENG2**
- AudioSource **storyV3ENG1**
- AudioSource **storyV3ENG2**
- AudioSource **storyV4ENG1**
- AudioSource **storyV4ENG2**
- AudioSource **storyV5ENG1**
- AudioSource **storyV5ENG2**
- AudioSource **storyV6ENG1**
- AudioSource **storyV6ENG2**
- AudioSource **storyV7ENG1**
- AudioSource **storyV7ENG2**
- AudioSource **storyV2FR1**
- AudioSource **storyV2FR2**
- AudioSource **storyV3FR1**
- AudioSource **storyV3FR2**
- AudioSource **storyV4FR1**
- AudioSource **storyV4FR2**
- AudioSource **storyV5FR1**
- AudioSource **storyV5FR2**
- AudioSource **storyV6FR1**
- AudioSource **storyV6FR2**
- AudioSource **storyV7FR1**
- AudioSource **storyV7FR2**
- AudioSource **introMenuFR1**
- AudioSource **introMenuFR2**
- AudioSource **introMenuFR3**
- AudioSource **introMenuEng1**
- AudioSource **introMenuEng2**
- AudioSource **introMenuEng3**
- AudioSource **actualMusic**
- AudioSource **lastVoiceHeard**

3.29.1 Detailed Description

The following script manages all the musics and sounds in the game. Other classes need to call a function from this class to obtain the wanted sound/music.

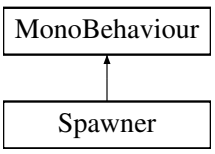
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/SoundManager.cs

3.30 Spawner Class Reference

The following script contains the logic of the spawner of the positions to imitate in the Dance Game (the blue helpers). Whenever a position is cleared by the player, a new one is spawned.

Inheritance diagram for Spawner:



Public Member Functions

- void SetDifficulty (int difficulty)

  This method sets the difficulty of the game (modifying which position can spawn).

Public Attributes

- bool **isMirror** = true

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **SpawnNewPose** ()

  This method is used to spawn a new pose to immitate.
- void **SpawnAfter** ()

  This method activates the pose helpers after timeAfterSpawn seconds.

Private Attributes

- Game1Logic **gameLogic**
- OpponentLogic **opponent**
- GameObject **straightPose**
- GameObject[ ] **mirroredPoses**
- GameObject[ ] **notMirroredPoses**
- Transform[ ] **points**
- float **timeAfterSpawn**
- bool **goBackStraight** = true
- GameObject **GOToActivate**
- List< GameObject > **possiblePositions** = new List<GameObject>()
- List< GameObject > **possiblePositionsNotMirrored** = new List<GameObject>()

3.30.1 Detailed Description

The following script contains the logic of the spawner of the positions to imitate in the Dance Game (the blue helpers). Whenever a position is cleared by the player, a new one is spawned.

3.30.2 Member Function Documentation

3.30.2.1 SetDifficulty()

```
void Spawner.SetDifficulty (
            int difficulty )
```

This method sets the difficulty of the game (modifying which position can spawn).

Parameters

difficulty	The difficulty set for the game.

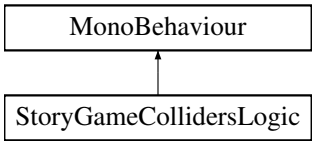
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Spawner.cs

3.31 StoryGameCollidersLogic Class Reference

This class handles the story game colliders used to activate the corresping dialogue as well as the corresping game whenever the player reaches them..

Inheritance diagram for StoryGameCollidersLogic:



Private Member Functions

- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is the player and if it is the case, depending on the collider, the correct dialogue as well as the corresping game are activated.

Private Attributes

- GameObject **danceGameHitBox1**
- GameObject **danceGameHitBox2**
- GameObject **beatHandsGameHitBox1**
- GameObject **beatHandsGameHitBox2**
- GameObject **balanceGameHitBox1**
- GameObject **balanceGameHitBox2**
- StoryGameLogic **gameLogic**
- OVRScreenFade **screenFade**
- WalkingInPlace **walkingInPlace**
- GameObject **victoryGO**

3.31.1 Detailed Description

This class handles the story game colliders used to activate the corresping dialogue as well as the corresping game whenever the player reaches them..

3.31.2 Member Function Documentation

3.31.2.1 OnTriggerEnter()

```
void StoryGameCollidersLogic.OnTriggerEnter (
            Collider other )  [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is the player and if it is the case, depending on the collider, the correct dialogue as well as the corresping game are activated.

Parameters

other	The other collider that hit the collider attached to this object.

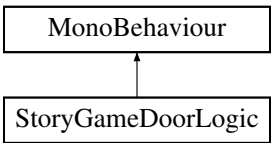
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/StoryGameScripts/StoryGameCollidersLogic.cs

3.32 StoryGameDoorLogic Class Reference

The following script contains the logic of the story game door. It is used to open or close the first house's door when the player walks to it.

Inheritance diagram for StoryGameDoorLogic:



Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is the player and if it is the case, it opens the door.
- void OnTriggerExit (Collider other)

  This method is triggered whenever another collider exits the collider attached to this object. It checks if the other collider is the player and if it is the case, it closes the door.
- void **toggleDoor** ()

  This method is used to open or close the door depending on the door's current state.
- void **doorMovement** ()

  This method handles the door's movements.
- void **openDoor** ()

  When this method is called, the door opens.
- void **closeDoor** ()

  When this method is called, the door closes.

Private Attributes

- GameObject **door**
- float **yAngle**
- float **desiredDuration** = 0.5f
- AnimationCurve **curve**
- bool **isOpen** = false
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startAngleTab** = new List<Vector3>()
- List< Vector3 > **endAngleTab** = new List<Vector3>()

3.32.1 Detailed Description

The following script contains the logic of the story game door. It is used to open or close the first house's door when the player walks to it.

3.32.2 Member Function Documentation

3.32.2.1 OnTriggerEnter()

```
void StoryGameDoorLogic.OnTriggerEnter (
            Collider other ) [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the other collider is the player and if it is the case, it opens the door.

Parameters

other	The other collider that hit the collider attached to this object.

3.32.2.2 OnTriggerExit()

```
void StoryGameDoorLogic.OnTriggerExit (
            Collider other )  [private]
```

This method is triggered whenever another collider exits the collider attached to this object. It checks if the other collider is the player and if it is the case, it closes the door.

Parameters

other	The other collider that hit the collider attached to this object.

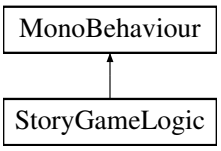
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/StoryGameScripts/StoryGameDoorLogic.cs

3.33 StoryGameLogic Class Reference

The following script contains the logic of the story Game. The launch of the different games, musics, etc... are done here.

Inheritance diagram for StoryGameLogic:



Public Member Functions

- void **RunDanceGame1** ()

  This method prepares what is needed to launch the first dance game in the story game.
- void **RunDanceGame2** ()

  This method prepares what is needed to launch the second dance game in the story game.
- void **RunBeatHandsGame1** ()

  This method prepares what is needed to launch the first BeatHands game in the story game.
- void **RunBeatHandsGame2** ()

  This method prepares what is needed to launch the second Beathands game in the story game.
- void **RunBalanceGame1** ()

  This method prepares what is needed to launch the first balance game in the story game.
- void **RunBalanceGame2** ()

This method prepares what is needed to launch the second balance game in the story game.

- void **RunShifumiGame1** ()

  This method prepares what is needed to launch the first shifumi game in the story game.

- void **RunEnd** ()

  This method terminates the story game and returns to the menu when called.

- void **goToDanceGame** ()

  This method launches the first dance game in the story game.

- void **goToBeatHandsGame** ()

  This method launches the BeatHands game in the story game.

- void **goToBalanceGame1** ()

  This method launches the first balance game in the story game.

- void **goToBalanceGame2** ()

  This method launches the second balance game in the story game.

- void **goToShifumiGame1** ()

  This method launches the shifumi game in the story game.

- void **resetGame** ()

  This method is called to reset the story game.

Public Attributes

- bool **finishedDanceGame1** = false
- bool **finishedDanceGame2** = false
- bool **finishedBeatHandsGame1** = false
- bool **finishedBeatHandsGame2** = false
- bool **finishedBalanceGame1** = false
- bool **finishedBalanceGame2** = false
- bool **finishedShifumiGame1** = false
- int **actualGameNb** = 0

Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.

- void **runIntro** ()

  This method runs the intro of the game. The first NPC talks to the player and explains the goal of the game as well as how to move in the virtual world.

- void **danceGame1Finished** ()

  This method is called whenever the first dance game is finished.

- void **danceGame2Finished** ()

  This method is called whenever the second dance game is finished.

- void **BeatHandsGame1Finished** ()

  This method is called whenever the first BeatHands game is finished.

- void **BeatHandsGame2Finished** ()

  This method is called whenever the second BeatHands game is finished.

- void **BalanceGame1Finished** ()

  This method is called whenever the first balance game is finished.

- void **BalanceGame2Finished** ()

  This method is called whenever the second balance game is finished.

- void **ShifumiGame1Finished** ()

  This method is called whenever the shifumi game is finished.

Private Attributes

- OVRScreenFade **screenFade**
- WalkingInPlace **WIP**
- GameObject **playerGO**
- GameObject **initialSpawnGO**
- MenuLogic **menu**
- SoundManager **soundManager**
- Material **danceGameSkyMat**
- Material **menuSkyMat**
- GameObject **introAvatarGO**
- JawMovement **introAvatarJaw**
- AvatarMovements **introAvatarMovements**
- GameObject **game1AvatarGO**
- JawMovement **game1AvatarJaw**
- AvatarMovements **game1AvatarMovements**
- GameObject **game2AvatarGO**
- JawMovement **game2AvatarJaw**
- AvatarMovements **game2AvatarMovements**
- GameObject **game3AvatarGO**
- JawMovement **game3AvatarJaw**
- AvatarMovements **game3AvatarMovements**
- GameObject **game4AvatarGO**
- JawMovement **game4AvatarJaw**
- AvatarMovements **game4AvatarMovements**
- GameObject **game5AvatarGO**
- JawMovement **game5AvatarJaw**
- AvatarMovements **game5AvatarMovements**
- GameObject **game6AvatarGO**
- JawMovement **game6AvatarJaw**
- AvatarMovements **game6AvatarMovements**
- GameObject **danceGameSpawnerGO**
- Game1Logic **danceGameLogic**
- int **danceGameDifficulty** = 1
- GameObject **beatHandsGameSpawnerGO**
- BeatHandsGameLogic **beathandsGameLogic**
- int **beatHandsGameDifficulty** = 1
- GameObject **balanceGame1SpawnerGO**
- GameObject **balanceGame2SpawnerGO**
- GameObject **danceGame1RespawnGO**
- bool **doneDanceGame1** = false
- GameObject **danceGame2RespawnGO**
- bool **doneDanceGame2** = false
- GameObject **beatHandsGame1RespawnGO**
- bool **doneBeatHandsGame1** = false
- GameObject **beatHandsGame2RespawnGO**
- bool **doneBeatHandsGame2** = false
- GameObject **balanceGame1RespawnGO**
- bool **doneBalanceGame1** = false
- DoorPuzzleLogic **balanceGame1Logic**
- GameObject **balanceGame2RespawnGO**
- bool **doneBalanceGame2** = false
- DoorPuzzleLogic **balanceGame2Logic**
- bool **doneShifumiGame1** = false
- HandPositions **shifumiLogic**
- bool **restart** = true

3.33.1 Detailed Description

The following script contains the logic of the story Game. The launch of the different games, musics, etc... are done here.

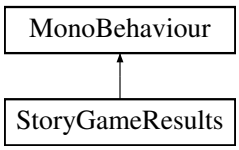
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/StoryGameScripts/StoryGameLogic.cs

3.34 StoryGameResults Class Reference

This class handles the finale results tab dispaying the results done by the player during the story game.

Inheritance diagram for StoryGameResults:



Public Member Functions

- void changeDanceScore1Text (string newText)

  This method changes the text on the finale results tab of the score done during the first dance game.
- void changeBeatHandsScore1Text (string newText)

  This method changes the text on the finale results tab of the score done during the first BeatHands game.
- void changeBalanceScore1Text (string newText)

  This method changes the text on the finale results tab of the score done during the first balance game.
- void changeDanceScore2Text (string newText)

  This method changes the text on the finale results tab of the score done during the second dance game.
- void changeBeatHandsScore2Text (string newText)

  This method changes the text on the finale results tab of the score done during the second BeatHands game.
- void changeBalanceScore2Text (string newText)

  This method changes the text on the finale results tab of the score done during the second balance game.
- void changeShifumiScoreText (string newText)

  This method changes the text on the finale results tab of the score done during the shifumi game.

Private Attributes

- TextMeshProUGUI **danceScore1Text**
- TextMeshProUGUI **beatHandsScore1Text**
- TextMeshProUGUI **balanceScore1Text**
- TextMeshProUGUI **danceScore2Text**
- TextMeshProUGUI **beatHandsScore2Text**
- TextMeshProUGUI **shifumiScoreText**
- TextMeshProUGUI **balanceScore2Text**

3.34.1 Detailed Description

This class handles the finale results tab dispaying the results done by the player during the story game.

3.34.2 Member Function Documentation

3.34.2.1 changeBalanceScore1Text()

```
void StoryGameResults.changeBalanceScore1Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the first balance game.

Parameters

newText	The new text to display.

3.34.2.2 changeBalanceScore2Text()

```
void StoryGameResults.changeBalanceScore2Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the second balance game.

Parameters

newText	The new text to display.

3.34.2.3 changeBeatHandsScore1Text()

```
void StoryGameResults.changeBeatHandsScore1Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the first BeatHands game.

Parameters

newText	The new text to display.

3.34.2.4 changeBeatHandsScore2Text()

```
void StoryGameResults.changeBeatHandsScore2Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the second BeatHands game.

Parameters

newText	The new text to display.

3.34.2.5 changeDanceScore1Text()

```
void StoryGameResults.changeDanceScore1Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the first dance game.

Parameters

newText	The new text to display.

3.34.2.6 changeDanceScore2Text()

```
void StoryGameResults.changeDanceScore2Text (
            string newText )
```

This method changes the text on the finale results tab of the score done during the second dance game.

Parameters

newText	The new text to display.

3.34.2.7 changeShifumiScoreText()

```
void StoryGameResults.changeShifumiScoreText (
            string newText )
```

This method changes the text on the finale results tab of the score done during the shifumi game.

Parameters

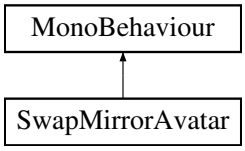
newText	The new text to display.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/StoryGameScripts/StoryGameResults.cs

3.35 SwapMirrorAvatar Class Reference

Inheritance diagram for SwapMirrorAvatar:



Private Member Functions

- void **Update** ()

Private Attributes

- GameObject **mirror1**
- GameObject **mirror2**
- GameObject **mirror3**
- GameObject **avatar1**
- GameObject **avatar2**
- GameObject **avatar3**
- bool **isMirror** = true

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Swap↩ MirrorAvatar.cs

3.36 SyncVar Class Reference

The following script has been done by Euge and is used as is in this project. This script represents the SyncVar which are used to communicate different values between the exoskeleton Autonomyo and the App.

Public Member Functions

- SyncVar **makeBoolSyncVar** (string name, string unit, bool var, VarAccess access, bool logToFile)
- SyncVar **makeIntSyncVar** (string name, string unit, int var, VarAccess access, bool logToFile)
- SyncVar **makeFloatSyncVar** (string name, string unit, float var, VarAccess access, bool logToFile)
- SyncVar **makeStringSyncVar** (string name, string unit, string var, VarAccess access, bool logToFile)

Static Public Attributes

- static int **SYNCVAR_NAME_COMM_LENGTH** = 100

  Length of SyncVar names during remote listing.
- static int **SYNCVAR_UNIT_COMM_LENGTH** = 20

  Length of SyncVar units during remote listing.
- static int **SYNCVAR_LIST_ITEM_SIZE** = (SYNCVAR_NAME_COMM_LENGTH + SYNCVAR_UNIT_COMM_LENGTH + 1 + 1 + 4)

  SyncVar description bytes size during remote listing.
- static char **SYNCVAR_NAME_SEPARATOR** = '/'

  SyncVar prefix separators.

Properties

- string **Name** `[get, set]`
- string **Unit** `[get, set]`
- VarType **Type** `[get, set]`
- VarAccess **Access** `[get, set]`
- bool **LogToFile** `[get, set]`
- uint **NBytes** `[get, set]`
- bool **BoolVar** `[get, set]`
- sbyte **SByteVar** `[get, set]`
- int **IntVar** `[get, set]`
- float **FloatVar** `[get, set]`
- ulong **ULongVar** `[get, set]`
- string **StringVar** `[get, set]`

Private Attributes

- string **sName**
- string **unit**
- VarType **type**
- VarAccess **access**
- bool **logToFile**
- uint **nBytes**
- bool **boolVar**
- sbyte **sByteVar**
- int **intVar**
- float **floatVar**
- ulong **uLongVar**
- string **stringVar**

3.36.1 Detailed Description

The following script has been done by Euge and is used as is in this project. This script represents the SyncVar which are used to communicate different values between the exoskeleton Autonomyo and the App.

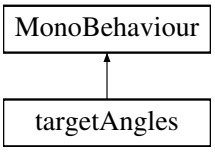
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/SyncVar.cs

3.37 targetAngles Class Reference

The following script represents the target angles of a position for the dance game.

Inheritance diagram for targetAngles:



Public Member Functions

- float[ ] getTargetAngles ()

  This method gives the target angles of the position.

Public Attributes

- string **poseName**

Private Attributes

- float **l_targetAbdAngle**
- float **r_targetAbdAngle**
- float **l_targetHipAngle**
- float **r_targetHipAngle**
- float **l_targetKneeAngle**
- float **r_targetKneeAngle**

3.37.1 Detailed Description

The following script represents the target angles of a position for the dance game.

3.37.2 Member Function Documentation

3.37.2.1 getTargetAngles()

```
float[] targetAngles.getTargetAngles ( )
```

This method gives the target angles of the position.

Returns

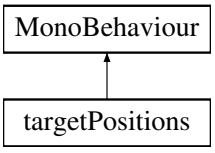
The target angles of this position.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/targetAngles.cs

3.38 targetPositions Class Reference

Inheritance diagram for targetPositions:



Public Member Functions

- Vector3[ ] **getTargetPositions** ()

Private Attributes

- Vector3 **f_l_targetKneePos**
- Vector3 **f_r_targetKneePos**
- Vector3 **f_l_targetFootPos**
- Vector3 **f_r_targetFootPos**
- Vector3 **s_l_targetKneePos**
- Vector3 **s_r_targetKneePos**
- Vector3 **s_l_targetFootPos**
- Vector3 **s_r_targetFootPos**

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/target↩ Positions.cs

3.39 ClientSide.TCP Class Reference

Public Member Functions

- void **StopStream** ()
- void **Connect** ()
- void **GetVarValue** (short varIndex)
- void **SendPacket** (short varIndex, uint varSize, bool b, int value)
- void **SendFloatPacket** (short varIndex, uint varSize, float value)
- void **SendStringPacket** (short varIndex, uint varSize, string str)
- void **SetVariable** (SyncVar sv)
- void **AddStreamedVars** (SyncVar sv)
- void **SetStreamedVars** (int period)

Public Attributes

- TcpClient **socket**

Private Member Functions

- void **OnHeartBeatTimeout** (object source, ElapsedEventArgs e)
- void **ConnectCallback** (IAsyncResult _result)
- void **UpdateText** ()
- void **ListenForData** ()
- void **ReceiveCallback** (IAsyncResult _result)
- void **SendCallback** (IAsyncResult _result)
- void **DecodingData** (byte[ ] _data, int _byteLength)

Private Attributes

- NetworkStream **stream**
- byte[ ] **receiveBuffer**
- byte[ ] **sendBuffer**

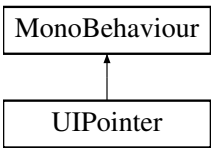
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/ClientSide.cs

3.40 UIPointer Class Reference

The following script handles the laser pointer used to navigate in the interface.

Inheritance diagram for UIPointer:



Public Member Functions

- void **TogglePointer** ()

  This method toggles the laser pointer.

Private Member Functions

- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **UpdateLength** ()

  This method updates the length of the laser.
- Vector3 CalculateEnd ()

  This method calculates where the laser should end.
- RaycastHit CreateFowardRaycast ()

  This method finds what is the first interface object hit by the laser.
- Vector3 DefaultEnd (float length)

  This method gives the position of the default end of the laser.

Private Attributes

- MenuLogic **menu**
- Controller **controller**
- LineRenderer **lineRenderer**
- LayerMask **layersToHit**
- GameObject **circleBarToSpawn**
- GameObject **teleportButtonGO**
- float **defaultLength** = 1000.0f
- GameObject **GOHit**
- GameObject **circleBar**
- float **validationCounter** = 0f
- CircularProgressBar **progressBar**
- float **timeToValidate** = 1.5f
- Button **button**
- Toggle **toggle**
- bool **isActive** = true

3.40.1 Detailed Description

The following script handles the laser pointer used to navigate in the interface.

3.40.2 Member Function Documentation

3.40.2.1 CalculateEnd()

```
Vector3 UIPointer.CalculateEnd ( )  [private]
```

This method calculates where the laser should end.

Returns

The position wherer the laser should end.

3.40.2.2 CreateFowardRaycast()

```
RaycastHit UIPointer.CreateFowardRaycast ( )  [private]
```

This method finds what is the first interface object hit by the laser.

Returns

The RaycastHit of the raycast.

3.40.2.3 DefaultEnd()

```
Vector3 UIPointer.DefaultEnd (
            float length )  [private]
```

This method gives the position of the default end of the laser.

Returns

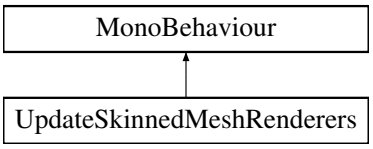
The position of the default end.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/UIPointer.cs

3.41 UpdateSkinnedMeshRenderers Class Reference

Inheritance diagram for UpdateSkinnedMeshRenderers:



Private Member Functions

- void **Update** ()

Private Attributes

- bool **done** = false

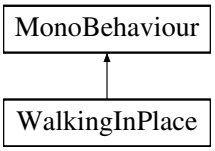
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DebugScripts/Update↩ SkinnedMeshRenderers.cs

3.42 WalkingInPlace Class Reference

The following script contains the logic of the Walking in place (static walking).

Inheritance diagram for WalkingInPlace:



Public Member Functions

- void **toggleWIP** ()

  This method toggles the walking in place.
- void **pressTPButton** ()

  This method is triggered when the player wants to teleport to the desired position.
- void **teleport** ()

  This method is used to teleport the player to the desired position.
- void changeWalkingMode (int newMode)

  This method is used to change the walking mode.

Public Attributes

- bool **WIP** = false
- int **walkingMode** = 1

Private Member Functions

- void **Start** ()

  Start method is called before the first frame update and is used to setup what is needed at the start of the App.
- void **Update** ()

  Update method is called once per frame and is used to update what needs to be updated each frame.
- void **RotatePlayer** ()

  This method rotates the avatar in the direction where the player is looking at.
- void OnTriggerEnter (Collider other)

  This method is triggered whenever another collider enters the collider attached to this object. It checks if the players hits a wall.
- void OnTriggerExit (Collider other)

  This method is triggered whenever another collider exits the collider attached to this object. It checks if the players hits a wall.
- void **movePlayer** (GameObject playerGO)

  This is used to avoid the player to move through the walls.

Private Attributes

- GameObject **camera**
- GameObject **centerEye**
- LowerLimbsMovements **lowerLimbs**
- GameObject **theClient**
- float **marginOfErrorAngles**
- float **marginOfErrorAnglesAbd**
- float[ ] **targetAnglesRight**
- float[ ] **targetAnglesLeft**
- float[ ] **targetAnglesDown**
- GameObject **playerGO**
- GameObject **teleportGO**
- SoundManager **soundManager**
- GameObject **leftFoot**
- GameObject **rightFoot**
- float **desiredDuration** = 2f
- AnimationCurve **curve**

- bool **alwaysStraight** = false
- ClientSide **clientSide**
- SyncVar **leftAbdAngle**
- SyncVar **rightAbdAngle**
- SyncVar **leftHipAngle**
- SyncVar **rightHipAngle**
- SyncVar **leftKneeAngle**
- SyncVar **rightKneeAngle**
- bool **done** = false
- bool **leftLegUp** = false
- bool **rightLegUp** = false
- bool **bothLegsDown** = false
- bool **leftLegUpDone** = false
- bool **rightLegUpDone** = false
- bool **bothLegsDownDone** = true
- bool **movedForward** = false
- List< bool > **doneTab** = new List<bool>()
- List< float > **startTimeTab** = new List<float>()
- List< Vector3 > **startPosTab** = new List<Vector3>()
- List< Vector3 > **endPosTab** = new List<Vector3>()
- float **lastForwardValueRight** = -1f
- float **lastForwardValueLeft** = -1f
- bool **isStraight**
- bool **canGoPositivX** = true
- bool **canGoNegativX** = true
- bool **canGoPositivZ** = true
- bool **canGoNegativZ** = true
- double **diffX** = 0
- double **diffZ** = 0
- bool **TPButtonPressed** = false
- bool **setupDone** = false

3.42.1 Detailed Description

The following script contains the logic of the Walking in place (static walking).

3.42.2 Member Function Documentation

3.42.2.1 changeWalkingMode()

```
void WalkingInPlace.changeWalkingMode (
            int newMode )
```

This method is used to change the walking mode.

Parameters

newMode	The new mode.

3.42.2.2 OnTriggerEnter()

```
void WalkingInPlace.OnTriggerEnter (
            Collider other )  [private]
```

This method is triggered whenever another collider enters the collider attached to this object. It checks if the players hits a wall.

Parameters

other	The other collider that hit the collider attached to this object.

3.42.2.3 OnTriggerExit()

```
void WalkingInPlace.OnTriggerExit (
            Collider other )  [private]
```

This method is triggered whenever another collider exits the collider attached to this object. It checks if the players hits a wall.

Parameters

other	The other collider that hit the collider attached to this object.

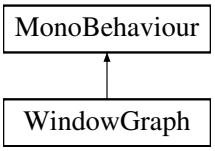
The documentation for this class was generated from the following file:

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/WalkingInPlace.cs

3.43 WindowGraph Class Reference

The following script is used to dispay a graph showing the time done for each position in the dance game by the player.

Inheritance diagram for WindowGraph:



Public Member Functions

- void showGraph (List< float > valueList)

  This method is to display the graph given the list of the values to display.

Private Member Functions

- GameObject createCircle (Vector2 anchoredPosition)

  This method is used to create a point on the graph at the correct position.
- void createDotConnection (Vector2 dotPositionA, Vector2 dotPositionB)

  This method creates a line between two points on the graph.
- void **destroyLastGraph** ()

  This method destroys the last graph.

Private Attributes

- Sprite **circleSprite**
- TextMeshProUGUI **maxYText**
- TextMeshProUGUI **avgText**
- RectTransform **graphContainer**
- MenuLogic **menu**
- List< GameObject > **toDestroy** = new List<GameObject>()

3.43.1 Detailed Description

The following script is used to dispay a graph showing the time done for each position in the dance game by the player.

3.43.2 Member Function Documentation

3.43.2.1 createCircle()

```
GameObject WindowGraph.createCircle (
            Vector2 anchoredPosition )  [private]
```

This method is used to create a point on the graph at the correct position.

Parameters

anchoredPosition	The position where to create the circle on the graph.

3.43.2.2 createDotConnection()

```
void WindowGraph.createDotConnection (
            Vector2 dotPositionA,
            Vector2 dotPositionB )  [private]
```

This method creates a line between two points on the graph.

Parameters

dotPositionA	The position of the first point.
dotPositionB	The position of the second point.

3.43.2.3 showGraph()

```
void WindowGraph.showGraph (
            List< float > valueList )
```

This method is to display the graph given the list of the values to display.

Parameters

valueList	The list containing the values to put on the graph.

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

- C:/Users/nicov/Nicolas-Vial-internship-Autonomyo/AutonomyoProject/Assets/Scripts/DanceGameScripts/Window↩ Graph.cs

Index