Thesis_V2

Generated by Doxygen 1.9.3

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 AccountManager Class Reference

Class responsible to manage the interaction with the database and the validation of the connections with the server. The maximum amount of points a user can have is stored in constant maxPointsPossible. The string that is used during the clientside salting is stored in constant sharedSalt. The number of iterations for the secure hashing algorithm is stored in constant secureHashIterations. The size in bytes of the hashed user password is stored in constant hashSize. The Encoding that has been chosen is stored in the readonly field currentEncoding. The separator in the database file is stored in constant separator. The relative path to the database folder is stored in constant relativeDatabasePath. The path to the database file is stored in constant databaseFile.

Inherits NetworkBehaviour.

Public Member Functions

- void SetNewAccount (bool value)

  External function to change the status of the next connection request, from known credentials to new credentials and viceversa. The function is public void and value parametrized on purpose so that it could be called from a toggle OnClick.
- void **Submit** ()

  Function to start a client and submit a login request. Checks username and password requirements on client side, If they are ok, the password is salted and hashed and then sent to the server to check if it is a valid login. The server will validate the sent data and accept or refuse the connection accordingly. The function is public void and parameterless on purpose so that it an be call from a button OnClick.
- void StorePoints (string username, ushort points)

  Function to update in the database the points of the given username. To facilitate the overwriting of the file without having to read all of it in memory, we store the points as a fixed length string padded with leading zeros.

Public Attributes

- TMP_InputField **username**
- TMP_InputField **password**
- bool **newAccount** = false
- NetworkWrapper **NW**
- MySceneManager **MSM**

3.1.1 Detailed Description

Class responsible to manage the interaction with the database and the validation of the connections with the server. The maximum amount of points a user can have is stored in constant maxPointsPossible. The string that is used during the clientside salting is stored in constant sharedSalt. The number of iterations for the secure hashing algorithm is stored in constant secureHashIterations. The size in bytes of the hashed user password is stored in constant hashSize. The Encoding that has been chosen is stored in the readonly field currentEncoding. The separator in the database file is stored in constant separator. The relative path to the database folder is stored in constant relativeDatabasePath. The path to the database file is stored in constant databaseFile.

3.1.2 Member Function Documentation

3.1.2.1 SetNewAccount()

```
void AccountManager.SetNewAccount (
            bool value )
```

External function to change the status of the next connection request, from known credentials to new credentials and viceversa. The function is public void and value parametrized on purpose so that it could be called from a toggle OnClick.

Parameters

value	true if the credentials should be considered new, false if the credentials should be considered known.

3.1.2.2 StorePoints()

```
void AccountManager.StorePoints (
            string username,
            ushort points )
```

Function to update in the database the points of the given username. To facilitate the overwriting of the file without having to read all of it in memory, we store the points as a fixed length string padded with leading zeros.

Parameters

username	Plaintext username.
points	Amount of points to be saved in the database.

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

- Assets/Scripts/Persistency/AccountManager.cs

3.2 AvatarManager Class Reference

Class responsible for spawning and managing the Avatars. See AvatarUI.

Inherits MonoBehaviour.

Public Member Functions

- void SelectThis (AvatarUI selectedAvatar)

  Function used by the AvatarUI class to change the color of the background panel of all displayed AvatarUI.

Public Attributes

- GameObject **avatarPrefab**
- RectTransform **avatarHolder**
- LocalizeStringEvent **currentPointsText**
- ushort **points**

3.2.1 Detailed Description

Class responsible for spawning and managing the Avatars. See AvatarUI.

3.2.2 Member Function Documentation

3.2.2.1 SelectThis()

```
void AvatarManager.SelectThis (
            AvatarUI selectedAvatar )
```

Function used by the AvatarUI class to change the color of the background panel of all displayed AvatarUI.

Parameters

selectedAvatar	The Avatar calling the function, the one that should be selected.

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

- Assets/Scripts/Avatars/AvatarManager.cs

3.3 AvatarUI Class Reference

Prefab class managing the details of an Avatar and exposing the functions to interact with it.

Inherits MonoBehaviour.

Public Member Functions

- void Setup (AvatarManager AM, string spriteName, int pointsThreshold)

  Function to initialize an Avatar prefab, it takes care of the visal UI and button OnClick delegate.
- void SetPanelColor (Color c)

  Utility function to change the color of the background to a given Color.

Public Attributes

- bool **isSceneObject**
- Image **backgroundPanel**
- Button **selectionButton**
- Image **spriteArea**
- string **points** = "100"
- LocalizeStringEvent **avatarInfoText**

3.3.1 Detailed Description

Prefab class managing the details of an Avatar and exposing the functions to interact with it.

3.3.2 Member Function Documentation

3.3.2.1 SetPanelColor()

```
void AvatarUI.SetPanelColor (
            Color c )
```

Utility function to change the color of the background to a given Color.

Parameters

c	Color to change the background to.

3.3.2.2 Setup()

```
void AvatarUI.Setup (
            AvatarManager AM,
            string spriteName,
            int pointsThreshold )
```

Function to initialize an Avatar prefab, it takes care of the visal UI and button OnClick delegate.

Parameters

AM	AvatarManager that spawned the prefab.
spriteName	Name of the avatar.
pointsThreshold	Points that the user currently has.

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

- Assets/Scripts/Avatars/AvatarUI.cs

3.4 codeQuestion Struct Reference

Struct representing the data associated with a codeQuestion. Because of its arbitrarely large nature, the properties are string so they CANNOT be sent over the network in a Rpc (packed in a struct). This struct's size is unknown at compile time.

Public Member Functions

- **codeQuestion** (string n, string d, string c, string[ ] t)

Public Attributes

- string **name**
- string **description**
- string **content**
- string[ ] **tags**

3.4.1 Detailed Description

Struct representing the data associated with a codeQuestion. Because of its arbitrarely large nature, the properties are string so they CANNOT be sent over the network in a Rpc (packed in a struct). This struct's size is unknown at compile time.

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

- Assets/Scripts/Statics/RequiredStructs.cs

3.5 CodeQuestionManager Class Reference

Class responsible for spawning and managing the codeQuestions. See CodeQuestionUI.

Inherits MonoBehaviour.

Public Member Functions

- void DisplayFilteredView (string value)

  Function called externally to display the spawned codeQuestions according to the filter on the tags. The function is public void and with parameter value on purpose so that it could be called on an input field change.
- void SelectThis (CodeQuestionUI selectedCodeQuestion)

  External function called by CodeQuestionUI. Triggers the background change of the caller into "selected", while changing all the others into the defaul color.

Public Attributes

- GameObject **codeQuestionPrefab**
- RectTransform **codeQuestionHolder**

3.5.1 Detailed Description

Class responsible for spawning and managing the codeQuestions. See CodeQuestionUI.

3.5.2 Member Function Documentation

3.5.2.1 DisplayFilteredView()

```
void CodeQuestionManager.DisplayFilteredView (
            string value )
```

Function called externally to display the spawned codeQuestions according to the filter on the tags. The function is public void and with parameter value on purpose so that it could be called on an input field change.

Parameters

value	The tags to filter by.

3.5.2.2 SelectThis()

```
void CodeQuestionManager.SelectThis (
            CodeQuestionUI selectedCodeQuestion )
```

External function called by CodeQuestionUI. Triggers the background change of the caller into "selected", while changing all the others into the defaul color.

Parameters

selectedCodeQuestion	The caller of function that wants to change its background color.

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

- Assets/Scripts/CodeQuestions/CodeQuestionManager.cs

3.6 CodeQuestionUI Class Reference

Prefab class managing the details of a codeQuestion and exposing the functions to interact with it.

Inherits MonoBehaviour.

Public Member Functions

- void Setup (CodeQuestionManager CQM, codeQuestion myQuestion, bool spawnSelected)

  Function to initialize a codeQuestion prefab, it takes care of the UI visuals and button OnClick delegate.
- void SetPanelColor (Color c)

  Utility function to change the color of the background to a given Color.

Public Attributes

- string **myCodeQuestionName**
- Image **backgroundPanel**
- Button **selectionButton**
- TextMeshProUGUI **nameText**
- TextMeshProUGUI **tagsText**

3.6.1 Detailed Description

Prefab class managing the details of a codeQuestion and exposing the functions to interact with it.

3.6.2 Member Function Documentation

3.6.2.1 SetPanelColor()

```
void CodeQuestionUI.SetPanelColor (
            Color c )
```

Utility function to change the color of the background to a given Color.

Parameters

c	Color to change the background to.

3.6.2.2 Setup()

```
void CodeQuestionUI.Setup (
            CodeQuestionManager CQM,
            codeQuestion myQuestion,
            bool spawnSelected )
```

Function to initialize a codeQuestion prefab, it takes care of the UI visuals and button OnClick delegate.

Parameters

CQM	CodeQuestionManager that spawned the prefab..
myQuestion	codeQuestion associated with this prefab.
spawnSelected	boolean to decide if the background color should be selected or selectable.

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

- Assets/Scripts/CodeQuestions/CodeQuestionUI.cs

3.7 databaseEntry Struct Reference

Struct representing the data associated with a user. Because of username and avatar being FixedString32Bytes, the whole struct CAN be sent over the network in a Rpc. This struct's size is fixed at: 32 + 2 + 2 + 32 + 8 = 76 bytes.

Public Member Functions

- **databaseEntry** (FixedString32Bytes u, ushort p, ushort s, FixedString32Bytes a, ulong o)
- **databaseEntry** (ushort p)
- **databaseEntry** (databaseEntry d)

Public Attributes

- FixedString32Bytes **username**
- ushort **progress**
- ushort **points**
- FixedString32Bytes **avatar**
- ulong **owner**

3.7.1 Detailed Description

Struct representing the data associated with a user. Because of username and avatar being FixedString32Bytes, the whole struct CAN be sent over the network in a Rpc. This struct's size is fixed at: 32 + 2 + 2 + 32 + 8 = 76 bytes.

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

- Assets/Scripts/Statics/RequiredStructs.cs

3.8 DisconnectionManager Class Reference

The DisconnectionManager handles the disconnection callback of clients in the general sense. It also exposes a public void parameterless function to disconnect either clients or server using an external buttonpress. Notice how it is VERY important that DisconnectionManager extends MonoBehaviour and NOT NetworkBehaviour, During a disconnection caused by a server rejection (wrong credentials), all scene NetworkObjects are despawned, making their functions unreachable, this is why this class exists.

Inherits MonoBehaviour.

Public Member Functions

- void **Disconnect** ()

  Function to disconnect a client and laod the first scene for a server. The function is purposefully public void and parameterless so that it can be used by external buttons.

Public Attributes

- MySceneManager **MSM**
- GameObject **cosmeticPanel**

3.8.1 Detailed Description

The DisconnectionManager handles the disconnection callback of clients in the general sense. It also exposes a public void parameterless function to disconnect either clients or server using an external buttonpress. Notice how it is VERY important that DisconnectionManager extends MonoBehaviour and NOT NetworkBehaviour, During a disconnection caused by a server rejection (wrong credentials), all scene NetworkObjects are despawned, making their functions unreachable, this is why this class exists.

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

- Assets/Scripts/Disconnections/DisconnectionManager.cs

3.9 doubt Struct Reference

Struct representing the data associated with a doubt. because of the use of FixedString32Bytes and Fixed↩String128Bytes, the whole struct CAN be sent over the network in a Rpc. This struct's size is fixed at: 1 + 8 + 8 + 32 + 32 + 32 + 1 + 128 + 128 = 370 bytes.

Public Member Functions

- **doubt** (ulong cId, ulong tId, string i, string o, string e, DOUBTTYPE t, string cd, string sd)
- void ProgressStatus (bool positiveBranch)

  Method to progress along the enum branches of the doubt.currentStatus.

Public Attributes

- STATUS **currentStatus**
- ulong **clientId**
- ulong **targetId**
- FixedString32Bytes **input**
- FixedString32Bytes **output**
- FixedString32Bytes **expected**
- DOUBTTYPE **doubtType**
- FixedString128Bytes **clientDoubt**
- FixedString128Bytes **serverDoubt**

3.9.1 Detailed Description

Struct representing the data associated with a doubt. because of the use of FixedString32Bytes and Fixed↩ String128Bytes, the whole struct CAN be sent over the network in a Rpc. This struct's size is fixed at: 1 + 8 + 8 + 32 + 32 + 32 + 1 + 128 + 128 = 370 bytes.

3.9.2 Member Function Documentation

3.9.2.1 ProgressStatus()

```
void doubt.ProgressStatus (
            bool positiveBranch )
```

Method to progress along the enum branches of the doubt.currentStatus.

Parameters

positiveBranch	true if the STATUS should progress on the positive branch, false otherwise.

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

- Assets/Scripts/Statics/RequiredStructs.cs

3.10 DoubtManager Class Reference

General manager of the doubting round.

Inherits NetworkBehaviour.

Public Member Functions

- void SetTarget (ulong clientId)

  Function to select the clicked PlayerBox as the current target.
- void SelectButton (Button b)

  External function to enforce exclusivity between the 4 options and to notify that the user has pressed a button. The function is public void and single parametrized on purpose so that it can be called by a button OnClick.
- void **NewText** ()

  External function to notify that new text has been entered in one of the 3 input fields. The function is public void and parameterless on purpose so that it can be called by an inputfield OnValueChanged.
- bool AllSetForDoubt ()

  Utility function to check for all requirements of the doubt panel. The user must select one of the 4 "expected" options. Inputs and outputs, when required, must respect the function signature's typings. Expected output and expected "perfect" output must differ.
- void **RemoveDoubt** ()

  External function to remove the doubt against the current target. The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void **CreateDoubt** ()

  External function to create a doubt against the current target. The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void **SendDoubts** ()

  External function called at the end of the available time. Sends all doubts to the server, one by one. Also opens the loading panel until the next scene is loaded.
- async void **CheckAllAndStartExecution** ()

  Function to start the server and client executions when all the clients are ready. The function is 'async' because the compilation and execution takes time (over 30 seconds usually). The function is 'async void' because it adheres to the "fire and forget" pattern, its termination is signaled by a side effect (the progressbar reaching 50%)
- void ParseServerDoubt (int lobbyIdx, string testResults)

  Function to parse completely a test result fn the server.
- void ParseClientDoubt (int lobbyIdx, int solutionIdx, string testResults)

  Function to parse completely a test result of a client.
- int **FindRelevantDoubt** (int lobbyIdx, int statusLevel, ulong targetId, testLineContents contents)
- int **FindRelevantDoubt** (int lobbyIdx, int statusLevel, ulong targetId, string input, bool includeTimeout, bool includeCrash)
- void **ReadyForNextSceneServerRpc** ()

  Remote Procedure Call, from client to server. Notifies the server that the client is ready for the next scene, when all the clients are ready, the next scene is loaded.
- void SendDoubtsServerRpc (int lobbyIdx, doubt singleDoubt)

  Remote Procedure Call, from client to server. Sends to the server a client doubt, must be called more than once so that every client sends all of its local doubts.
- void SendExecutionResultServerRpc (int lobbyIdx, int clientRank, string userResult)

  Remote Procedure Call, from client to server. Sends to the server the result of the local execution of the client's own solution. When all solutions have been parsed, the leaderboards are updated and all doubt are shared for the slideshow in the next scene.
- async void OffloadExecutionClientRpc (int lobbyIdx, int clientRank, string cppContent, string fileName, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Gives to each client the required strings to execute compilation and execution of the user solution locally. The result is sent back to the server with SendExecutionResultServerRpc(int, int, string).
- void UpdateDoubtsClientRpc (doubt[ ] allDoubts, string[ ] serverResults, string[ ] userResults, ClientRpc↩ Params clientRpcParams=default)

  Remote Procedure Call, from server to client. Updates the doubts and string relative to the result of server and user execution of each doubt.
- void UpdateLeaderboardClientRpc (int[ ] pointDeltas, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Updates the leaderboard of the lobby.

Public Attributes

- int **doneCounter** = 0
- int **totalMatrixSize** = 0
- List< int > **matrixSize** = new List<int>()
- List< string[ ]> **serverResults** = new List<string[ ]>()
- List< string[ ]> **userResults** = new List<string[ ]>()
- List< int[ ]> **pointDeltas** = new List<int[ ]>()
- MySceneManager **MSM**
- ExecutionManager **EM**
- ulong **targetId**
- List< doubt[ ]> **doubtList** = new List<doubt[ ]>()
- Slider **doneSlider**
- GameObject **serverPanel**
- LocalizeStringEvent[ ] **readyListText**
- GameObject **loadingPanel**
- Button **openDoubtPanelButton**
- TMP_InputField **givenInput**
- TMP_InputField **expectedOutput**
- TMP_InputField **correctOutput**
- Button **wrongReturnButton**
- Button **noCompileButton**
- Button **timeoutButton**
- Button **crashButton**
- Button **doubtButton**
- Button **removeButton**
- int

  Utility function to retrieve which client created a doubt just from the strings returned to the terminal. This Overload checks for the contents of contents to retrieve the doubter.

3.10.1 Detailed Description

General manager of the doubting round.

3.10.2 Member Function Documentation

3.10.2.1 AllSetForDoubt()

```
bool DoubtManager.AllSetForDoubt ( )
```

Utility function to check for all requirements of the doubt panel. The user must select one of the 4 "expected" options. Inputs and outputs, when required, must respect the function signature's typings. Expected output and expected "perfect" output must differ.

Returns

true if the requirements are all met, false otherwise.

3.10.2.2 OffloadExecutionClientRpc()

```
async void DoubtManager.OffloadExecutionClientRpc (
            int lobbyIdx,
            int clientRank,
            string cppContent,
            string fileName,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Gives to each client the required strings to execute compilation and execution of the user solution locally. The result is sent back to the server with SendExecutionResultServerRpc(int, int, string).

Parameters

lobbyIdx	Index of the lobby of the client.
clientRank	Position of the client in the leaderboard.
cppContent	Full user solution, with all the tests already attached.
fileName	Name of the file with which to call the temporary .cpp.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients in the same lobby will receive a different Rpc

3.10.2.3 ParseClientDoubt()

```
void DoubtManager.ParseClientDoubt (
            int lobbyIdx,
            int solutionIdx,
            string testResults )
```

Function to parse completely a test result of a client.

Parameters

lobbyIdx	Index of the lobby of the client.
solutionIdx	Index of the client solution that was executed.
testResults	String containing all test results of a client execution.

3.10.2.4 ParseServerDoubt()

```
void DoubtManager.ParseServerDoubt (
            int lobbyIdx,
            string testResults )
```

Function to parse completely a test result fn the server.

Parameters

lobbyIdx	Index of the lobby currently examining.
testResults	String containing all test results of a server execution.

3.10.2.5 SelectButton()

```
void DoubtManager.SelectButton (
            Button b )
```

External function to enforce exclusivity between the 4 options and to notify that the user has pressed a button. The function is public void and single parametrized on purpose so that it can be called by a button OnClick.

Parameters

b	The button that has just been pressed.

3.10.2.6 SendDoubtsServerRpc()

```
void DoubtManager.SendDoubtsServerRpc (
            int lobbyIdx,
            doubt singleDoubt )
```

Remote Procedure Call, from client to server. Sends to the server a client doubt, must be called more than once so that every client sends all of its local doubts.

Parameters

lobbyIdx	Index of the lobby of the client.
singleDoubt	doubt to be sent to the server.

3.10.2.7 SendExecutionResultServerRpc()

```
void DoubtManager.SendExecutionResultServerRpc (
            int lobbyIdx,
            int clientRank,
            string userResult )
```

Remote Procedure Call, from client to server. Sends to the server the result of the local execution of the client's own solution. When all solutions have been parsed, the leaderboards are updated and all doubt are shared for the slideshow in the next scene.

Parameters

lobbyIdx	Index of the lobby of the client.
clientRank	Position in the leaderboard of the client.
userResult	Result of the user solution's execution, null if it did not compile.

3.10.2.8 SetTarget()

```
void DoubtManager.SetTarget (
            ulong clientId )
```

Function to select the clicked PlayerBox as the current target.

Parameters

clientId	The id of the client corresponding to the clicked PlayerBox.

3.10.2.9 UpdateDoubtsClientRpc()

```
void DoubtManager.UpdateDoubtsClientRpc (
            doubt[] allDoubts,
            string[] serverResults,
            string[] userResults,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Updates the doubts and string relative to the result of server and user execution of each doubt.

Parameters

allDoubts	Array of all doubts of the lobby.
serverResults	Array of all results of the server executions.
userResults	Array of all results of the user executions.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients in the same lobby will receive a different Rpc

3.10.2.10 UpdateLeaderboardClientRpc()

```
void DoubtManager.UpdateLeaderboardClientRpc (
            int[] pointDeltas,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Updates the leaderboard of the lobby.

Parameters

pointDeltas	New leaderboard order.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients in the same lobby will receive a different Rpc

3.10.3 Member Data Documentation

3.10.3.1 int

`DoubtManager.int`

Utility function to retrieve which client created a doubt just from the strings returned to the terminal. This Overload checks for the contents of *contents* to retrieve the doubter.

Utility function to retrieve which client created a doubt just from the strings returned to the terminal. This Overload checks for *includeTimeout* and *includeCrash* strings to retieve the doubter.

Parameters

lobbyIdx	Index of the lobby of the client.
statusLevel	The STATUS level of the expected doubt.
targetId	Id of the client targeted by the expected doubt.
contents	An initially parsed test result line.

Returns

Tuple containing the index of the found doubt and the position of the doubter on the leaderboard, (-1,-1) if it is not found.

Parameters

lobbyIdx	Index of the lobby of the client.
statusLevel	The STATUS level of the expected doubt.
targetId	Id of the client targeted by the expected doubt.
input	String containing the input given to the expected doubt.
includeTimeout	Bool representing a timeout from the expected doubt.
includeCrash	Bool representing a crash from the expected doubt.

Returns

Tuple containing the index of the found doubt and the position of the doubter on the leaderboard, (-1,-1) if it is not found.

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

- Assets/Scripts/Gameplay/DoubtManager.cs

3.11 ExecutionManager Class Reference

Class responsible for all the out-of-unity executions, as well as preparing and storing execution data.

Inherits MonoBehaviour.

Public Member Functions

- string GetBaseTests ()

  Getter for the base tests of the current codeQuestion.
- string GetFinalTests ()

  Getter for the final tests of the current codeQuestion.
- string GetIntendedSolution ()

  Getter for the intended solution of the current codeQuestion.
- string GetUserSolution (string suppliedSolution)

  Utility function to obtain a solution after a possible main function substitution.
- async Task< string > PreCompile ()

  Function to create the file catch_main.o. The file it's required for execution but takes some time to compile , so it is compiled once at the beginning and then linked during the next compilations. The function is 'async' because the compilation takes some time (usually between 10 and 45 seconds). The function returns a Task becuase we need the result of the compilation.
- async Task< string > Compile (string fileName)

  Function to compile the user solution. The function is 'async' because the compilation takes some time (usually between 10 and 30 seconds). The function returns a Task because we need the result of the compilation.
- async Task< string > Test (string fileName, string tags)

  Function to execute the compiled executable. Depending on the round, the tests that are executed change using the Catch2 [tags]. The function is 'async' because the execution takes some time (usually less than 2 seconds). The function returns a Task because we need the result of the compilation.
- async Task< string > Test ()

  Overload to execute the compiled executable, with the same name as the codeQuestion name, with no tags, meaning that all tests will be run. The function is 'async' because the execution takes some time (usually less than 2 seconds). The function returns a Task because we need the result of the compilation.
- async void **BasicTest** ()

  Function to start the basic tests on the user solution, compilation errors and result of the execution are printed to the user log. The function is public void and parameterless so that it can be called by a button OnClick. The function is 'async' because the compilation and execution takes time (between 10 and 30 seconds usually). The function is 'async void' because it adheres to the "fire and forget" pattern, its termination is signaled by a side effect (the button becoming active)
- async Task< string > TestReadySolution (string solution, string filename)

  Function to start the creation, compilation and execution of all tests of the given solution. The function is 'async' because the execution takes some time (usually around 20 seconds). The function returns a Task because we need the result of the execution.
- void Create (bool notMine)

  Function that creates a .cpp file containing exactly what is written on a user solution. The user can ask to have a copy of its answer or of the best answer in the path that it selected in the settings. The function is public void and single parameter so that it can be called by a button OnClick.
- void Create (string solution, string fileName)

  Overload to create a .cpp file containing what is given in input in the temporary directory.
- void Create (string solution)

  Overload to create a .cpp containing what is given in input in the temporary directory, using the name of the codeQuestion as filename. The function is public void and single parameter so that it can be called by a button OnClick.
- void **CreateAll** ()

Funtion to create a .cpp file for all known user solutions (server only). The function is public void and parameterless so that it can be called by a button OnClick.

- string Boilerplate (string solution, bool createFile)

  Utility function to create a valid solution. Every solution requires: The import header and a definition of the TIMEOUT value. The additional setup code required to run the codeQuestion, if it exists. The main solution. The correct codeQuestion wrapper and the base tests. Call this function before every compilation.
- string Boilerplate ()

  Overload of Boilerplate(string, bool), it creates a valid solution from the user notepad and creates its corresponding cpp file.

Public Attributes

- NotepadManager **NM**
- TextManager **TM**
- Button **readyButton**
- string **functionType**
- string **functionName**
- string[ ] **argumentsType**
- string[ ] **argumentsName**
- inputLimits[ ] **argumentsLimits**
- TextAsset **main**
- TextAsset **imports**
- TextAsset **wrapper**

3.11.1 Detailed Description

Class responsible for all the out-of-unity executions, as well as preparing and storing execution data.

3.11.2 Member Function Documentation

3.11.2.1 Boilerplate() [1/2]

```
string ExecutionManager.Boilerplate ( )
```

Overload of Boilerplate(string, bool), it creates a valid solution from the user notepad and creates its corresponding cpp file.

Returns

The string containing the solution in the valid format.

3.11.2.2 Boilerplate() [2/2]

```
string ExecutionManager.Boilerplate (
            string solution,
            bool createFile )
```

Utility function to create a valid solution. Every solution requires: The import header and a definition of the TIMEOUT value. The additional setup code required to run the codeQuestion, if it exists. The main solution. The correct codeQuestion wrapper and the base tests. Call this function before every compilation.

Parameters

solution	Main solution to include.
createFile	true if a cpp file should be created, false otherwise.

Returns

The string containing a solution in the valid format.

3.11.2.3 Compile()

```
async Task< string > ExecutionManager.Compile (
            string fileName )
```

Function to compile the user solution. The function is 'async' because the compilation takes some time (usually between 10 and 30 seconds). The function returns a Task because we need the result of the compilation.

Parameters

fileName	The name of the file to compile, with extension excluded.

Returns

The result of the compilation, if it is not empty something went wrong.

3.11.2.4 Create() [1/3]

```
void ExecutionManager.Create (
            bool notMine )
```

Function that creates a .cpp file containing exactly what is written on a user solution. The user can ask to have a copy of its answer or of the best answer in the path that it selected in the settings. The function is public void and single parameter so that it can be called by a button OnClick.

Parameters

notMine	true if the .cpp should be created from the best solution in the lobby, false if the .cpp should be created from the solution of the current user.

3.11.2.5 Create() [2/3]

```
void ExecutionManager.Create (
            string solution )
```

Overload to create a .cpp containing what is given in input in the temporary directory, using the name of the codeQuestion as filename. The function is public void and single parameter so that it can be called by a button OnClick.

Parameters

solution	Content to be inserted in the .cpp.

3.11.2.6 Create() [3/3]

```
void ExecutionManager.Create (
            string solution,
            string fileName )
```

Overload to create a .cpp file containing what is given in input in the temporary directory.

Parameters

solution	Content to be inserted in the .cpp.
fileName	New name of the .cpp that will be created.

3.11.2.7 GetBaseTests()

```
string ExecutionManager.GetBaseTests ( )
```

Getter for the base tests of the current codeQuestion.

Returns

The contents of the [basic] tests.

3.11.2.8 GetFinalTests()

```
string ExecutionManager.GetFinalTests ( )
```

Getter for the final tests of the current codeQuestion.

Returns

The contents of the [final] tests.

3.11.2.9 GetIntendedSolution()

```
string ExecutionManager.GetIntendedSolution ( )
```

Getter for the intended solution of the current codeQuestion.

Returns

The contents of the solution function from the codeQuestion file.

3.11.2.10 GetUserSolution()

```
string ExecutionManager.GetUserSolution (
            string suppliedSolution )
```

Utility function to obtain a solution after a possible main function substitution.

Parameters

suppliedSolution	String to use instead of the user notepad.

Returns

The solution, either given from the notepad or after a possible main function substitution.

3.11.2.11 PreCompile()

```
async Task< string > ExecutionManager.PreCompile ( )
```

Function to create the file catch_main.o. The file it's required for execution but takes some time to compile , so it is compiled once at the beginning and then linked during the next compilations. The function is 'async' because the compilation takes some time (usually between 10 and 45 seconds). The function returns a Task becuase we need the result of the compilation.

Returns

The result of the compilation, if it is not empty something went wrong.

3.11.2.12 Test() [1/2]

```
async Task< string > ExecutionManager.Test ( )
```

Overload to execute the compiled executable, with the same name as the codeQuestion name, with no tags, meaning that all tests will be run. The function is 'async' because the execution takes some time (usually less than 2 seconds). The function returns a Task because we need the result of the compilation.

Returns

The result of the execution, if it is empty something went wrong.

3.11.2.13 Test() [2/2]

```
async Task< string > ExecutionManager.Test (
            string fileName,
            string tags )
```

Function to execute the compiled executable. Depending on the round, the tests that are executed change using the Catch2 [tags]. The function is 'async' because the execution takes some time (usually less than 2 seconds). The function returns a Task because we need the result of the compilation.

Parameters

fileName	Name of the executable to test.
tags	Tags to select which tests to execute, [base], [user], [final] or no tag, meaning all of them.

Returns

The result of the execution, if it is empty something went wrong.

3.11.2.14 TestReadySolution()

```
async Task< string > ExecutionManager.TestReadySolution (
            string solution,
            string filename )
```

Function to start the creation, compilation and execution of all tests of the given solution. The function is 'async' because the execution takes some time (usually around 20 seconds). The function returns a Task because we need the result of the execution.

Parameters

solution	A solution already filled with boilerplate and doubts.
filename	The name to use for the .cpp and .exe names.

Returns

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

- Assets/Scripts/Gameplay/ExecutionManager.cs

3.12 HintBox Class Reference

Class exclusively attached to a HintBox prefab. Can be updated externally using Setup.

Inherits MonoBehaviour.

Public Member Functions

- void Setup (int head, string body)

  Sets the 2 texts of an hint: the number at the top and the label of the body.

Public Attributes

- int **hintNumber**
- LocalizeStringEvent **hintBody**

3.12.1 Detailed Description

Class exclusively attached to a HintBox prefab. Can be updated externally using Setup.

3.12.2 Member Function Documentation

3.12.2.1 Setup()

```
void HintBox.Setup (
            int head,
            string body )
```

Sets the 2 texts of an hint: the number at the top and the label of the body.

Parameters

head	Number of the hint, from 0 to how many are present in the codeQuestion.
body	Label to be assigned to the hint so that it will get localized at runtime.

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

- Assets/Scripts/TextManagement/HintBox.cs

3.13 inputLimits Struct Reference

Struct representing a limit imposed on a codeQuestion input argument. Only ever used locally.

Public Member Functions

- void **Init** ()

Public Attributes

- bool **isMalformed**
- int **leftValue**
- int **rightValue**
- bool **leftIncluded**
- bool **rightIncluded**
- List< string > **setValues**

3.13.1 Detailed Description

Struct representing a limit imposed on a codeQuestion input argument. Only ever used locally.

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

- Assets/Scripts/Statics/RequiredStructs.cs

3.14 InvokableDataManager Class Reference

The InvokableDataManager class exists solely for the purpose of exposing public void and value parametrized functions that will then called by external UI elements. The functions have purposefully the same name that they have in the DataManager class.

Inherits MonoBehaviour.

Public Member Functions

- void **SetVolume** (float value)
- void **SetTimeout** (int value)
- void **SetPath** (string value)
- void **SetTimer** (string value)

3.14.1 Detailed Description

The InvokableDataManager class exists solely for the purpose of exposing public void and value parametrized functions that will then called by external UI elements. The functions have purposefully the same name that they have in the DataManager class.

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

- Assets/Scripts/Invokables/InvokableDataManager.cs

3.15 IpManager Class Reference

Class to manage the retieval of the local Ipv4 address and the correct setup of the connection address for server and clients so that they may communicate.

Inherits MonoBehaviour.

Public Member Functions

- void **IpSelection** ()

  After having inserted an Ip address. If it is valid, it becomes the new connection address for the client. If it is invalid, we fallback to localhost.

Public Attributes

- LocalizeStringEvent **currentAddressText**
- LocalizeStringEvent **feedbackText**
- string **address**
- TMP_InputField **ipField**
- Button **playButton**

3.15.1 Detailed Description

Class to manage the retieval of the local Ipv4 address and the correct setup of the connection address for server and clients so that they may communicate.

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

- Assets/Scripts/TextManagement/IpManager.cs

3.16 LobbyManager Class Reference

Class responsible for spawning and managing the lobbies. See LobbyUI. The maximum amount of possible lobbies is stored in constant maxNumberOfLobbies.

Inherits MonoBehaviour.

Public Member Functions

- void **SpawnLobbies** ()

  Function to spawn the lobbies in the server interface. Each lobby is saved in a list and setup as needed. Notice how their NetworkObject component needs to be spawned, so that other NetworkObjects might be parented to the lobby.
- bool AssignLobby (RectTransform playerTransform, ulong clientId)

  Utility function to assign a player prefab to the first available spot, returns if the operation was successfull or not.
- void DeassignLobby (ulong clientId)

  Utility function to deassign a player from its lobby.
- void **RebalanceLobbies** ()

  Function to rebalance the lobbies using the following scheme: Divide the number of clients by the maximum capacity of a lobby to get the number of needed lobbies. Divide the number of clients by the needed lobbies to get the minimum guaranteed number of clients in each lobby. Distribute the minimum guarantedd number of clients to the needed lobbies. The remaining clients are distributed in round-robin order to all lobbies. To reduce computation, clients that are already assigned to a lobby correctly are not moved.
- List< ulong[ ]> GetClientsInLobbies ()

  Utility function to return all the clients inside all the lobbies in a list.

Public Attributes

- RectTransform **lobbyHolder**
- GameObject **lobbyPrefab**
- GameObject **userHolderPrefab**
- List< GameObject > **listOfLobbies** = new List<GameObject>()

3.16.1 Detailed Description

Class responsible for spawning and managing the lobbies. See LobbyUI. The maximum amount of possible lobbies is stored in constant maxNumberOfLobbies.

3.16.2 Member Function Documentation

3.16.2.1 AssignLobby()

```
bool LobbyManager.AssignLobby (
            RectTransform playerTransform,
            ulong clientId )
```

Utility function to assign a player prefab to the first available spot, returns if the operation was successfull or not.

Parameters

playerTransform	The player prefab transform of the client to assign.
clientId	The id client to assign.

Returns

3.16.2.2 DeassignLobby()

```
void LobbyManager.DeassignLobby (
            ulong clientId )
```

Utility function to deassign a player from its lobby.

Parameters

clientId	The id of the client to deassign.

3.16.2.3 GetClientsInLobbies()

```
List< ulong[]> LobbyManager.GetClientsInLobbies ( )
```

Utility function to return all the clients inside all the lobbies in a list.

Returns

List containing the arrays of the client ids inside all lobbies.

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

- Assets/Scripts/Lobbies/LobbyManager.cs

3.17 LobbyUI Class Reference

Prefab class managing the details of a lobby and exposing the functions to interact with it. The maximum number of players that can be in a lobby is stored in constant maxCapacityOfLobby.

Inherits MonoBehaviour.

Public Member Functions

- void AddClient (RectTransform playerTransform, ulong clientId)

  Function add given player to this lobby. The RectTransform of the player is needed for easy parenting.
- void RemoveClient (ulong clientId)

  Function to remove the given player from this lobby. Because this function is called for player disconnection or lobby rebalancing, there is no need to keep track of the clients' RectTransform, it either gets despawned or moved with AddClient(RectTransform, ulong).
- int FirstSeat (ulong val)

  Uility function to return the first spot in the lobby that corresponds to the given Id. When given 0, it will return the first empty spot.
- void Setup (int n, GameObject usersHolderPrefab)

  Function called externally by LobbyManager to setup the details of a lobby.
- ulong[ ] GetClientsInLobby ()

  Utility function to return all Ids of the clients in this lobby. Very useful to extract the targets for ClientRpcParams.
- int GetMaxCapacityOfLobby ()

  Getter of the constant for the maximum amount of client that a lobby can handle.

Public Attributes

- int **lobbyNumber**
- int **lobbyCount** = 0
- int **lobbyCapacity** = maxCapacityOfLobby
- LocalizeStringEvent **lobbyInfoText**
- ulong[ ] **freeSeats** = new ulong[maxCapacityOfLobby]

3.17.1 Detailed Description

Prefab class managing the details of a lobby and exposing the functions to interact with it. The maximum number of players that can be in a lobby is stored in constant maxCapacityOfLobby.

3.17.2 Member Function Documentation

3.17.2.1 AddClient()

```
void LobbyUI.AddClient (
            RectTransform playerTransform,
            ulong clientId )
```

Function add given player to this lobby. The RectTransform of the player is needed for easy parenting.

Parameters

playerTransform	The transform of the player spawned prefab.
clientId	The id of the player to add.

3.17.2.2 FirstSeat()

```
int LobbyUI.FirstSeat (
            ulong val )
```

Uility function to return the first spot in the lobby that corresponds to the given Id. When given 0, it will return the first empty spot.

Parameters

val	Value to be found in the lobby.

Returns

The index of the position in the lobby for *val* , or -1 if it was not found.

3.17.2.3 GetClientsInLobby()

```
ulong[] LobbyUI.GetClientsInLobby ( )
```

Utility function to return all Ids of the clients in this lobby. Very useful to extract the targets for ClientRpcParams.

Returns

The array of the ids of the clients in this lobby.

3.17.2.4 GetMaxCapacityOfLobby()

```
int LobbyUI.GetMaxCapacityOfLobby ( )
```

Getter of the constant for the maximum amount of client that a lobby can handle.

Returns

maxCapacityOfLobby.

3.17.2.5 RemoveClient()

```
void LobbyUI.RemoveClient (
            ulong clientId )
```

Function to remove the given player from this lobby. Because this function is called for player disconnection or lobby rebalancing, there is no need to keep track of the clients' RectTransform, it either gets despawned or moved with AddClient(RectTransform, ulong).

Parameters

clientId	The id of the player to remove.

3.17.2.6 Setup()

```
void LobbyUI.Setup (
            int n,
            GameObject usersHolderPrefab )
```

Function called externally by LobbyManager to setup the details of a lobby.

Parameters

n	Number of the lobby.
usersHolderPrefab	Prefab of the area where the player prefabs will be parented.

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

- Assets/Scripts/Lobbies/LobbyUI.cs

3.18 MySceneManager Class Reference

The MySceneManager class exists to expose UnityEngine.SceneManagement methods. Because of the synchronized nature of the project, the actual scene management is handled mostly by the NetworkManager.SceneManager, which we interface in SetupAndLoadNextScene.

Inherits MonoBehaviour.

Public Member Functions

- void **Quit** ()

  Utility function to Invoke the client disconnection and termination of the application. The function is purposefully public void and parameterless so that it can be used by external buttons.
- void **SetupAndLoadNextScene** ()

  Take care of despawning everything that has been spawned on the network and then load the next scene. The function is purposefully public void and parameterless so that it can be used by external buttons.
- void **LoadSceneZero** ()

  Turns off and destroys the NetworkManager, then loads the first scene. Note that the NetworkManager is a scene object in the main menu, meaning that the destruction is consistent with its Singleton pattern. The function is purposefully public void and parameterless so that it can be used by external buttons.

3.18.1 Detailed Description

The MySceneManager class exists to expose UnityEngine.SceneManagement methods. Because of the synchronized nature of the project, the actual scene management is handled mostly by the NetworkManager.SceneManager, which we interface in SetupAndLoadNextScene.

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

- Assets/Scripts/SceneManagement/MySceneManager.cs

3.19 NetworkWrapper Class Reference

General manager class for the main menu. The minimum number of clients required before being able to start a session is saved in constant minNumberOfPlayers.

Inherits NetworkBehaviour.

Public Member Functions

- void **Server** ()

  External function to start the Server and subscribe to the relevant callbacks. The function is public void and parameterless on purpose so that it could be called from a button OnClick
- void RequirementsCheck (ulong clientId)

  Function that responds to a client connecting to the server or the codeQuestion being chosen. Makes sure that the codeQuestion had been chosen and the minimum amount of players is connected.
- void ValidLogin (ulong clientId, ClientRpcParams clientRpcParams=default)

  External function to execute the operations after a connection has been confirmed as valid. In case that the maximum amount of players has been reached, the client is disconnected. Otherwise it is assigned a spot in the lobbies and the cosmetic panel is opened.
- bool NewConfirmedClient (ulong clientId, databaseEntry userData)

  Function to add the user into the dictionary if the credentials are not already in the session. If they are, the connection is refused.
- void **LoadNotepadScene** ()

  Function to start the last necessary steps before starting a game session. Rebalance lobbies, give Avatars to clients without one, share the lobby information and session information with the clients The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void CosmeticChoiceServerRpc (ulong clientId, string spriteName)

  Remote Procedure Call, from client to server. Signals the server that the user has selected a different Avatar and updates it over the network accordingly.
- void **CheckAllAndNextSceneServerRpc** ()

  Remote Procedure Call, from client to server. Signals the server that whatever operation the client had to perform has finished. When all clients have terminated all their operations, the next scene is loaded.
- void FillDataClientRpc (databaseEntry userData, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Updates the clients data.
- void **OpenCosmeticsClientRpc** (ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Opens the client's cosmetics panel.

Parameters

clientRpcParams	*Necessary parameter to edit which clients will receive the Rpc, in this case all clients will receive a different Rpc*

- void AddLobbiesClientRpc (int lobbyIdx, int lobbySize, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Updates the lobby data for each client, then signals the server that it has completed the operation.
- void UpdateStaticDataClientRpc (int currentTimer, string questionName, string questionLabel, string questionContent, string[ ] questionTags)

  Remote Procedure Call, from server to client. Updates the static data decided by the server for each client, the it signals the server that it has completed the operation.. Specifically: how much time is available and which codeQuestion was selected. The codeQuestion is sent piece by piece and then reconstructed on the client because of 2 limitation↩ : A struct property cannot be a reference type (like string) if it is to be sent over the network. The content of a codeQuestion could be arbitrarely large, so converting the struct's strings into fixed byte strings would not work.

Public Attributes

- bool **isServerBuild**
- Button **continueButton**
- LocalizeStringEvent **continueButtonText**
- MySceneManager **MSM**
- GameObject **serverPanel**
- LocalizeStringEvent **numberOfPlayers**

- LocalizeStringEvent **codeQuestionSolved**
- GameObject **cosmeticPanel**
- GameObject **mainMenuPanel**
- Button **startButton**
- LobbyManager **LM**

3.19.1 Detailed Description

General manager class for the main menu. The minimum number of clients required before being able to start a session is saved in constant minNumberOfPlayers.

3.19.2 Member Function Documentation

3.19.2.1 AddLobbiesClientRpc()

```
void NetworkWrapper.AddLobbiesClientRpc (
            int lobbyIdx,
            int lobbySize,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Updates the lobby data for each client, then signals the server that it has completed the operation.

Parameters

lobbyIdx	The client's lobby.
lobbySize	The size of the lobby.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients in the same lobby will receive a different Rpc

3.19.2.2 CosmeticChoiceServerRpc()

```
void NetworkWrapper.CosmeticChoiceServerRpc (
            ulong clientId,
            string spriteName )
```

Remote Procedure Call, from client to server. Signals the server that the user has selected a different Avatar and updates it over the network accordingly.

Parameters

clientId	The id of the client that called it.
spriteName	The name of the new Avatar.

3.19.2.3 FillDataClientRpc()

```
void NetworkWrapper.FillDataClientRpc (
            databaseEntry userData,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Updates the clients data.

Parameters

userData	The new client data.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients will receive a different Rpc

3.19.2.4 NewConfirmedClient()

```
bool NetworkWrapper.NewConfirmedClient (
            ulong clientId,
            databaseEntry userData )
```

Function to add the user into the dictionary if the credentials are not already in the session. If they are, the connection is refused.

Parameters

clientId	The id of the client that just connected.
userData	The data (user creadentials) of the client that just connected.

Returns

3.19.2.5 RequirementsCheck()

```
void NetworkWrapper.RequirementsCheck (
            ulong clientId )
```

Function that responds to a client connecting to the server or the codeQuestion being chosen. Makes sure that the codeQuestion had been chosen and the minimum amount of players is connected.

Parameters

clientId	The id of the client that just connected, not needed.

3.19.2.6 UpdateStaticDataClientRpc()

```
void NetworkWrapper.UpdateStaticDataClientRpc (
            int currentTimer,
            string questionName,
            string questionLabel,
            string questionContent,
            string[] questionTags )
```

Remote Procedure Call, from server to client. Updates the static data decided by the server for each client, the it signals the server that it has completed the operation.. Specifically: how much time is available and which codeQuestion was selected. The codeQuestion is sent piece by piece and then reconstructed on the client because of 2 limitation: A struct property cannot be a reference type (like string) if it is to be sent over the network. The content of a codeQuestion could be arbitrarely large, so converting the struct's strings into fixed byte strings would not work.

Parameters

currentTimer	Amount of available time to create a solution.
questionName	Name of the selected codeQuestion.
questionLabel	Description (in label format) of the selected codeQuestion.
questionContent	Content of the selected codeQuestion.
questionTags	Tags of the selected codeQuestion.

3.19.2.7 ValidLogin()

```
void NetworkWrapper.ValidLogin (
            ulong clientId,
            ClientRpcParams clientRpcParams = default )
```

External function to execute the operations after a connection has been confirmed as valid. In case that the maximum amount of players has been reached, the client is disconnected. Otherwise it is assigned a spot in the lobbies and the cosmetic panel is opened.

Parameters

clientId	The id of the client that completed the login.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients will receive a different Rpc

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

- Assets/Scripts/GeneralWrapper/NetworkWrapper.cs

3.20 NotepadManager Class Reference

Class that manages the notepads, implements autocentering and Undo-Redo. The maximum amount of zoom possible is stored in constant maxAllowedZoom. The minimum amount of zoom possible is stored in constant minAllowedZoom. The zoom granularity is stored in constant zoomStepSize. The offset required to avoid that the cursor goes off screen while typing is stored in constant typingOffset.

Inherits MonoBehaviour.

Public Member Functions

- void DoText (string value)

  External function to save the current text into the stack of past texts. The function is public void and value parametrized on purpose so that it can be called from a inputfield OnValueChanged.
- void **UndoText** ()

  External function to restore a previous text, only triggered by pressing Ctrl+Z.
- void **RedoText** ()

  External function to restore a previously undone text, only triggered by pressing Ctrl+Y.
- string GetSolution ()

  Getter of the content of the notepad.
- void SetSolution (string solution)

  Setter of the content of the notepad.
- void **SwapInteractability** ()

  External function to set the notepad as interactable or viceversa. The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void ContentChanged (bool focusOnBeginning)

  Adjusts the size of the notepad on the text inside of it. If notepad contains more text that it can show, the scrollbars will activate. It also automatically calls MoveView(bool, bool, bool) so that the content is centered where the text is being modified. The function is public void and single parameter so it can be called by the inputfield OnValueChanged.
- void **ZoomIn** ()

  Increases the zoom on of the notepad to a maximum of 200%. The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void **ZoomOut** ()

  Decreases the zoom on of the notepad to a minimum of 20%. The function is public void and parameterless on purpose so that it can be called by a button OnClick.

Public Attributes

- ScrollRect **scrollView**
- RectTransform **content**
- bool **readOnly**

3.20.1 Detailed Description

Class that manages the notepads, implements autocentering and Undo-Redo. The maximum amount of zoom possible is stored in constant maxAllowedZoom. The minimum amount of zoom possible is stored in constant minAllowedZoom. The zoom granularity is stored in constant zoomStepSize. The offset required to avoid that the cursor goes off screen while typing is stored in constant typingOffset.

3.20.2 Member Function Documentation

3.20.2.1 ContentChanged()

```
void NotepadManager.ContentChanged (
            bool focusOnBeginning )
```

Adjusts the size of the notepad on the text inside of it. If notepad contains more text that it can show, the scrollbars will activate. It also automatically calls MoveView(bool, bool, bool) so that the content is centered where the text is being modified. The function is public void and single parameter so it can be called by the inputfield OnValue↩ Changed.

Parameters

focusOnBeginning	true if the view should be set at the beginning, false otherwise.

3.20.2.2 DoText()

```
void NotepadManager.DoText (
            string value )
```

External function to save the current text into the stack of past texts. The function is public void and value parametrized on purpose so that it can be called from a inputfield OnValueChanged.

Parameters

value	The new text of the notepad.

3.20.2.3 GetSolution()

```
string NotepadManager.GetSolution ( )
```

Getter of the content of the notepad.

Returns

The current string in the notepad.

3.20.2.4 SetSolution()

```
void NotepadManager.SetSolution (
            string solution )
```

Setter of the content of the notepad.

Parameters

solution	The new string to put inside the notepad.

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

- Assets/Scripts/Gameplay/NotepadManager.cs

3.21 PlayerController Class Reference

Prefab class managing the details of a UserBox and exposing the functions to interact with it. This class is the NetworkManager's player class, so it should be spawned by it to be propagated on the network.

Inherits NetworkBehaviour.

Public Member Functions

- void DataHasChanged (databaseEntry oldData, databaseEntry newData)

  Function that responds to the change in value of myData. Calls SetPlayerData(databaseEntry) to update the client data.
- void UpdateLayoutState (bool oldData, bool newData)

  Function that responds to the change in value of disableLayout. Removes or reintroduces the restriction on the LayoutElement component that makes it stretch for all the space available.
- void UpdateBackgroundColor (bool oldData, bool newData)

  Function that responds to the change in value of isSelected. Changes the color of the background panel.

Public Attributes

- Image **backgroundPanel**
- Button **selectionButton**
- TextMeshProUGUI **nametag**
- Image **spriteArea**
- bool **retrySettingClicks** = false
- NetworkVariable< bool > **disableLayout** = new NetworkVariable<bool>(false)
- NetworkVariable< bool > **isSelected** = new NetworkVariable<bool>(false)
- NetworkVariable< bool > **canBeClicked** = new NetworkVariable<bool>()
- NetworkVariable< int > **myAvatarState** = new NetworkVariable<int>(0)
- NetworkVariable< databaseEntry > **myData** = new NetworkVariable<databaseEntry>()

3.21.1 Detailed Description

Prefab class managing the details of a UserBox and exposing the functions to interact with it. This class is the NetworkManager's player class, so it should be spawned by it to be propagated on the network.

3.21.2 Member Function Documentation

3.21.2.1 DataHasChanged()

```
void PlayerController.DataHasChanged (
            databaseEntry oldData,
            databaseEntry newData )
```

Function that responds to the change in value of myData. Calls SetPlayerData(databaseEntry) to update the client data.

Parameters

oldData	Data before the change, not needed.
newData	Data after the change.

3.21.2.2 UpdateBackgroundColor()

```
void PlayerController.UpdateBackgroundColor (
            bool oldData,
            bool newData )
```

Function that responds to the change in value of isSelected. Changes the color of the background panel.

Parameters

oldData	Data before the change, not needed.
newData	Data after the change.

3.21.2.3 UpdateLayoutState()

```
void PlayerController.UpdateLayoutState (
            bool oldData,
            bool newData )
```

Function that responds to the change in value of disableLayout. Removes or reintroduces the restriction on the LayoutElement component that makes it stretch for all the space available.

Parameters

oldData	Data before the change, not needed.
newData	Data after the change.

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

- Assets/Scripts/Players/PlayerController.cs

3.22 PlayerSpawner Class Reference

Class responsible for spawning and managing the userBoxes. See PlayerController.

Inherits NetworkBehaviour.

Public Member Functions

- void SpawnAllLobbyLeaderboards (RectTransform[ ] holdersTransform, bool serverOwnership, bool useOld↩ Leaderboard, bool staggeredSpawning)

  Utility function to spawn the player prefabs using the same function for all lobbies.
- void HighlightPlayerBoxServerRpc (int lobbyIdx, ulong doubterId, ulong targetId)

  Remote Procedure Call, from client to server. Triggers the change in background color of the player prefabs, both in doubtersList and targetsList. To reduce the computation, the background changes only if the ids are different from the last time the function was called.

Public Attributes

- DoubtManager **DM**
- NotepadManager **NM**
- GameObject **playerPrefab**
- RectTransform[ ] **playerHolders**
- bool **isDoubtScene**

3.22.1 Detailed Description

Class responsible for spawning and managing the userBoxes. See PlayerController.

3.22.2 Member Function Documentation

3.22.2.1 HighlightPlayerBoxServerRpc()

```
void PlayerSpawner.HighlightPlayerBoxServerRpc (
            int lobbyIdx,
            ulong doubterId,
            ulong targetId )
```

Remote Procedure Call, from client to server. Triggers the change in background color of the player prefabs, both in doubtersList and targetsList. To reduce the computation, the background changes only if the ids are different from the last time the function was called.

Parameters

lobbyIdx	The lobby of the clients.
doubter↩ Id	The id of the client that made the doubt (top list).
targetId	The id of the client that was doubted (left list).

3.22.2.2 SpawnAllLobbyLeaderboards()

```
void PlayerSpawner.SpawnAllLobbyLeaderboards (
            RectTransform[] holdersTransform,
            bool serverOwnership,
            bool useOldLeaderboard,
            bool staggeredSpawning )
```

Utility function to spawn the player prefabs using the same function for all lobbies.

Parameters

holdersTransform	Array of transforms that the players will be parented to.
serverOwnership	true if the player prefab represents the local client, false if the prefab represents a different client.
useOldLeaderboard	true if the order should follow the not yet updated leaderboard, false to use the updated leaderboard.
staggeredSpawning	true if the player prefabs should be spread over many RectTransforms, false if they should be duplicated instead.

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

- Assets/Scripts/Players/PlayerSpawner.cs

3.23 ReadyManager Class Reference

General manager class for the first "round", the scene with the creation of the user solutions.

Inherits NetworkBehaviour.

Public Member Functions

- void **SendReady** ()

  Function to trigger the ServerRpc to notify of a change in ready status: from not ready to ready and viceversa. The function is public void and parameterless on purpose so that it can be called by a button OnClick.
- void **SendSolution** ()

  External function called by RoundTimer. It triggers the ServerRpc to force the client to send its solution.
- void SendReadyServerRpc (int lobbyIdx, databaseEntry userData)

  Remote Procedure Call, from client to server. Triggers a change in the ready list, either by adding the new client or removing an old one.
- void SendSolutionServerRpc (int lobbyIdx, databaseEntry userData, string solution)

  Remote Procedure Call, from client to server. Sends the solution to the server that will store it accordingly.
- void **ReadyForNextSceneServerRpc** ()

  Remote Procedure Call, from client to server. Notifies the server that the client is ready for the next scene. When all clients are ready, the next scene is loaded.
- void ShareSolutionsAndLeaderboardClientRpc (databaseEntry[ ] lobbyLeaderboard, string[ ] lobbySolutions, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Shares all the solutions and leaderboard of their own lobby to each client.

Public Attributes

- int **doneCounter** = 0
- MySceneManager **MSM**
- NotepadManager **NM**
- ExecutionManager **EM**
- Button **testButton**
- Slider **doneSlider**
- GameObject **serverPanel**
- LocalizeStringEvent[ ] **readyListText**

3.23.1 Detailed Description

General manager class for the first "round", the scene with the creation of the user solutions.

3.23.2 Member Function Documentation

3.23.2.1 SendReadyServerRpc()

```
void ReadyManager.SendReadyServerRpc (
            int lobbyIdx,
            databaseEntry userData )
```

Remote Procedure Call, from client to server. Triggers a change in the ready list, either by adding the new client or removing an old one.

Parameters

lobbyIdx	The lobby of the client.
userData	The general data of the client.

3.23.2.2 SendSolutionServerRpc()

```
void ReadyManager.SendSolutionServerRpc (
            int lobbyIdx,
            databaseEntry userData,
            string solution )
```

Remote Procedure Call, from client to server. Sends the solution to the server that will store it accordingly.

Parameters

lobbyIdx	The lobby of the client.
userData	The general data of the client.
solution	The solution of the client.

3.23.2.3 ShareSolutionsAndLeaderboardClientRpc()

```
void ReadyManager.ShareSolutionsAndLeaderboardClientRpc (
            databaseEntry[] lobbyLeaderboard,
            string[] lobbySolutions,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Shares all the solutions and leaderboard of their own lobby to each client.

Parameters

lobbyLeaderboard	The leaderboard of the client's lobby.
lobbySolutions	The solutions of the client's lobby.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients will receive a different Rpc depending on their lobby

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

- Assets/Scripts/RoundManagement/ReadyManager.cs

3.24 RoundTimer Class Reference

Class to keep track of the time passing during a round. The amount of additional time that the players receive during the doubting "round" is stored in constant percentageTimeIncreaseForEachClient.

Inherits MonoBehaviour.

Public Attributes

- ReadyManager **RM**
- DoubtManager **DM**
- Image **timerImage**
- bool **isDoubtScene**
- LocalizeStringEvent **timerText**
- int **displayTime** = 0

3.24.1 Detailed Description

Class to keep track of the time passing during a round. The amount of additional time that the players receive during the doubting "round" is stored in constant percentageTimeIncreaseForEachClient.

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

- Assets/Scripts/RoundManagement/RoundTimer.cs

3.25 SliderWithValueOnKnob Class Reference

Asset-like class, introduces the possibility of having a Slider with a TextMeshProUGUI on its knob.

Inherits MonoBehaviour.

Public Member Functions

- void **UpdateKnob** ()

  Utility function that updates the position of the TextMeshProUGUI component to be exactly on the Slider's knob.

Public Attributes

- float **farLeft**
- float **farRight**

3.25.1 Detailed Description

Asset-like class, introduces the possibility of having a Slider with a TextMeshProUGUI on its knob.

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

- Assets/Scripts/SelfContained/SliderWithValueOnKnob.cs

3.26 SlideshowManager Class Reference

General manager class for the slideshow scene. The amount of seconds given to the players to check each doubt is stored in constant waitTime.

Inherits NetworkBehaviour.

Public Member Functions

- async void **FinalExecution** ()

  Function that executes the last test battery on the user solution and then notifies the server of the result. The function is 'async' because the test takes time (between 1 and 10 seconds usually). The function is 'async void' because it adheres to the "fire and forget" pattern, its termination is signaled by a side effect (the call to SendFinalResultServerRpc(int, int, string))
- int ParseFinalResult (string finalResult)

  Function to parse the Catch2 test execution.
- void SendFinalResultServerRpc (int lobbyIdx, int clientRank, string userResult)

  Remote Procedure Call, from client to server. Sends the results of the users' final execution to the server, after which the server calculates the final scores and updates the clients leaderboard.
- void **PerformanceFinishedServerRpc** ()

  Remote Procedure Call, from client to server. Keeps track of all players that have finished watching the slideshow and are ready to start the final execution.
- void ReadyForNextSceneServerRpc (databaseEntry userData)

  Remote Procedure Call, from client to server. Keeps track of all players that have finished the final execution and are ready to load the next scene.
- void **FinalExecutionClientRpc** ()

  Remote Procedure Call, from server to client. Starts the FinalExecution to execute the last final battery on the client's own solution.
- void UpdateLeaderboardClientRpc (int[] pointDeltas, ClientRpcParams clientRpcParams=default)

  Remote Procedure Call, from server to client. Updates the clients leaderboard.

Public Attributes

- LocalizeStringEvent **mainText**
- LocalizeStringEvent **finalEvaluation**
- PlayerSpawner **PS**
- ExecutionManager **EM**
- MySceneManager **MSM**
- AccountManager **AM**
- ScrollRect **doubtersScrollView**
- ScrollRect **targetsScrollView**
- GameObject **leaderboardPanel**
- RectTransform **playerHolder**
- Slider **doneSlider**
- GameObject **serverPanel**

3.26.1 Detailed Description

General manager class for the slideshow scene. The amount of seconds given to the players to check each doubt is stored in constant waitTime.

3.26.2 Member Function Documentation

3.26.2.1 ParseFinalResult()

```
int SlideshowManager.ParseFinalResult (
            string finalResult )
```

Function to parse the Catch2 test execution.

Parameters

finalResult	Complete string from the test execution.

Returns

The number of points given to this execution.

3.26.2.2 ReadyForNextSceneServerRpc()

```
void SlideshowManager.ReadyForNextSceneServerRpc (
            databaseEntry userData )
```

Remote Procedure Call, from client to server. Keeps track of all players that have finished the final execution and are ready to load the next scene.

Parameters

userData	The user data of the current client.

3.26.2.3 SendFinalResultServerRpc()

```
void SlideshowManager.SendFinalResultServerRpc (
            int lobbyIdx,
            int clientRank,
            string userResult )
```

Remote Procedure Call, from client to server. Sends the results of the users' final execution to the server, after which the server calculates the final scores and updates the clients leaderboard.

Parameters

lobbyIdx	The lobby of the client.
clientRank	The position of the client in the current leaderboard.
userResult	The result of the final execution of the user solution.

3.26.2.4 UpdateLeaderboardClientRpc()

```
void SlideshowManager.UpdateLeaderboardClientRpc (
            int[] pointDeltas,
            ClientRpcParams clientRpcParams = default )
```

Remote Procedure Call, from server to client. Updates the clients leaderboard.

Parameters

pointDeltas	Points for all the clients in the lobby of this round.
clientRpcParams	Necessary parameter to edit which clients will receive the Rpc, in this case all clients in the same lobby will receive a different Rpc

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

- Assets/Scripts/AutomaticDisplay/SlideshowManager.cs

3.27 testLineContents Struct Reference

Summary struct representing an initial parse of a test result line. Only ever used locally.

Public Attributes

- string **input**
- string **correct**
- string **expected**
- int **endOfInputIdx**
- char **followingChar**

3.27.1 Detailed Description

Summary struct representing an initial parse of a test result line. Only ever used locally.

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

- Assets/Scripts/Statics/RequiredStructs.cs

3.28 TextManager Class Reference

Class to manage text in general. Maintains: localization initialization and visuals, hints initializaion and visuals, tutorial initialization and visuals, codeQuestion visuals and the log panel.

Inherits MonoBehaviour.

Public Member Functions

- void **Start** ()
- void ChangeTutorial (int dir)

  Function called externally by buttons, pressing the left one will decrease the tutorialIdx, the right one will increase it, then the tutorial data is updated using UpdateTutorial.
- void **UpdateLanguage** ()

  Loads the correct locale for the localization using languageIdx;
- void **ChangeLanguage** ()

  Changes the current language by increasing languageIdx and then calling UpdateLanguage
- void AddToLog (string message)

  Adds a message to the log, it implicitly adds a 2 newlines after it.
- void SelectCurrentPath (string value)

  External function to change the path of the permanent cpps and to keep the TMP_InputField synchronized with the variable. The function is public void and with a value parameter on purpose so that it could be called externally by an TMP_InputField OnEndEdit.
- void PassHints (string[ ] hintsFromExecutionManager)

  Function called externally to pass the hints that have been extracted in the ExecutionManager.
- void PassLabel (string label)

  Function called externally to pass the codeQuestion request that has been extracted in the ExecutionManager.
- void CreateHintBox (GameObject fromHintShop)

  Function called externally by a button click (from the "Buy" button). Creates a new HintBox prefab carrying all the information of a HintBox, additionally disables the buyHintButton if there are no more hints for this codeQuestion and updates the Client data appropriately for the final scoring.

Public Attributes

- TextMeshProUGUI **log**
- TMP_InputField **selectCppsPath**
- string **codeQuestionLabel**
- LocalizeStringEvent **codeQuestionHintText**
- LocalizeStringEvent **codeQuestionGoalText**
- GameObject **hintsPanel**
- GameObject **hintBoxPrefab**
- LocalizeStringEvent **startingPrize**
- Button **buyHintButton**
- int **languageIdx** = 0
- LocalizeStringEvent **feedbackText**
- int **tutorialIdx** = 0
- Texture[ ] **tutorialTextures**
- RawImage **tutorialImage**
- LocalizeStringEvent **tutorialText**

3.28.1 Detailed Description

Class to manage text in general. Maintains: localization initialization and visuals, hints initializaion and visuals, tutorial initialization and visuals, codeQuestion visuals and the log panel.

3.28.2 Member Function Documentation

3.28.2.1 AddToLog()

```
void TextManager.AddToLog (
            string message )
```

Adds a message to the log, it implicitly adds a 2 newlines after it.

Parameters

message	Message to add to the log<./param>

3.28.2.2 ChangeTutorial()

```
void TextManager.ChangeTutorial (
            int dir )
```

Function called externally by buttons, pressing the left one will decrease the tutorialIdx, the right one will increase it, then the tutorial data is updated using UpdateTutorial.

Parameters

dir	A negative number to reduce the tutorialIdx, a positive one to increase it.

3.28.2.3 CreateHintBox()

```
void TextManager.CreateHintBox (
            GameObject fromHintShop )
```

Function called externally by a button click (from the "Buy" button). Creates a new HintBox prefab carrying all the information of a HintBox, additionally disables the buyHintButton if there are no more hints for this codeQuestion and updates the Client data appropriately for the final scoring.

Parameters

fromHintShop	The GameObject representing the HintShop.

3.28.2.4 PassHints()

```
void TextManager.PassHints (
            string[] hintsFromExecutionManager )
```

Function called externally to pass the hints that have been extracted in the ExecutionManager.

Parameters

hintsFromExecutionManager	The hint labels in a string array.

3.28.2.5 PassLabel()

```
void TextManager.PassLabel (
            string label )
```

Function called externally to pass the codeQuestion request that has been extracted in the ExecutionManager.

Parameters

label	The label to display (before localization).

3.28.2.6 SelectCurrentPath()

```
void TextManager.SelectCurrentPath (
            string value )
```

External function to change the path of the permanent cpps and to keep the TMP_InputField synchronized with the variable. The function is public void and with a value parameter on purpose so that it could be called externally by an TMP_InputField OnEndEdit.

Parameters

value	New path inserted.

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

- Assets/Scripts/TextManagement/TextManager.cs

Index