Interaktywna wizualizacja urządzeń sterujących dla jonowego komputera kwantowego w środowisku wirtualnej rzeczywistości

Generated by Doxygen 1.9.8

1 Hierarchical Index **1**

1.1 Class Hierarchy 1

2 Class Index **3**

2.1 Class List 3

3 File Index **5**

3.1 File List 5

4 Class Documentation **7**

4.1 ButtonFollowVisual Class Reference 7

4.1.1 Detailed Description 8

4.1.2 Member Function Documentation 8

4.1.2.1 Follow() 8

4.1.2.2 Freeze() 8

4.1.2.3 ResetButton() 8

4.2 ConfigData Class Reference 10

4.2.1 Detailed Description 10

4.2.2 Member Data Documentation 10

4.2.2.1 pathToExport 10

4.2.2.2 pathToIonsFile 10

4.2.2.3 pathToSave 11

4.3 ExportData Class Reference 11

4.3.1 Detailed Description 11

4.3.2 Member Data Documentation 11

4.3.2.1 IonName 11

4.3.2.2 lasers 11

4.3.2.3 piTime 12

4.4 Graph Struct Reference 12

4.4.1 Detailed Description 12

4.4.2 Member Data Documentation 12

4.4.2.1 lineColor 12

4.4.2.2 pointColor 12

4.4.2.3 points 13

4.5 GraphControll Class Reference 13

4.5.1 Detailed Description 14

4.5.2 Member Function Documentation 14

4.5.2.1 GetShaderData() 14

4.6 GraphMaximize Class Reference 14

4.6.1 Detailed Description 15

4.7 GraphScaleDescControll Class Reference 15

4.7.1 Detailed Description 16

4.8 GraphShaderData Class Reference 16

4.8.1 Detailed Description 18
4.8.2 Member Enumeration Documentation 18
4.8.2.1 lineVariantEnum 18
4.8.2.2 pointShapeEnum 18
4.8.3 Member Function Documentation 19
4.8.3.1 AddPointToGraph() 19
4.8.3.2 CopyValuesFrom() 19
4.8.3.3 GetClosestPiontIndex() 19
4.8.3.4 SetOffsetOnX() 19
4.8.3.5 SetOffsetOnY() 20
4.8.3.6 SetScaleOnX() 20
4.8.3.7 SetScaleOnY() 20
4.8.4 Member Data Documentation 21
4.8.4.1 graphOffsetOnX 21
4.8.4.2 graphOffsetOnXMax 21
4.8.4.3 graphOffsetOnXMin 21
4.8.4.4 graphOffsetOnY 21
4.8.4.5 graphOffsetOnYMax 21
4.8.4.6 graphOffsetOnYMin 21
4.8.4.7 graphs 21
4.8.4.8 graphScaleOnX 22
4.8.4.9 graphScaleOnXMax 22
4.8.4.10 graphScaleOnXMin 22
4.8.4.11 graphScaleOnY 22
4.8.4.12 graphScaleOnYMax 22
4.8.4.13 graphScaleOnYMin 22
4.8.4.14 int 22
4.8.4.15 lineColor 23
4.8.4.16 lineSize 23
4.8.4.17 lineVariant 23
4.8.4.18 maxPointAmount 23
4.8.4.19 maxXPointValue 23
4.8.4.20 maxYPointValue 23
4.8.4.21 meshRenderer 23
4.8.4.22 minPointAmount 24
4.8.4.23 minXPointValue 24
4.8.4.24 minYPointValue 24
4.8.4.25 pointColor 24
4.8.4.26 pointShape 24
4.8.4.27 pointSize 24
4.8.4.28 unitPerGridOnX 24
4.8.4.29 unitPerGridOnY 25

4.9 GridFiller Class Reference 25
4.9.1 Detailed Description 25
4.10 InOutDataController Class Reference 26
4.10.1 Detailed Description 27
4.10.2 Member Function Documentation 27
4.10.2.1 CheckIfSaved() 27
4.10.2.2 LoadIonData() 27
4.10.3 Member Data Documentation 27
4.10.3.1 ionData 27
4.10.4 Property Documentation 28
4.10.4.1 instance 28
4.11 IonData Class Reference 28
4.11.1 Detailed Description 28
4.11.2 Member Data Documentation 28
4.11.2.1 laserAmount 28
4.11.2.2 name 29
4.12 Laser Struct Reference 29
4.12.1 Detailed Description 29
4.12.2 Member Data Documentation 29
4.12.2.1 points 29
4.13 MaximizeController Class Reference 30
4.13.1 Detailed Description 30
4.13.2 Member Function Documentation 31
4.13.2.1 MaximizeSection() 31
4.13.3 Property Documentation 31
4.13.3.1 instance 31
4.14 QuitButton Class Reference 31
4.14.1 Detailed Description 32
4.14.2 Member Function Documentation 32
4.14.2.1 QuitApplication() 32
4.15 SaveData Class Reference 33
4.15.1 Detailed Description 33
4.15.2 Member Data Documentation 33
4.15.2.1 graphs 33
4.15.2.2 IonName 34
4.15.2.3 numberOfLasers 34
4.15.2.4 piTime 34
4.16 ShowKeyboard Class Reference 34
4.16.1 Detailed Description 35
4.16.2 Member Function Documentation 35
4.16.2.1 OpenKeyboard() 35
4.16.2.2 SetCaretColorAlpha() 35

4.17 TwoStateButton Class Reference 36
4.17.1 Detailed Description 37
4.17.2 Member Function Documentation 37
4.17.2.1 ChangeValue() 37
4.17.2.2 GetValue() 37
4.17.2.3 SetValue() 37

5 File Documentation 39

5.1 DataScripts/ConfigData.cs File Reference 39
5.2 ConfigData.cs 39
5.3 DataScripts/ExportData.cs File Reference 39
5.4 ExportData.cs 39
5.5 DataScripts/Graph.cs File Reference 40
5.6 Graph.cs 40
5.7 DataScripts/IonData.cs File Reference 40
5.8 IonData.cs 40
5.9 DataScripts/Laser.cs File Reference 40
5.10 Laser.cs 41
5.11 DataScripts/SaveData.cs File Reference 41
5.12 SaveData.cs 41
5.13 GlobalScripts/GridFiller.cs File Reference 41
5.14 GridFiller.cs 41
5.15 GlobalScripts/InOutDataController.cs File Reference 42
5.16 InOutDataController.cs 42
5.17 GlobalScripts/MaximizeController.cs File Reference 45
5.18 MaximizeController.cs 45
5.19 GraphScripts/GraphControll.cs File Reference 46
5.20 GraphControll.cs 46
5.21 GraphScripts/GraphMaximize.cs File Reference 48
5.22 GraphMaximize.cs 49
5.23 GraphScripts/GraphScaleDescControll.cs File Reference 49
5.24 GraphScaleDescControll.cs 49
5.25 GraphScripts/GraphShaderData.cs File Reference 50
5.26 GraphShaderData.cs 50
5.27 InteractableScripts/ButtonFollowVisual.cs File Reference 53
5.28 ButtonFollowVisual.cs 53
5.29 InteractableScripts/QuitButton.cs File Reference 54
5.30 QuitButton.cs 54
5.31 InteractableScripts/ShowKeyboard.cs File Reference 54
5.32 ShowKeyboard.cs 55
5.33 InteractableScripts/TwoStateButton.cs File Reference 55
5.34 TwoStateButton.cs 55

Index **57**

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

- ConfigData 10
- ExportData 11
- Graph 12
- IonData 28
- Laser 29
- MonoBehaviour
  - ButtonFollowVisual 7
  - GraphControll 13
  - GraphMaximize 14
  - GraphScaleDescControll 15
  - GraphShaderData 16
  - GridFiller 25
  - InOutDataController 26
  - MaximizeController 30
  - QuitButton 31
  - ShowKeyboard 34
  - TwoStateButton 36
- SaveData 33

Chapter 2

Class Index

2.1 Class List

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

ButtonFollowVisual
Class `ButtonFollowVisual` controls visual of interactable button 7
ConfigData
Class `ConfigData` is a data class used in process of deserialization of config data 10
ExportData
Class `ExportData` is a data class used in process of serialization of export data 11
Graph
Class `Graph` is a data class used to describe one graph 12
GraphControll
Class `GraphControll` controls and manages interactions and actions of user with graph . 13
GraphMaximize
Class `GraphMaximize` hepl with process of maximization and minimization of graph section 14
GraphScaleDescControll
Class `GraphScaleDescControll` controls graph scale descriptions 15
GraphShaderData
Class `GraphShaderData` responsible for set up and controll of Graph 16
GridFiller
Class `GridFiller` responsible for filling of `GraphGrid` with right number of `Graph↩ Section` prefabs 25
InOutDataController
Class `InOutDataController` responsible for importing, saving and exporting data 26
IonData
Class `IonData` is a data class used in process of deserialization of ions data 28
Laser
Class `Laser` is a data class used to describe one laser 29
MaximizeController
Class `MaximizeController` control process of maximization and minimization of graph sections 30
QuitButton
Class `QuitButton` used to close application 31
SaveData
Class `SaveData` is a data class used in process of serialization of save data 33
ShowKeyboard
Class `ShowKeyboard` shows and hides interactable keyboard 34
TwoStateButton
Class `TwoStateButton` is responsible to controll two state button interaction 36

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

DataScripts/ConfigData.cs 39
DataScripts/ExportData.cs 39
DataScripts/Graph.cs 40
DataScripts/IonData.cs 40
DataScripts/Laser.cs 40
DataScripts/SaveData.cs 41
GlobalScripts/GridFiller.cs 41
GlobalScripts/InOutDataController.cs 42
GlobalScripts/MaximizeController.cs 45
GraphScripts/GraphControll.cs 46
GraphScripts/GraphMaximize.cs 48
GraphScripts/GraphScaleDescControll.cs 49
GraphScripts/GraphShaderData.cs 50
InteractableScripts/ButtonFollowVisual.cs 53
InteractableScripts/QuitButton.cs 54
InteractableScripts/ShowKeyboard.cs 54
InteractableScripts/TwoStateButton.cs 55

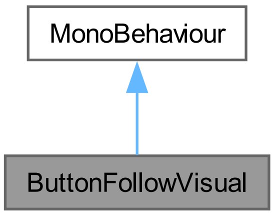
Chapter 4

Class Documentation

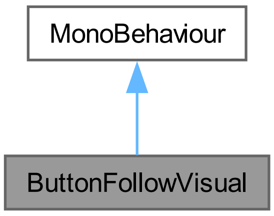
4.1 ButtonFollowVisual Class Reference

Class `ButtonFollowVisual` controls visual of interactable button.

Inheritance diagram for ButtonFollowVisual:



Collaboration diagram for ButtonFollowVisual:



Public Member Functions

- void Follow (BaseInteractionEventArgs hover)

  Method `Follow` is used to start process of folowing interactor by visual.
- void ResetButton (BaseInteractionEventArgs hover)

  Method `ResetButton` is used to end process of folowing interactor by visual and resets visual.
- void Freeze (BaseInteractionEventArgs hover)

  Method `Freeze` is used to freez visual.

4.1.1 Detailed Description

Class `ButtonFollowVisual` controls visual of interactable button.

Definition at line 7 of file ButtonFollowVisual.cs.

4.1.2 Member Function Documentation

4.1.2.1 Follow()

```
void ButtonFollowVisual.Follow (
            BaseInteractionEventArgs hover )
```

Method `Follow` is used to start process of folowing interactor by visual.

Parameters

hover	Information on BaseInteraction event

Definition at line 43 of file ButtonFollowVisual.cs.

4.1.2.2 Freeze()

```
void ButtonFollowVisual.Freeze (
            BaseInteractionEventArgs hover )
```

Method `Freeze` is used to freez visual.

Parameters

hover	Information on BaseInteraction event

Definition at line 78 of file ButtonFollowVisual.cs.

4.1.2.3 ResetButton()

```
void ButtonFollowVisual.ResetButton (
            BaseInteractionEventArgs hover )
```

Method `ResetButton` is used to end process of folowing interactor by visual and resets visual.

Parameters

hover	Information on BaseInteraction event

Definition at line 65 of file ButtonFollowVisual.cs.

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

- InteractableScripts/ButtonFollowVisual.cs

4.2 ConfigData Class Reference

Class `ConfigData` is a data class used in process of deserialization of config data.

Public Attributes

- string pathToSave

  Variable `pathToSave` contains path to save files folder.
- string pathToExport

  Variable `pathToExport` contains path to export files folder.
- string pathToIonsFile

  Variable `pathToIonsFile` contains path to file with ion configurations.

4.2.1 Detailed Description

Class `ConfigData` is a data class used in process of deserialization of config data.

Definition at line 6 of file ConfigData.cs.

4.2.2 Member Data Documentation

4.2.2.1 pathToExport

```
string ConfigData.pathToExport
```

Variable `pathToExport` contains path to export files folder.

Definition at line 15 of file ConfigData.cs.

4.2.2.2 pathToIonsFile

```
string ConfigData.pathToIonsFile
```

Variable `pathToIonsFile` contains path to file with ion configurations.

Definition at line 19 of file ConfigData.cs.

4.2.2.3 pathToSave

```
string ConfigData.pathToSave
```

Variable `pathToSave` contains path to save files folder.

Definition at line 11 of file ConfigData.cs.

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

- DataScripts/ConfigData.cs

4.3 ExportData Class Reference

Class `ExportData` is a data class used in process of serialization of export data.

Public Attributes

- string IonName

  Variable `IonName` contains name of exported ion.
- float piTime

  Variable `piTime` contains value of pitime parameter.
- List< Laser > lasers

  Variable `lasers` contains list of configured lasers for exported ion.

4.3.1 Detailed Description

Class `ExportData` is a data class used in process of serialization of export data.

Definition at line 8 of file ExportData.cs.

4.3.2 Member Data Documentation

4.3.2.1 IonName

```
string ExportData.IonName
```

Variable `IonName` contains name of exported ion.

Definition at line 13 of file ExportData.cs.

4.3.2.2 lasers

```
List<Laser> ExportData.lasers
```

Variable `lasers` contains list of configured lasers for exported ion.

Definition at line 21 of file ExportData.cs.

4.3.2.3 piTime

```
float ExportData.piTime
```

Variable `piTime` contains value of pitime parameter.

Definition at line 17 of file ExportData.cs.

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

- DataScripts/ExportData.cs

4.4 Graph Struct Reference

Class `Graph` is a data class used to describe one graph.

Public Attributes

- Color pointColor

  Variable `pointColor` contains color of points of graph.
- Color lineColor

  Variable `lineColor` contains color of graph.
- Vector2[ ] points

  Variable `points` contains points of graph.

4.4.1 Detailed Description

Class `Graph` is a data class used to describe one graph.

Definition at line 8 of file Graph.cs.

4.4.2 Member Data Documentation

4.4.2.1 lineColor

```
Color Graph.lineColor
```

Variable `lineColor` contains color of graph.

Definition at line 17 of file Graph.cs.

4.4.2.2 pointColor

```
Color Graph.pointColor
```

Variable `pointColor` contains color of points of graph.

Definition at line 13 of file Graph.cs.

4.4.2.3 points

```
Vector2 [ ] Graph.points
```

Variable `points` contains points of graph.

Definition at line 21 of file Graph.cs.

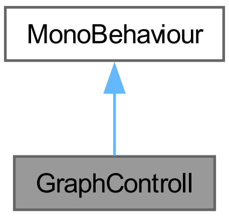
The documentation for this struct was generated from the following file:

- DataScripts/Graph.cs

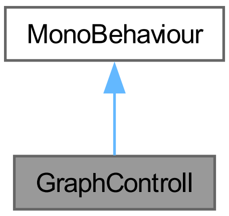
4.5 GraphControll Class Reference

Class `GraphControll` controls and manages interactions and actions of user with graph.

Inheritance diagram for GraphControll:



Collaboration diagram for GraphControll:



Public Member Functions

- GraphShaderData GetShaderData ()

  Method `GetShaderData` returns `GraphShaderData` of graph.

4.5.1 Detailed Description

Class `GraphControll` controls and manages interactions and actions of user with graph.

Definition at line 9 of file GraphControll.cs.

4.5.2 Member Function Documentation

4.5.2.1 GetShaderData()

```
GraphShaderData GraphControll.GetShaderData ( )
```

Method `GetShaderData` returns `GraphShaderData` of graph.

Returns

Value of `graphShaderData` variable

Definition at line 213 of file GraphControll.cs.

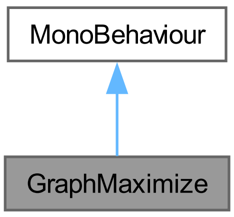
The documentation for this class was generated from the following file:

- GraphScripts/GraphControll.cs

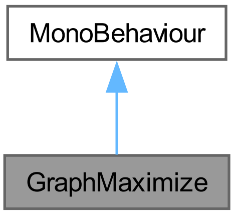
4.6 GraphMaximize Class Reference

Class `GraphMaximize` hepl with process of maximization and minimization of graph section.

Inheritance diagram for GraphMaximize:



Collaboration diagram for GraphMaximize:



4.6.1 Detailed Description

Class `GraphMaximize` hepl with process of maximization and minimization of graph section.

Definition at line 7 of file GraphMaximize.cs.

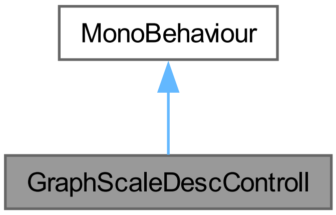
The documentation for this class was generated from the following file:

- GraphScripts/GraphMaximize.cs

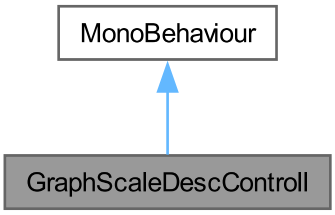
4.7 GraphScaleDescControll Class Reference

Class `GraphScaleDescControll` controls graph scale descriptions.

Inheritance diagram for GraphScaleDescControll:



Collaboration diagram for GraphScaleDescControll:



4.7.1 Detailed Description

Class `GraphScaleDescControll` controls graph scale descriptions.

Definition at line 7 of file GraphScaleDescControll.cs.

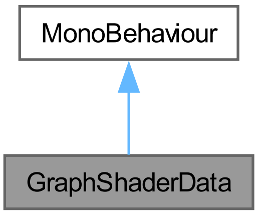
The documentation for this class was generated from the following file:

- GraphScripts/GraphScaleDescControll.cs

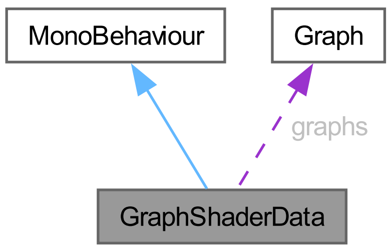
4.8 GraphShaderData Class Reference

Class `GraphShaderData` responsible for set up and controll of Graph.

Inheritance diagram for GraphShaderData:



Collaboration diagram for GraphShaderData:



Public Types

- enum pointShapeEnum { Square = 1 , Circle = 2 }
- enum lineVariantEnum { Normal = 1 , AllwaysConnected = 2 , Catmull = 3 }

Public Member Functions

- int GetClosestPiontIndex (float x, float y)
- void SetScaleOnY (float scale)

  Method `SetScaleOnY` set `graphScaleOnY` variable.
- void SetScaleOnX (float scale)

  Method `SetScaleOnX` set `graphScaleOnX` variable.
- void SetOffsetOnY (float offset)

  Method `SetOffsetOnY` set `graphOffsetOnY` variable.
- void SetOffsetOnX (float offset)

  Method `SetOffsetOnX` set `graphOffsetOnX` variable.
- void CopyValuesFrom (GraphShaderData input)

  Method `CopyValuesFrom` copy value from other `GraphShaderData` object.
- void AddPointToGraph (int graphIndex, Vector2 point)

  Method `AddPointToGraph` adds new point to graph of given index.

Public Attributes

- MeshRenderer meshRenderer
- float graphOffsetOnX = -2f
- float graphOffsetOnY = -2f
- float graphOffsetOnXMin = -6f
- float graphOffsetOnYMin = -6f
- float graphOffsetOnXMax = 6f
- float graphOffsetOnYMax = 6f
- float graphScaleOnX = 16f
- float graphScaleOnY = 16f
- float graphScaleOnXMin = 4f
- float graphScaleOnYMin = 4f

- float graphScaleOnXMax = 16f
- float graphScaleOnYMax = 16f
- float unitPerGridOnX = 1f
- float unitPerGridOnY = 1f
- pointShapeEnum pointShape = pointShapeEnum.Square
- Color pointColor = Color.black
- float pointSize = 0.2f
- float minXPointValue = -1f
- float maxXPointValue = 9f
- float minYPointValue = -1f
- float maxYPointValue = 9f
- int minPointAmount = 0
- int maxPointAmount = 255
- lineVariantEnum lineVariant = lineVariantEnum.Normal
- Color lineColor = Color.black
- float lineSize = 0.2f
- Graph[ ] graphs
- int

    Method `GetClosestPiontIndex` give coordinates of closest point on graph.

4.8.1 Detailed Description

Class `GraphShaderData` responsible for set up and controll of Graph.

Definition at line 8 of file GraphShaderData.cs.

4.8.2 Member Enumeration Documentation

4.8.2.1 lineVariantEnum

`enum GraphShaderData.lineVariantEnum`

Enumerator

Normal	
AllwaysConnected	
Catmull	

Definition at line 58 of file GraphShaderData.cs.

4.8.2.2 pointShapeEnum

`enum GraphShaderData.pointShapeEnum`

Enumerator

Square	
Circle	

Definition at line 40 of file GraphShaderData.cs.

4.8.3 Member Function Documentation

4.8.3.1 AddPointToGraph()

```
void GraphShaderData.AddPointToGraph (
             int graphIndex,
             Vector2 point )
```

Method `AddPointToGraph` adds new point to graph of given index.

Parameters

graphIndex	Index of graph to add point to
point	Point to add to the graph

Definition at line 261 of file GraphShaderData.cs.

4.8.3.2 CopyValuesFrom()

```
void GraphShaderData.CopyValuesFrom (
             GraphShaderData input )
```

Method `CopyValuesFrom` copy value from other `GraphShaderData` object.

Parameters

input	Object of `GraphShaderData` class to copy form

Definition at line 211 of file GraphShaderData.cs.

4.8.3.3 GetClosestPiontIndex()

```
int GraphShaderData.GetClosestPiontIndex (
             float x,
             float y )
```

Definition at line 146 of file GraphShaderData.cs.

4.8.3.4 SetOffsetOnX()

```
void GraphShaderData.SetOffsetOnX (
             float offset )
```

Method `SetOffsetOnX` set `graphOffsetOnX` variable.

Parameters

offset	Value of new offset

Definition at line 202 of file GraphShaderData.cs.

4.8.3.5 SetOffsetOnY()

```
void GraphShaderData.SetOffsetOnY (
             float offset )
```

Method `SetOffsetOnY` set `graphOffsetOnY` variable.

Parameters

offset	Value of new offset

Definition at line 193 of file GraphShaderData.cs.

4.8.3.6 SetScaleOnX()

```
void GraphShaderData.SetScaleOnX (
             float scale )
```

Method `SetScaleOnX` set `graphScaleOnX` variable.

Parameters

scale	Value of new scale

Definition at line 184 of file GraphShaderData.cs.

4.8.3.7 SetScaleOnY()

```
void GraphShaderData.SetScaleOnY (
             float scale )
```

Method `SetScaleOnY` set `graphScaleOnY` variable.

Parameters

scale	Value of new scale

Definition at line 175 of file GraphShaderData.cs.

4.8.4 Member Data Documentation

4.8.4.1 graphOffsetOnX

```
float GraphShaderData.graphOffsetOnX = -2f
```

Definition at line 16 of file GraphShaderData.cs.

4.8.4.2 graphOffsetOnXMax

```
float GraphShaderData.graphOffsetOnXMax = 6f
```

Definition at line 22 of file GraphShaderData.cs.

4.8.4.3 graphOffsetOnXMin

```
float GraphShaderData.graphOffsetOnXMin = -6f
```

Definition at line 19 of file GraphShaderData.cs.

4.8.4.4 graphOffsetOnY

```
float GraphShaderData.graphOffsetOnY = -2f
```

Definition at line 17 of file GraphShaderData.cs.

4.8.4.5 graphOffsetOnYMax

```
float GraphShaderData.graphOffsetOnYMax = 6f
```

Definition at line 23 of file GraphShaderData.cs.

4.8.4.6 graphOffsetOnYMin

```
float GraphShaderData.graphOffsetOnYMin = -6f
```

Definition at line 20 of file GraphShaderData.cs.

4.8.4.7 graphs

```
Graph [] GraphShaderData.graphs
```

Definition at line 71 of file GraphShaderData.cs.

4.8.4.8 graphScaleOnX

```
float GraphShaderData.graphScaleOnX = 16f
```

Definition at line 26 of file GraphShaderData.cs.

4.8.4.9 graphScaleOnXMax

```
float GraphShaderData.graphScaleOnXMax = 16f
```

Definition at line 32 of file GraphShaderData.cs.

4.8.4.10 graphScaleOnXMin

```
float GraphShaderData.graphScaleOnXMin = 4f
```

Definition at line 29 of file GraphShaderData.cs.

4.8.4.11 graphScaleOnY

```
float GraphShaderData.graphScaleOnY = 16f
```

Definition at line 27 of file GraphShaderData.cs.

4.8.4.12 graphScaleOnYMax

```
float GraphShaderData.graphScaleOnYMax = 16f
```

Definition at line 33 of file GraphShaderData.cs.

4.8.4.13 graphScaleOnYMin

```
float GraphShaderData.graphScaleOnYMin = 4f
```

Definition at line 30 of file GraphShaderData.cs.

4.8.4.14 int

```
GraphShaderData.int
```

Method `GetClosestPiontIndex` give coordinates of closest point on graph.

Parameters

x	Coordinate on X
y	Coordinate on Y

Returns

Returns coordinates of closest point

Definition at line 146 of file GraphShaderData.cs.

4.8.4.15 lineColor

```
Color GraphShaderData.lineColor = Color.black
```

Definition at line 67 of file GraphShaderData.cs.

4.8.4.16 lineSize

```
float GraphShaderData.lineSize = 0.2f
```

Definition at line 68 of file GraphShaderData.cs.

4.8.4.17 lineVariant

```
lineVariantEnum GraphShaderData.lineVariant = lineVariantEnum.Normal
```

Definition at line 66 of file GraphShaderData.cs.

4.8.4.18 maxPointAmount

```
int GraphShaderData.maxPointAmount = 255
```

Definition at line 55 of file GraphShaderData.cs.

4.8.4.19 maxXPointValue

```
float GraphShaderData.maxXPointValue = 9f
```

Definition at line 50 of file GraphShaderData.cs.

4.8.4.20 maxYPointValue

```
float GraphShaderData.maxYPointValue = 9f
```

Definition at line 52 of file GraphShaderData.cs.

4.8.4.21 meshRenderer

```
MeshRenderer GraphShaderData.meshRenderer
```

Definition at line 14 of file GraphShaderData.cs.

4.8.4.22 minPointAmount

```
int GraphShaderData.minPointAmount = 0
```

Definition at line 54 of file GraphShaderData.cs.

4.8.4.23 minXPointValue

```
float GraphShaderData.minXPointValue = -1f
```

Definition at line 49 of file GraphShaderData.cs.

4.8.4.24 minYPointValue

```
float GraphShaderData.minYPointValue = -1f
```

Definition at line 51 of file GraphShaderData.cs.

4.8.4.25 pointColor

```
Color GraphShaderData.pointColor = Color.black
```

Definition at line 47 of file GraphShaderData.cs.

4.8.4.26 pointShape

```
pointShapeEnum GraphShaderData.pointShape = pointShapeEnum.Square
```

Definition at line 46 of file GraphShaderData.cs.

4.8.4.27 pointSize

```
float GraphShaderData.pointSize = 0.2f
```

Definition at line 48 of file GraphShaderData.cs.

4.8.4.28 unitPerGridOnX

```
float GraphShaderData.unitPerGridOnX = 1f
```

Definition at line 37 of file GraphShaderData.cs.

4.8.4.29 unitPerGridOnY

```
float GraphShaderData.unitPerGridOnY = 1f
```

Definition at line 38 of file GraphShaderData.cs.

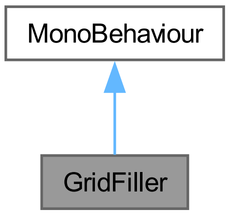
The documentation for this class was generated from the following file:

- GraphScripts/GraphShaderData.cs

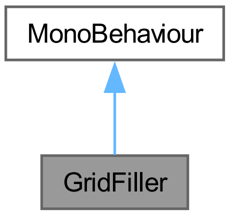
4.9 GridFiller Class Reference

Class `GridFiller` responsible for filling of `GraphGrid` with right number of `GraphSection` prefabs.

Inheritance diagram for GridFiller:



Collaboration diagram for GridFiller:



4.9.1 Detailed Description

Class `GridFiller` responsible for filling of `GraphGrid` with right number of `GraphSection` prefabs.

Definition at line 7 of file GridFiller.cs.

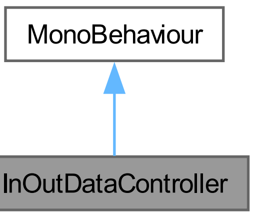
The documentation for this class was generated from the following file:

- GlobalScripts/GridFiller.cs

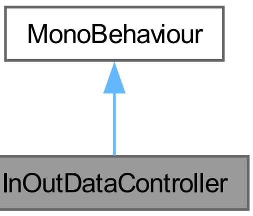
4.10 InOutDataController Class Reference

Class `InOutDataController` responsible for importing, saving and exporting data.

Inheritance diagram for InOutDataController:



Collaboration diagram for InOutDataController:



Public Member Functions

- bool CheckIfSaved (string nameOfIon)

  Method `CheckIfSaved` provide option to check if there is a save file for given ion name.
- SaveData LoadIonData (string nameOfIon)

  Method `LoadIonData` provide option to load data from save file of given ion.

Public Attributes

- List< IonData > ionData

Properties

- static InOutDataController instance `[get]`

  Instance of `InOutDataController` class.

4.10.1 Detailed Description

Class `InOutDataController` responsible for importing, saving and exporting data.

Definition at line 13 of file InOutDataController.cs.

4.10.2 Member Function Documentation

4.10.2.1 CheckIfSaved()

```
bool InOutDataController.CheckIfSaved (
             string nameOfIon )
```

Method `CheckIfSaved` provide option to check if there is a save file for given ion name.

Parameters

nameOfIon	Name of ion to check save for

Returns

A bool with value of "true" if there is a save or "false" if there is not

Definition at line 58 of file InOutDataController.cs.

4.10.2.2 LoadIonData()

```
SaveData InOutDataController.LoadIonData (
             string nameOfIon )
```

Method `LoadIonData` provide option to load data from save file of given ion.

Parameters

nameOfIon	Name of ion to load save data

Returns

`SaveData` object that contain save data for given ion name or returns `null` if there is no save file for this ion name

Definition at line 78 of file InOutDataController.cs.

4.10.3 Member Data Documentation

4.10.3.1 ionData

```
List<IonData> InOutDataController.ionData
```

Definition at line 32 of file InOutDataController.cs.

4.10.4 Property Documentation

4.10.4.1 instance

```
InOutDataController InOutDataController.instance  [static], [get]
```

Instance of `InOutDataController` class.

Definition at line 18 of file InOutDataController.cs.

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

- GlobalScripts/InOutDataController.cs

4.11 IonData Class Reference

Class `IonData` is a data class used in process of deserialization of ions data.

Public Attributes

- string name

  Variable `name` contains name of ion.
- int laserAmount

  Variable `laserAmount` contains number of lasers in configuration for this ion.

4.11.1 Detailed Description

Class `IonData` is a data class used in process of deserialization of ions data.

Definition at line 7 of file IonData.cs.

4.11.2 Member Data Documentation

4.11.2.1 laserAmount

```
int IonData.laserAmount
```

Variable `laserAmount` contains number of lasers in configuration for this ion.

Definition at line 16 of file IonData.cs.

4.11.2.2 name

```
string IonData.name
```

Variable `name` contains name of ion.

Definition at line 12 of file IonData.cs.

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

- DataScripts/IonData.cs

4.12 Laser Struct Reference

Class `Laser` is a data class used to describe one laser.

Public Attributes

- Vector2[ ] points
  Variable `points` contains points of laser.

4.12.1 Detailed Description

Class `Laser` is a data class used to describe one laser.

Definition at line 8 of file Laser.cs.

4.12.2 Member Data Documentation

4.12.2.1 points

```
Vector2 [] Laser.points
```

Variable `points` contains points of laser.

Definition at line 13 of file Laser.cs.

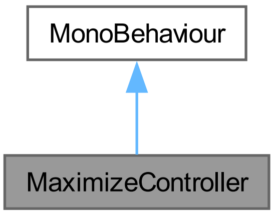
The documentation for this struct was generated from the following file:

- DataScripts/Laser.cs

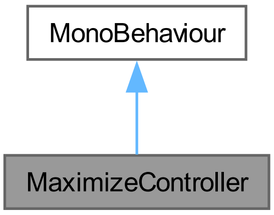
4.13 MaximizeController Class Reference

Class `MaximizeController` control process of maximization and minimization of graph sections.

Inheritance diagram for MaximizeController:



Collaboration diagram for MaximizeController:



Public Member Functions

- void MaximizeSection ()

  Method `MaximizeSection` maximize choseen sections or minimize bigger section if its shown.

Properties

- static MaximizeController instance `[get]`

  Instance of `MaximizeController` class.

4.13.1 Detailed Description

Class `MaximizeController` control process of maximization and minimization of graph sections.

Definition at line 9 of file MaximizeController.cs.

4.13.2 Member Function Documentation

4.13.2.1 MaximizeSection()

```
void MaximizeController.MaximizeSection ( )
```

Method `MaximizeSection` maximize choseen sections or minimize bigger section if its shown.

Definition at line 34 of file MaximizeController.cs.

4.13.3 Property Documentation

4.13.3.1 instance

```
MaximizeController MaximizeController.instance  [static], [get]
```

Instance of `MaximizeController` class.

Definition at line 14 of file MaximizeController.cs.

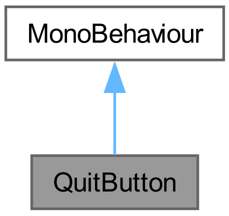
The documentation for this class was generated from the following file:

- GlobalScripts/MaximizeController.cs

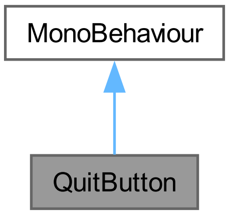
4.14 QuitButton Class Reference

Class `QuitButton` used to close application.

Inheritance diagram for QuitButton:



Collaboration diagram for QuitButton:



Public Member Functions

- void QuitApplication ()

  Method `QuitApplication` quits application.

4.14.1 Detailed Description

Class `QuitButton` used to close application.

Definition at line 6 of file QuitButton.cs.

4.14.2 Member Function Documentation

4.14.2.1 QuitApplication()

```
void QuitButton.QuitApplication ( )
```

Method `QuitApplication` quits application.

Definition at line 11 of file QuitButton.cs.

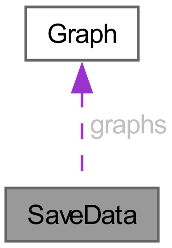
The documentation for this class was generated from the following file:

- InteractableScripts/QuitButton.cs

4.15 SaveData Class Reference

Class `SaveData` is a data class used in process of serialization of save data.

Collaboration diagram for SaveData:



Public Attributes

- string IonName

    Variable `IonName` contains name of saved ion.
- int numberOfLasers

    Variable `piTime` contains number of lasers in saved ion.
- float piTime

    Variable `piTime` contains value of pitime parameter.
- Graph[ ] graphs

    Variable `graphs` contains array of graphs in given saved ion.

4.15.1 Detailed Description

Class `SaveData` is a data class used in process of serialization of save data.

Definition at line 7 of file SaveData.cs.

4.15.2 Member Data Documentation

4.15.2.1 graphs

```
Graph [] SaveData.graphs
```

Variable `graphs` contains array of graphs in given saved ion.

Definition at line 24 of file SaveData.cs.

4.15.2.2 IonName

```
string SaveData.IonName
```

Variable `IonName` contains name of saved ion.

Definition at line 12 of file SaveData.cs.

4.15.2.3 numberOfLasers

```
int SaveData.numberOfLasers
```

Variable `piTime` contains number of lasers in saved ion.

Definition at line 16 of file SaveData.cs.

4.15.2.4 piTime

```
float SaveData.piTime
```

Variable `piTime` contains value of pitime parameter.

Definition at line 20 of file SaveData.cs.

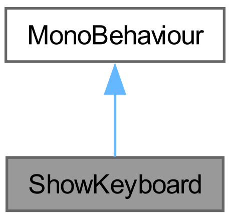
The documentation for this class was generated from the following file:

- DataScripts/SaveData.cs

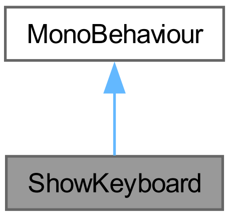
4.16 ShowKeyboard Class Reference

Class `ShowKeyboard` shows and hides interactable keyboard.

Inheritance diagram for ShowKeyboard:



Collaboration diagram for ShowKeyboard:



Public Member Functions

- void OpenKeyboard ()

    Method `OpenKeyboard` *shows interactable keyboard.*
- void SetCaretColorAlpha (float alpha)

    Method `SetCaretColorAlpha` *setting caret color alpha value to given value.*

4.16.1 Detailed Description

Class `ShowKeyboard` shows and hides interactable keyboard.

Definition at line 8 of file ShowKeyboard.cs.

4.16.2 Member Function Documentation

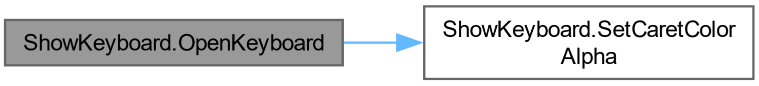
4.16.2.1 OpenKeyboard()

`void ShowKeyboard.OpenKeyboard ( )`

Method `OpenKeyboard` shows interactable keyboard.

Definition at line 28 of file ShowKeyboard.cs.

Here is the call graph for this function:



4.16.2.2 SetCaretColorAlpha()

```
void ShowKeyboard.SetCaretColorAlpha (
             float alpha )
```

Method `SetCaretColorAlpha` setting caret color alpha value to given value.

Parameters

alpha	Value of alpha in color to set

Definition at line 60 of file ShowKeyboard.cs.

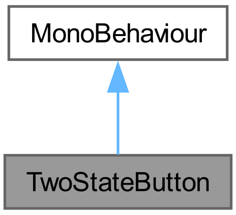
The documentation for this class was generated from the following file:

- InteractableScripts/ShowKeyboard.cs

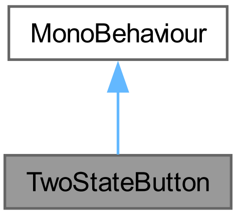
4.17 TwoStateButton Class Reference

Class `TwoStateButton` is responsible to controll two state button interaction.

Inheritance diagram for TwoStateButton:



Collaboration diagram for TwoStateButton:



Public Member Functions

- void ChangeValue ()

  Method `ChangeValue` *changes value of button to opposite.*
- bool GetValue ()

  Method `GetValue` *returns value of button.*
- void SetValue (bool newValue)

  Method `SetValue` *set value of button to given value.*

4.17.1 Detailed Description

Class `TwoStateButton` is responsible to controll two state button interaction.

Definition at line 7 of file TwoStateButton.cs.

4.17.2 Member Function Documentation

4.17.2.1 ChangeValue()

```
void TwoStateButton.ChangeValue ( )
```

Method `ChangeValue` changes value of button to opposite.

Definition at line 26 of file TwoStateButton.cs.

4.17.2.2 GetValue()

```
bool TwoStateButton.GetValue ( )
```

Method `GetValue` returns value of button.

Returns

Definition at line 36 of file TwoStateButton.cs.

4.17.2.3 SetValue()

```
void TwoStateButton.SetValue (
             bool newValue )
```

Method `SetValue` set value of button to given value.

Parameters

newValue	New value to set

Definition at line 47 of file TwoStateButton.cs.

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

- InteractableScripts/TwoStateButton.cs

Chapter 5

File Documentation

5.1 DataScripts/ConfigData.cs File Reference

Classes

- class ConfigData

  Class `ConfigData` is a data class used in process of deserialization of config data.

5.2 ConfigData.cs

Go to the documentation of this file.

```
using System;
[Serializable]
public class ConfigData
{
    public string pathToSave;
    public string pathToExport;
    public string pathToIonsFile;
}

```

5.3 DataScripts/ExportData.cs File Reference

Classes

- class ExportData

  Class `ExportData` is a data class used in process of serialization of export data.

5.4 ExportData.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;

[Serializable]
public class ExportData
{
    public string IonName;
    public float piTime;
    public List<Laser> lasers;
}
```

5.5 DataScripts/Graph.cs File Reference

Classes

- struct Graph

  Class Graph is a data class used to describe one graph.

5.6 Graph.cs

Go to the documentation of this file.

```
using System;
using UnityEngine;

[Serializable]
public struct Graph
{
    public Color pointColor;
    public Color lineColor;
    public Vector2[] points;
}
```

5.7 DataScripts/IonData.cs File Reference

Classes

- class IonData

  Class IonData is a data class used in process of deserialization of ions data.

5.8 IonData.cs

Go to the documentation of this file.

```
using System;

[Serializable]
public class IonData
{
    public string name;
    public int laserAmount;
}
```

5.9 DataScripts/Laser.cs File Reference

Classes

- struct Laser

  Class Laser is a data class used to describe one laser.

5.10 Laser.cs

Go to the documentation of this file.
```
using System;
using UnityEngine;

[Serializable]
public struct Laser
{
    public Vector2[] points;
}
```

5.11 DataScripts/SaveData.cs File Reference

Classes

- class SaveData

  Class `SaveData` is a data class used in process of serialization of save data.

5.12 SaveData.cs

Go to the documentation of this file.
```
using System;

[Serializable]
public class SaveData
{
    public string IonName;
    public int numberOfLasers;
    public float piTime;
    public Graph[] graphs;
}
```

5.13 GlobalScripts/GridFiller.cs File Reference

Classes

- class GridFiller

  Class `GridFiller` responsible for filling of `GraphGrid` with right number of `GraphSection` prefabs.

5.14 GridFiller.cs

Go to the documentation of this file.
```
using TMPro;
using UnityEngine;

public class GridFiller : MonoBehaviour
{
    [SerializeField] private GameObject sectionPrefab;
    [SerializeField] private TMP_Dropdown ionDropdown;

    private void Start()
    {
        ionDropdown.onValueChanged.AddListener(Fill);
        Fill(ionDropdown.value);
    }

    private void Fill(int arg0)
    {
        int numberOfElements = InOutDataController.instance.ionData[arg0].laserAmount;
        string ionName = ionDropdown.options[arg0].text;
        SaveData saveData = null;
        if (InOutDataController.instance.CheckIfSaved(ionName))
        {
            saveData = InOutDataController.instance.LoadIonData(ionName);
        }

        foreach (Transform child in transform)
        {
            GameObject.Destroy(child.gameObject);
        }
        for(int i = 0;i< numberOfElements; i++)
        {
            GameObject child =
Instantiate(sectionPrefab,transform.position,Quaternion.identity,transform);
            child.GetComponentInChildren<GraphShaderData>().graphs[0].lineColor = new Color((float)i /
numberOfElements, 0, (float)(numberOfElements- i) / numberOfElements, 1);
            if(saveData != null)
            {
                child.GetComponentInChildren<GraphShaderData>().graphs[0].points =
saveData.graphs[i].points;
            }

        }
    }

}
```

5.15 GlobalScripts/InOutDataController.cs File Reference

Classes

- class InOutDataController

    Class `InOutDataController` responsible for importing, saving and exporting data.

5.16 InOutDataController.cs

Go to the documentation of this file.

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class InOutDataController : MonoBehaviour
{
    public static InOutDataController instance { get; private set; }

    [SerializeField] private GameObject gridSection;
    [SerializeField] private XRSimpleInteractable buttonSave;
    [SerializeField] private XRSimpleInteractable buttonExport;
    [SerializeField] private TMP_Dropdown dropdownIon;
    [SerializeField] private TMP_InputField inputPitime;
    [SerializeField] private TMP_Dropdown dropdownPitimeUnit;
    [SerializeField] private TMP_InputField inputIonName;
    [SerializeField] private TwoStateButton buttonNewIon;
    [SerializeField] private GameObject newMenu;
    [SerializeField] private string pathToSave;
    [SerializeField] private string pathToExport;
    [SerializeField] private string pathToIonsFile;
    public List<IonData> ionData;


    void Awake()
    {
        if (instance != null && instance != this)
            Destroy(this);
        else
            instance = this;

        dropdownIon.value = 0;
        buttonSave.selectEntered.AddListener(x => SaveIon());
        buttonExport.selectEntered.AddListener(x => ExportIon());
        InitializeConfigData();
        InitializeDropdownIon();
    }
    public bool CheckIfSaved(string nameOfIon)
    {
        string pathToIonDirectory = pathToSave + "/" + nameOfIon;
        if(Directory.Exists(pathToIonDirectory))
        {
            if(File.Exists(pathToIonDirectory +"/Ion.json"))
            {
                return true;
            }
        }
        return false;
    }

    public SaveData LoadIonData(string nameOfIon)
    {
        string pathToIonDirectory = pathToSave + "/" + nameOfIon;
        if (Directory.Exists(pathToIonDirectory))
        {
            string filePath = pathToIonDirectory + "/Ion.json";
            if (File.Exists(filePath))
            {
                string json = File.ReadAllText(filePath);
                return JsonConvert.DeserializeObject<SaveData>(json);
            }
        }
        return null;
    }

    private void ExportIon()
    {
        string nameOfIon = (buttonNewIon.GetValue()) ? inputIonName.text : dropdownIon.options[dropdownIon.value].text;

        string pathToIonDirectory = pathToExport + "/" + nameOfIon;
        string pathToIonFile = pathToIonDirectory + "/" + "Ion.json";
        float piTimeValue = float.Parse(inputPitime.text) * (float)Math.Pow(10, -3 * (dropdownPitimeUnit.value + 1));
        List<GraphShaderData> data = gridSection.GetComponentsInChildren<GraphShaderData>().ToList();

        ExportData exportData = new ExportData();
        exportData.IonName = nameOfIon;
        exportData.piTime = piTimeValue;
        exportData.lasers = new List<Laser>();

        int juliaIndex = 1;
        string julliaScript = "include(\""+ Directory.GetCurrentDirectory().Replace(’\\’,’/’) + "/scripts/laserValueBaseScript.jl\")\r\n" + "ionFile = \"" + pathToIonFile.Replace(’\\’, ’/’)+ "\"\r\n";
        string pathToJuliaScript = pathToIonDirectory + "/" + nameOfIon + "Lasers.jl";
        foreach (var graph in data)
        {
            List<Vector2> tmpPoints = new List<Vector2>{ new Vector2(0, 0) };
            tmpPoints = tmpPoints.Concat(graph.graphs[0].points.ToList().Distinct().OrderBy(v => v.x)).ToList();
            tmpPoints.Add(new Vector2(tmpPoints.Last().x+1.0f,tmpPoints.Last().y));

            Vector2[] points = tmpPoints.ToArray();
            for (int i = 0; i < points.Length; i++)
            {
                points[i].x = points[i].x / 16.0f * piTimeValue;
                points[i].y = points[i].y / 8.0f;
            }
            Laser laser = new Laser();
            laser.points = points;
            exportData.lasers.Add(laser);
            julliaScript += "function Laser" + juliaIndex + "(timeValue::Float64)\r\n    laserIndex = "+ juliaIndex + "\r\n    return laserValue(laserIndex,timeValue,ionFile)\r\nend\r\n";
            juliaIndex++;

        }

        string outputJSON = JsonConvert.SerializeObject(exportData, Formatting.Indented);

        if (!Directory.Exists(pathToIonDirectory))
        {
            Directory.CreateDirectory(pathToIonDirectory);
            File.WriteAllText(pathToJuliaScript, julliaScript);
            File.WriteAllText(pathToIonFile, outputJSON);
        }
        else
        {
            File.WriteAllText(pathToJuliaScript, julliaScript);
            File.WriteAllText(pathToIonFile, outputJSON);
        }
    }

    private void InitializeDropdownIon()
    {
        ionData = JsonConvert.DeserializeObject<IonData[]>(File.ReadAllText(pathToIonsFile)).ToList();
        dropdownIon.options.Clear();
        dropdownIon.value = 0;
        foreach (var ion in ionData) {
            dropdownIon.options.Add(new TMP_Dropdown.OptionData(ion.name));
        }
    }

    private void InitializeConfigData()
    {
        ConfigData confData =
JsonConvert.DeserializeObject<ConfigData>(File.ReadAllText(Directory.GetCurrentDirectory() +
"/config/config.json"));
        Debug.Log(Directory.GetCurrentDirectory() + "/config/config.json");
        Debug.Log(confData.pathToIonsFile);

        pathToExport = confData.pathToExport;
        pathToSave = confData.pathToSave;
        pathToIonsFile = confData.pathToIonsFile;
    }

    private void AddDropdownIon(string newOption, int numberofLasers)
    {
        IonData newIon = new IonData();
        newIon.name = newOption;
        newIon.laserAmount = numberofLasers;
        ionData.Add(newIon);

        string outputJSON = JsonConvert.SerializeObject(ionData, Formatting.Indented);
        File.WriteAllText(pathToIonsFile, outputJSON);
        dropdownIon.options.Add(new TMP_Dropdown.OptionData(newOption));
        dropdownIon.value = dropdownIon.options.Count - 1;

    }

    private void SaveIon()
    {
        string nameOfIon = (buttonNewIon.GetValue()) ? inputIonName.text :
dropdownIon.options[dropdownIon.value].text;
        string pathToIonDirectory = pathToSave + "/" + nameOfIon;
        float piTimeValue = float.Parse(inputPitime.text) * (float)Math.Pow(10, -3 *
(dropdownPitimeUnit.value + 1));


        List<GraphShaderData> data = gridSection.GetComponentsInChildren<GraphShaderData>().ToList();
        List<Graph> list = new List<Graph>();
        foreach (var graph in data)
        {
            list.Add(graph.graphs[0]);
        }

        SaveData saveData = new SaveData();
        saveData.IonName = nameOfIon;
        saveData.numberOfLasers = list.Count;
        saveData.piTime = piTimeValue;
        saveData.graphs = list.ToArray();

        string outputJSON = JsonConvert.SerializeObject(saveData, Formatting.Indented);

        string pathToIonFile = pathToIonDirectory + "/" + "Ion.json";

        if (!Directory.Exists(pathToIonDirectory))
        {
            Directory.CreateDirectory(pathToIonDirectory);
            File.WriteAllText(pathToIonFile, outputJSON);
            if (buttonNewIon.GetValue()) AddDropdownIon(nameOfIon, saveData.graphs.Length);
        }
        else
        {
            File.WriteAllText(pathToIonFile, outputJSON);
        }

    }

    void Update()
    {
        if (buttonNewIon.GetValue())
        {
            newMenu.SetActive(true);
        }
        else
        {
            newMenu.SetActive(false);
        }

    }
}
```

5.17 GlobalScripts/MaximizeController.cs File Reference

Classes

- class MaximizeController

  Class `MaximizeController` control process of maximization and minimization of graph sections.

5.18 MaximizeController.cs

Go to the documentation of this file.

```
using System.Collections.Generic;
using Unity.XR.CoreUtils;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class MaximizeController : MonoBehaviour
{
    public static MaximizeController instance { get; private set; }


    [SerializeField] private GameObject multipleGraphSection;
    [SerializeField] private GameObject gridSection;
    [SerializeField] private GraphShaderData graphShaderData;
    [SerializeField] private bool multipleGraphShown = false;
    List<int> indexesList;

    void Awake()
    {
        if (instance != null && instance != this)
            Destroy(this);
        else
            instance = this;
    }

    public void MaximizeSection()
    {
        if (multipleGraphShown)
        {
            List<GameObject> list = new List<GameObject>();
            gridSection.GetChildGameObjects(list);
            GraphShaderData multipleshaderData =
multipleGraphSection.GetComponentInChildren<GraphControll>().GetShaderData();
            int graphIndex = 0;
            foreach(int i in indexesList)
            {

                GraphShaderData shaderData =
list[i].GetComponentInChildren<GraphControll>().GetShaderData();
                Graph graph = multipleshaderData.graphs[graphIndex];
                shaderData.graphs[0].points = graph.points;
                shaderData.lineColor = graph.lineColor;
                shaderData.pointColor = graph.pointColor;
                graphIndex++;
            }
            gridSection.SetActive(true);
            multipleGraphSection.SetActive(false);
            multipleGraphShown = false;
        }
        else
        {
            List<GameObject> list = new List<GameObject>();
            gridSection.GetChildGameObjects(list);
            List<Graph> graphs = new List<Graph>();
            indexesList = new List<int>();
            for (int i = 0;i<list.Count;i++)
            {
                bool choosen =
list[i].GetNamedChild("ButtonChoose").GetComponentInChildren<TwoStateButton>().GetValue();
                if(choosen)
                {
                    GraphShaderData shaderData =
list[i].GetComponentInChildren<GraphControll>().GetShaderData();
                    Graph graph = new Graph();
                    graph.points = shaderData.graphs[0].points;
                    graph.lineColor = shaderData.graphs[0].lineColor;
                    graph.pointColor = shaderData.graphs[0].pointColor;
                    graphs.Add(graph);
                    indexesList.Add(i);
                }
            }
            graphShaderData.graphs = graphs.ToArray();

            gridSection.SetActive(false);
            multipleGraphSection.SetActive(true);
            multipleGraphShown = true;
        }
    }
}
```

5.19 GraphScripts/GraphControll.cs File Reference

Classes

- class GraphControll

  Class `GraphControll` controls and manages interactions and actions of user with graph.

5.20 GraphControll.cs

Go to the documentation of this file.

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
using Unity.Mathematics;
using UnityEngine.XR.Content.Interaction;

public class GraphControll : MonoBehaviour
{
    [SerializeField] private XRKnob knobScaleX;
    [SerializeField] private XRSimpleInteractable buttonBackOffset;
    [SerializeField] private XRSimpleInteractable buttonNextOffset;
    [SerializeField] private XRSimpleInteractable buttonModeChange;
    [SerializeField] private GameObject buttonModeChangeGO;
    [SerializeField] private GameObject buttonModeChangeDescGO;
    [SerializeField] private XRSimpleInteractable simpleInteractable;


    MeshCollider meshCollider;
    IXRSelectInteractor xrInteractor;
    GraphShaderData graphShaderData;
    bool createMode = false;
    bool attached = false;
    int grapchIndex;
    int pointIndex;
    float nowMinX;
    float nowMaxX;
    float nowMinY;
    float nowMaxY;

    void Start()
    {
        simpleInteractable = GetComponent<XRSimpleInteractable>();
        meshCollider = GetComponent<MeshCollider>();
        graphShaderData = GetComponent<GraphShaderData>();

        simpleInteractable.selectEntered.AddListener(Attach);
        simpleInteractable.selectExited.AddListener(DisAttach);

        knobScaleX.onValueChange.AddListener(ScaleChangeX);

        nowMinX = graphShaderData.graphOffsetOnX;
        nowMaxX = graphShaderData.graphOffsetOnX + graphShaderData.graphScaleOnX;
        nowMinY = graphShaderData.graphOffsetOnY;
        nowMaxY = graphShaderData.graphOffsetOnY + graphShaderData.graphScaleOnY;

        buttonBackOffset.selectEntered.AddListener(OffsetBack);
        buttonNextOffset.selectEntered.AddListener(OffsetNext);

        buttonModeChange.selectEntered.AddListener(ChangeMode);
        buttonModeChange.GetComponent<TwoStateButton>().SetValue(createMode);

    }

    private void ChangeMode(SelectEnterEventArgs arg0)
    {
        createMode = !createMode;
    }

    private void OffsetNext(SelectEnterEventArgs arg0)
    {
        int addition = 16;
        if (graphShaderData.graphScaleOnX == 12)
        {
            addition = 8;
        }
        else if(graphShaderData.graphScaleOnX == 8)
        {
            addition = 4;
        }
        else if(graphShaderData.graphScaleOnX == 6)
        {
            addition = 2;
        }
        float newOffset = graphShaderData.graphOffsetOnX + addition;
        graphShaderData.SetOffsetOnX(newOffset);
    }
    private void OffsetBack(SelectEnterEventArgs arg0)
    {
        int addition = 16;
        if (graphShaderData.graphScaleOnX == 12)
        {
            addition = 8;
        }
        else if (graphShaderData.graphScaleOnX == 8)
        {
            addition = 4;
        }
        else if (graphShaderData.graphScaleOnX == 6)
        {
            addition = 2;
        }
        float newOffset = graphShaderData.graphOffsetOnX - addition;
        if(newOffset >= -2)graphShaderData.SetOffsetOnX(newOffset);
    }

    private void ScaleChangeX(float arg0)
    {

        int exponent = (int)math.remap(0, 1, 1, 6, knobScaleX.value);
        float newScale = 4 + math.pow(2, exponent);

       if(graphShaderData.graphScaleOnX <=12)
        {
            float temp = (graphShaderData.graphOffsetOnX + 2);
            //Correction of offset
            if (temp % (newScale-4) != 0)
                graphShaderData.graphOffsetOnX = graphShaderData.graphOffsetOnX - temp;

        }

        graphShaderData.SetScaleOnX(newScale);
    }

    private void DisAttach(SelectExitEventArgs arg0)
    {
        attached = false;
        xrInteractor = null;
    }

    private void Attach(SelectEnterEventArgs arg0)
    {
        Vector3 inteactionPoint =
meshCollider.ClosestPointOnBounds(arg0.interactorObject.transform.position);
        double interactorX = math.remap(-0.5f, 0.5f, nowMinX, nowMaxX,
transform.InverseTransformPoint(inteactionPoint).x);
        double interactorY = math.remap(-0.5f, 0.5f, nowMinY, nowMaxY, transform.InverseTransformPoint(inteactionPoint).y);

        (grapchIndex, pointIndex) = graphShaderData.GetClosestPiontIndex((float)interactorX, (float)interactorY);

        xrInteractor = arg0.interactorObject;

        if (createMode)
        {
            graphShaderData.AddPointToGraph(0,new Vector2((float)interactorX, (float)interactorY));
        }
        else
        {
            attached = true;
        }




    }


    void Update()
    {
        nowMinX = graphShaderData.graphOffsetOnX;
        nowMaxX = graphShaderData.graphOffsetOnX + graphShaderData.graphScaleOnX;
        nowMinY = graphShaderData.graphOffsetOnY;
        nowMaxY = graphShaderData.graphOffsetOnY + graphShaderData.graphScaleOnY;

        if (attached)
        {
            Vector3 inteactionPoint = meshCollider.ClosestPointOnBounds(xrInteractor.transform.position);
            double interactorX = math.remap(-0.5f, 0.5f, nowMinX, nowMaxX, transform.InverseTransformPoint(inteactionPoint).x);
            double interactorY = math.remap(-0.5f, 0.5f, nowMinY, nowMaxY, transform.InverseTransformPoint(inteactionPoint).y);


            graphShaderData.graphs[grapchIndex].points[pointIndex].x = (float)interactorX;
            graphShaderData.graphs[grapchIndex].points[pointIndex].y = (float)interactorY;
        }

        if(graphShaderData.graphs.Length != 1 )
        {
            createMode = false;
            buttonModeChange.GetComponent<TwoStateButton>().SetValue(false);
            buttonModeChangeGO.SetActive(false);
            buttonModeChangeDescGO.SetActive(false);
        }
        else
        {
            buttonModeChangeGO.SetActive(true);
            buttonModeChangeDescGO.SetActive(true);
        }
    }

    public GraphShaderData GetShaderData()
    {
        return graphShaderData;
    }


}
```

5.21 GraphScripts/GraphMaximize.cs File Reference

Classes

- class GraphMaximize

    Class `GraphMaximize` hepl with process of maximization and minimization of graph section.

5.22 GraphMaximize.cs

Go to the documentation of this file.

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class GraphMaximize : MonoBehaviour
{
    [SerializeField] private TwoStateButton chooseButton;
    [SerializeField] private XRSimpleInteractable interactable;

    // Start is called before the first frame update
    void Start()
    {
        interactable.selectEntered.AddListener(x => StartMaximize());
    }

    void StartMaximize()
    {
        chooseButton.SetValue(true);
        MaximizeController.instance.MaximizeSection();
    }
}
```

5.23 GraphScripts/GraphScaleDescControll.cs File Reference

Classes

- class GraphScaleDescControll

    Class `GraphScaleDescControll` controls graph scale descriptions.

5.24 GraphScaleDescControll.cs

Go to the documentation of this file.

```
using TMPro;
using UnityEngine;

public class GraphScaleDescControll : MonoBehaviour
{
    [SerializeField] private GraphShaderData shaderData;
    [SerializeField] private RectTransform scaleDesc1Transform;
    [SerializeField] private RectTransform scaleDesc3Transform;
    [SerializeField] private RectTransform scaleDescY0Transform;
    [SerializeField] private RectTransform scaleDescY1Transform;

    [SerializeField] private TextMeshProUGUI scaleDesc1;
    [SerializeField] private TextMeshProUGUI scaleDesc2;
    [SerializeField] private TextMeshProUGUI scaleDesc3;
    [SerializeField] private TextMeshProUGUI scaleDescAxiX;

    void Update()
    {
        float scaleGraphElement = 19.5f;
        float sizeOfUnit = scaleGraphElement / shaderData.graphScaleOnX;
        float scale = shaderData.graphScaleOnX - 4;

        float offset = shaderData.graphOffsetOnX + 2f;
        float startingText = offset / 8f;
        float startingTextPosition = -(scaleGraphElement/2) + sizeOfUnit * 2;

        float middleText = (offset + (scale / 2f)) / 8f;

        float lastText = (offset + scale) / 8f;
        float lastTextPosition = (scaleGraphElement / 2) - sizeOfUnit * 2;

        scaleDesc1.text = startingText + "PI";
        scaleDesc1Transform.localPosition = new Vector3(startingTextPosition,
scaleDesc1Transform.localPosition.y, scaleDesc1Transform.localPosition.z);
        scaleDesc2.text = middleText + "PI";
        scaleDesc3.text = lastText + "PI";
        scaleDesc3Transform.localPosition = new Vector3(lastTextPosition, scaleDesc3Transform.localPosition.y, scaleDesc3Transform.localPosition.z);

        float yTextPosition ;
        if (offset == 0  ) {
            yTextPosition = -(scaleGraphElement / 2) + sizeOfUnit * 1.5f;
        }
        else if( offset ==1)
        {
            yTextPosition = -(scaleGraphElement / 2) + sizeOfUnit * 0.5f;
        }
        else
        {
            yTextPosition = -(scaleGraphElement / 2) + sizeOfUnit * 0.25f; ;
        }

        scaleDescY0Transform.localPosition = new Vector3(yTextPosition, scaleDescY0Transform.localPosition.y, scaleDescY0Transform.localPosition.z);
        scaleDescY1Transform.localPosition = new Vector3(yTextPosition, scaleDescY1Transform.localPosition.y, scaleDescY1Transform.localPosition.z);

    }

}
```

5.25 GraphScripts/GraphShaderData.cs File Reference

Classes

- class GraphShaderData

    Class `GraphShaderData` responsible for set up and controll of Graph.

5.26 GraphShaderData.cs

Go to the documentation of this file.

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class GraphShaderData : MonoBehaviour
{


    private MaterialPropertyBlock materialPB;

    public MeshRenderer meshRenderer;

    public float graphOffsetOnX = -2f;
    public float graphOffsetOnY = -2f;

    public float graphOffsetOnXMin = -6f;
    public float graphOffsetOnYMin = -6f;

    public float graphOffsetOnXMax = 6f;
    public float graphOffsetOnYMax = 6f;


    public float graphScaleOnX = 16f;
    public float graphScaleOnY = 16f;

    public float graphScaleOnXMin = 4f;
    public float graphScaleOnYMin = 4f;

    public float graphScaleOnXMax = 16f;
    public float graphScaleOnYMax = 16f;



    public float unitPerGridOnX = 1f;
    public float unitPerGridOnY = 1f;

    public enum pointShapeEnum
    {
        Square = 1,
        Circle = 2
    }

    public pointShapeEnum pointShape = pointShapeEnum.Square;
    public Color pointColor = Color.black;
    public float pointSize = 0.2f;
    public float minXPointValue = -1f;
    public float maxXPointValue = 9f;
    public float minYPointValue = -1f;
    public float maxYPointValue = 9f;

    public int minPointAmount = 0;
    public int maxPointAmount = 255;


    public enum lineVariantEnum
    {
        Normal = 1,
        AllwaysConnected = 2,
        Catmull = 3
    }


    public lineVariantEnum lineVariant = lineVariantEnum.Normal;
    public Color lineColor = Color.black;
    public float lineSize = 0.2f;


    public Graph[] graphs;

    private void Start()
    {
        materialPB = new MaterialPropertyBlock();
    }
    void Update()
    {
        int desireLength = 0;
        foreach (Graph graph in graphs)
        {

            desireLength += graph.points.Distinct().ToArray().Length;
            desireLength += 3;
        }
        Texture2D input = new Texture2D(desireLength, 1, TextureFormat.RGBAFloat, false);
        input.filterMode = FilterMode.Point;
        input.wrapMode = TextureWrapMode.Clamp;
        int index = 0;
        for (int i = 0; i < graphs.Length; i++)
        {
            Vector2[] orderedPoints = graphs[i].points.Distinct().ToArray().OrderBy(v =>
v.x).ToArray<Vector2>();
            input.SetPixel(index, 0, graphs[i].pointColor);
            index++;
            input.SetPixel(index, 0, graphs[i].lineColor);
            index++;
            input.SetPixel(index, 0, new
Color(NormalizeValue(orderedPoints.Length,minPointAmount,maxPointAmount),0f,0f,1f));
            index++;



            for (int j = 0; j < orderedPoints.Length; j++, index++)
            {
                //Debug.Log(normalizeValue(orderedPoints[j].x, minXPointValue, maxXPointValue));
                input.SetPixel(index, 0, new
Color(NormalizeValue(orderedPoints[j].x,minXPointValue,maxXPointValue),
NormalizeValue(orderedPoints[j].y,minYPointValue,maxYPointValue), 1.0f,1.0f));
            }
        }
        input.Apply();
        meshRenderer.GetPropertyBlock(materialPB);
        materialPB.SetFloat("_ScaleX", graphScaleOnX);
        materialPB.SetFloat("_ScaleY", graphScaleOnY);
        materialPB.SetFloat("_OffsetX", graphOffsetOnX);
        materialPB.SetFloat("_OffsetY", graphOffsetOnY);
        materialPB.SetFloat("_UnitPerGridX", unitPerGridOnX);
        materialPB.SetFloat("_UnitPerGridY", unitPerGridOnY);
        materialPB.SetInteger("_LineVariant", (int)lineVariant);
        materialPB.SetColor("_LineColor", lineColor);
        materialPB.SetFloat("_LineSize", lineSize);
        materialPB.SetInteger("_PointShape", (int)pointShape);
        materialPB.SetColor("_PointColor", pointColor);
        materialPB.SetFloat("_PointSize", pointSize);
        materialPB.SetTexture("_GraphsTex", input);
        materialPB.SetInt("_GraphsAmount", graphs.Length);
        materialPB.SetInt("_TexAmount", desireLength);
        materialPB.SetInt("_MinPointsAmount", minPointAmount);
        materialPB.SetInt("_MaxPointsAmount", maxPointAmount);
        materialPB.SetFloat("_MinXValue", minXPointValue);
        materialPB.SetFloat("_MaxXValue", maxXPointValue);
        materialPB.SetFloat("_MinYValue", minYPointValue);
        materialPB.SetFloat("_MaxYValue", maxYPointValue);
        meshRenderer.SetPropertyBlock(materialPB);
    }

    public (int,int) GetClosestPiontIndex(float x, float y)
    {
        Vector2 point = new Vector2(x, y);

        double distance = double.MaxValue;
        int graphIndex = 0;
        int pointIndex = 0;
        for (int i = 0; i < graphs.Length; i++)
        {
            for (int j = 0;j < graphs[i].points.Length; j++)
            {
                if (Vector2.Distance(point, graphs[i].points[j]) < distance)
                {
                    distance = Vector2.Distance(point, graphs[i].points[j]);
                    graphIndex = i;
                    pointIndex = j;
                }
            }

        }
        Vector2 closestPoint = new Vector2(graphIndex, pointIndex);

        return (graphIndex, pointIndex);
    }

    public void SetScaleOnY(float scale)
    {
        graphScaleOnY = scale;
    }

    public void SetScaleOnX(float scale)
    {
        graphScaleOnX = scale;
    }

    public void SetOffsetOnY(float offset)
    {
        graphOffsetOnY = offset;
    }

    public void SetOffsetOnX(float offset)
    {
        graphOffsetOnX = offset;
    }

    public void CopyValuesFrom(GraphShaderData input)
    {
        graphOffsetOnX = input.graphOffsetOnX;
        graphOffsetOnXMin = input.graphOffsetOnXMin;
        graphOffsetOnXMax = input.graphOffsetOnXMax;
        graphOffsetOnY = input.graphOffsetOnY;
        graphOffsetOnYMin = input.graphOffsetOnYMin;
        graphOffsetOnYMax = input.graphOffsetOnYMax;

        graphScaleOnX = input.graphScaleOnX;
        graphScaleOnXMin = input.graphScaleOnXMin;
        graphScaleOnXMax = input.graphScaleOnXMax;
        graphScaleOnY = input.graphScaleOnY;
        graphScaleOnYMin = input.graphScaleOnYMin;
        graphScaleOnYMax = input.graphScaleOnYMax;

        unitPerGridOnX = input.unitPerGridOnX;
        unitPerGridOnY = input.unitPerGridOnY;

        pointShape = input.pointShape;
        pointColor = input.pointColor;
        pointSize = input.pointSize;
        minXPointValue = input.minXPointValue;
        maxXPointValue = input.maxXPointValue;
        minYPointValue = input.minYPointValue;
        maxYPointValue = input.maxYPointValue;

        lineColor = input.lineColor;
        lineSize = input.lineSize;
        lineVariant = input.lineVariant;

    }

    private float NormalizeValue(float value, float minValue, float maxValue)
    {
        return (value - minValue) / (maxValue - minValue);
    }

    public void AddPointToGraph(int graphIndex, Vector2 point)
    {
        List<Vector2> list = graphs[graphIndex].points.ToList<Vector2>();
        list.Add(point);
        graphs[graphIndex].points = list.ToArray();
        Debug.Log("Dodano punkt:" + point);
    }


}
```

5.27 InteractableScripts/ButtonFollowVisual.cs File Reference

Classes

- class ButtonFollowVisual

  Class `ButtonFollowVisual` *controls visual of interactable button.*

5.28 ButtonFollowVisual.cs

Go to the documentation of this file.

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class ButtonFollowVisual : MonoBehaviour
{
    [SerializeField] private Transform visualTarget;
    [SerializeField] private Vector3 localAxis;
    [SerializeField] private float resetSpeed =5f;
    [SerializeField] private float followAngleTreshold=45f;



    private bool freeze = false;

    private Vector3 initialLocalPosition;

    private Vector3 offset;
    private Transform pokeAttachTransform;

    private XRBaseInteractable interactable;
    private bool isFollowing = false;


    void Start()
    {
        initialLocalPosition = visualTarget.localPosition;
        interactable = GetComponent<XRBaseInteractable>();
        interactable.hoverEntered.AddListener(Follow);
        interactable.hoverExited.AddListener(ResetButton);
        interactable.selectEntered.AddListener(Freeze);
    }

    public void Follow(BaseInteractionEventArgs hover)
    {
        if(hover.interactorObject is XRPokeInteractor)
        {
            XRPokeInteractor interactor = (XRPokeInteractor)hover.interactorObject;

            pokeAttachTransform = interactor.attachTransform;
            offset = visualTarget.position - pokeAttachTransform.position;

            float pokeAngle = Vector3.Angle(offset, visualTarget.TransformDirection(localAxis));
            if (pokeAngle < followAngleTreshold)
            {
                isFollowing = true;
                freeze = false;
            }
        }
    }

    public void ResetButton(BaseInteractionEventArgs hover)
    {
        if (hover.interactorObject is XRPokeInteractor)
        {
            isFollowing = false;
            freeze = false;
        }
    }

    public void Freeze(BaseInteractionEventArgs hover)
    {
        if (hover.interactorObject is XRPokeInteractor)
        {
            freeze = true;
        }
    }


    void Update()
    {
        if(freeze)
        {
            return;
        }
        if(isFollowing)
        {
            Vector3 localTargetPosition =
visualTarget.InverseTransformPoint(pokeAttachTransform.position + offset);
            Vector3 constrainedLocalTargetPosition = Vector3.Project(localTargetPosition, localAxis);
            visualTarget.position = visualTarget.TransformPoint(constrainedLocalTargetPosition);
        }
        else
        {
            visualTarget.localPosition = Vector3.Lerp(visualTarget.localPosition,
initialLocalPosition,Time.deltaTime * resetSpeed);
        }
    }
}
```

5.29 InteractableScripts/QuitButton.cs File Reference

Classes

- class QuitButton

    Class `QuitButton` used to close application.

5.30 QuitButton.cs

Go to the documentation of this file.

```
using UnityEngine;

public class QuitButton : MonoBehaviour
{
    public void QuitApplication()
    {
        Application.Quit();
    }
}
```

5.31 InteractableScripts/ShowKeyboard.cs File Reference

Classes

- class ShowKeyboard

    Class `ShowKeyboard` shows and hides interactable keyboard.

5.32 ShowKeyboard.cs

Go to the documentation of this file.

```
using Microsoft.MixedReality.Toolkit.Experimental.UI;
using TMPro;
using UnityEngine;

public class ShowKeyboard : MonoBehaviour
{
    private TMP_InputField inputField;
    [SerializeField] private float distance = 0.5f;
    [SerializeField] private float verticaloffset = -0.5f;
    [SerializeField] private Transform positionSource;


    void Start()
    {
        inputField =GetComponent<TMP_InputField>();
        inputField.onSelect.AddListener(x => OpenKeyboard());
    }

    public void OpenKeyboard()
    {
        NonNativeKeyboard.Instance.InputField = inputField;
        NonNativeKeyboard.Instance.PresentKeyboard(inputField.text);

        Vector3 direction = positionSource.forward;
        direction.y = 0f;
        direction.Normalize();

        Vector3 targetPosition = positionSource.position + direction * distance + Vector3.up *
verticaloffset;

        NonNativeKeyboard.Instance.RepositionKeyboard(targetPosition);
        SetCaretColorAlpha(1);

        NonNativeKeyboard.Instance.OnClosed += Instance_OnClosed;
    }

    private void Instance_OnClosed(object sender, System.EventArgs e)
    {
        SetCaretColorAlpha(0);
        NonNativeKeyboard.Instance.OnClosed -= Instance_OnClosed;
    }

    public void SetCaretColorAlpha(float alpha)
    {
        inputField.customCaretColor = true;
        Color caretColor = inputField.caretColor;
        caretColor.a = alpha;
        inputField.caretColor = caretColor;
    }
}
```

5.33 InteractableScripts/TwoStateButton.cs File Reference

Classes

- class TwoStateButton

  Class `TwoStateButton` is responsible to controll two state button interaction.

5.34 TwoStateButton.cs

Go to the documentation of this file.

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class TwoStateButton : MonoBehaviour
{
    [SerializeField] private MeshRenderer meshRenderer;
    [SerializeField] private Material materialOn;
    [SerializeField] private Material materialOff;
    [SerializeField] private bool value =false;
    [SerializeField] private XRSimpleInteractable interactable;

    void Start()
    {
        interactable.selectEntered.AddListener(x => ChangeValue());

    }
    public void ChangeValue()
    {
        value =!value;

    }

    public bool GetValue()
    {
        return value;
    }



    public void SetValue(bool newValue)
    {
        value = newValue;
    }


    private void Update()
    {
        if (value)
        {
            meshRenderer.sharedMaterial = materialOn;
        }
        else
        {
            meshRenderer.sharedMaterial = materialOff;
        }
    }
}
```

Index

AddPointToGraph
    GraphShaderData, 19
AllwaysConnected
    GraphShaderData, 18

ButtonFollowVisual, 7
    Follow, 8
    Freeze, 8
    ResetButton, 8

Catmull
    GraphShaderData, 18
ChangeValue
    TwoStateButton, 37
CheckIfSaved
    InOutDataController, 27
Circle
    GraphShaderData, 18
ConfigData, 10
    pathToExport, 10
    pathToIonsFile, 10
    pathToSave, 10
CopyValuesFrom
    GraphShaderData, 19

DataScripts/ConfigData.cs, 39
DataScripts/ExportData.cs, 39
DataScripts/Graph.cs, 40
DataScripts/IonData.cs, 40
DataScripts/Laser.cs, 40, 41
DataScripts/SaveData.cs, 41

ExportData, 11
    IonName, 11
    lasers, 11
    piTime, 11

Follow
    ButtonFollowVisual, 8
Freeze
    ButtonFollowVisual, 8

GetClosestPiontIndex
    GraphShaderData, 19
GetShaderData
    GraphControll, 14
GetValue
    TwoStateButton, 37
GlobalScripts/GridFiller.cs, 41
GlobalScripts/InOutDataController.cs, 42
GlobalScripts/MaximizeController.cs, 45
Graph, 12
    lineColor, 12
    pointColor, 12
    points, 12
GraphControll, 13
    GetShaderData, 14
GraphMaximize, 14
graphOffsetOnX
    GraphShaderData, 21
graphOffsetOnXMax
    GraphShaderData, 21
graphOffsetOnXMin
    GraphShaderData, 21
graphOffsetOnY
    GraphShaderData, 21
graphOffsetOnYMax
    GraphShaderData, 21
graphOffsetOnYMin
    GraphShaderData, 21
graphs
    GraphShaderData, 21
    SaveData, 33
GraphScaleDescControll, 15
graphScaleOnX
    GraphShaderData, 21
graphScaleOnXMax
    GraphShaderData, 22
graphScaleOnXMin
    GraphShaderData, 22
graphScaleOnY
    GraphShaderData, 22
graphScaleOnYMax
    GraphShaderData, 22
graphScaleOnYMin
    GraphShaderData, 22
GraphScripts/GraphControll.cs, 46
GraphScripts/GraphMaximize.cs, 48, 49
GraphScripts/GraphScaleDescControll.cs, 49
GraphScripts/GraphShaderData.cs, 50
GraphShaderData, 16
    AddPointToGraph, 19
    AllwaysConnected, 18
    Catmull, 18
    Circle, 18
    CopyValuesFrom, 19
    GetClosestPiontIndex, 19
    graphOffsetOnX, 21
    graphOffsetOnXMax, 21
    graphOffsetOnXMin, 21

graphOffsetOnY, 21
graphOffsetOnYMax, 21
graphOffsetOnYMin, 21
graphs, 21
graphScaleOnX, 21
graphScaleOnXMax, 22
graphScaleOnXMin, 22
graphScaleOnY, 22
graphScaleOnYMax, 22
graphScaleOnYMin, 22
int, 22
lineColor, 23
lineSize, 23
lineVariant, 23
lineVariantEnum, 18
maxPointAmount, 23
maxXPointValue, 23
maxYPointValue, 23
meshRenderer, 23
minPointAmount, 23
minXPointValue, 24
minYPointValue, 24
Normal, 18
pointColor, 24
pointShape, 24
pointShapeEnum, 18
pointSize, 24
SetOffsetOnX, 19
SetOffsetOnY, 20
SetScaleOnX, 20
SetScaleOnY, 20
Square, 18
unitPerGridOnX, 24
unitPerGridOnY, 24
GridFiller, 25

InOutDataController, 26
CheckIfSaved, 27
instance, 28
ionData, 27
LoadIonData, 27
instance
InOutDataController, 28
MaximizeController, 31
int
GraphShaderData, 22
InteractableScripts/ButtonFollowVisual.cs, 53
InteractableScripts/QuitButton.cs, 54
InteractableScripts/ShowKeyboard.cs, 54, 55
InteractableScripts/TwoStateButton.cs, 55
IonData, 28
laserAmount, 28
name, 28
ionData
InOutDataController, 27
IonName
ExportData, 11
SaveData, 33

Laser, 29
points, 29
laserAmount
IonData, 28
lasers
ExportData, 11
lineColor
Graph, 12
GraphShaderData, 23
lineSize
GraphShaderData, 23
lineVariant
GraphShaderData, 23
lineVariantEnum
GraphShaderData, 18
LoadIonData
InOutDataController, 27

MaximizeController, 30
instance, 31
MaximizeSection, 31
MaximizeSection
MaximizeController, 31
maxPointAmount
GraphShaderData, 23
maxXPointValue
GraphShaderData, 23
maxYPointValue
GraphShaderData, 23
meshRenderer
GraphShaderData, 23
minPointAmount
GraphShaderData, 23
minXPointValue
GraphShaderData, 24
minYPointValue
GraphShaderData, 24

name
IonData, 28
Normal
GraphShaderData, 18
numberOfLasers
SaveData, 34

OpenKeyboard
ShowKeyboard, 35

pathToExport
ConfigData, 10
pathToIonsFile
ConfigData, 10
pathToSave
ConfigData, 10
piTime
ExportData, 11
SaveData, 34
pointColor
Graph, 12

GraphShaderData, 24
points
    Graph, 12
    Laser, 29
pointShape
    GraphShaderData, 24
pointShapeEnum
    GraphShaderData, 18
pointSize
    GraphShaderData, 24

QuitApplication
    QuitButton, 32
QuitButton, 31
    QuitApplication, 32

ResetButton
    ButtonFollowVisual, 8

SaveData, 33
    graphs, 33
    IonName, 33
    numberOfLasers, 34
    piTime, 34
SetCaretColorAlpha
    ShowKeyboard, 35
SetOffsetOnX
    GraphShaderData, 19
SetOffsetOnY
    GraphShaderData, 20
SetScaleOnX
    GraphShaderData, 20
SetScaleOnY
    GraphShaderData, 20
SetValue
    TwoStateButton, 37
ShowKeyboard, 34
    OpenKeyboard, 35
    SetCaretColorAlpha, 35
Square
    GraphShaderData, 18

TwoStateButton, 36
    ChangeValue, 37
    GetValue, 37
    SetValue, 37

unitPerGridOnX
    GraphShaderData, 24
unitPerGridOnY
    GraphShaderData, 24