

Java - Puissance 4

Generated by Doxygen 1.9.3

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 Cell Class Reference	7
4.1.1 Detailed Description	9
4.1.2 Constructor & Destructor Documentation	9
4.1.2.1 Cell()	9
4.1.3 Member Function Documentation	10
4.1.3.1 check()	10
4.1.3.2 compareTo()	10
4.1.3.3 equals()	11
4.1.3.4 getColor()	12
4.1.3.5 getNeighbor()	12
4.1.3.6 getToken()	13
4.1.3.7 isEmpty()	14
4.1.3.8 numberOfSameNeighbor() [1/2]	14
4.1.3.9 numberOfSameNeighbor() [2/2]	15
4.1.3.10 setNeighbor()	16
4.1.3.11 setToken()	16
4.1.3.12 toString()	17
4.1.4 Member Data Documentation	18
4.1.4.1 neighborhood	18
4.1.4.2 outOfBoundCell	18
4.1.4.3 token	18
4.2 Color Enum Reference	19
4.2.1 Detailed Description	19
4.2.2 Member Function Documentation	20
4.2.2.1 ansiColorOf() [1/2]	20
4.2.2.2 ansiColorOf() [2/2]	20
4.2.2.3 colorOf()	21
4.2.3 Member Data Documentation	21
4.2.3.1 EMPTY	21
4.2.3.2 RED	21
4.2.3.3 YELLOW	22
4.3 Comparable Class Reference	22
4.4 Direction Enum Reference	23

4.4.1 Detailed Description	24
4.4.2 Member Function Documentation	24
4.4.2.1 getDiagonales()	24
4.4.2.2 getOpposite()	24
4.4.3 Member Data Documentation	25
4.4.3.1 DOWN	25
4.4.3.2 LEFT	25
4.4.3.3 RIGHT	25
4.4.3.4 UP	26
4.5 Entity Enum Reference	26
4.5.1 Detailed Description	26
4.5.2 Member Function Documentation	27
4.5.2.1 of()	27
4.5.3 Member Data Documentation	27
4.5.3.1 IA	27
4.5.3.2 LOCAL	27
4.6 Game Class Reference	28
4.6.1 Detailed Description	29
4.6.2 Constructor & Destructor Documentation	30
4.6.2.1 Game()	30
4.6.3 Member Function Documentation	30
4.6.3.1 askForInput()	30
4.6.3.2 askForLoadSave()	31
4.6.3.3 askForPlayers()	32
4.6.3.4 chooseAColumn()	32
4.6.3.5 createPlayer()	33
4.6.3.6 endMessage()	34
4.6.3.7 getPlayerFromId()	34
4.6.3.8 hasWin()	35
4.6.3.9 initLocalPlayer()	36
4.6.3.10 isEnd()	37
4.6.3.11 isTie()	37
4.6.3.12 loadSave()	38
4.6.3.13 menu()	39
4.6.3.14 nextPlayer()	40
4.6.3.15 play()	40
4.6.3.16 playAToken()	41
4.6.3.17 printCurrentPlayer()	42
4.6.3.18 printPartyChoice()	43
4.6.3.19 run()	43
4.6.3.20 setOptionGame()	44
4.6.3.21 toString()	45

4.6.3.22 valideColumn()	45
4.6.3.23 valideGameOption()	46
4.6.3.24 welcomeMessage()	47
4.6.4 Member Data Documentation	47
4.6.4.1 arrayPlayer	47
4.6.4.2 currentPlayerId	47
4.6.4.3 end	48
4.6.4.4 gameplay	48
4.6.4.5 grid	48
4.6.4.6 iteration	48
4.6.4.7 lastMove	48
4.6.4.8 numberOfPlayers	48
4.6.4.9 numberOfTokenToWin	49
4.6.4.10 save	49
4.7 Grid Class Reference	49
4.7.1 Detailed Description	51
4.7.2 Constructor & Destructor Documentation	51
4.7.2.1 Grid()	52
4.7.3 Member Function Documentation	52
4.7.3.1 getHeight()	52
4.7.3.2 getNextEmptyCellAt()	52
4.7.3.3 getSize()	53
4.7.3.4 getTopCellAt()	54
4.7.3.5 getWidth()	55
4.7.3.6 initGrid()	55
4.7.3.7 loadGrid()	56
4.7.3.8 print()	56
4.7.3.9 printAvailableColumn()	57
4.7.3.10 printGrid()	58
4.7.3.11 toString()	58
4.7.3.12 UpToNextEmptyCellAt()	59
4.7.3.13 valideColumn()	60
4.7.4 Member Data Documentation	61
4.7.4.1 arrayNextEmptyCell	61
4.7.4.2 HEIGHT	61
4.7.4.3 WIDTH	61
4.8 Main Class Reference	62
4.8.1 Detailed Description	62
4.8.2 Member Function Documentation	62
4.8.2.1 main()	63
4.9 Player Class Reference	63
4.9.1 Detailed Description	65

4.9.2 Constructor & Destructor Documentation	65
4.9.2.1 Player()	65
4.9.3 Member Function Documentation	66
4.9.3.1 getColor()	66
4.9.3.2 getToken()	67
4.9.3.3 getUsername()	67
4.9.3.4 initToken()	67
4.9.3.5 toString()	68
4.9.3.6 validePlayer()	68
4.9.3.7 whatIs()	69
4.9.4 Member Data Documentation	69
4.9.4.1 gameplay	69
4.9.4.2 pseudo	70
4.9.4.3 token	70
4.10 Save Class Reference	70
4.10.1 Detailed Description	71
4.10.2 Constructor & Destructor Documentation	71
4.10.2.1 Save()	71
4.10.3 Member Function Documentation	72
4.10.3.1 delete() [1/2]	72
4.10.3.2 delete() [2/2]	72
4.10.3.3 isEmpty()	73
4.10.3.4 read() [1/2]	74
4.10.3.5 read() [2/2]	74
4.10.3.6 write() [1/2]	75
4.10.3.7 write() [2/2]	76
4.10.4 Member Data Documentation	77
4.10.4.1 filename	77
4.11 Token Class Reference	77
4.11.1 Detailed Description	78
4.11.2 Constructor & Destructor Documentation	78
4.11.2.1 Token()	78
4.11.3 Member Function Documentation	79
4.11.3.1 getColor()	79
4.11.3.2 toString()	79
4.11.4 Member Data Documentation	80
4.11.4.1 colorToken	80
4.11.4.2 emptyToken	80
5 File Documentation	81
5.1 /home/hozen/cur/projet-java/src/Cell.java File Reference	81
5.2 Cell.java	81

5.3 /home/hozen/cur/projet-java/src/Color.java File Reference	82
5.4 Color.java	83
5.5 /home/hozen/cur/projet-java/src/Direction.java File Reference	83
5.6 Direction.java	83
5.7 /home/hozen/cur/projet-java/src/Entity.java File Reference	84
5.8 Entity.java	84
5.9 /home/hozen/cur/projet-java/src/Game.java File Reference	84
5.10 Game.java	85
5.11 /home/hozen/cur/projet-java/src/Grid.java File Reference	89
5.12 Grid.java	89
5.13 /home/hozen/cur/projet-java/src/Main.java File Reference	91
5.14 Main.java	91
5.15 /home/hozen/cur/projet-java/src/Player.java File Reference	91
5.16 Player.java	92
5.17 /home/hozen/cur/projet-java/src/Save.java File Reference	92
5.18 Save.java	92
5.19 /home/hozen/cur/projet-java/src/Token.java File Reference	93
5.20 Token.java	94
Index	95

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Color	19
Comparable	22
Cell	7
Direction	23
Entity	26
Game	28
Grid	49
Main	62
Player	63
Save	70
Token	77

Chapter 2

Class Index

2.1 Class List

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

Cell	Logical representation of a case in the grid of the Game of Puissance 4	7
Color	Represent colors globally	19
Comparable	22
Direction	Represent different direction with diagonales	23
Entity	Logical representation of a Player (IA, LOCAL(human))	26
Game	Represent a Game of Puissance 4	28
Grid	Logical representation of a board of the Game of Puissance 4	49
Main	Main class, program entry point	62
Player	Represent a Player of a Game using Token	63
Save	Represent a global Save class	70
Token	Represent a Token of Puissance 4	77

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

/home/hozen/cur/projet-java/src/ Cell.java	81
/home/hozen/cur/projet-java/src/ Color.java	82
/home/hozen/cur/projet-java/src/ Direction.java	83
/home/hozen/cur/projet-java/src/ Entity.java	84
/home/hozen/cur/projet-java/src/ Game.java	84
/home/hozen/cur/projet-java/src/ Grid.java	89
/home/hozen/cur/projet-java/src/ Main.java	91
/home/hozen/cur/projet-java/src/ Player.java	91
/home/hozen/cur/projet-java/src/ Save.java	92
/home/hozen/cur/projet-java/src/ Token.java	93

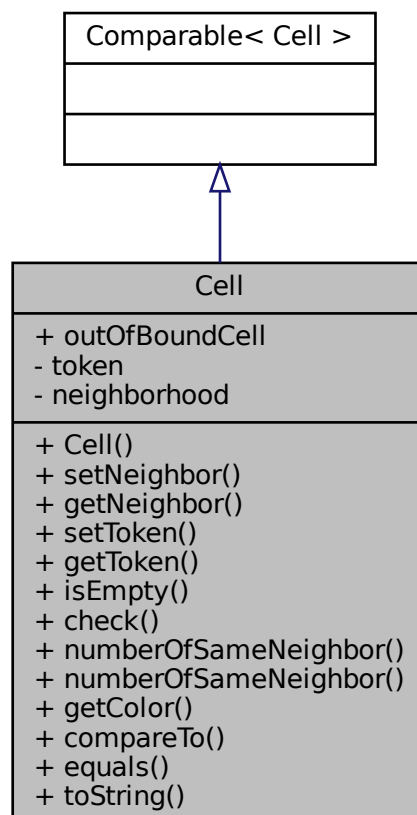
Chapter 4

Class Documentation

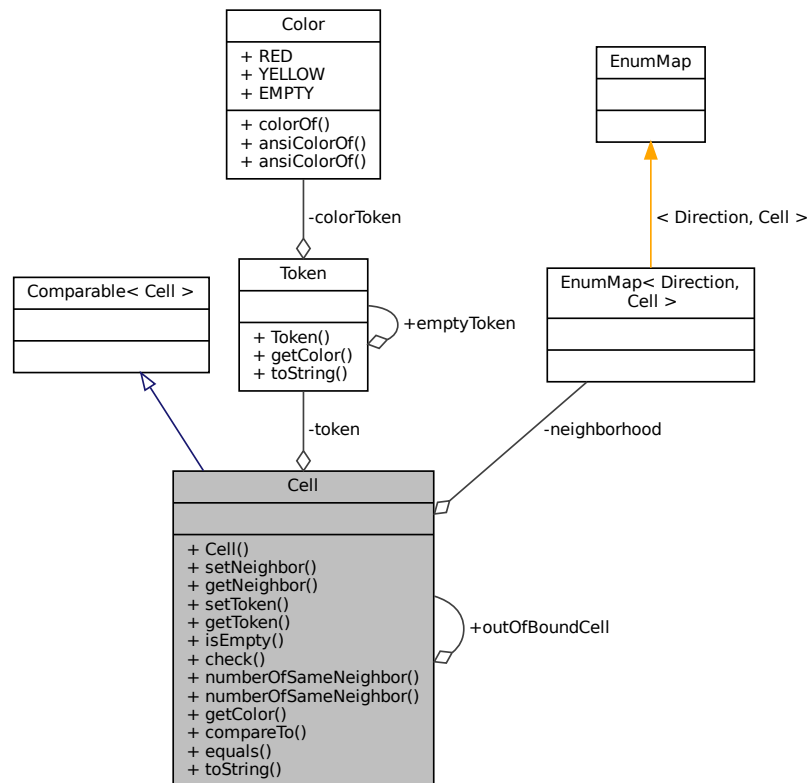
4.1 Cell Class Reference

Logical representation of a case in the grid of the [Game](#) of Puissance 4.

Inheritance diagram for Cell:



Collaboration diagram for Cell:



Public Member Functions

- [Cell](#) ([Token](#) token)
- void [setNeighbor](#) ([Cell](#) c, [Direction](#) d)
- [Cell](#) [getNeighbor](#) ([Direction](#) d)
- void [setToken](#) ([Token](#) t)
- [Token](#) [getToken](#) ()
- boolean [isEmpty](#) ()
- boolean [check](#) ()
- int [numberOfSameNeighbor](#) ([Direction](#) d)
- int [numberOfSameNeighbor](#) ([Direction](#) d1, [Direction](#) d2)
- [Color](#) [getColor](#) ()
- int [compareTo](#) ([Cell](#) other)
- boolean [equals](#) (Object other)
- String [toString](#) ()

Static Public Attributes

- static final [Cell](#) [outOfBoundCell](#) = new [Cell](#)([Token.emptyToken](#))

Private Attributes

- [Token](#) `token`
- `EnumMap< Direction, Cell >` `neighborhood`

4.1.1 Detailed Description

Logical representation of a case in the grid of the [Game](#) of Puissance 4.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line [11](#) of file [Cell.java](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 `Cell()`

```
Cell.Cell (
    Token token )
```

[Cell](#) constructor

Parameters

<code>token</code>	Token corresponding to the cell
--------------------	---

Definition at line [18](#) of file [Cell.java](#).

Here is the call graph for this function:



4.1.3 Member Function Documentation

4.1.3.1 check()

```
boolean Cell.check ( )
```

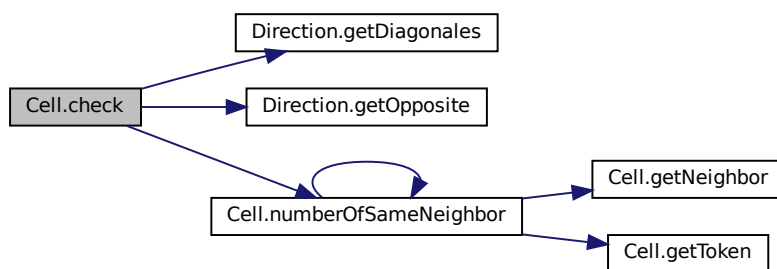
Check if this [Cell](#) has the number required of same [Cell](#) in a [Direction](#) to "win"

Returns

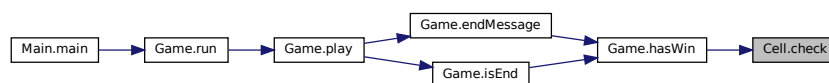
a boolean true if there is [Game.numberOfTokenToWin](#) neighbor with same token

Definition at line 83 of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.2 compareTo()

```
int Cell.compareTo (
    Cell other )
```

Override the `compareTo` function, test same [Token](#) reference

Parameters

<i>other</i>	the other Cell to compare
--------------	---

Returns

the comparison

Definition at line [153](#) of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.3 equals()

```
boolean Cell.equals (  
    Object other )
```

Override the equals method

Parameters

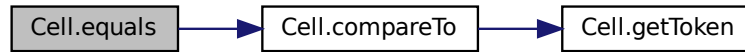
<i>other</i>	the other object to test equality
--------------	-----------------------------------

Returns

boolean true if it's equal, else false

Definition at line 165 of file [Cell.java](#).

Here is the call graph for this function:



4.1.3.4 getColor()

```
Color Cell.getColor ( )
```

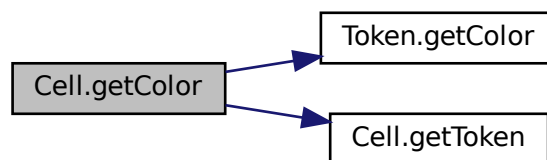
Get the [Token Color](#) of the [Cell](#)'s [Token](#).

Returns

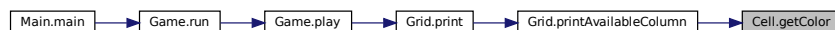
the specific [Color](#)

Definition at line 143 of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.5 getNeighbor()

```
Cell Cell.getNeighbor (
    Direction d )
```

Get a [Cell](#) neighbor in a direction, if null return an invalide valide cell

Parameters

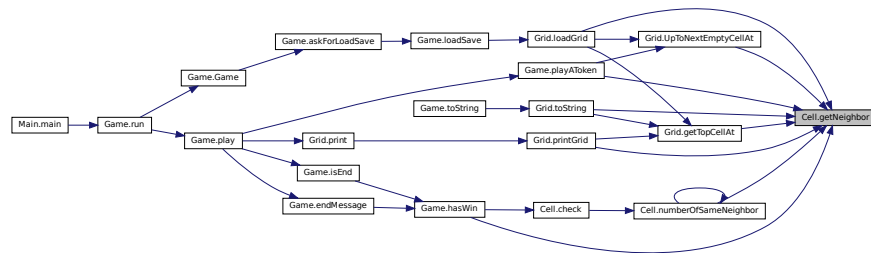
<i>d</i>	Direction where to get the neighbor
----------	-------------------------------------

Returns

the [Cell](#) corresponding : if null return outOfBoundCell

Definition at line 38 of file [Cell.java](#).

Here is the caller graph for this function:



4.1.3.6 getToken()

[Token](#) [Cell.getToken](#) ()

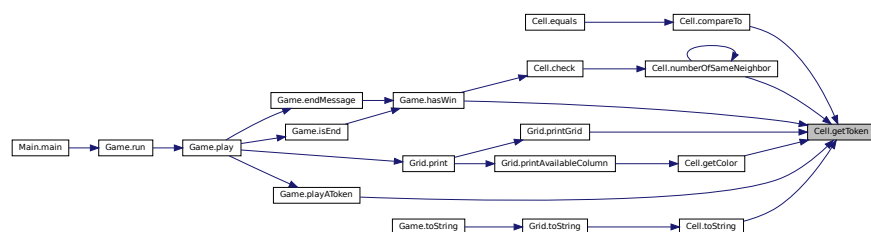
Get the [Cell Token](#).

Returns

the [Token](#) corresponding.

Definition at line 65 of file [Cell.java](#).

Here is the caller graph for this function:



4.1.3.7 isEmpty()

```
boolean Cell.isEmpty ( )
```

Test if the cell is EMPTY, if it's token if the emptyToken

Returns

a boolean true if it's EMPTY, false if it is not

Definition at line 74 of file [Cell.java](#).

Here is the call graph for this function:



4.1.3.8 numberOfSameNeighbor() [1/2]

```
int Cell.numberOfSameNeighbor (
    Direction d )
```

Recursive function count his number of [Cell](#) which has same token

Parameters

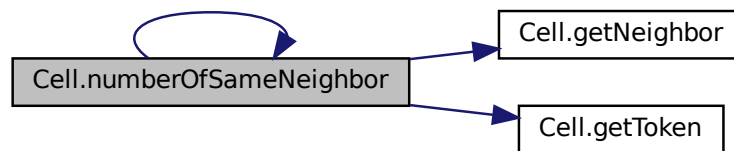
<i>d</i>	Direction to check
----------	------------------------------------

Returns

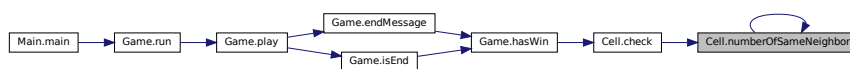
the number of same neighbor

Definition at line 112 of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.3.9 numberOfSameNeighbor() [2/2]**

```
int Cell.numberOfSameNeighbor (
    Direction d1,
    Direction d2 )
```

Recursive functin count his diagonales number of [Cell](#) which has same token

Parameters

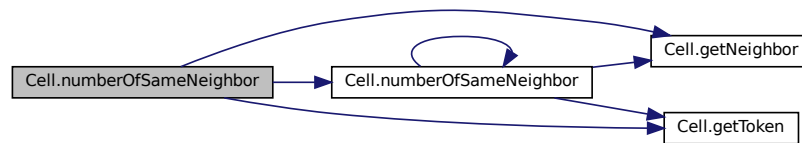
<i>d1</i>	first Direction diagonale
<i>d2</i>	second Direction diagonale

Returns

the number of same neighbor

Definition at line 127 of file [Cell.java](#).

Here is the call graph for this function:



4.1.3.10 setNeighbor()

```
void Cell.setNeighbor (
    Cell c,
    Direction d )
```

Set the [Cell](#) neighbor to this in a [Direction](#)

Parameters

<i>c</i>	Cell to set as a neighbor
<i>d</i>	Direction where to set the Cell neighbor

Definition at line 28 of file [Cell.java](#).

Here is the caller graph for this function:



4.1.3.11 setToken()

```
void Cell.setToken (
    Token t )
```

Set a [Token](#) to the [Cell](#), can't set an EMPTY one

Parameters

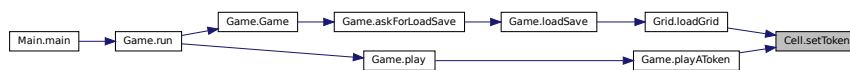
<i>t</i>	not EMPTY Token to place in
----------	---

Definition at line 53 of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.12 toString()

```
String Cell.toString ( )
```

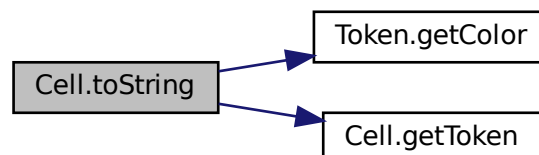
Override `toString` method, print the [Color](#) of the [Token](#) of the [Cell](#)

Returns

a String corresponding to the [Color](#)

Definition at line 179 of file [Cell.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4 Member Data Documentation

4.1.4.1 neighborhood

```
EnumMap<Direction, Cell> Cell.neighborhood [private]
```

Definition at line 16 of file [Cell.java](#).

4.1.4.2 outOfBoundCell

```
final Cell Cell.outOfBoundCell = new Cell(Token.emptyToken) [static]
```

Definition at line 13 of file [Cell.java](#).

4.1.4.3 token

```
Token Cell.token [private]
```

Definition at line 15 of file [Cell.java](#).

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

- [/home/hozen/cur/projet-java/src/Cell.java](#)

4.2 Color Enum Reference

Represent colors globally.

Collaboration diagram for Color:



Static Public Member Functions

- static [Color](#) [colorOf](#) (String colorString)
- static String [ansiColorOf](#) ([Color](#) c)
- static String [ansiColorOf](#) (String c)

Public Attributes

- [RED](#)
- [YELLOW](#)
- [EMPTY](#)

4.2.1 Detailed Description

Represent colors globally.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 7 of file [Color.java](#).

4.2.2 Member Function Documentation

4.2.2.1 ansiColorOf() [1/2]

```
static String Color ansiColorOf (
    Color c ) [static]
```

Give the ansi color corresponding to a [Color](#) given

Parameters

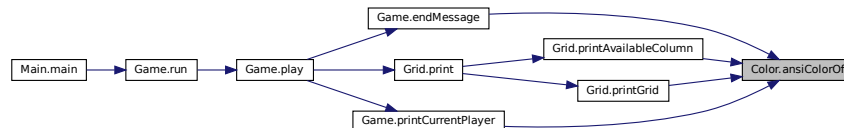
<i>c</i>	Color source
----------	------------------------------

Returns

String ansi representation

Definition at line 27 of file [Color.java](#).

Here is the caller graph for this function:



4.2.2.2 ansiColorOf() [2/2]

```
static String Color ansiColorOf (
    String c ) [static]
```

Give the ansi color corresponding of a [String](#) given

Parameters

<i>c</i>	String color source
----------	---------------------

Returns

String ansi representation

Definition at line 43 of file [Color.java](#).

4.2.2.3 colorOf()

```
static Color Color.colorOf (
    String colorString ) [static]
```

Give the color for a specify String

Parameters

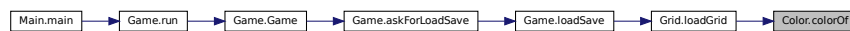
<i>colorString</i>	the color String representation you need
--------------------	--

Returns

the [Color](#) of the colorString

Definition at line 11 of file [Color.java](#).

Here is the caller graph for this function:



4.2.3 Member Data Documentation

4.2.3.1 EMPTY

`Color.EMPTY`

Definition at line 9 of file [Color.java](#).

4.2.3.2 RED

`Color.RED`

Definition at line 9 of file [Color.java](#).

4.2.3.3 YELLOW

`Color.YELLOW`

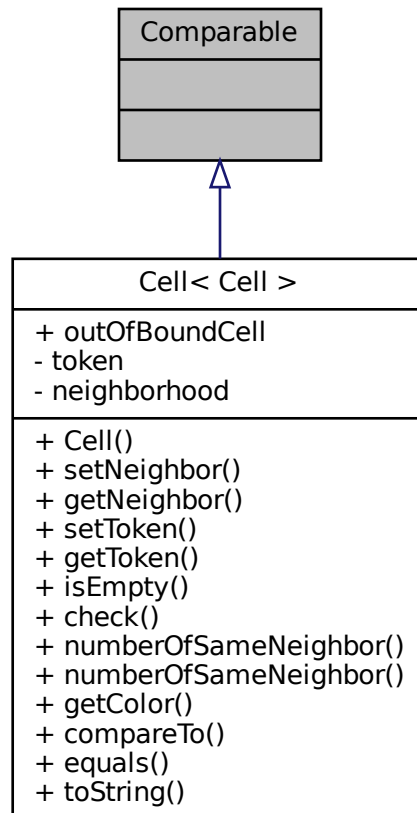
Definition at line 9 of file [Color.java](#).

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

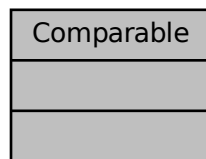
- [/home/hozen/cur/projet-java/src/Color.java](#)

4.3 Comparable Class Reference

Inheritance diagram for Comparable:



Collaboration diagram for Comparable:



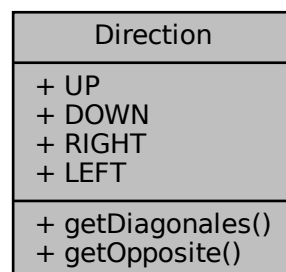
The documentation for this class was generated from the following file:

- [/home/hozen/cur/projet-java/src/Cell.java](#)

4.4 Direction Enum Reference

Represent different direction with diagonales.

Collaboration diagram for Direction:



Static Public Member Functions

- static EnumMap< [Direction](#), [Direction](#) > [getDiagonales](#) ()
- static [Direction](#) [getOpposite](#) ([Direction](#) d)

Public Attributes

- [UP](#)
- [DOWN](#)
- [RIGHT](#)
- [LEFT](#)

4.4.1 Detailed Description

Represent different direction with diagonales.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 9 of file [Direction.java](#).

4.4.2 Member Function Documentation

4.4.2.1 getDiagonales()

```
static EnumMap< Direction, Direction > Direction.getDiagonales ( ) [static]
```

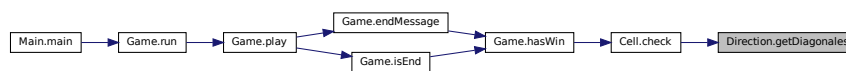
Give an EnumMap of [Direction](#) which each key is a [Direction](#) and its value the diagonale [Direction](#) corresponding.

Returns

EnumMap representing diagonales

Definition at line 13 of file [Direction.java](#).

Here is the caller graph for this function:



4.4.2.2 getOpposite()

```
static Direction Direction.getOpposite ( Direction d ) [static]
```

Give the opposite [Direction](#) to a [Direction](#) given

Parameters

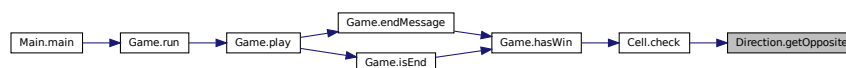
<i>d</i>	Direction source
----------	----------------------------------

Returns

the opposite of *d* [Direction](#)

Definition at line 28 of file [Direction.java](#).

Here is the caller graph for this function:



4.4.3 Member Data Documentation

4.4.3.1 DOWN

`Direction.DOWN`

Definition at line 11 of file [Direction.java](#).

4.4.3.2 LEFT

`Direction.LEFT`

Definition at line 11 of file [Direction.java](#).

4.4.3.3 RIGHT

`Direction.RIGHT`

Definition at line 11 of file [Direction.java](#).

4.4.3.4 UP

`Direction.UP`

Definition at line 11 of file [Direction.java](#).

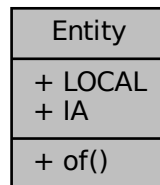
The documentation for this enum was generated from the following file:

- [/home/hozen/cur/projet-java/src/Direction.java](#)

4.5 Entity Enum Reference

Logical representation of a [Player](#) (IA, LOCAL(human))

Collaboration diagram for Entity:



Static Public Member Functions

- static [Entity of](#) (String e)

Public Attributes

- [LOCAL](#)
- [IA](#)

4.5.1 Detailed Description

Logical representation of a [Player](#) (IA, LOCAL(human))

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 7 of file [Entity.java](#).

4.5.2 Member Function Documentation

4.5.2.1 of()

```
static Entity Entity.of (
    String e ) [static]
```

Give the [Entity](#) corresponding to the given String

Parameters

<i>e</i>	name of the entity
----------	--------------------

Returns

the [Entity](#) of the string

Definition at line 11 of file [Entity.java](#).

Here is the caller graph for this function:



4.5.3 Member Data Documentation

4.5.3.1 IA

`Entity.IA`

Definition at line 9 of file [Entity.java](#).

4.5.3.2 LOCAL

`Entity.LOCAL`

Definition at line 9 of file [Entity.java](#).

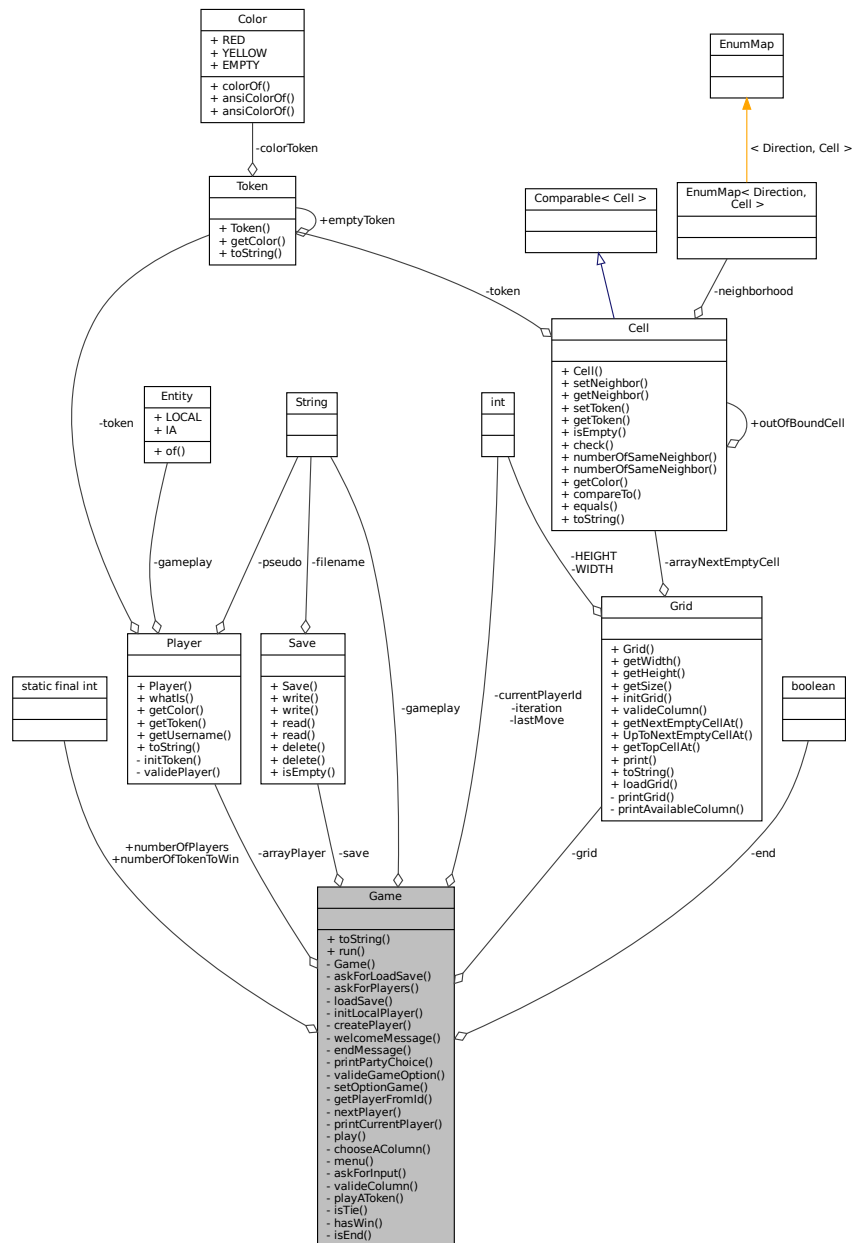
The documentation for this enum was generated from the following file:

- [/home/hozen/cur/projet-java/src/Entity.java](#)

4.6 Game Class Reference

Represent a [Game](#) of Puissance 4.

Collaboration diagram for Game:



Public Member Functions

- String [toString](#) ()

Static Public Member Functions

- static void [run](#) ()

Static Public Attributes

- static final int `numberOfPlayers` = 2
- static final int `numberOfTokenToWin` = 4

Private Member Functions

- `Game` ()
- void `askForLoadSave` ()
- void `askForPlayers` ()
- void `loadSave` (String saveText)
- void `initLocalPlayer` (int position)
- void `createPlayer` (String name, int id, `Entity` og)
- void `welcomeMessage` ()
- void `endMessage` ()
- void `printPartyChoice` ()
- int `valideGameOption` (int option)
- void `setOptionGame` ()
- `Player` `getPlayerFromId` (int id)
- `Player` `nextPlayer` (`Player` current)
- void `printCurrentPlayer` ()
- void `play` ()
- int `chooseAColumn` ()
- void `menu` ()
- int `askForInput` ()
- int `valideColumn` (int column)
- boolean `playAToken` (`Token` token, int column)
- boolean `isTie` ()
- boolean `hasWin` ()
- void `isEnd` ()

Private Attributes

- `Player`[] `arrayPlayer`
- `Grid` `grid`
- final `Save` `save`
- boolean `end`
- int `iteration`
- String `gameplay`
- int `currentPlayerId`
- int `lastMove`

4.6.1 Detailed Description

Represent a `Game` of Puissance 4.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 14 of file `Game.java`.

4.6.2 Constructor & Destructor Documentation

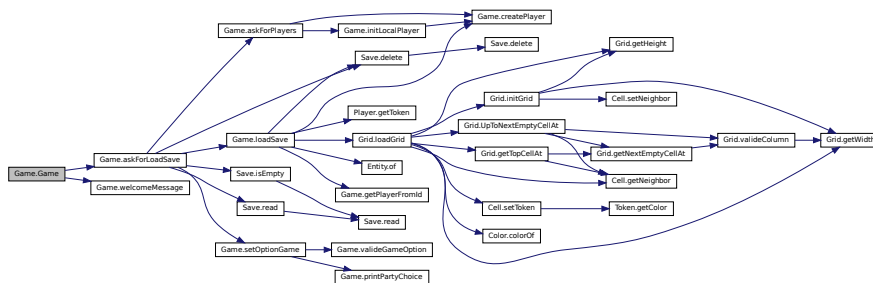
4.6.2.1 Game()

```
Game.Game ( ) [private]
```

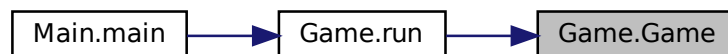
[Game](#) class constructor

Definition at line 34 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3 Member Function Documentation

4.6.3.1 askForInput()

```
int Game.askForInput ( ) [private]
```

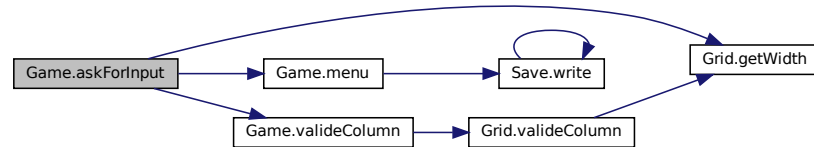
If the player is LOCAL ask what he want to do (play, menu).

Returns

the column choosen.

Definition at line 354 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

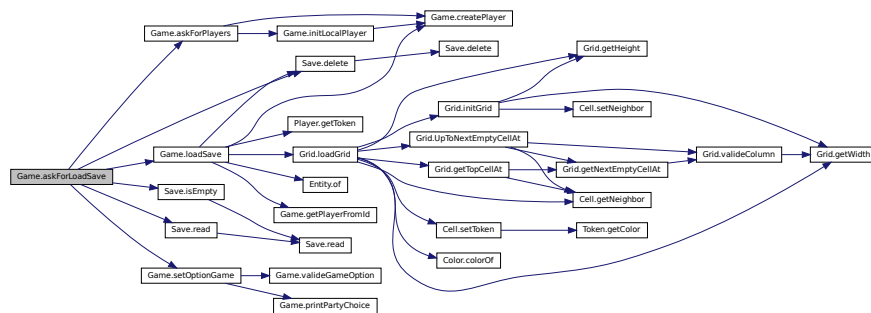
**4.6.3.2 askForLoadSave()**

```
void Game.askForLoadSave ( ) [private]
```

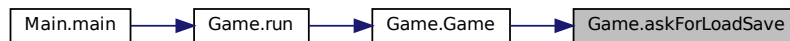
Ask the user if he wants to load an existing save

Definition at line 58 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



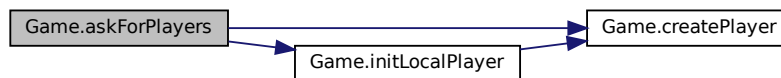
4.6.3.3 askForPlayers()

```
void Game.askForPlayers ( ) [private]
```

Initialisation of the Players

Definition at line 96 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.4 chooseAColumn()

```
int Game.chooseAColumn ( ) [private]
```

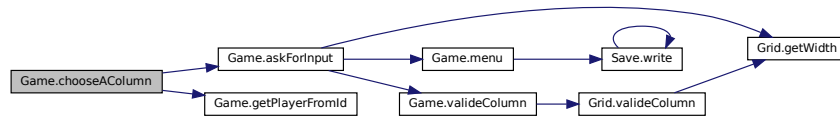
Choose a column depending if it's local or ia player

Returns

An int corresponding to the chosen column

Definition at line 308 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.5 createPlayer()**

```

void Game.createPlayer (
    String name,
    int id,
    Entity og ) [private]
  
```

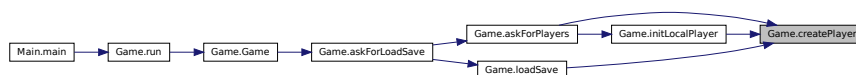
Create a player by call [Player](#) constructor

Parameters

<i>name</i>	name of the player
<i>id</i>	index of the player
<i>og</i>	what is the player (LOCAL or IA)

Definition at line 147 of file [Game.java](#).

Here is the caller graph for this function:



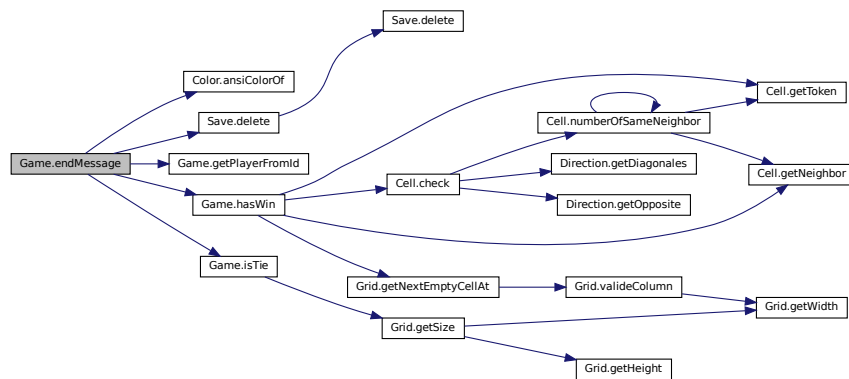
4.6.3.6 endMessage()

```
void Game.endMessage ( ) [private]
```

Print an end message

Definition at line 165 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.7 getPlayerFromId()

```
Player Game.getPlayerFromId (
    int id ) [private]
```

Get a player for a specific id

Parameters

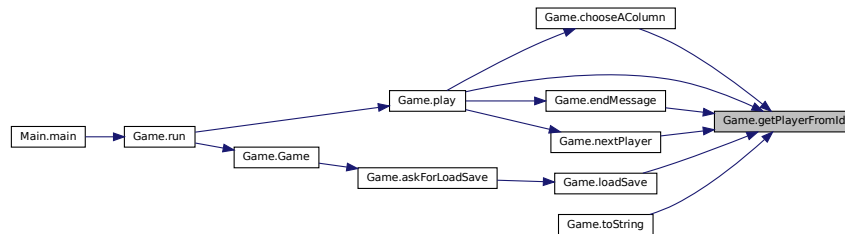
<i>id</i>	index of the Player
-----------	-------------------------------------

Returns

the [Player](#) corresponding to the index

Definition at line 240 of file [Game.java](#).

Here is the caller graph for this function:

**4.6.3.8 hasWin()**

```
boolean Game.hasWin ( ) [private]
```

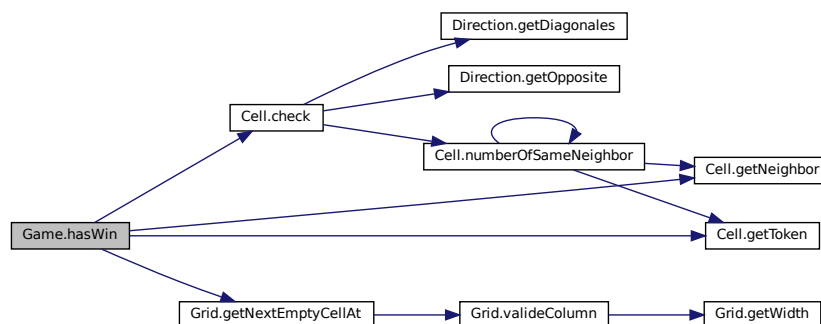
Win condition function: call check cell function on the last move

Returns

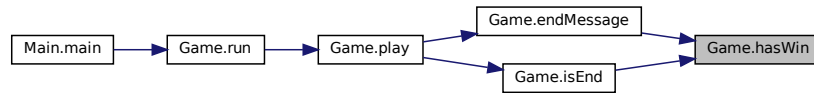
a boolean corresponding if it's a win or not

Definition at line 428 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.9 initLocalPlayer()

```
void Game.initLocalPlayer (
    int position ) [private]
```

Initialisation of a LOCAL [Player](#), ask his username

Parameters

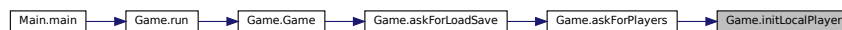
<i>position</i>	player position in his physical representation
-----------------	--

Definition at line [136](#) of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



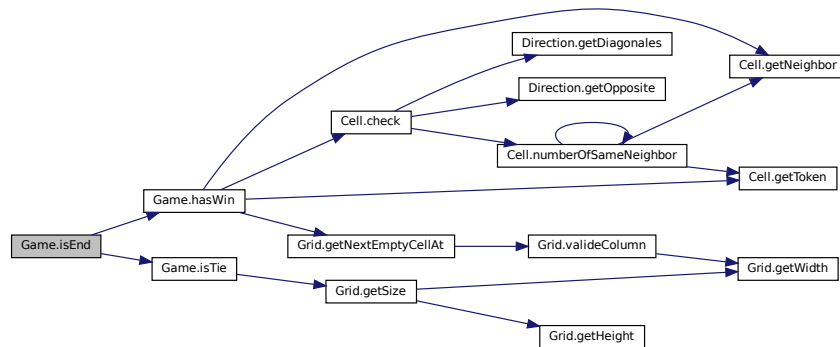
4.6.3.10 isEnd()

```
void Game.isEnd ( ) [private]
```

End condition : if it's a tie or a win it makes the game to end

Definition at line 440 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.11 isTie()

```
boolean Game.isTie ( ) [private]
```

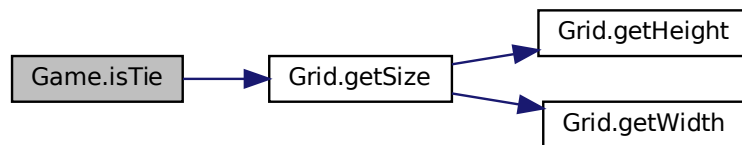
Tie function : if the number of play is equal to the size of the grid

Returns

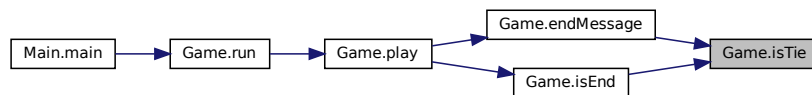
boolean corresponding if it's a tie or not

Definition at line 418 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.12 loadSave()**

```
void Game.loadSave (
    String saveText ) [private]
```

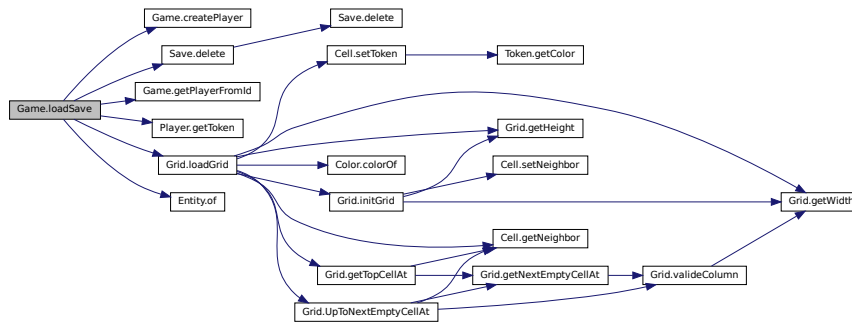
Initialisation of the game by an existing save

Parameters

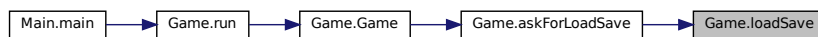
<i>saveText</i>	string representing a save
-----------------	----------------------------

Definition at line 105 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



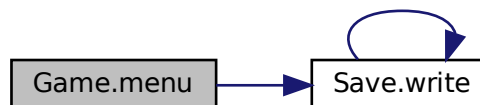
4.6.3.13 menu()

```
void Game.menu ( ) [private]
```

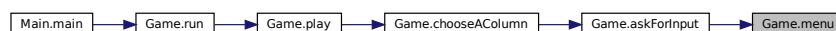
[Game](#) menu (play, save, quit) loop.

Definition at line 322 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.14 nextPlayer()

```
Player Game.nextPlayer (
    Player current ) [private]
```

Update the currentPlayerId by set the next in the physical representation

Parameters

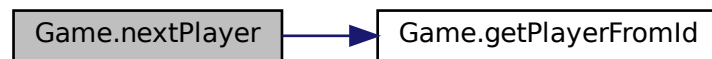
<i>current</i>	the last Player who have played
----------------	---

Returns

the next player to play

Definition at line 253 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



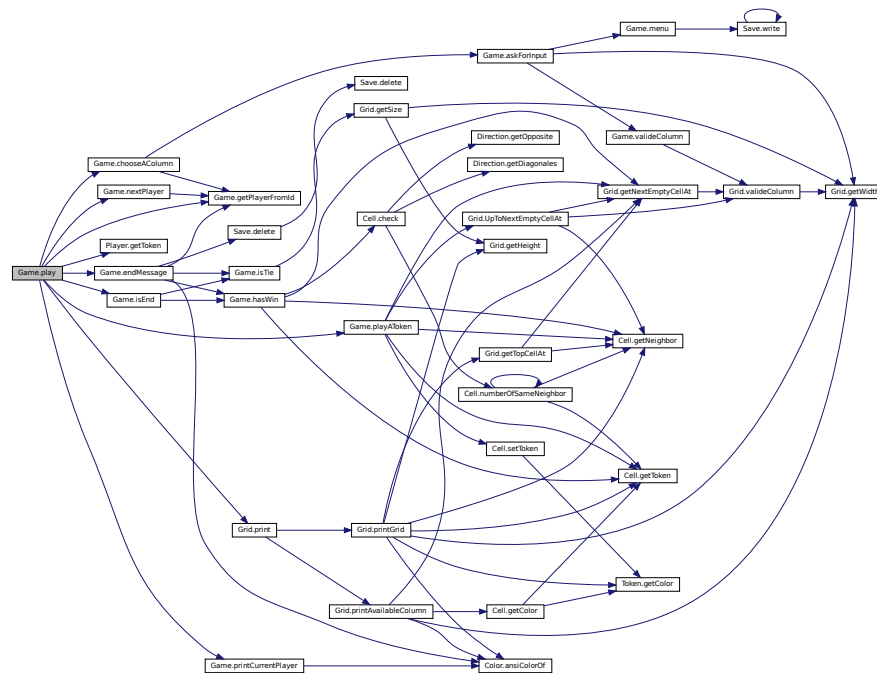
4.6.3.15 play()

```
void Game.play ( ) [private]
```

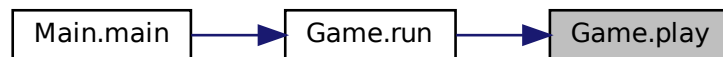
[Game](#) loop, end when game finish (Tie or Win or Quit)

Definition at line 276 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.16 playAToken()

```
boolean Game.playAToken (
    Token token,
    int column ) [private]
```

The player play a token to a given column

Parameters

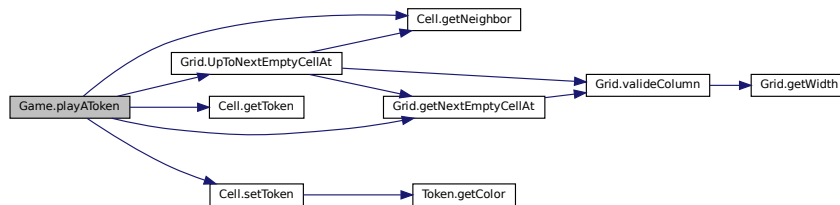
<i>token</i>	The token to play
<i>column</i>	The column where to play

Returns

a boolean if the play is a success or not

Definition at line 393 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

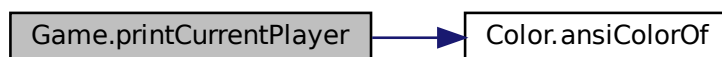
**4.6.3.17 printCurrentPlayer()**

```
void Game.printCurrentPlayer ( ) [private]
```

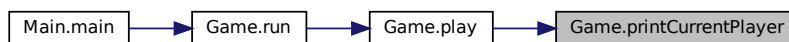
Print the `toString()` method in his personal token color with ANSI

Definition at line 265 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



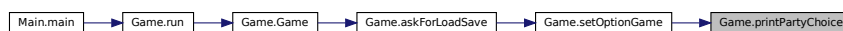
4.6.3.18 printPartyChoice()

```
void Game.printPartyChoice ( ) [private]
```

Print option game choice

Definition at line 188 of file [Game.java](#).

Here is the caller graph for this function:



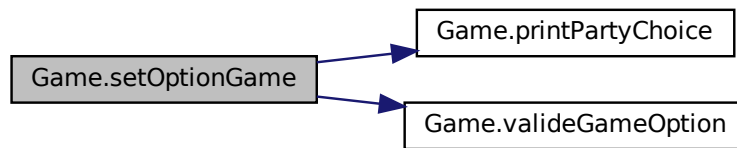
4.6.3.19 run()

```
static void Game.run ( ) [static]
```

Run a game of puissance 4

Definition at line 232 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.21 toString()

```
String Game.toString ( )
```

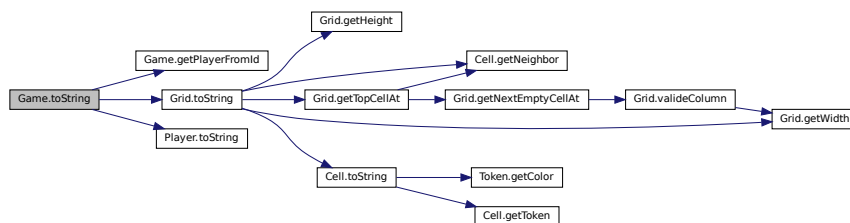
Representing color of grid cells, players informations and the next player who will play

Returns

the String representation

Definition at line 448 of file [Game.java](#).

Here is the call graph for this function:



4.6.3.22 valideColumn()

```
int Game.valideColumn (
    int column ) [private]
```

Delegation on valideColumn [Grid](#) function

Parameters

<i>column</i>	column to validate
---------------	--------------------

Returns

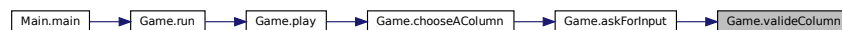
the valide column

Definition at line 383 of file [Game.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.23 valideGameOption()**

```
int Game.valideGameOption (
    int option ) [private]
```

Validation of game option

Parameters

<i>option</i>	game option to validate
---------------	-------------------------

Exceptions

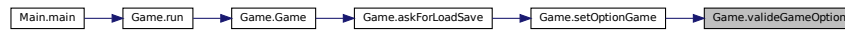
<i>IllegalArgumentException</i>	
---------------------------------	--

Returns

valide game option

Definition at line 197 of file [Game.java](#).

Here is the caller graph for this function:



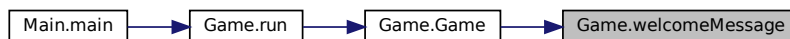
4.6.3.24 welcomeMessage()

```
void Game.welcomeMessage ( ) [private]
```

Print a welcome message

Definition at line 158 of file [Game.java](#).

Here is the caller graph for this function:



4.6.4 Member Data Documentation

4.6.4.1 arrayPlayer

```
Player [ ] Game.arrayPlayer [private]
```

Definition at line 20 of file [Game.java](#).

4.6.4.2 currentPlayerId

```
int Game.currentPlayerId [private]
```

Definition at line 31 of file [Game.java](#).

4.6.4.3 end

```
boolean Game.end [private]
```

Definition at line 27 of file [Game.java](#).

4.6.4.4 gameplay

```
String Game.gameplay [private]
```

Definition at line 29 of file [Game.java](#).

4.6.4.5 grid

```
Grid Game.grid [private]
```

Definition at line 21 of file [Game.java](#).

4.6.4.6 iteration

```
int Game.iteration [private]
```

Definition at line 28 of file [Game.java](#).

4.6.4.7 lastMove

```
int Game.lastMove [private]
```

Definition at line 32 of file [Game.java](#).

4.6.4.8 numberOfPlayers

```
final int Game.numberOfPlayers = 2 [static]
```

Definition at line 16 of file [Game.java](#).

4.6.4.9 numberOfTokenToWin

```
final int Game.numberOfTokenToWin = 4 [static]
```

Definition at line 17 of file [Game.java](#).

4.6.4.10 save

```
final Save Game.save [private]
```

Definition at line 24 of file [Game.java](#).

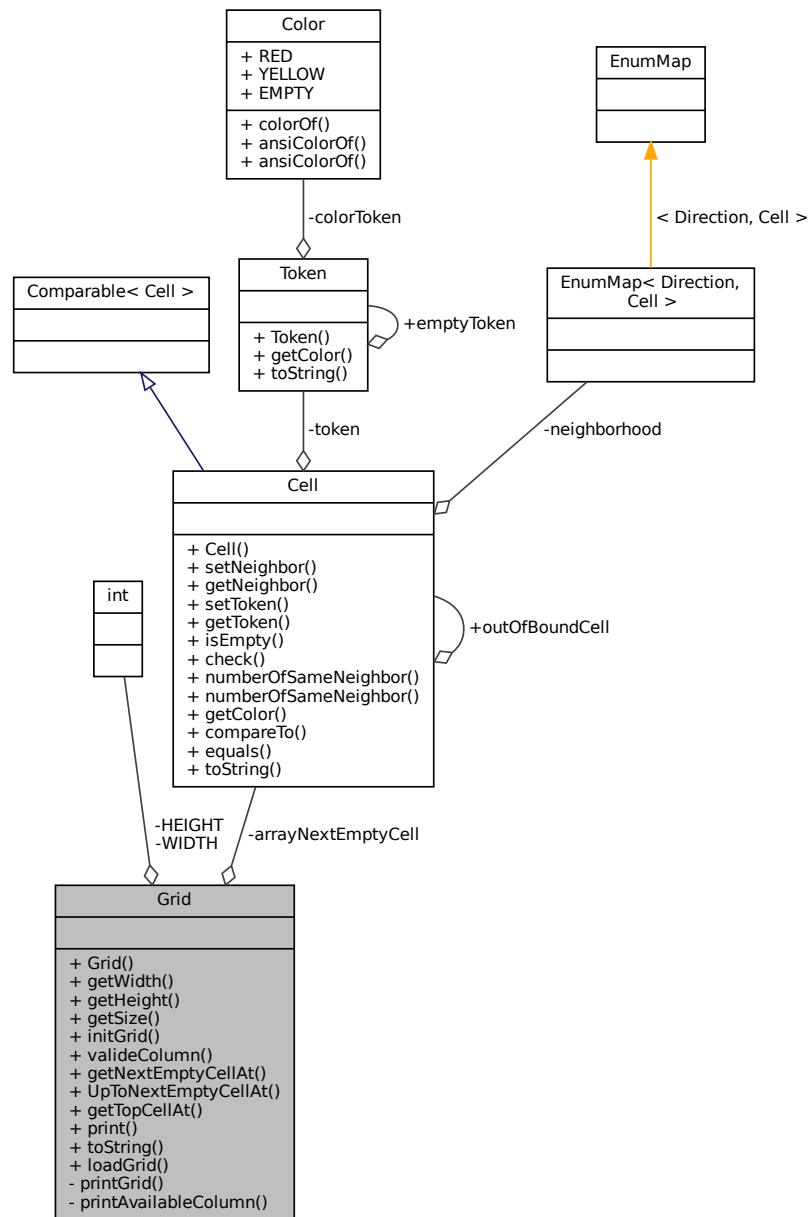
The documentation for this class was generated from the following file:

- [/home/hozen/cur/projet-java/src/Game.java](#)

4.7 Grid Class Reference

Logical representation of a board of the [Game](#) of Puissance 4.

Collaboration diagram for Grid:



Public Member Functions

- [Grid](#) ()
- [int getWidth](#) ()
- [int getHeight](#) ()
- [int getSize](#) ()
- [void initGrid](#) ()
- [int valideColumn](#) (int column)
- [Cell getNextEmptyCellAt](#) (int column)
- [void UpToNextEmptyCellAt](#) (int column)

- [Cell](#) [getTopCellAt](#) (int column)
- void [print](#) ()
- String [toString](#) ()
- void [loadGrid](#) (String schema, [Token](#)[] tokenOfPlayers)

Private Member Functions

- void [printGrid](#) ()
- void [printAvailableColumn](#) ()

Private Attributes

- [Cell](#)[] [arrayNextEmptyCell](#)

Static Private Attributes

- static final int [WIDTH](#) = 7
- static final int [HEIGHT](#) = 6

4.7.1 Detailed Description

Logical representation of a board of the [Game](#) of Puissance 4.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 10 of file [Grid.java](#).

4.7.2 Constructor & Destructor Documentation

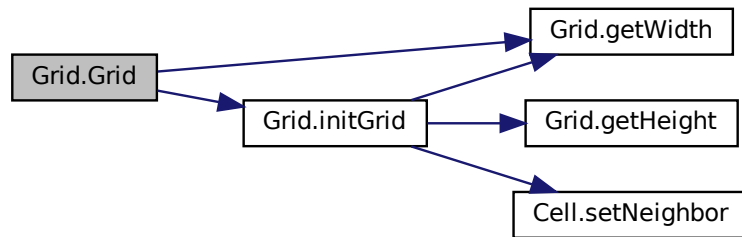
4.7.2.1 Grid()

`Grid.Grid ()`

[Grid](#) constructor

Definition at line 19 of file [Grid.java](#).

Here is the call graph for this function:



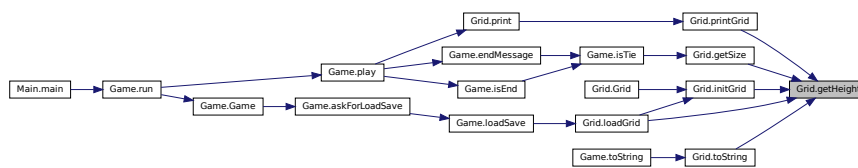
4.7.3 Member Function Documentation

4.7.3.1 getHeight()

`int Grid.getHeight ()`

Definition at line 29 of file [Grid.java](#).

Here is the caller graph for this function:



4.7.3.2 getNextEmptyCellAt()

`Cell Grid.getNextEmptyCellAt (`
`int column)`

Return the next empty cell in the specify column, if the column is full, return the top [Cell](#) of the column in the grid

Parameters

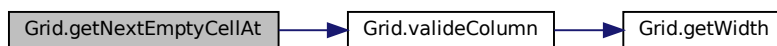
<i>column</i>	the column of the Grid where is the Cell
---------------	--

Returns

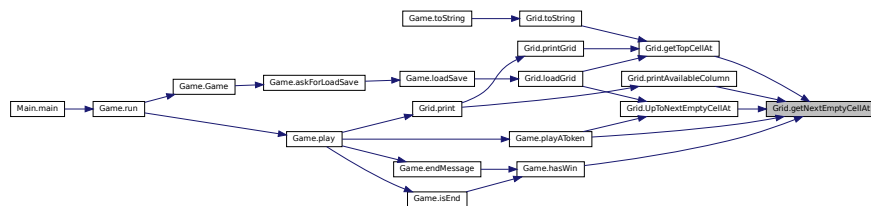
the [Cell](#) corresponding to the given column

Definition at line 78 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

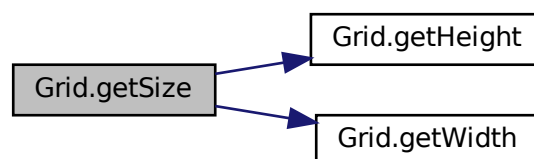


4.7.3.3 getSize()

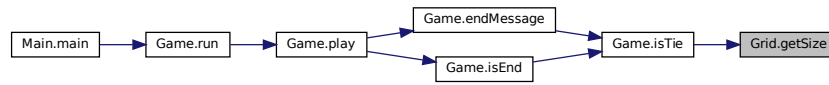
```
int Grid.getSize ( )
```

Definition at line 31 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.4 getTopCellAt()

```
Cell Grid.getTopCellAt (
    int column )
```

Return the top [Cell](#) in the specify column

Parameters

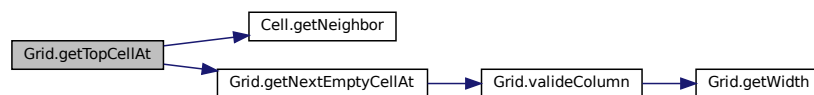
<i>column</i>	the column to get the top Cell
---------------	--

Returns

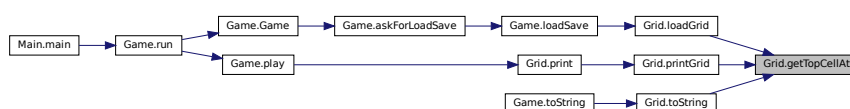
the top [Cell](#) of the column

Definition at line 99 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

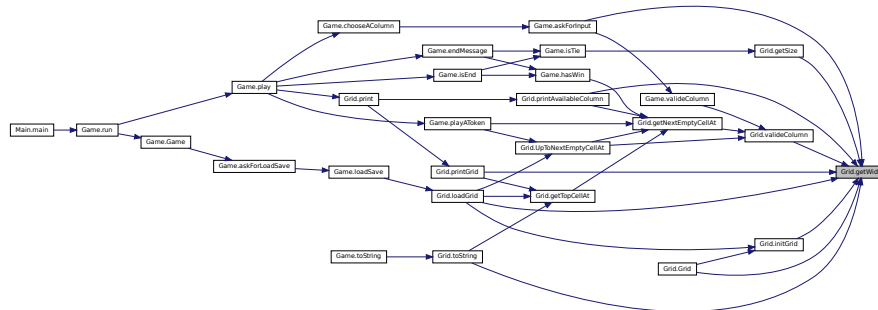


4.7.3.5 getWidth()

```
int Grid.getWidth ( )
```

Definition at line 27 of file Grid.java.

Here is the caller graph for this function:



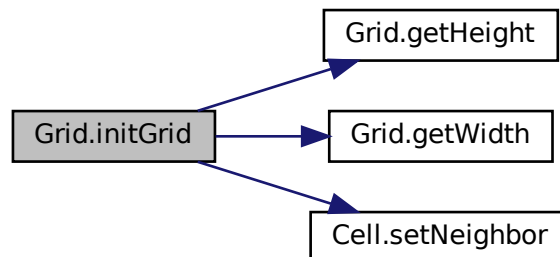
4.7.3.6 initGrid()

```
void Grid::initGrid ( )
```

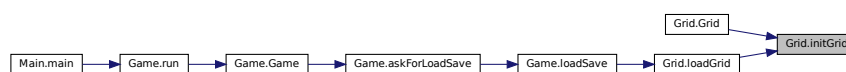
Initialisation of **Cell** of the **Grid** with their 4 neighbor

Definition at line 33 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.7 loadGrid()

```
void Grid.loadGrid (
    String schema,
    Token[] tokenOfPlayers )
```

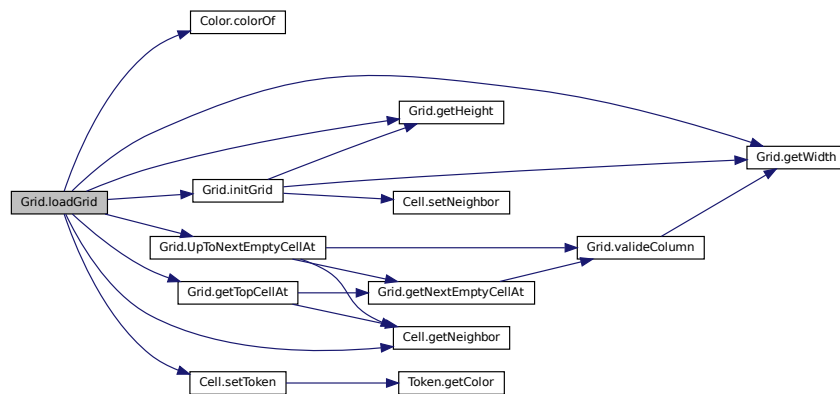
Load a [Grid](#) from a save (based on its own [toString\(\)](#) method)

Parameters

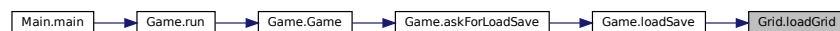
<i>schema</i>	the grid representation
<i>tokenOfPlayers</i>	a tab of Player Token

Definition at line 231 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



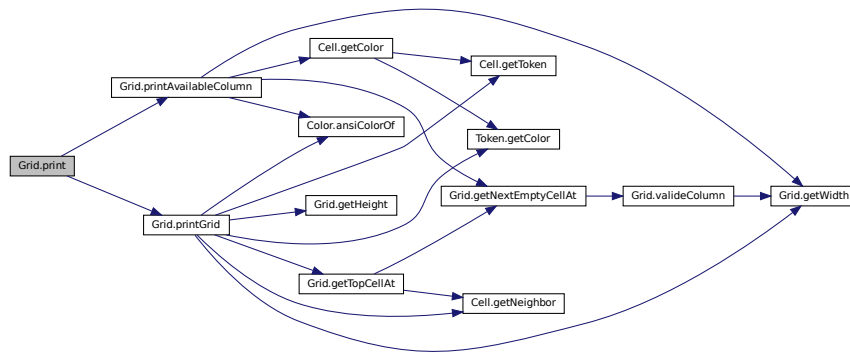
4.7.3.8 print()

```
void Grid.print ( )
```

Print the pretty print of the grid and column available

Definition at line 199 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



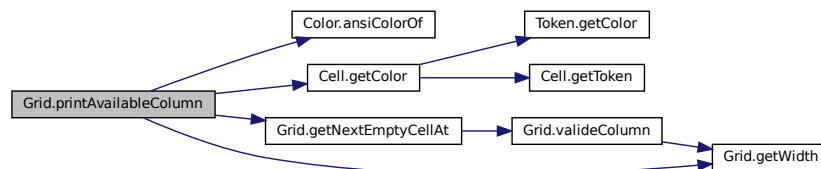
4.7.3.9 printAvailableColumn()

```
void Grid.printAvailableColumn ( ) [private]
```

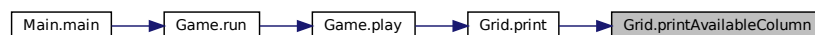
Pretty print of the column available to put a token in, print a 'X' when the column is full of [Token](#)

Definition at line 171 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



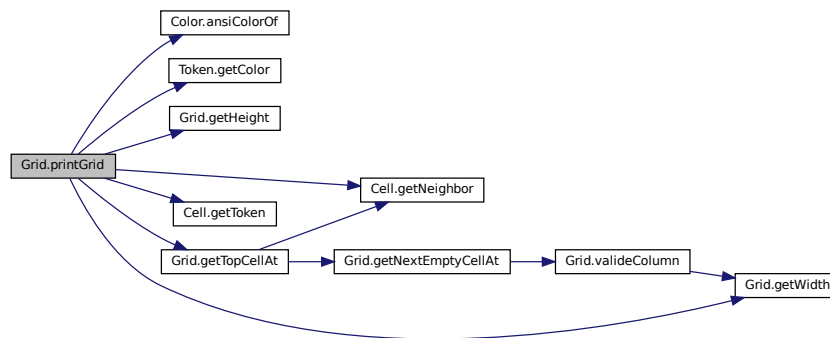
4.7.3.10 printGrid()

```
void Grid.printGrid ( ) [private]
```

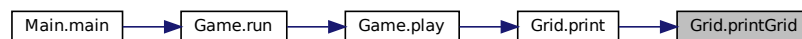
Pretty print of the grid with ansi color corresponding to the tokens in it

Definition at line 114 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.11 toString()

```
String Grid.toString ( )
```

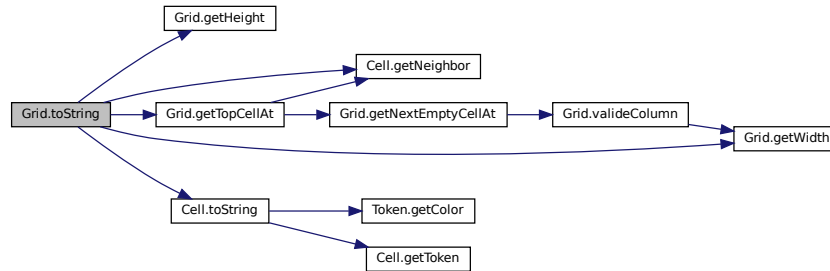
Return the String representation of the [Grid](#) : the color of each [Token](#) of each [Cell](#)

Returns

the String of [Color](#) separated by ';'.

Definition at line 212 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.12 UpToNextEmptyCellAt()**

```
void Grid.UpToNextEmptyCellAt (
    int column )
```

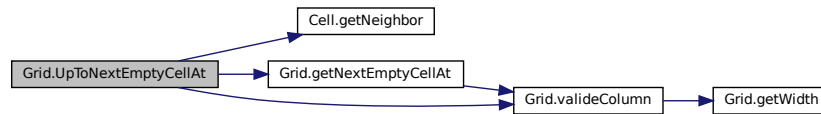
Update the next EMPTY [Cell](#) in the column

Parameters

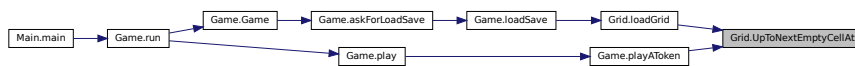
<i>column</i>	the column of the Grid where is the Cell
---------------	--

Definition at line 89 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.13 valideColumn()

```
int Grid.valideColumn (
    int column )
```

Verify the validity of a column (range of width)

Parameters

<i>column</i>	the column to verify
---------------	----------------------

Exceptions

<i>IllegalArgumentException</i>	
---------------------------------	--

Returns

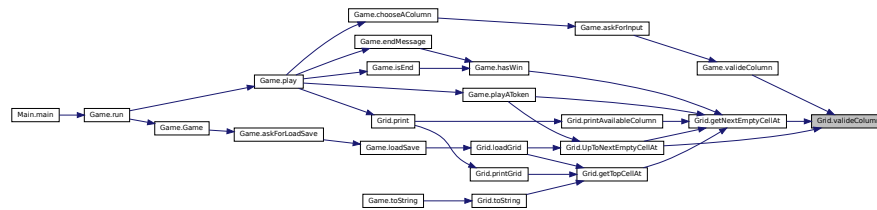
the valide column

Definition at line 64 of file [Grid.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.4 Member Data Documentation

4.7.4.1 arrayNextEmptyCell

```
Cell [] Grid.arrayNextEmptyCell [private]
```

Definition at line 15 of file [Grid.java](#).

4.7.4.2 HEIGHT

```
final int Grid.HEIGHT = 6 [static], [private]
```

Definition at line 13 of file [Grid.java](#).

4.7.4.3 WIDTH

```
final int Grid.WIDTH = 7 [static], [private]
```

Definition at line 12 of file [Grid.java](#).

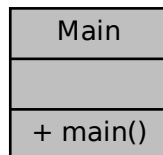
The documentation for this class was generated from the following file:

- [/home/hozen/cur/projet-java/src/Grid.java](#)

4.8 Main Class Reference

[Main](#) class, program entry point.

Collaboration diagram for Main:



Static Public Member Functions

- static void [main](#) (String[] args)

4.8.1 Detailed Description

[Main](#) class, program entry point.

Author

Durel Enzo
Villepreux Thibault

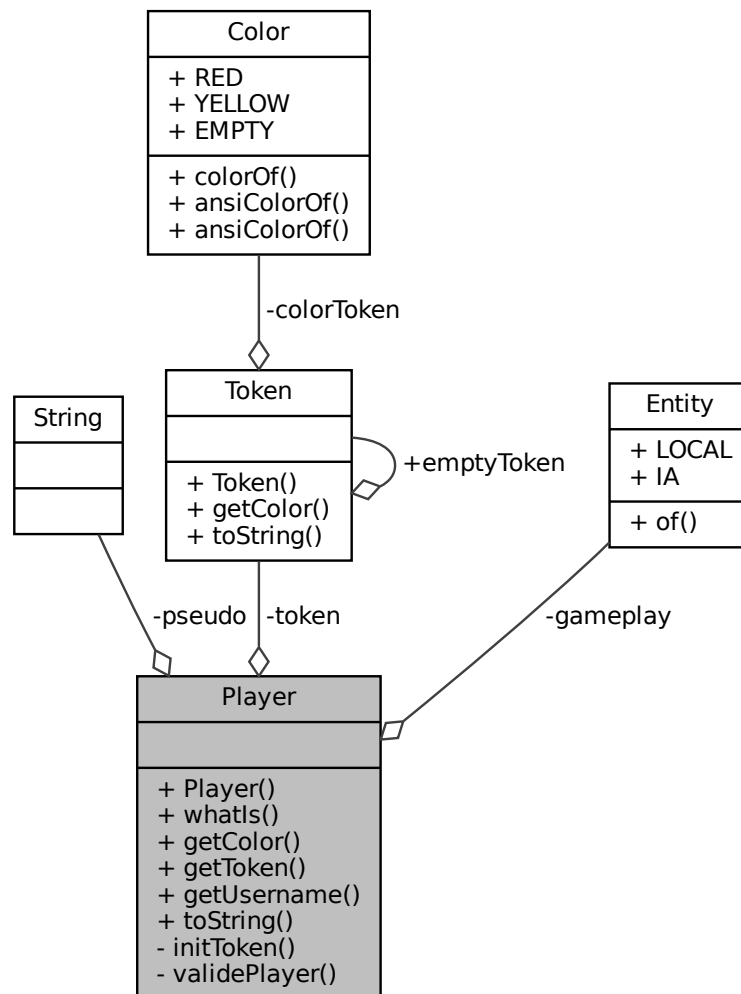
Version

1.0

Definition at line 7 of file [Main.java](#).

4.8.2 Member Function Documentation

Collaboration diagram for Player:



Public Member Functions

- `Player` (`String pseudo`, `int indexPlayer`, `Entity e`)
- `Entity whatIs ()`
- `Color getColor ()`
- `Token getToken ()`
- `String getUsername ()`
- `String toString ()`

Private Member Functions

- `Token initToken` (`int id`)
- `int validePlayer` (`int id`)

Private Attributes

- final String [pseudo](#)
- final [Token](#) token
- final [Entity](#) gameplay

4.9.1 Detailed Description

Represent a [Player](#) of a [Game](#) using [Token](#).

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 7 of file [Player.java](#).

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Player()

```
Player.Player (
    String pseudo,
    int indexPlayer,
    Entity e )
```

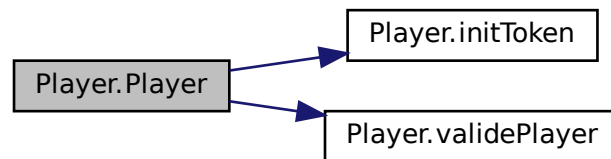
[Player](#) constructor

Parameters

<i>pseudo</i>	username of the player
<i>indexPlayer</i>	id of the player, decide its Token Color
<i>e</i>	Entity of the Player (LOCAL or IA)

Definition at line 13 of file [Player.java](#).

Here is the call graph for this function:



4.9.3 Member Function Documentation

4.9.3.1 getColor()

```
Color Player.getColor ( )
```

Give the [Color](#) of the [Token](#) of the [Player](#) (Delegation on [Token getColor\(\)](#))

Returns

[Color](#)

Definition at line [63](#) of file [Player.java](#).

Here is the call graph for this function:



4.9.3.2 getToken()

```
Token Player.getToken ( )
```

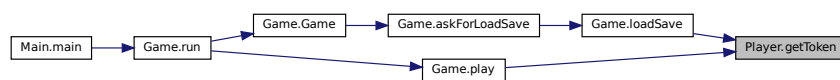
Give the reference of the [Token](#) of the [Player](#)

Returns

[Player's Token](#)

Definition at line 72 of file [Player.java](#).

Here is the caller graph for this function:



4.9.3.3 getUsername()

```
String Player.getUsername ( )
```

Give the username of the [Player](#)

Returns

String corresponding to the username

Definition at line 81 of file [Player.java](#).

4.9.3.4 initToken()

```
Token Player.initToken (
    int id ) [private]
```

Initialisation of the [Player Token](#) : give the [Token](#) in function of index

Parameters

<i>id</i>	index of player
-----------	-----------------

Returns

the [Token](#) of the [Player](#)

Definition at line 26 of file [Player.java](#).

Here is the caller graph for this function:

**4.9.3.5 toString()**

```
String Player.toString ( )
```

Give the String representation of the [Player](#)

Returns

String format ([Entity](#), String username, ([Token](#)'s [Color](#)))

Definition at line 91 of file [Player.java](#).

Here is the caller graph for this function:

**4.9.3.6 validePlayer()**

```
int Player.validePlayer (
    int id ) [private]
```

Validate the index of a [Player](#)

Parameters

<i>id</i>	index to validate
-----------	-------------------

Exceptions

<i>IllegalArgumentException</i>	
---------------------------------	--

Returns

the valide index

Definition at line 40 of file [Player.java](#).

Here is the caller graph for this function:



4.9.3.7 whatIs()

```
Entity Player.whatIs ( )
```

Give the [Entity](#) of the [Player](#)

Returns

[Entity](#)

Definition at line 54 of file [Player.java](#).

4.9.4 Member Data Documentation

4.9.4.1 gameplay

```
final Entity Player.gameplay [private]
```

Definition at line 11 of file [Player.java](#).

4.9.4.2 pseudo

```
final String Player.pseudo [private]
```

Definition at line 9 of file [Player.java](#).

4.9.4.3 token

```
final Token Player.token [private]
```

Definition at line 10 of file [Player.java](#).

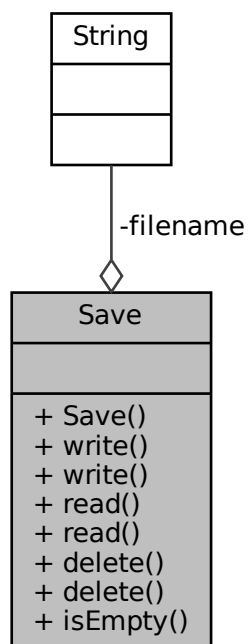
The documentation for this class was generated from the following file:

- [/home/hozen/cur/projet-java/src/Player.java](#)

4.10 Save Class Reference

Represent a global [Save](#) class.

Collaboration diagram for Save:



Public Member Functions

- [Save](#) (String [filename](#))
- void [write](#) (boolean verbose, Object src)
- void [write](#) (Object src)
- String [read](#) (boolean verbose)
- String [read](#) ()
- void [delete](#) (boolean verbose)
- void [delete](#) ()
- boolean [isEmpty](#) ()

Private Attributes

- String [filename](#)

4.10.1 Detailed Description

Represent a global [Save](#) class.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line [13](#) of file [Save.java](#).

4.10.2 Constructor & Destructor Documentation

4.10.2.1 [Save\(\)](#)

```
Save.Save (  
    String filename )
```

[Save](#) constructor, access with fullname path of the file

Parameters

<i>filename</i>	filename to write, read, delete
-----------------	---------------------------------

Definition at line [17](#) of file [Save.java](#).

4.10.3 Member Function Documentation

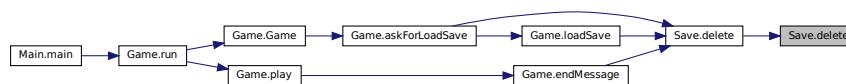
4.10.3.1 `delete()` [1/2]

```
void Save.delete ( )
```

Delete all the content from the file

Definition at line 99 of file [Save.java](#).

Here is the caller graph for this function:



4.10.3.2 `delete()` [2/2]

```
void Save.delete (
    boolean verbose )
```

Verbose method of [delete\(\)](#)

Parameters

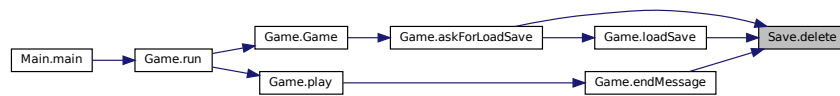
<i>verbose</i>	if true print a successful message
----------------	------------------------------------

Definition at line 89 of file [Save.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.3 isEmpty()

```
boolean Save.isEmpty ( )
```

Response if the file content is empty

Returns

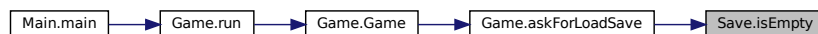
boolean true if file is empty, else false

Definition at line 111 of file [Save.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.4 read() [1/2]

```
String Save.read ( )
```

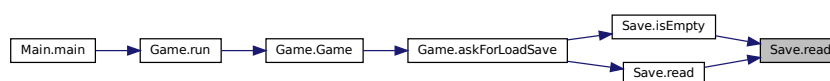
Read from a file and give its contents as a String

Returns

String representation of the content of the file

Definition at line 65 of file [Save.java](#).

Here is the caller graph for this function:

**4.10.3.5 read()** [2/2]

```
String Save.read (
    boolean verbose )
```

Verbose method of [read\(\)](#)

Parameters

<i>verbose</i>	if true print a successful message
----------------	------------------------------------

Returns

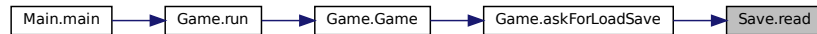
String representation of the content of the file

Definition at line 53 of file [Save.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.6 write() [1/2]

```
void Save.write (
    boolean verbose,
    Object src )
```

Verbose method of [write\(Object src\)](#)

Parameters

<i>verbose</i>	if true print a successful message
<i>src</i>	Object to write in the save

Definition at line 26 of file [Save.java](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.7 write() [2/2]

```
void Save.write (
    Object src )
```

Call the `toString` of the object and write it on the file

Parameters

src	Object to write in the save
-----	-----------------------------

Definition at line 37 of file [Save.java](#).

4.10.4 Member Data Documentation

4.10.4.1 filename

```
String Save.filename [private]
```

Definition at line 15 of file [Save.java](#).

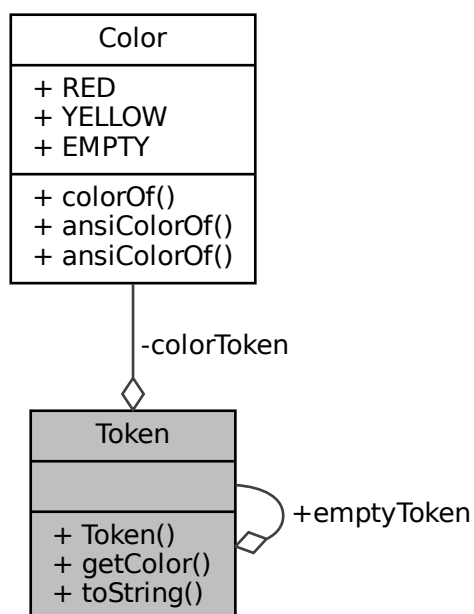
The documentation for this class was generated from the following file:

- [/home/hozen/cur/projet-java/src/Save.java](#)

4.11 Token Class Reference

Represent a [Token](#) of Puissance 4.

Collaboration diagram for Token:



Public Member Functions

- [Token](#) ([Color](#) c)
- [Color](#) [getColor](#) ()
- [String](#) [toString](#) ()

Static Public Attributes

- static final [Token](#) [emptyToken](#) = new [Token](#)([Color.EMPTY](#))

Private Attributes

- final [Color](#) [colorToken](#)

4.11.1 Detailed Description

Represent a [Token](#) of Puissance 4.

Author

Durel Enzo
Villepreux Thibault

Version

1.0

Definition at line 7 of file [Token.java](#).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 [Token](#)()

```
Token.Token (
    Color c )
```

[Token](#) constructor

Parameters

c	Color of the Token
-------------------	--

Definition at line 13 of file [Token.java](#).

4.11.3 Member Function Documentation

4.11.3.1 getColor()

`Color Token.getColor ()`

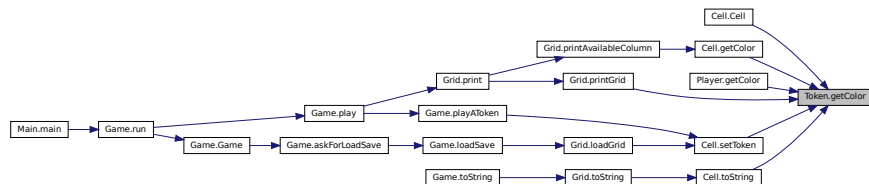
Give the `Color` of the `Token`

Returns

`Color`

Definition at line 22 of file `Token.java`.

Here is the caller graph for this function:



4.11.3.2 toString()

`String Token.toString ()`

Give the String representation of a `Token`

Returns

String representing the `Color` of the `Token`

Definition at line 32 of file `Token.java`.

Here is the caller graph for this function:



4.11.4 Member Data Documentation

4.11.4.1 colorToken

```
final Color Token.colorToken [private]
```

Definition at line 11 of file [Token.java](#).

4.11.4.2 emptyToken

```
final Token Token.emptyToken = new Token(Color.EMPTY) [static]
```

Definition at line 9 of file [Token.java](#).

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

- [/home/hozen/cur/projet-java/src/Token.java](#)

Chapter 5

File Documentation

5.1 /home/hozen/cur/projet-java/src/Cell.java File Reference

Classes

- class [Cell](#)

Logical representation of a case in the grid of the [Game](#) of Puissance 4.

5.2 Cell.java

[Go to the documentation of this file.](#)

```
00001 import static java.util.Objects.requireNonNull;
00002 import java.util.EnumMap;
00003
00011 public class Cell implements Comparable<Cell> {
00012
00013     public final static Cell outOfBoundCell = new Cell(Token.emptyToken);
00014
00015     private Token token;
00016     private EnumMap<Direction, Cell> neighborhood;
00017
00018     public Cell(Token token) {
00024         this.neighborhood = new EnumMap<>(Direction.class);
00025         if (token.getColor() == Color.EMPTY) this.token = Token.emptyToken;
00026     }
00027
00028     public void setNeighbor(Cell c, Direction d) {
00035         this.neighborhood.put(requireNonNull(d), requireNonNull(c));
00036     }
00037
00038     public Cell getNeighbor(Direction d) {
00045         Cell get;
00046         get = this.neighborhood.get(requireNonNull(d));
00047         if (get == null) {
00048             get = this.outOfBoundCell;
00049         }
00050         return get;
00051     }
00052
00053     public void setToken(Token t) {
00059         if (this.token.getColor() != Color.EMPTY) {
00060             throw new IllegalArgumentException("There is already a token to this cell");
00061         }
00062         this.token = requireNonNull(t);
00063     }
00064
00065     public Token getToken() {
00071         return this.token;
00072     }
00073
00074     public boolean isEmpty() {
```

```

00080     return this.token.toString() == "EMPTY";
00081 }
00082
00083 public boolean check() {
00090     EnumMap<Direction, Direction> diagonales = Direction.getDiagonales();
00091     int count;
00092
00093     for (Direction d: Direction.values()) {
00094         /* Horizontal and vertical check */
00095         count = this.numberOfSameNeighbor(d);
00096         count += this.numberOfSameNeighbor(Direction.getOpposite(d));
00097         count++; // Pour compter le pion actuel
00098
00099         if (count >= Game.numberOfTokenToWin) return true;
00100
00101         /* Diagonales check */
00102         count = this.numberOfSameNeighbor(d, diagonales.get(d));
00103         count += this.numberOfSameNeighbor(Direction.getOpposite(d),
00104             Direction.getOpposite(diagonales.get(d)));
00105         count++;
00106
00107         if (count >= Game.numberOfTokenToWin) return true;
00108     }
00109     return false;
00110 }
00111
00112 public int numberOfSameNeighbor(Direction d) {
00119     Cell next = this.getNeighbor(d);
00120     if (next == this.outOfBoundCell ||
00121         this.getToken() != next.getToken()) {
00122         return 0;
00123     }
00124     else return 1 + next.numberOfSameNeighbor(d);
00125 }
00126
00127 public int numberOfSameNeighbor(Direction d1, Direction d2) {
00135     Cell next = this.getNeighbor(d1).getNeighbor(d2);
00136     if (next == this.outOfBoundCell ||
00137         this.getToken() != next.getToken()) {
00138         return 0;
00139     }
00140     else return 1 + next.numberOfSameNeighbor(d1, d2);
00141 }
00142
00143 public Color getColor() {
00149     return this.getToken().getColor();
00150 }
00151
00152 @Override
00153 public int compareTo(Cell other) {
00160     if (this.getToken() == other.getToken()) return 0;
00161     return 1;
00162 }
00163
00164 @Override
00165 public boolean equals(Object other) {
00172     if (other == null) return false;
00173     if (other == this) return true;
00174     if (!(other instanceof Cell)) return false;
00175     return this.compareTo((Cell) other) == 0;
00176 }
00177
00178 @Override
00179 public String toString() {
00185     return this.getToken().getColor().toString();
00186 }
00187 }

```

5.3 /home/hozen/cur/projet-java/src/Color.java File Reference

Classes

- enum [Color](#)

Represent colors globally.

5.4 Color.java

[Go to the documentation of this file.](#)

```

00001
00007 public enum Color {
00008
00009     RED, YELLOW, EMPTY;
00010
00011     public static Color colorOf(String colorString) {
00012         switch (colorString) {
00013             case "RED":
00014                 return Color.RED;
00015             case "YELLOW":
00016                 return Color.YELLOW;
00017         }
00018         return Color.EMPTY;
00019     }
00020
00027     public static String ansiColorOf(Color c) {
00028         switch (c) {
00029             case RED :
00030                 return ansiColorOf("RED");
00031             case YELLOW :
00032                 return ansiColorOf("YELLOW");
00033         }
00034         return ansiColorOf("WHITE");
00035     }
00036
00043     public static String ansiColorOf(String c) {
00044         switch (c) {
00045             case "RED":
00046                 return "\u001B[31m"; // RED
00047             case "YELLOW":
00048                 return "\u001B[33m"; // YELLOW
00049             case "BLUE":
00050                 return "\u001B[36m"; // BLUE
00051             case "GREEN":
00052                 return "\u001B[32m"; // GREEN
00053             case "WHITE":
00054                 return "\u001B[37m"; // WHITE
00055         }
00056         System.out.println("There is no color corresponding to the argument: white is returning");
00057         return ansiColorOf("WHITE");
00058     }
00059 }

```

5.5 /home/hozen/cur/projet-java/src/Direction.java File Reference

Classes

- enum [Direction](#)

Represent different direction with diagonales.

5.6 Direction.java

[Go to the documentation of this file.](#)

```

00001 import java.util.EnumMap;
00002
00009 public enum Direction {
00010
00011     UP, DOWN, RIGHT, LEFT;
00012
00013     public static EnumMap<Direction, Direction> getDiagonales() {
00014         EnumMap<Direction, Direction> diagonales = new EnumMap<>(Direction.class);
00015         diagonales.put(Direction.UP, Direction.LEFT);
00016         diagonales.put(Direction.DOWN, Direction.RIGHT);
00017         diagonales.put(Direction.LEFT, Direction.DOWN);
00018         diagonales.put(Direction.RIGHT, Direction.UP);
00019         return diagonales;
00020     }
00021 }

```

```

00028     public static Direction getOpposite(Direction d) {
00035     switch (d) {
00036     case UP:
00037         return DOWN;
00038     case DOWN:
00039         return UP;
00040     case RIGHT:
00041         return LEFT;
00042     case LEFT:
00043         return RIGHT;
00044     }
00045     return UP;
00046     }
00047 }

```

5.7 /home/hozen/cur/projet-java/src/Entity.java File Reference

Classes

- enum [Entity](#)
Logical representation of a [Player](#) (IA, LOCAL(human))

5.8 Entity.java

[Go to the documentation of this file.](#)

```

00001
00007 public enum Entity {
00008
00009     LOCAL, IA;
00010
00011     public static Entity of(String e) {
00018     Entity option = LOCAL;
00019     switch (e) {
00020     case "LOCAL":
00021         option = LOCAL;
00022         break;
00023     case "IA":
00024         option = IA;
00025         break;
00026     }
00027     return option;
00028     }
00029 }

```

5.9 /home/hozen/cur/projet-java/src/Game.java File Reference

Classes

- class [Game](#)
Represent a [Game](#) of Puissance 4.

5.10 Game.java

[Go to the documentation of this file.](#)

```

00001 import java.util.Random;
00002 import java.util.Scanner;
00003 import java.util.InputMismatchException;
00004 import static java.util.Objects.requireNonNull;
00005 import java.util.EnumMap;
00006 import java.lang.Math;
00007
00014 public class Game {
00015     // Const
00016     public final static int numberOfPlayers = 2;
00017     public final static int numberOfTokenToWin = 4;
00018
00019     // Gameplay attributs
00020     private Player[] arrayPlayer;
00021     private Grid grid;
00022
00023     // Save
00024     private final Save save;
00025
00026     // Game state attributs
00027     private boolean end;
00028     private int iteration;
00029     private String gameplay; // IA or LOCAL
00030
00031     private int currentPlayerId;
00032     private int lastMove; // represent the last column played
00033
00034     private Game() {
00035         this.welcomeMessage();
00036
00037         // Init Player physical representation
00038         this.arrayPlayer = new Player[numberOfPlayers];
00039
00040         // Init Grid
00041         this.grid = new Grid();
00042
00043         // Init random first Player
00044         this.currentPlayerId = (int) Math.round(Math.random());
00045
00046         // Init Save
00047         this.save = new Save("../sauv");
00048         this.askForLoadSave();
00049
00050         // Init Game State
00051         this.end = false;
00052         this.iteration = 0;
00053     }
00054
00055     private void askForLoadSave() {
00056         if (!this.save.isEmpty()) {
00057             boolean done = false;
00058             while (!done) {
00059                 System.out.println("Sauvegarde est existante, voulez-vous la charger ?");
00060                 System.out.println("[1:Oui] [2:Non] [3:Supprimer]");
00061                 Scanner input = (new Scanner(System.in));
00062                 try {
00063                     switch (input.nextInt()) {
00064                         case 1:
00065                             this.loadSave(this.save.read());
00066                             break;
00067                         case 3:
00068                             this.save.delete();
00069                             this.setOptionGame();
00070                             this.askForPlayers();
00071                             break;
00072                         default:
00073                             this.setOptionGame();
00074                             this.askForPlayers();
00075                             break;
00076                     }
00077                     done = true;
00078                 } catch (InputMismatchException e) {
00079                     System.out.println("Ce choix n'existe pas");
00080                 }
00081             }
00082         } else {
00083             this.setOptionGame();
00084             this.askForPlayers();
00085         }
00086     }
00087
00088     private void welcomeMessage() {
00089         System.out.println("Bienvenue dans le jeu de la bataille navale");
00090     }
00091
00092     private void setOptionGame() {
00093         System.out.println("Choisissez l'option de jeu");
00094     }
00095
00096     private void askForPlayers() {
00097         System.out.println("Entrez le nombre de joueurs");
00098     }
00099
00100     private void loadSave(String save) {
00101         System.out.println("Chargement de la sauvegarde");
00102     }
00103
00104     private void deleteSave() {
00105         System.out.println("Suppression de la sauvegarde");
00106     }
00107
00108     private void setGrid() {
00109         System.out.println("Initialisation du jeu");
00110     }
00111
00112     private void setPlayer() {
00113         System.out.println("Initialisation des joueurs");
00114     }
00115
00116     private void setIteration() {
00117         System.out.println("Initialisation de l'iteration");
00118     }
00119
00120     private void setEnd() {
00121         System.out.println("Initialisation de l'etat du jeu");
00122     }
00123
00124     private void setGameplay() {
00125         System.out.println("Initialisation du jeu");
00126     }
00127
00128     private void setLastMove() {
00129         System.out.println("Initialisation du jeu");
00130     }
00131
00132     private void setCurrentPlayerId() {
00133         System.out.println("Initialisation du jeu");
00134     }
00135
00136     private void setArrayPlayer() {
00137         System.out.println("Initialisation du jeu");
00138     }
00139
00140     private void setGrid() {
00141         System.out.println("Initialisation du jeu");
00142     }
00143
00144     private void setPlayer() {
00145         System.out.println("Initialisation des joueurs");
00146     }
00147
00148     private void setIteration() {
00149         System.out.println("Initialisation de l'iteration");
00150     }
00151
00152     private void setEnd() {
00153         System.out.println("Initialisation de l'etat du jeu");
00154     }
00155
00156     private void setGameplay() {
00157         System.out.println("Initialisation du jeu");
00158     }
00159
00160     private void setLastMove() {
00161         System.out.println("Initialisation du jeu");
00162     }
00163
00164     private void setCurrentPlayerId() {
00165         System.out.println("Initialisation du jeu");
00166     }
00167
00168     private void setArrayPlayer() {
00169         System.out.println("Initialisation du jeu");
00170     }
00171
00172     private void setGrid() {
00173         System.out.println("Initialisation du jeu");
00174     }
00175
00176     private void setPlayer() {
00177         System.out.println("Initialisation des joueurs");
00178     }
00179
00180     private void setIteration() {
00181         System.out.println("Initialisation de l'iteration");
00182     }
00183
00184     private void setEnd() {
00185         System.out.println("Initialisation de l'etat du jeu");
00186     }
00187
00188     private void setGameplay() {
00189         System.out.println("Initialisation du jeu");
00190     }
00191
00192     private void setLastMove() {
00193         System.out.println("Initialisation du jeu");
00194     }
00195
00196     private void setCurrentPlayerId() {
00197         System.out.println("Initialisation du jeu");
00198     }
00199
00200     private void setArrayPlayer() {
00201         System.out.println("Initialisation du jeu");
00202     }
00203
00204     private void setGrid() {
00205         System.out.println("Initialisation du jeu");
00206     }
00207
00208     private void setPlayer() {
00209         System.out.println("Initialisation des joueurs");
00210     }
00211
00212     private void setIteration() {
00213         System.out.println("Initialisation de l'iteration");
00214     }
00215
00216     private void setEnd() {
00217         System.out.println("Initialisation de l'etat du jeu");
00218     }
00219
00220     private void setGameplay() {
00221         System.out.println("Initialisation du jeu");
00222     }
00223
00224     private void setLastMove() {
00225         System.out.println("Initialisation du jeu");
00226     }
00227
00228     private void setCurrentPlayerId() {
00229         System.out.println("Initialisation du jeu");
00230     }
00231
00232     private void setArrayPlayer() {
00233         System.out.println("Initialisation du jeu");
00234     }
00235
00236     private void setGrid() {
00237         System.out.println("Initialisation du jeu");
00238     }
00239
00240     private void setPlayer() {
00241         System.out.println("Initialisation des joueurs");
00242     }
00243
00244     private void setIteration() {
00245         System.out.println("Initialisation de l'iteration");
00246     }
00247
00248     private void setEnd() {
00249         System.out.println("Initialisation de l'etat du jeu");
00250     }
00251
00252     private void setGameplay() {
00253         System.out.println("Initialisation du jeu");
00254     }
00255
00256     private void setLastMove() {
00257         System.out.println("Initialisation du jeu");
00258     }
00259
00260     private void setCurrentPlayerId() {
00261         System.out.println("Initialisation du jeu");
00262     }
00263
00264     private void setArrayPlayer() {
00265         System.out.println("Initialisation du jeu");
00266     }
00267
00268     private void setGrid() {
00269         System.out.println("Initialisation du jeu");
00270     }
00271
00272     private void setPlayer() {
00273         System.out.println("Initialisation des joueurs");
00274     }
00275
00276     private void setIteration() {
00277         System.out.println("Initialisation de l'iteration");
00278     }
00279
00280     private void setEnd() {
00281         System.out.println("Initialisation de l'etat du jeu");
00282     }
00283
00284     private void setGameplay() {
00285         System.out.println("Initialisation du jeu");
00286     }
00287
00288     private void setLastMove() {
00289         System.out.println("Initialisation du jeu");
00290     }
00291
00292     private void setCurrentPlayerId() {
00293         System.out.println("Initialisation du jeu");
00294     }
00295
00296     private void setArrayPlayer() {
00297         System.out.println("Initialisation du jeu");
00298     }
00299
00300     private void setGrid() {
00301         System.out.println("Initialisation du jeu");
00302     }
00303
00304     private void setPlayer() {
00305         System.out.println("Initialisation des joueurs");
00306     }
00307
00308     private void setIteration() {
00309         System.out.println("Initialisation de l'iteration");
00310     }
00311
00312     private void setEnd() {
00313         System.out.println("Initialisation de l'etat du jeu");
00314     }
00315
00316     private void setGameplay() {
00317         System.out.println("Initialisation du jeu");
00318     }
00319
00320     private void setLastMove() {
00321         System.out.println("Initialisation du jeu");
00322     }
00323
00324     private void setCurrentPlayerId() {
00325         System.out.println("Initialisation du jeu");
00326     }
00327
00328     private void setArrayPlayer() {
00329         System.out.println("Initialisation du jeu");
00330     }
00331
00332     private void setGrid() {
00333         System.out.println("Initialisation du jeu");
00334     }
00335
00336     private void setPlayer() {
00337         System.out.println("Initialisation des joueurs");
00338     }
00339
00340     private void setIteration() {
00341         System.out.println("Initialisation de l'iteration");
00342     }
00343
00344     private void setEnd() {
00345         System.out.println("Initialisation de l'etat du jeu");
00346     }
00347
00348     private void setGameplay() {
00349         System.out.println("Initialisation du jeu");
00350     }
00351
00352     private void setLastMove() {
00353         System.out.println("Initialisation du jeu");
00354     }
00355
00356     private void setCurrentPlayerId() {
00357         System.out.println("Initialisation du jeu");
00358     }
00359
00360     private void setArrayPlayer() {
00361         System.out.println("Initialisation du jeu");
00362     }
00363
00364     private void setGrid() {
00365         System.out.println("Initialisation du jeu");
00366     }
00367
00368     private void setPlayer() {
00369         System.out.println("Initialisation des joueurs");
00370     }
00371
00372     private void setIteration() {
00373         System.out.println("Initialisation de l'iteration");
00374     }
00375
00376     private void setEnd() {
00377         System.out.println("Initialisation de l'etat du jeu");
00378     }
00379
00380     private void setGameplay() {
00381         System.out.println("Initialisation du jeu");
00382     }
00383
00384     private void setLastMove() {
00385         System.out.println("Initialisation du jeu");
00386     }
00387
00388     private void setCurrentPlayerId() {
00389         System.out.println("Initialisation du jeu");
00390     }
00391
00392     private void setArrayPlayer() {
00393         System.out.println("Initialisation du jeu");
00394     }
00395
00396     private void setGrid() {
00397         System.out.println("Initialisation du jeu");
00398     }
00399
00400     private void setPlayer() {
00401         System.out.println("Initialisation des joueurs");
00402     }
00403
00404     private void setIteration() {
00405         System.out.println("Initialisation de l'iteration");
00406     }
00407
00408     private void setEnd() {
00409         System.out.println("Initialisation de l'etat du jeu");
00410     }
00411
00412     private void setGameplay() {
00413         System.out.println("Initialisation du jeu");
00414     }
00415
00416     private void setLastMove() {
00417         System.out.println("Initialisation du jeu");
00418     }
00419
00420     private void setCurrentPlayerId() {
00421         System.out.println("Initialisation du jeu");
00422     }
00423
00424     private void setArrayPlayer() {
00425         System.out.println("Initialisation du jeu");
00426     }
00427
00428     private void setGrid() {
00429         System.out.println("Initialisation du jeu");
00430     }
00431
00432     private void setPlayer() {
00433         System.out.println("Initialisation des joueurs");
00434     }
00435
00436     private void setIteration() {
00437         System.out.println("Initialisation de l'iteration");
00438     }
00439
00440     private void setEnd() {
00441         System.out.println("Initialisation de l'etat du jeu");
00442     }
00443
00444     private void setGameplay() {
00445         System.out.println("Initialisation du jeu");
00446     }
00447
00448     private void setLastMove() {
00449         System.out.println("Initialisation du jeu");
00450     }
00451
00452     private void setCurrentPlayerId() {
00453         System.out.println("Initialisation du jeu");
00454     }
00455
00456     private void setArrayPlayer() {
00457         System.out.println("Initialisation du jeu");
00458     }
00459
00460     private void setGrid() {
00461         System.out.println("Initialisation du jeu");
00462     }
00463
00464     private void setPlayer() {
00465         System.out.println("Initialisation des joueurs");
00466     }
00467
00468     private void setIteration() {
00469         System.out.println("Initialisation de l'iteration");
00470     }
00471
00472     private void setEnd() {
00473         System.out.println("Initialisation de l'etat du jeu");
00474     }
00475
00476     private void setGameplay() {
00477         System.out.println("Initialisation du jeu");
00478     }
00479
00480     private void setLastMove() {
00481         System.out.println("Initialisation du jeu");
00482     }
00483
00484     private void setCurrentPlayerId() {
00485         System.out.println("Initialisation du jeu");
00486     }
00487
00488     private void setArrayPlayer() {
00489         System.out.println("Initialisation du jeu");
00490     }
00491
00492     private void setGrid() {
00493         System.out.println("Initialisation du jeu");
00494     }
00495
00496     private void setPlayer() {
00497         System.out.println("Initialisation des joueurs");
00498     }
00499
00500     private void setIteration() {
00501         System.out.println("Initialisation de l'iteration");
00502     }
00503
00504     private void setEnd() {
00505         System.out.println("Initialisation de l'etat du jeu");
00506     }
00507
00508     private void setGameplay() {
00509         System.out.println("Initialisation du jeu");
00510     }
00511
00512     private void setLastMove() {
00513         System.out.println("Initialisation du jeu");
00514     }
00515
00516     private void setCurrentPlayerId() {
00517         System.out.println("Initialisation du jeu");
00518     }
00519
00520     private void setArrayPlayer() {
00521         System.out.println("Initialisation du jeu");
00522     }
00523
00524     private void setGrid() {
00525         System.out.println("Initialisation du jeu");
00526     }
00527
00528     private void setPlayer() {
00529         System.out.println("Initialisation des joueurs");
00530     }
00531
00532     private void setIteration() {
00533         System.out.println("Initialisation de l'iteration");
00534     }
00535
00536     private void setEnd() {
00537         System.out.println("Initialisation de l'etat du jeu");
00538     }
00539
00540     private void setGameplay() {
00541         System.out.println("Initialisation du jeu");
00542     }
00543
00544     private void setLastMove() {
00545         System.out.println("Initialisation du jeu");
00546     }
00547
00548     private void setCurrentPlayerId() {
00549         System.out.println("Initialisation du jeu");
00550     }
00551
00552     private void setArrayPlayer() {
00553         System.out.println("Initialisation du jeu");
00554     }
00555
00556     private void setGrid() {
00557         System.out.println("Initialisation du jeu");
00558     }
00559
00560     private void setPlayer() {
00561         System.out.println("Initialisation des joueurs");
00562     }
00563
00564     private void setIteration() {
00565         System.out.println("Initialisation de l'iteration");
00566     }
00567
00568     private void setEnd() {
00569         System.out.println("Initialisation de l'etat du jeu");
00570     }
00571
00572     private void setGameplay() {
00573         System.out.println("Initialisation du jeu");
00574     }
00575
00576     private void setLastMove() {
00577         System.out.println("Initialisation du jeu");
00578     }
00579
00580     private void setCurrentPlayerId() {
00581         System.out.println("Initialisation du jeu");
00582     }
00583
00584     private void setArrayPlayer() {
00585         System.out.println("Initialisation du jeu");
00586     }
00587
00588     private void setGrid() {
00589         System.out.println("Initialisation du jeu");
00590     }
00591
00592     private void setPlayer() {
00593         System.out.println("Initialisation des joueurs");
00594     }
00595
00596     private void setIteration() {
00597         System.out.println("Initialisation de l'iteration");
00598     }
00599
00600     private void setEnd() {
00601         System.out.println("Initialisation de l'etat du jeu");
00602     }
00603
00604     private void setGameplay() {
00605         System.out.println("Initialisation du jeu");
00606     }
00607
00608     private void setLastMove() {
00609         System.out.println("Initialisation du jeu");
00610     }
00611
00612     private void setCurrentPlayerId() {
00613         System.out.println("Initialisation du jeu");
00614     }
00615
00616     private void setArrayPlayer() {
00617         System.out.println("Initialisation du jeu");
00618     }
00619
00620     private void setGrid() {
00621         System.out.println("Initialisation du jeu");
00622     }
00623
00624     private void setPlayer() {
00625         System.out.println("Initialisation des joueurs");
00626     }
00627
00628     private void setIteration() {
00629         System.out.println("Initialisation de l'iteration");
00630     }
00631
00632     private void setEnd() {
00633         System.out.println("Initialisation de l'etat du jeu");
00634     }
00635
00636     private void setGameplay() {
00637         System.out.println("Initialisation du jeu");
00638     }
00639
00640     private void setLastMove() {
00641         System.out.println("Initialisation du jeu");
00642     }
00643
00644     private void setCurrentPlayerId() {
00645         System.out.println("Initialisation du jeu");
00646     }
00647
00648     private void setArrayPlayer() {
00649         System.out.println("Initialisation du jeu");
00650     }
00651
00652     private void setGrid() {
00653         System.out.println("Initialisation du jeu");
00654     }
00655
00656     private void setPlayer() {
00657         System.out.println("Initialisation des joueurs");
00658     }
00659
00660     private void setIteration() {
00661         System.out.println("Initialisation de l'iteration");
00662     }
00663
00664     private void setEnd() {
00665         System.out.println("Initialisation de l'etat du jeu");
00666     }
00667
00668     private void setGameplay() {
00669         System.out.println("Initialisation du jeu");
00670     }
00671
00672     private void setLastMove() {
00673         System.out.println("Initialisation du jeu");
00674     }
00675
00676     private void setCurrentPlayerId() {
00677         System.out.println("Initialisation du jeu");
00678     }
00679
00680     private void setArrayPlayer() {
00681         System.out.println("Initialisation du jeu");
00682     }
00683
00684     private void setGrid() {
00685         System.out.println("Initialisation du jeu");
00686     }
00687
00688     private void setPlayer() {
00689         System.out.println("Initialisation des joueurs");
00690     }
00691
00692     private void setIteration() {
00693         System.out.println("Initialisation de l'iteration");
00694     }
00695
00696     private void setEnd() {
00697         System.out.println("Initialisation de l'etat du jeu");
00698     }
00699
00700     private void setGameplay() {
00701         System.out.println("Initialisation du jeu");
00702     }
00703
00704     private void setLastMove() {
00705         System.out.println("Initialisation du jeu");
00706     }
00707
00708     private void setCurrentPlayerId() {
00709         System.out.println("Initialisation du jeu");
00710     }
00711
00712     private void setArrayPlayer() {
00713         System.out.println("Initialisation du jeu");
00714     }
00715
00716     private void setGrid() {
00717         System.out.println("Initialisation du jeu");
00718     }
00719
00720     private void setPlayer() {
00721         System.out.println("Initialisation des joueurs");
00722     }
00723
00724     private void setIteration() {
00725         System.out.println("Initialisation de l'iteration");
00726     }
00727
00728     private void setEnd() {
00729         System.out.println("Initialisation de l'etat du jeu");
00730     }
00731
00732     private void setGameplay() {
00733         System.out.println("Initialisation du jeu");
00734     }
00735
00736     private void setLastMove() {
00737         System.out.println("Initialisation du jeu");
00738     }
00739
00740     private void setCurrentPlayerId() {
00741         System.out.println("Initialisation du jeu");
00742     }
00743
00744     private void setArrayPlayer() {
00745         System.out.println("Initialisation du jeu");
00746     }
00747
00748     private void setGrid() {
00749         System.out.println("Initialisation du jeu");
00750     }
00751
00752     private void setPlayer() {
00753         System.out.println("Initialisation des joueurs");
00754     }
00755
00756     private void setIteration() {
00757         System.out.println("Initialisation de l'iteration");
00758     }
00759
00760     private void setEnd() {
00761         System.out.println("Initialisation de l'etat du jeu");
00762     }
00763
00764     private void setGameplay() {
00765         System.out.println("Initialisation du jeu");
00766     }
00767
00768     private void setLastMove() {
00769         System.out.println("Initialisation du jeu");
00770     }
00771
00772     private void setCurrentPlayerId() {
00773         System.out.println("Initialisation du jeu");
00774     }
00775
00776     private void setArrayPlayer() {
00777         System.out.println("Initialisation du jeu");
00778     }
00779
00780     private void setGrid() {
00781         System.out.println("Initialisation du jeu");
00782     }
00783
00784     private void setPlayer() {
00785         System.out.println("Initialisation des joueurs");
00786     }
00787
00788     private void setIteration() {
00789         System.out.println("Initialisation de l'iteration");
00790     }
00791
00792     private void setEnd() {
00793         System.out.println("Initialisation de l'etat du jeu");
00794     }
00795
00796     private void setGameplay() {
00797         System.out.println("Initialisation du jeu");
00798     }
00799
00800     private void setLastMove() {
00801         System.out.println("Initialisation du jeu");
00802     }
00803
00804     private void setCurrentPlayerId() {
00805         System.out.println("Initialisation du jeu");
00806     }
00807
00808     private void setArrayPlayer() {
00809         System.out.println("Initialisation du jeu");
00810     }
00811
00812     private void setGrid() {
00813         System.out.println("Initialisation du jeu");
00814     }
00815
00816     private void setPlayer() {
00817         System.out.println("Initialisation des joueurs");
00818     }
00819
00820     private void setIteration() {
00821         System.out.println("Initialisation de l'iteration");
00822     }
00823
00824     private void setEnd() {
00825         System.out.println("Initialisation de l'etat du jeu");
00826     }
00827
00828     private void setGameplay() {
00829         System.out.println("Initialisation du jeu");
00830     }
00831
00832     private void setLastMove() {
00833         System.out.println("Initialisation du jeu");
00834     }
00835
00836     private void setCurrentPlayerId() {
00837         System.out.println("Initialisation du jeu");
00838     }
00839
00840     private void setArrayPlayer() {
00841         System.out.println("Initialisation du jeu");
00842     }
00843
00844     private void setGrid() {
00845         System.out.println("Initialisation du jeu");
00846     }
00847
00848     private void setPlayer() {
00849         System.out.println("Initialisation des joueurs");
00850     }
00851
00852     private void setIteration() {
00853         System.out.println("Initialisation de l'iteration");
00854     }
00855
00856     private void setEnd() {
00857         System.out.println("Initialisation de l'etat du jeu");
00858     }
00859
00860     private void setGameplay() {
00861         System.out.println("Initialisation du jeu");
00862     }
00863
00864     private void setLastMove() {
00865         System.out.println("Initialisation du jeu");
00866     }
00867
00868     private void setCurrentPlayerId() {
00869         System.out.println("Initialisation du jeu");
00870     }
00871
00872     private void setArrayPlayer() {
00873         System.out.println("Initialisation du jeu");
00874     }
00875
00876     private void setGrid() {
00877         System.out.println("Initialisation du jeu");
00878     }
00879
00880     private void setPlayer() {
00881         System.out.println("Initialisation des joueurs");
00882     }
00883
00884     private void setIteration() {
00885         System.out.println("Initialisation de l'iteration");
00886     }
00887
00888     private void setEnd() {
00889         System.out.println("Initialisation de l'etat du jeu");
00890     }
00891
00892     private void setGameplay() {
00893         System.out.println("Initialisation du jeu");
00894     }
00895
00896     private void setLastMove() {
00897         System.out.println("Initialisation du jeu");
00898     }
00899
00900     private void setCurrentPlayerId() {
00901         System.out.println("Initialisation du jeu");
00902     }
00903
00904     private void setArrayPlayer() {
00905         System.out.println("Initialisation du jeu");
00906     }
00907
00908     private void setGrid() {
00909         System.out.println("Initialisation du jeu");
00910     }
00911
00912     private void setPlayer() {
00913         System.out.println("Initialisation des joueurs");
00914     }
00915
00916     private void setIteration() {
00917         System.out.println("Initialisation de l'iteration");
00918     }
00919
00920     private void setEnd() {
00921         System.out.println("Initialisation de l'etat du jeu");
00922     }
00923
00924     private void setGameplay() {
00925         System.out.println("Initialisation du jeu");
00926     }
00927
00928     private void setLastMove() {
00929         System.out.println("Initialisation du jeu");
00930     }
00931
00932     private void setCurrentPlayerId() {
00933         System.out.println("Initialisation du jeu");
00934     }
00935
00936     private void setArrayPlayer() {
00937         System.out.println("Initialisation du jeu");
00938     }
00939
00940     private void setGrid() {
00941         System.out.println("Initialisation du jeu");
00942     }
00943
00944     private void setPlayer() {
00945         System.out.println("Initialisation des joueurs");
00946     }
00947
00948     private void setIteration() {
00949         System.out.println("Initialisation de l'iteration");
00950     }
00951
00952     private void setEnd() {
00953         System.out.println("Initialisation de l'etat du jeu");
00954     }
00955
00956     private void setGameplay() {
00957         System.out.println("Initialisation du jeu");
00958     }
00959
00960     private void setLastMove() {
00961         System.out.println("Initialisation du jeu");
00962     }
00963
00964     private void setCurrentPlayerId() {
00965         System.out.println("Initialisation du jeu");
00966     }
00967
00968     private void setArrayPlayer() {
00969         System.out.println("Initialisation du jeu");
00970     }
00971
00972     private void setGrid() {
00973         System.out.println("Initialisation du jeu");
00974     }
00975
00976     private void setPlayer() {
00977         System.out.println("Initialisation des joueurs");
00978     }
00979
00980     private void setIteration() {
00981         System.out.println("Initialisation de l'iteration");
00982     }
00983
00984     private void setEnd() {
00985         System.out.println("Initialisation de l'etat du jeu");
00986     }
00987
00988     private void setGameplay() {
00989         System.out.println("Initialisation du jeu");
00990     }
00991
00992     private void setLastMove() {
00993         System.out.println("Initialisation du jeu");
00994     }
00995
00996     private void setCurrentPlayerId() {
00997         System.out.println("Initialisation du jeu");
00998     }
00999
01000     private void setArrayPlayer() {
01001         System.out.println("Initialisation du jeu");
01002     }
01003
01004     private void setGrid() {
01005         System.out.println("Initialisation du jeu");
01006     }
01007
01008     private void setPlayer() {
01009         System.out.println("Initialisation des joueurs");
01010     }
01011
01012     private void setIteration() {
01013         System.out.println("Initialisation de l'iteration");
01014     }
01015
01016     private void setEnd() {
01017         System.out.println("Initialisation de l'etat du jeu");
01018     }
01019
01020     private void setGameplay() {
01021         System.out.println("Initialisation du jeu");
01022     }
01023
01024     private void setLastMove() {
01025         System.out.println("Initialisation du jeu");
01026     }
01027
01028     private void setCurrentPlayerId() {
01029         System.out.println("Initialisation du jeu");
01030     }
01031
01032     private void setArrayPlayer() {
01033         System.out.println("Initialisation du jeu");
01034     }
01035
01036     private void setGrid() {
01037         System.out.println("Initialisation du jeu");
01038     }
01039
01040     private void setPlayer() {
01041         System.out.println("Initialisation des joueurs");
01042     }
01043
01044     private void setIteration() {
01045         System.out.println("Initialisation de l'iteration");
01046     }
01047
01048     private void setEnd() {
01049         System.out.println("Initialisation de l'etat du jeu");
01050     }
01051
01052     private void setGameplay() {
01053         System.out.println("Initialisation du jeu");
01054     }
01055
01056     private void setLastMove() {
01057         System.out.println("Initialisation du jeu");
01058     }
01059
01060     private void setCurrentPlayerId() {
01061         System.out.println("Initialisation du jeu");
01062     }
01063
01064     private void setArrayPlayer() {
01065         System.out.println("Initialisation du jeu");
01066     }
01067
01068     private void setGrid() {
01069         System.out.println("Initialisation du jeu");
01070     }
01071
01072     private void setPlayer() {
01073         System.out.println("Initialisation des joueurs");
01074     }
01075
01076     private void setIteration() {
01077         System.out.println("Initialisation de l'iteration");
01078     }
01079
01080     private void setEnd() {
01081         System.out.println("Initialisation de l'etat du jeu");
01082     }
01083
01084     private void setGameplay() {
01085         System.out.println("Initialisation du jeu");
01086     }
01087
01088     private void setLastMove() {
01089         System.out.println("Initialisation du jeu");
01090     }
01091
01092     private void setCurrentPlayerId() {
01093         System.out.println("Initialisation du jeu");
01094     }
01095
01096     private void setArrayPlayer() {
01097         System.out.println("Initialisation du jeu");
01098     }
01099
01100     private void setGrid() {
01101         System.out.println("Initialisation du jeu");
01102     }
01103
01104     private void setPlayer() {
```

```

00095
00096 private void askForPlayers() {
00100     this.initLocalPlayer(0); // premier joueur toujours humain
00101     if(this.gameplay.equals("LOCAL")) initLocalPlayer(1);
00102     else createPlayer("Ordinateur", 1, Entity.IA);
00103 }
00104
00105 private void loadSave(String saveText) {
00111     String[] saveSplit = saveText.split("&");
00112     if (saveSplit.length < this.numberOfPlayers+2) {
00113         this.save.delete();
00114         throw new IllegalArgumentException("Sauvegarde corrompue");
00115     }
00116
00117     // Initialise the players //
00118     for (int i=0; i<this.numberOfPlayers; i++) {
00119         Entity og = Entity.of(saveSplit[i+1].split(" ")[0]);
00120         String playerName = saveSplit[i+1].split(" ")[1];
00121         this.createPlayer(playerName, i, og);
00122     }
00123
00124     // Initialise the current player //
00125     this.currentPlayerId = Integer.parseInt(saveSplit[this.numberOfPlayers+1]);
00126
00127     // Initialise the grid //
00128     Token[] tokenArray = new Token[Color.values().length];
00129     for (int i=0; i<this.numberOfPlayers; i++) {
00130         tokenArray[i] = this.getPlayerFromId(i).getToken();
00131     }
00132     tokenArray[Color.values().length-1] = Token.emptyToken;
00133     this.grid.loadGrid(saveSplit[0], tokenArray);
00134 }
00135
00136 private void initLocalPlayer(int position) {
00142     System.out.println("Veuillez saisir le pseudo du joueur " + position + " :");
00143     String username = (new Scanner(System.in)).nextLine();
00144     this.createPlayer(username, position, Entity.LOCAL);
00145 }
00146
00147 private void createPlayer(String name, int id, Entity og) {
00155     this.arrayPlayer[id] = new Player(name, id, og);
00156 }
00157
00158 private void welcomeMessage() {
00162     System.out.println("Bienvenu au jeu du puissance 4.\n");
00163 }
00164
00165 private void endMessage() {
00169     // Print game ended
00170     System.out.println("*****");
00171     System.out.println("+" + Color.ansiColorOf("RED") +
00172         "          Game Ended          "
00173         + Color.ansiColorOf("WHITE") + "+");
00174     System.out.println("*****");
00175
00176     // Print the winner or if pat
00177     if (this.isTie()) {
00178         this.save.delete();
00179         System.out.println("Egalité");
00180     }
00181     else if (this.hasWin()) {
00182         this.save.delete();
00183         System.out.println("Le gagnant est : " +
00184             this.getPlayerFromId(this.currentPlayerId));
00185     }
00186 }
00187
00188 private void printPartyChoice() {
00192     System.out.println("Vous avez 2 options de jeu :");
00193     System.out.println("-[1] 1 VS 1 avec 2 joueurs en local");
00194     System.out.println("-[2] 1 VS 1 contre une IA\n");
00195 }
00196
00197 private int valideGameOption(int option) {
00205     if (option <= 0 || option > 2) throw new IllegalArgumentException();
00206     return option;
00207 }
00208
00209 private void setOptionGame() {
00213     while (true) {
00214         try {
00215             printPartyChoice();
00216             System.out.println("A quel mode de jeu voulez-vous jouer ? ");
00217             int option = valideGameOption((new Scanner(System.in)).nextInt());
00218             if (option == 1) this.gameplay = "LOCAL";
00219             else this.gameplay = "IA";
00220             break;

```

```

00221         }
00222         catch (IllegalArgumentException e) {
00223             System.out.println("Ce mode de jeu n'existe pas.\n");
00224         }
00225         catch (InputMismatchException e) {
00226             System.out.println("Erreur lors de la saisie," +
00227                 "veuillez saisir l'entier '1' ou '2'\n");
00228         }
00229     }
00230 }
00231
00232 public static void run() {
00233     Game game = new Game();
00234     game.play();
00235 }
00236
00237 private Player getPlayerFromId(int id) {
00238     if (id < 0 || id > this.numberOfPlayers) {
00239         throw new IllegalArgumentException("Bad player id");
00240     }
00241     return this.arrayPlayer[id];
00242 }
00243
00244 private Player nextPlayer(Player current) {
00245     this.currentPlayerId = (this.currentPlayerId+1)%this.numberOfPlayers;
00246     return this.getPlayerFromId(this.currentPlayerId);
00247 }
00248
00249 private void printCurrentPlayer() {
00250     System.out.println(
00251         Color ansiColorOf(this.getPlayerFromId(this.currentPlayerId).getColor())
00252         + "["
00253         + this.getPlayerFromId(this.currentPlayerId) + "]"
00254         + Color ansiColorOf("WHITE"));
00255 }
00256
00257 private void play() {
00258     boolean playerHasPlay;
00259     Player currentPlayer = this.getPlayerFromId(this.currentPlayerId);
00260
00261     while (!this.end) {
00262
00263         this.iteration += 1;
00264         this.grid.print();
00265         playerHasPlay = false;
00266
00267         while (!playerHasPlay) {
00268             this.printCurrentPlayer();
00269             System.out.println("[m: Menu]");
00270             int columnChosen = chooseAColumn();
00271             if (this.end) break;
00272             playerHasPlay = this.playAToken(currentPlayer.getToken(),
00273                 columnChosen);
00274         }
00275
00276         this.isEnd();
00277
00278         if (!this.end) {
00279             currentPlayer = nextPlayer(currentPlayer);
00280         }
00281     }
00282     this.grid.print();
00283     this.endMessage();
00284 }
00285
00286 private int chooseAColumn() {
00287     if (this.getPlayerFromId(this.numberOfPlayers-1).whatIs() == Entity.IA) {
00288         if (this.currentPlayerId == this.numberOfPlayers-1) {
00289             return (new Random()).nextInt(7);
00290         }
00291     }
00292     return this.askForInput();
00293 }
00294
00295 private void menu() {
00296     boolean done = false;
00297     while (!done) {
00298         System.out.println("[1:Continuer][2:Sauvegarder][3:Quitter le jeu]");
00299         System.out.print("Choisissez une option : ");
00300         Scanner input = new Scanner(System.in);
00301         try {
00302             switch (input.nextInt()) {
00303                 case 1:
00304                     done = true;
00305                     break;
00306                 case 2:
00307                     this.save.write(this);

```

```

00338         break;
00339     case 3:
00340         done = true;
00341         this.end = true;
00342         break;
00343     default:
00344         done = true;
00345         break;
00346     }
00347 }
00348 catch (InputMismatchException e) {
00349     System.out.println("Ce choix n'existe pas.");
00350 }
00351 }
00352 }
00353
00354 private int askForInput() {
00355     int column = 0;
00356     while (true) {
00357         System.out.print("Choisissez une colonne : ");
00358         Scanner input = new Scanner(System.in);
00359         String sInput = input.nextLine();
00360         if (sInput.equals("m")) {
00361             this.menu();
00362             if (this.end) break;
00363         }
00364         else {
00365             try {
00366                 column = valideColumn(Integer.parseInt(sInput)-1);
00367                 break;
00368             }
00369             catch (IllegalArgumentException e) {
00370                 System.out.println("Colonne invalide, un entier en 1 et " +
00371                                     this.grid.getWidth());
00372             }
00373         }
00374     }
00375     return column;
00376 }
00377
00378 private int valideColumn(int column) {
00379     return this.grid.valideColumn(column);
00380 }
00381
00382 private boolean playAToken(Token token, int column) {
00383     if (token == Token.emptyToken) {
00384         throw new IllegalArgumentException("You can't play an empty Token");
00385     }
00386     Cell played = this.grid.getNextEmptyCellAt(column);
00387     if (played.getToken() != Token.emptyToken) {
00388         System.out.println("La colonne est pleine");
00389         return false;
00390     }
00391     played.setToken(requireNonNull(token));
00392     if (played.getNeighbor(Direction.UP) != Cell.outOfBoundCell) {
00393         this.grid.UpToNextEmptyCellAt(column);
00394     }
00395     this.lastMove = column;
00396     return true;
00397 }
00398
00399 private boolean isTie() {
00400     return this.iteration == this.grid.getSize();
00401 }
00402
00403 private boolean hasWin() {
00404     Cell lastCellPlayed = this.grid.getNextEmptyCellAt(this.lastMove);
00405     if (lastCellPlayed.getToken() != Token.emptyToken);
00406     else lastCellPlayed = lastCellPlayed.getNeighbor(Direction.DOWN);
00407     return lastCellPlayed.check();
00408 }
00409
00410 private void isEnd() {
00411     if (this.isTie() || this.hasWin()) this.end = true;
00412 }
00413
00414 @Override
00415 public String toString() {
00416     return this.grid.toString() + "&"
00417         + this.getPlayerFromId(0).toString() + "&"
00418         + this.getPlayerFromId(1).toString() + "&"
00419         + String.valueOf(this.currentPlayerId);
00420 }
00421 }

```


5.11 /home/hozen/cur/projet-java/src/Grid.java File Reference

Classes

- class [Grid](#)

Logical representation of a board of the [Game](#) of Puissance 4.

5.12 Grid.java

[Go to the documentation of this file.](#)

```
00001 import static java.util.Objects.requireNonNull;
00002 import java.util.EnumMap;
00003
00010 public class Grid {
00011
00012     private static final int WIDTH = 7;
00013     private static final int HEIGHT = 6;
00014
00015     private Cell[] arrayNextEmptyCell; /* contient la référence vers la prochaine cellule
00016                                         vide de chaque colonne */
00017     /* Constructeurs */
00018
00019     public Grid() {
00023         this.arrayNextEmptyCell = new Cell[this.getWidth()];
00024         this.initGrid();
00025     }
00026
00027     public int getWidth() {return this.WIDTH;}
00028
00029     public int getHeight() {return this.HEIGHT;}
00030
00031     public int getSize() {return this.getWidth()*this.getHeight();}
00032
00033     public void initGrid() {
00037         Cell[][] tempGrid2D = new Cell[this.getHeight()][this.getWidth()];
00038         for(int i = 0; i < this.getHeight(); i++) {
00039             for(int j = 0; j < this.getWidth(); ++j) {
00040                 tempGrid2D[i][j] = new Cell(Token.emptyToken);
00041             }
00042         }
00043         for(int i = 0; i < this.getHeight(); i++) {
00044             for(int j = 0; j < this.getWidth(); ++j) {
00045                 if (i > 0) {
00046                     tempGrid2D[i][j].setNeighbor(tempGrid2D[i-1][j], Direction.UP);
00047                 }
00048                 if (i < this.getHeight()-1) {
00049                     tempGrid2D[i][j].setNeighbor(tempGrid2D[i+1][j], Direction.DOWN);
00050                 }
00051                 if (j > 0) {
00052                     tempGrid2D[i][j].setNeighbor(tempGrid2D[i][j-1], Direction.LEFT);
00053                 }
00054                 if (j < this.getWidth()-1) {
00055                     tempGrid2D[i][j].setNeighbor(tempGrid2D[i][j+1], Direction.RIGHT);
00056                 }
00057             }
00058         }
00059         for(int j = 0; j<this.getWidth() ; ++j) {
00060             this.arrayNextEmptyCell[j] = tempGrid2D[this.getHeight()-1][j] ;
00061         }
00062     }
00063
00064     public int valideColumn(int column) {
00072         if (column < 0 || column >= this.getWidth()) {
00073             throw new IllegalArgumentException("column outOfBound");
00074         }
00075         return column;
00076     }
00077
00078     public Cell getNextEmptyCellAt(int column) {
00086         return this.arrayNextEmptyCell[valideColumn(column)];
00087     }
00088
00089     public void UpToNextEmptyCellAt(int column) {
00095         this.arrayNextEmptyCell[valideColumn(column)] =
00096             this.getNextEmptyCellAt(column).getNeighbor(Direction.UP);
00097     }
00098 }
```

```

00099     public Cell getTopCellAt(int column) {
00106         // Parcours jusqu'à la dernière cellule haute de la colonne i
00107         Cell top = this.getNextEmptyCellAt(column);
00108         while (top.getNeighbor(Direction.UP) != Cell.outOfBoundCell) {
00109             top = top.getNeighbor(Direction.UP);
00110         }
00111         return top;
00112     }
00113
00114     private void printGrid() {
00118         char[][] tempArray = new char[this.getHeight()][this.getWidth()];
00119         for(int i=0; i < this.getWidth(); ++i) {
00120             Cell cellTemp = this.arrayNextEmptyCell[i];
00121             cellTemp = this.getTopCellAt(i);
00122             int j = 0;
00123             // Parcours jusqu'à la dernière cellule basse de la colonne i
00124             while (cellTemp != Cell.outOfBoundCell) {
00125                 if (cellTemp.getToken().getColor() == Color.RED) {
00126                     tempArray[j][i] = 'r';
00127                 }
00128                 else if (cellTemp.getToken().getColor() == Color.YELLOW) {
00129                     tempArray[j][i] = 'y';
00130                 }
00131                 else if (cellTemp.getToken().getColor() == Color.EMPTY) {
00132                     tempArray[j][i] = ' ';
00133                 }
00134                 ++j;
00135                 cellTemp = cellTemp.getNeighbor(Direction.DOWN);
00136             }
00137         }
00138
00139         System.out.print(Color ansiColorOf("BLUE"));
00140
00141         for(int i=0; i<this.getHeight(); ++i) {
00142             for(int j=0; j<this.getWidth(); ++j) {
00143                 System.out.print("----");
00144             }
00145             System.out.println("+");
00146             System.out.print("|");
00147             for(int j=0; j<this.getWidth(); ++j) {
00148                 if(tempArray[i][j] == 'r') {
00149                     System.out.print(Color ansiColorOf("RED") +
00150                         " O " +
00151                         Color ansiColorOf("BLUE"));
00152                 }
00153                 else if (tempArray[i][j] == 'y') {
00154                     System.out.print(Color ansiColorOf("YELLOW") +
00155                         " O " +
00156                         Color ansiColorOf("BLUE"));
00157                 }
00158                 else {
00159                     System.out.print("   ");
00160                 }
00161             }
00162             System.out.print("|");
00163             System.out.println();
00164         }
00165         for(int j=0; j<this.getWidth(); ++j) {
00166             System.out.print("----");
00167         }
00168         System.out.println("+ " + Color ansiColorOf("WHITE"));
00169     }
00170
00171     private void printAvailableColumn() {
00176         System.out.print(Color ansiColorOf("BLUE"));
00177         for(int j=0; j<this.getWidth(); ++j) {
00178             System.out.print("+--^--");
00179         }
00180         System.out.println("+");
00181         for(int j=0; j<this.getWidth(); ++j) {
00182             if (this.getNextEmptyCellAt(j).getColor() != Color.EMPTY) {
00183                 System.out.print("| X ");
00184             }
00185             else {
00186                 System.out.print("| "
00187                     + Color ansiColorOf("GREEN")
00188                     + (j+1) + Color ansiColorOf("BLUE")
00189                     + " ");
00190             }
00191         }
00192         System.out.println("|");
00193         for(int j=0; j<this.getWidth(); ++j) {
00194             System.out.print("----");
00195         }
00196         System.out.println("+ " + Color ansiColorOf("WHITE"));
00197     }
00198

```

```

00199     public void print() {
00203         System.out.println();
00204         this.printGrid();
00205         System.out.println();
00206         this.printAvailableColumn();
00207         System.out.println();
00208     }
00209 }
00210
00211 @Override
00212 public String toString() {
00219     StringBuilder s = new StringBuilder();
00220     Cell c;
00221     for (int i=0; i<this.getWidth(); i++) {
00222         c = this.getTopCellAt(i);
00223         for(int j=0; j<this.getHeight(); j++) {
00224             s.append(c.toString()+"");
00225             c = c.getNeighbor(Direction.DOWN);
00226         }
00227     }
00228     return s.toString();
00229 }
00230
00231 public void loadGrid(String schema, Token[] tokenOfPlayers) {
00238     this.initGrid();
00239     Cell current;
00240     String[] cells = schema.split(";");
00241     for (int i=0; i<this.getWidth(); i++) {
00242         current = this.getTopCellAt(i);
00243         for (int j=0; j<this.getHeight(); j++) {
00244             for (int k=0; k<Color.values().length; k++) {
00245                 if (tokenOfPlayers[k].getColor() ==
00246                     Color.valueOf(cells[j+i*this.getHeight()]))
00247                 {
00248                     current.setToken(tokenOfPlayers[k]);
00249                     if (tokenOfPlayers[k] != Token.emptyToken)
00250                         this.UpToNextEmptyCellAt(i);
00251                     break;
00252                 }
00253             }
00254             current = current.getNeighbor(Direction.DOWN);
00255         }
00256     }
00257 }
00258 }

```

5.13 /home/hozen/cur/projet-java/src/Main.java File Reference

Classes

- class [Main](#)
Main class, program entry point.

5.14 Main.java

[Go to the documentation of this file.](#)

```

00001
00007 public class Main {
00008     public static void main(String[] args) {
00009         Game.run();
00010     }
00011 }

```

5.15 /home/hozen/cur/projet-java/src/Player.java File Reference

Classes

- class [Player](#)
Represent a [Player](#) of a [Game](#) using [Token](#).

5.16 Player.java

[Go to the documentation of this file.](#)

```

00001
00007 public class Player {
00008
00009     private final String pseudo;
00010     private final Token token;
00011     private final Entity gameplay;
00012
00013     public Player(String pseudo, int indexPlayer, Entity e) {
00021         this.token = initToken(validePlayer(indexPlayer));
00022         this.pseudo = pseudo;
00023         this.gameplay = e;
00024     }
00025
00026     private Token initToken(int id) {
00033         switch (id) {
00034             case 0: return new Token(Color.RED);
00035             case 1: return new Token(Color.YELLOW);
00036         }
00037         return Token.emptyToken;
00038     }
00039
00040     private int validePlayer(int id) {
00048         if (id < 0 || id >= Game.numberOfPlayers) {
00049             throw new IllegalArgumentException("indexPlayer must equal 0 if first player else 1.");
00050         }
00051         return id;
00052     }
00053
00054     public Entity whatIs() {
00060         return this.gameplay;
00061     }
00062
00063     public Color getColor() {
00069         return this.token.getColor();
00070     }
00071
00072     public Token getToken() {
00078         return this.token;
00079     }
00080
00081     public String getUsername() {
00087         return this.pseudo;
00088     }
00089
00090     @Override
00091     public String toString(){
00097         return String.format("%s %s (%s)",
00098             this.gameplay.toString(),
00099             this.pseudo,
00100             this.getColor());
00101     }
00102 }

```

5.17 /home/hozen/cur/projet-java/src/Save.java File Reference

Classes

- class [Save](#)

Represent a global [Save](#) class.

5.18 Save.java

[Go to the documentation of this file.](#)

```

00001 import java.io.File; // Import the File class
00002 import java.io.FileNotFoundException; // Import this class to handle errors
00003 import java.util.Scanner; // Import the Scanner class to read text files
00004 import java.io.FileWriter; // Import the FileWriter class
00005 import java.io.IOException; // Import the IOException class to handle errors

```

```

00006
00013 public class Save {
00014
00015     private String filename;
00016
00017     public Save(String filename) {
00023         this.filename = filename;
00024     }
00025
00026     public void write(boolean verbose, Object src) {
00033         this.write(src);
00034         if (verbose) System.out.println("Successfully wrote to the file.");
00035     }
00036
00037     public void write(Object src) {
00043         try {
00044             FileWriter myWriter = new FileWriter(this.filename);
00045             myWriter.write(src.toString());
00046             myWriter.close();
00047         } catch (IOException e) {
00048             System.out.println("An error occurred.");
00049             e.printStackTrace();
00050         }
00051     }
00052
00053     public String read(boolean verbose) {
00060         String result = this.read();
00061         System.out.println("Succesfully read the file.");
00062         return result;
00063     }
00064
00065     public String read() {
00071         StringBuilder s = new StringBuilder();
00072         try {
00073
00074             File myObj = new File(this.filename);
00075             Scanner myReader = new Scanner(myObj);
00076             while (myReader.hasNextLine()) {
00077                 String data = myReader.nextLine();
00078                 // System.out.println(data);
00079                 s.append(data);
00080             }
00081             myReader.close();
00082         } catch (FileNotFoundException e) {
00083             System.out.println("An error occurred.");
00084             e.printStackTrace();
00085         }
00086         return s.toString();
00087     }
00088
00089     public void delete(boolean verbose) {
00095         this.delete();
00096         if (verbose) System.out.println("Successfully wrote to the file.");
00097     }
00098
00099     public void delete() {
00103         try {
00104             new FileWriter(filename, false).close();
00105         } catch (IOException e) {
00106             System.out.println("An error occurred.");
00107             e.printStackTrace();
00108         }
00109     }
00110
00111     public boolean isEmpty() {
00117         return this.read().equals("");
00118     }
00119 }

```

5.19 /home/hozen/cur/projet-java/src/Token.java File Reference

Classes

- class [Token](#)

Represent a [Token](#) of Puissance 4.

5.20 Token.java

[Go to the documentation of this file.](#)

```
00001
00007 public class Token {
00008
00009     public static final Token emptyToken = new Token(Color.EMPTY);
00010
00011     private final Color colorToken;
00012
00013     public Token(Color c) {
00019         this.colorToken = c;
00020     }
00021
00022     public Color getColor() {
00028         return this.colorToken;
00029     }
00030
00031     @Override
00032     public String toString() {
00038         return this.colorToken.toString();
00039     }
00040 }
```

Index

- /home/hozen/cur/projet-java/src/Cell.java, [81](#)
- /home/hozen/cur/projet-java/src/Color.java, [82](#), [83](#)
- /home/hozen/cur/projet-java/src/Direction.java, [83](#)
- /home/hozen/cur/projet-java/src/Entity.java, [84](#)
- /home/hozen/cur/projet-java/src/Game.java, [84](#), [85](#)
- /home/hozen/cur/projet-java/src/Grid.java, [89](#)
- /home/hozen/cur/projet-java/src/Main.java, [91](#)
- /home/hozen/cur/projet-java/src/Player.java, [91](#), [92](#)
- /home/hozen/cur/projet-java/src/Save.java, [92](#)
- /home/hozen/cur/projet-java/src/Token.java, [93](#), [94](#)

- ansiColorOf
 - Color, [20](#)
- arrayNextEmptyCell
 - Grid, [61](#)
- arrayPlayer
 - Game, [47](#)
- askForInput
 - Game, [30](#)
- askForLoadSave
 - Game, [31](#)
- askForPlayers
 - Game, [32](#)

- Cell, [7](#)
 - Cell, [9](#)
 - check, [10](#)
 - compareTo, [10](#)
 - equals, [11](#)
 - getColor, [12](#)
 - getNeighbor, [12](#)
 - getToken, [13](#)
 - isEmpty, [13](#)
 - neighborhood, [18](#)
 - numberOfSameNeighbor, [14](#), [15](#)
 - outOfBoundCell, [18](#)
 - setNeighbor, [16](#)
 - setToken, [16](#)
 - token, [18](#)
 - toString, [17](#)
- check
 - Cell, [10](#)
- chooseAColumn
 - Game, [32](#)
- Color, [19](#)
 - ansiColorOf, [20](#)
 - colorOf, [21](#)
 - EMPTY, [21](#)
 - RED, [21](#)
 - YELLOW, [21](#)

- colorOf
 - Color, [21](#)
- colorToken
 - Token, [80](#)
- Comparable, [22](#)
- compareTo
 - Cell, [10](#)
- createPlayer
 - Game, [33](#)
- currentPlayerId
 - Game, [47](#)

- delete
 - Save, [72](#)
- Direction, [23](#)
 - DOWN, [25](#)
 - getDiagonales, [24](#)
 - getOpposite, [24](#)
 - LEFT, [25](#)
 - RIGHT, [25](#)
 - UP, [25](#)
- DOWN
 - Direction, [25](#)

- EMPTY
 - Color, [21](#)
- emptyToken
 - Token, [80](#)
- end
 - Game, [47](#)
- endMessage
 - Game, [34](#)
- Entity, [26](#)
 - IA, [27](#)
 - LOCAL, [27](#)
 - of, [27](#)
- equals
 - Cell, [11](#)

- filename
 - Save, [77](#)

- Game, [28](#)
 - arrayPlayer, [47](#)
 - askForInput, [30](#)
 - askForLoadSave, [31](#)
 - askForPlayers, [32](#)
 - chooseAColumn, [32](#)
 - createPlayer, [33](#)
 - currentPlayerId, [47](#)

- end, 47
- endMessage, 34
- Game, 30
- gameplay, 48
- getPlayerFromId, 34
- grid, 48
- hasWin, 35
- initLocalPlayer, 36
- isEnd, 36
- isTie, 37
- iteration, 48
- lastMove, 48
- loadSave, 38
- menu, 39
- nextPlayer, 39
- numberOfPlayers, 48
- numberOfTokenToWin, 48
- play, 40
- playAToken, 41
- printCurrentPlayer, 42
- printPartyChoice, 43
- run, 43
- save, 49
- setOptionGame, 44
- toString, 45
- valideColumn, 45
- valideGameOption, 46
- welcomeMessage, 47
- gameplay
 - Game, 48
 - Player, 69
- getColor
 - Cell, 12
 - Player, 66
 - Token, 79
- getDiagonales
 - Direction, 24
- getHeight
 - Grid, 52
- getNeighbor
 - Cell, 12
- getNextEmptyCellAt
 - Grid, 52
- getOpposite
 - Direction, 24
- getPlayerFromId
 - Game, 34
- getSize
 - Grid, 53
- getToken
 - Cell, 13
 - Player, 66
- getTopCellAt
 - Grid, 54
- getUsername
 - Player, 67
- getWidth
 - Grid, 54
- Grid, 49
 - arrayNextEmptyCell, 61
 - getHeight, 52
 - getNextEmptyCellAt, 52
 - getSize, 53
 - getTopCellAt, 54
 - getWidth, 54
 - Grid, 51
 - HEIGHT, 61
 - initGrid, 55
 - loadGrid, 55
 - print, 56
 - printAvailableColumn, 57
 - printGrid, 58
 - toString, 58
 - UpToNextEmptyCellAt, 59
 - valideColumn, 60
 - WIDTH, 61
- grid
 - Game, 48
- hasWin
 - Game, 35
- HEIGHT
 - Grid, 61
- IA
 - Entity, 27
- initGrid
 - Grid, 55
- initLocalPlayer
 - Game, 36
- initToken
 - Player, 67
- isEmpty
 - Cell, 13
 - Save, 73
- isEnd
 - Game, 36
- isTie
 - Game, 37
- iteration
 - Game, 48
- lastMove
 - Game, 48
- LEFT
 - Direction, 25
- loadGrid
 - Grid, 55
- loadSave
 - Game, 38
- LOCAL
 - Entity, 27
- Main, 62
 - main, 62
- main
 - Main, 62

menu
 Game, 39

neighborhood
 Cell, 18

nextPlayer
 Game, 39

numberOfPlayers
 Game, 48

numberOfSameNeighbor
 Cell, 14, 15

numberOfTokenToWin
 Game, 48

of
 Entity, 27

outOfBoundCell
 Cell, 18

play
 Game, 40

playAToken
 Game, 41

Player, 63
 gameplay, 69
 getColor, 66
 getToken, 66
 getUsername, 67
 initToken, 67
 Player, 65
 pseudo, 69
 token, 70
 toString, 68
 validePlayer, 68
 whatIs, 69

print
 Grid, 56

printAvailableColumn
 Grid, 57

printCurrentPlayer
 Game, 42

printGrid
 Grid, 58

printPartyChoice
 Game, 43

pseudo
 Player, 69

read
 Save, 73, 74

RED
 Color, 21

RIGHT
 Direction, 25

run
 Game, 43

Save, 70
 delete, 72

filename, 77

isEmpty, 73

read, 73, 74

Save, 71

write, 75

save
 Game, 49

setNeighbor
 Cell, 16

setOptionGame
 Game, 44

setToken
 Cell, 16

Token, 77
 colorToken, 80
 emptyToken, 80
 getColor, 79
 Token, 78
 toString, 79

token
 Cell, 18
 Player, 70

toString
 Cell, 17
 Game, 45
 Grid, 58
 Player, 68
 Token, 79

UP
 Direction, 25

UpToNextEmptyCellAt
 Grid, 59

valideColumn
 Game, 45
 Grid, 60

valideGameOption
 Game, 46

validePlayer
 Player, 68

welcomeMessage
 Game, 47

whatIs
 Player, 69

WIDTH
 Grid, 61

write
 Save, 75

YELLOW
 Color, 21