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

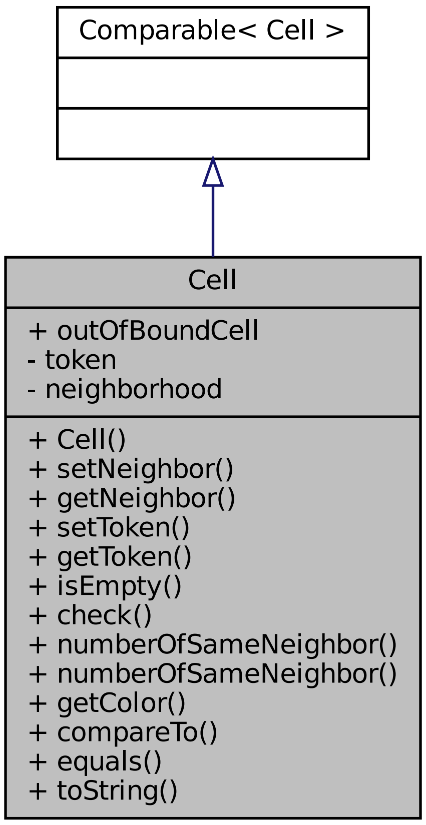
Chapter 4

Class Documentation

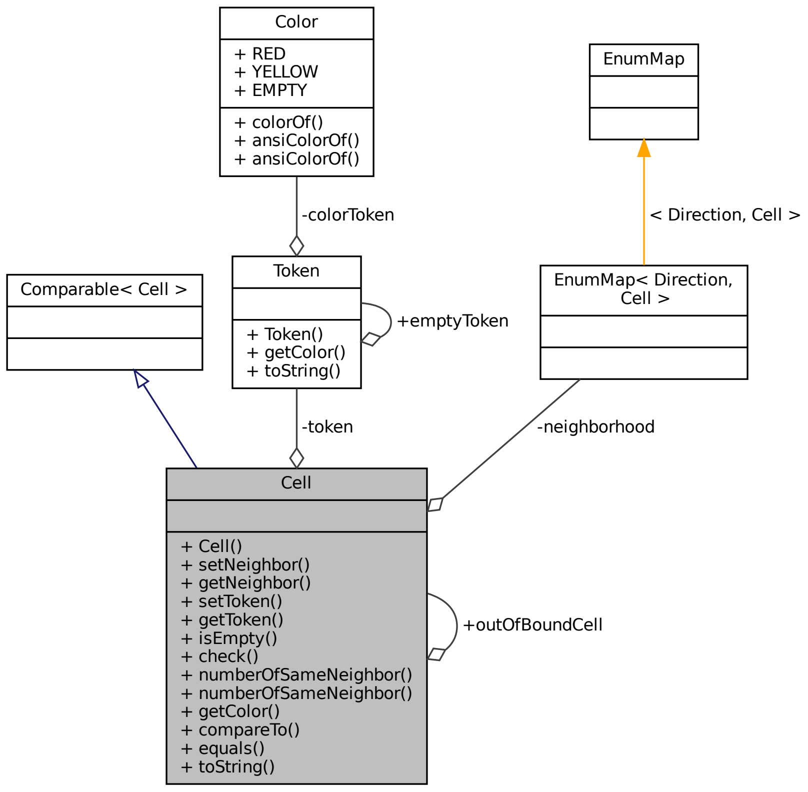
4.1 Cell Class Reference

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 )
```

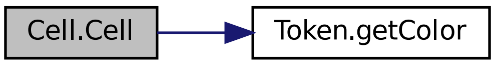
Cell constructor

Parameters

token	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 ( )
```

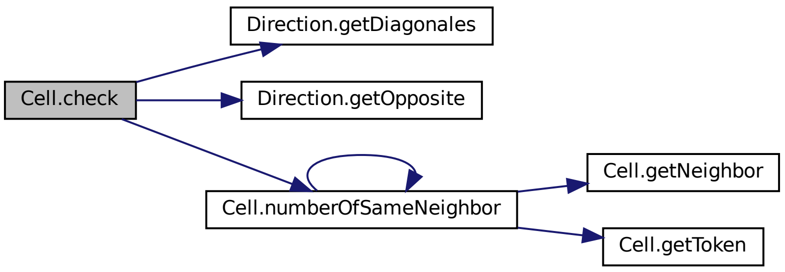
Check if this Cell has the number required of same Cell in a Direction to "win"

Returns

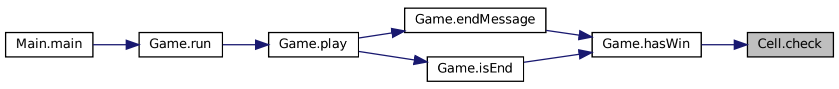
a boolean true if there is Game.numberOfTokenToWin neighbor with same token

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

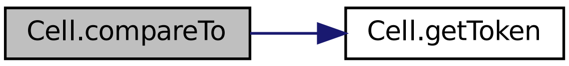
other	the other Cell to compare

Returns

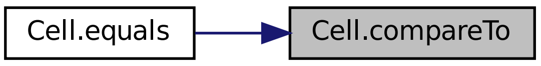
the comparaison

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

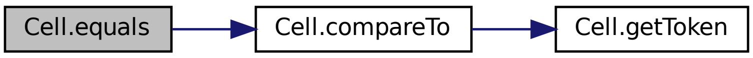
other	the other object to test equality

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 ( )
```

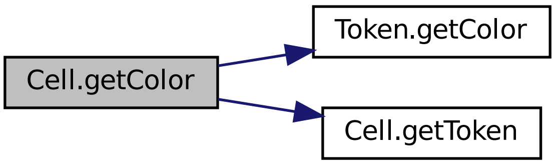
Get the Token Color of the Cell's Token.

Returns

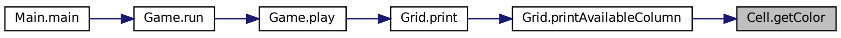
the specific Color

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

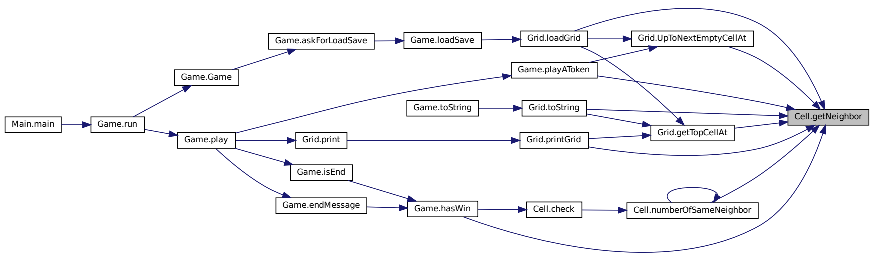
d	Direction where to get the neighbor

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 ( )
```

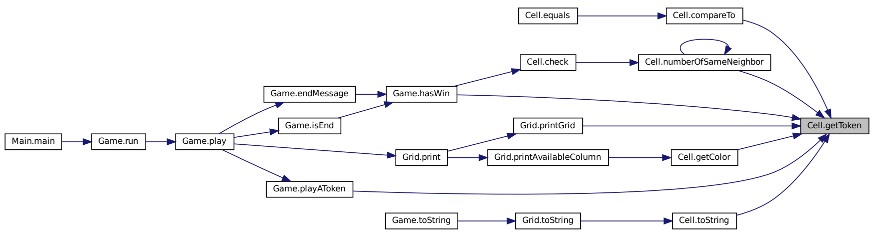
Get the Cell Token.

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 ( )
```

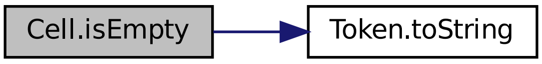
Test if the cell is EMPTY, if it's token if the emptyToken

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

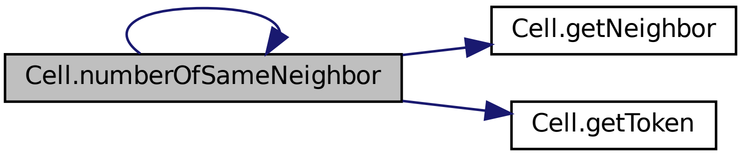
d	Direction to check

Returns

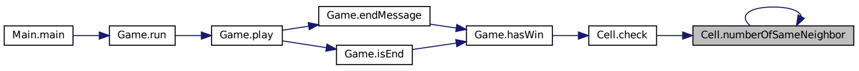
the number of same neighbor

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

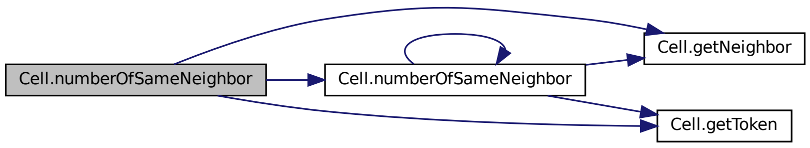
d1	first Direction diagonale
d2	second Direction diagonale

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 )
```

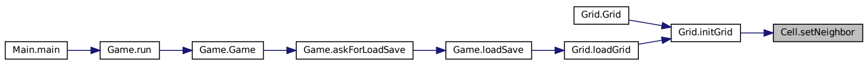
Set the Cell neighbor to this in a Direction

Parameters

c	Cell to set as a neighbor
d	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 )
```

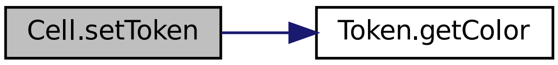
Set a Token to the Cell, can't set an EMPTY one

Parameters

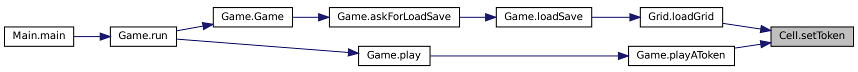
t	not EMPTY Token to place in

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 ( )
```

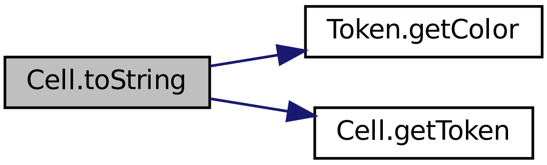
Override toString method, print the Color of the Token of the Cell

Returns

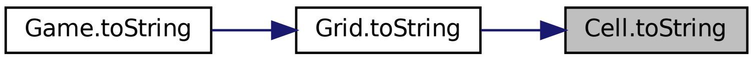
a String corresponding to the Color

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.

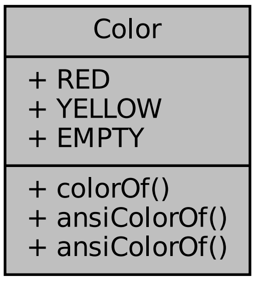
The documentation for this class was generated from the following file:

- /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

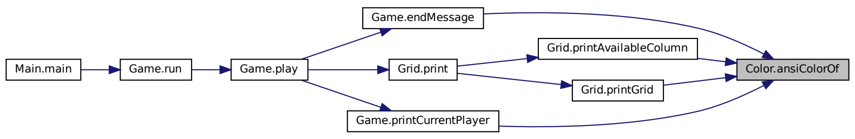
c	Color source

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

c	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

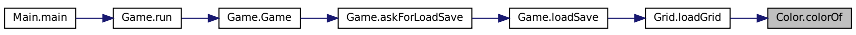
colorString	the color String representation you need

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
```

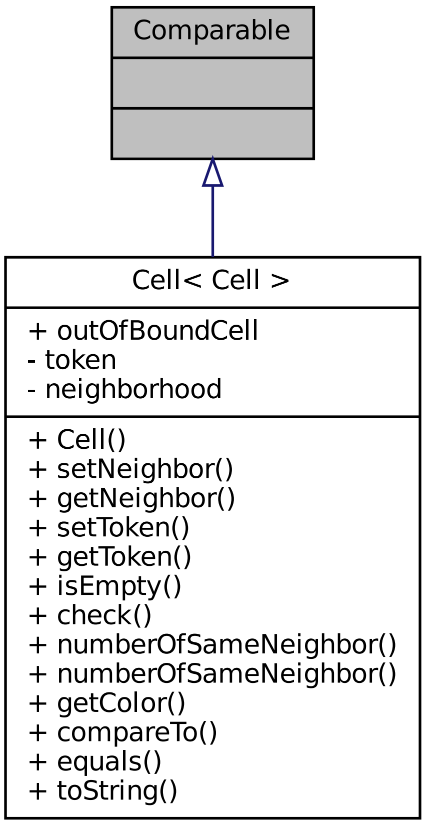
Definition at line 9 of file Color.java.

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

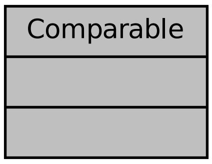
- /home/hozen/cur/projet-java/src/Color.java

4.3 Comparable Class Reference

Inheritance diagram for Comparable:



Collaboration diagram for Comparable:



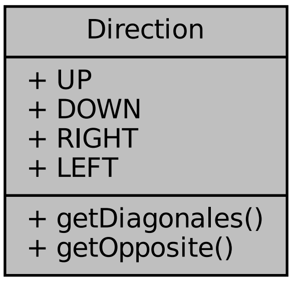
The documentation for this class was generated from the following file:

- /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]
```

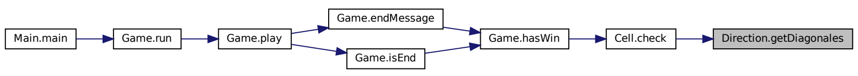
Give an EnumMap of Direction which each key is a Direction and its value the diagonale Direction corresponding.

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

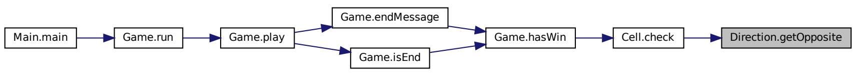
d	Direction source

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.

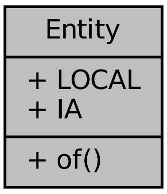
The documentation for this enum was generated from the following file:

- /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

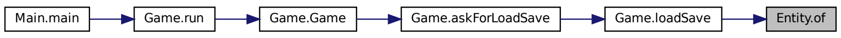
e	name of the entity

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.

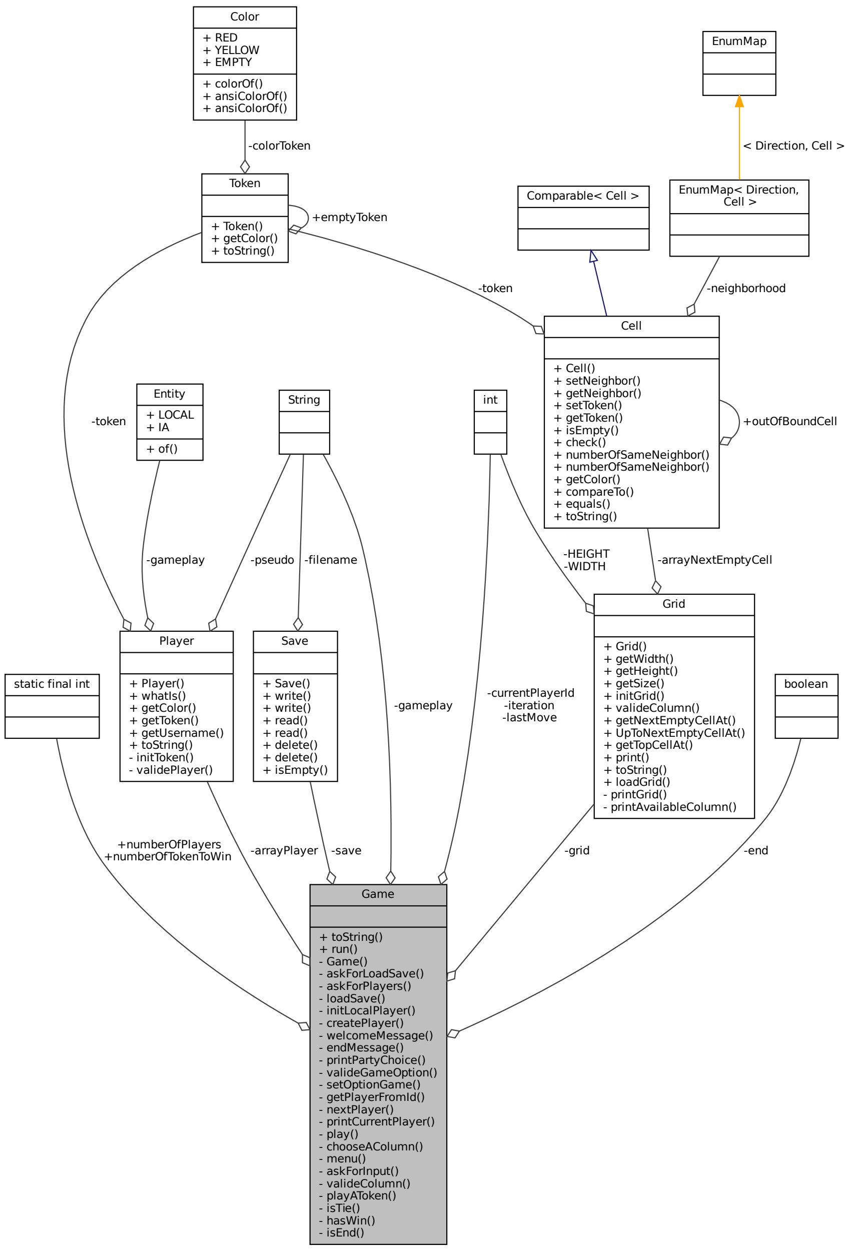
The documentation for this enum was generated from the following file:

- /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

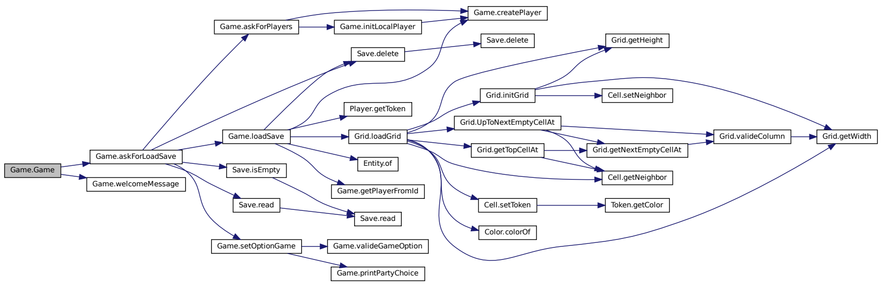
4.6.2.1 Game()

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

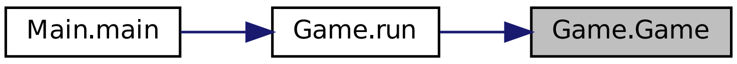
Game class constructor

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]
```

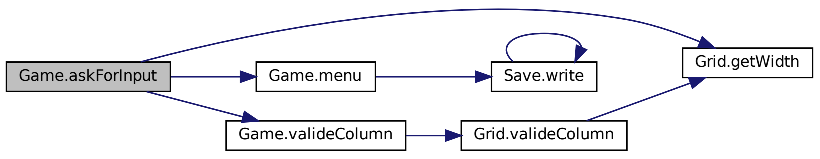
If the player is LOCAL ask what he want to do (play, menu).

Returns

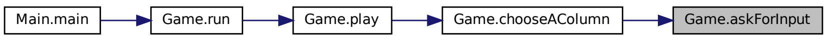
the column choosen.

Definition at line 354 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



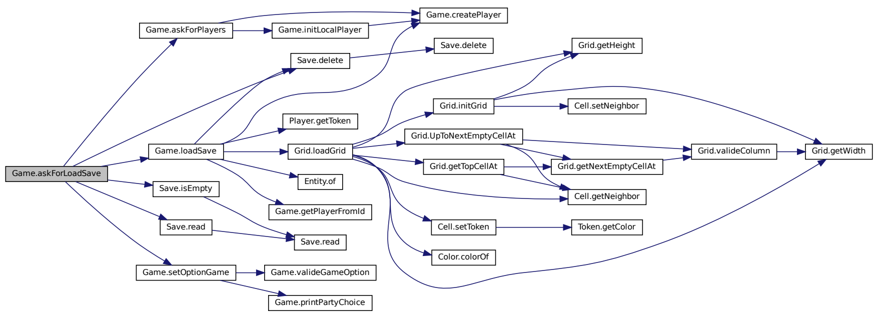
4.6.3.2 askForLoadSave()

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

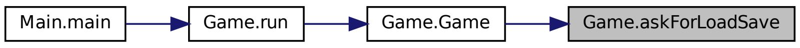
Ask the user if he wants to load an existing save

Definition at line 58 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



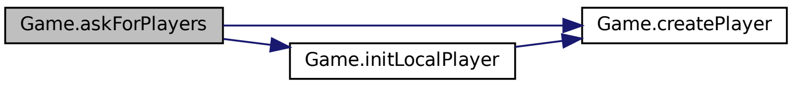
4.6.3.3 askForPlayers()

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

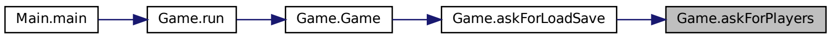
Initialisation of the Players

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]
```

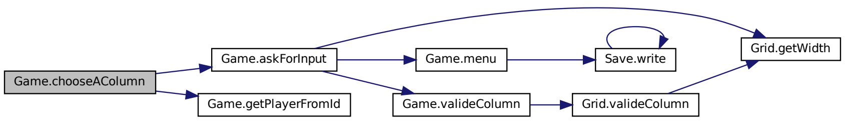
Choose a column depending if it's local or ia player

Returns

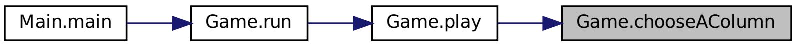
An int corresponding to the choosen column

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]
```

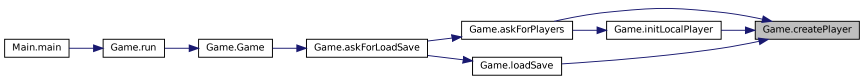
Create a player by call Player constructor

Parameters

name	name of the player
id	index of the player
og	what is the player (LOCAL or IA)

Definition at line 147 of file Game.java.

Here is the caller graph for this function:



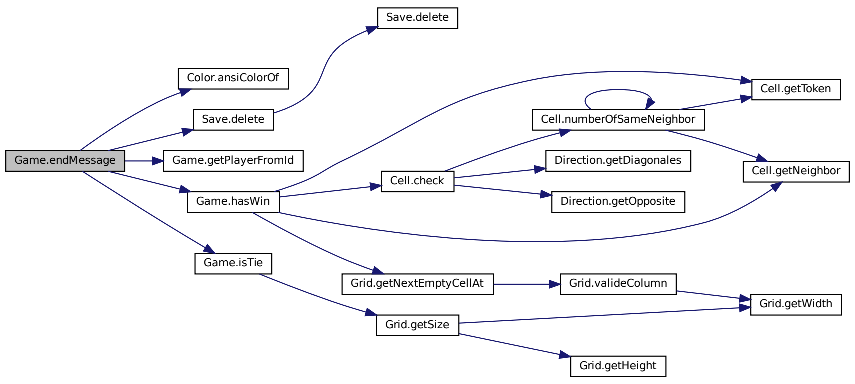
4.6.3.6 endMessage()

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

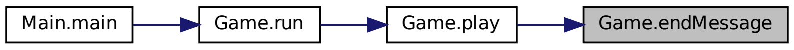
Print an end message

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

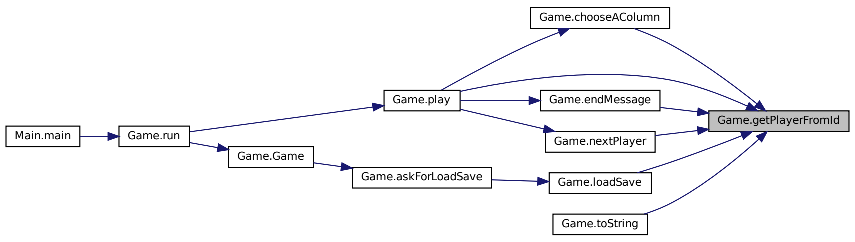
id	index of the Player

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]
```

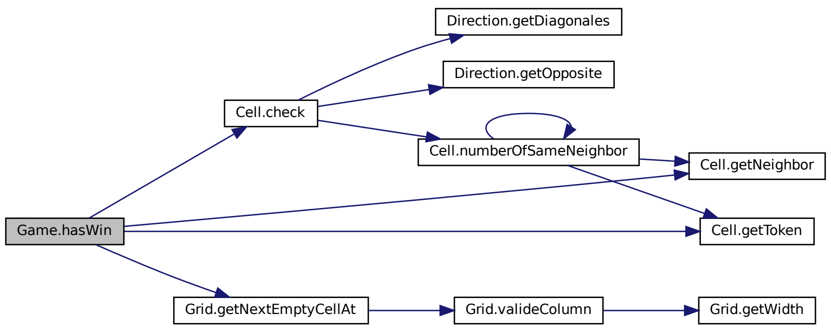
Win condition function: call check cell function on the last move

Returns

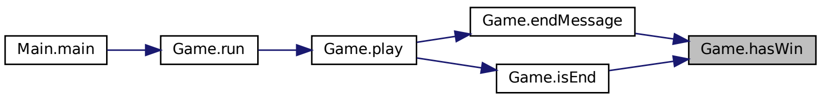
a boolean corresponding if it's a win or not

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]
```

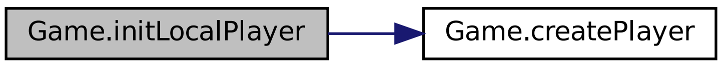
Initialisation of a LOCAL Player, ask his username

Parameters

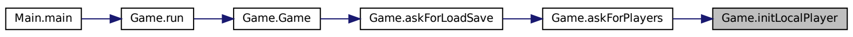
position	player position in his physical representation

Definition at line 136 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



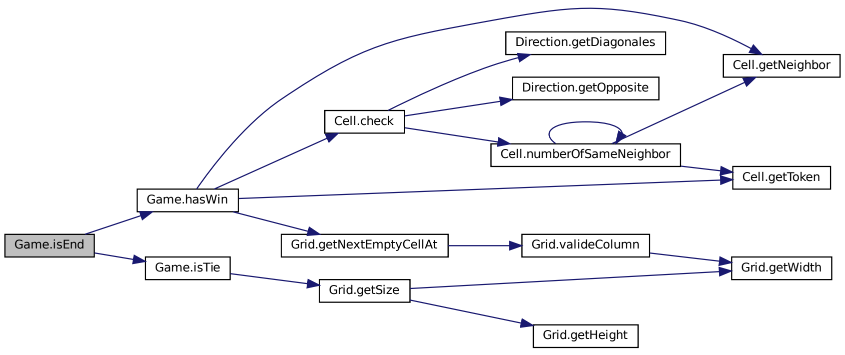
4.6.3.10 isEnd()

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

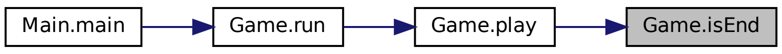
End condition : if it's a tie or a win it makes the game to end

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]
```

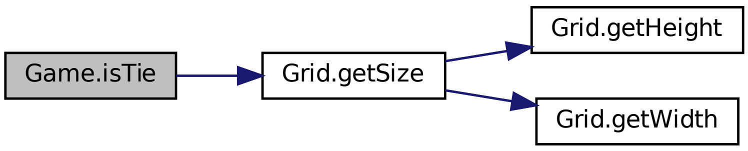
Tie function : if the number of play is equal to the size of the grid

Returns

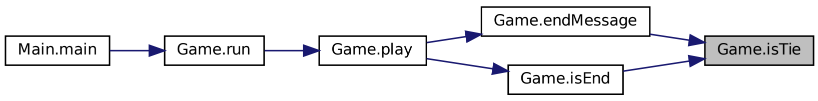
boolean corresponding if it's a tie or not

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]
```

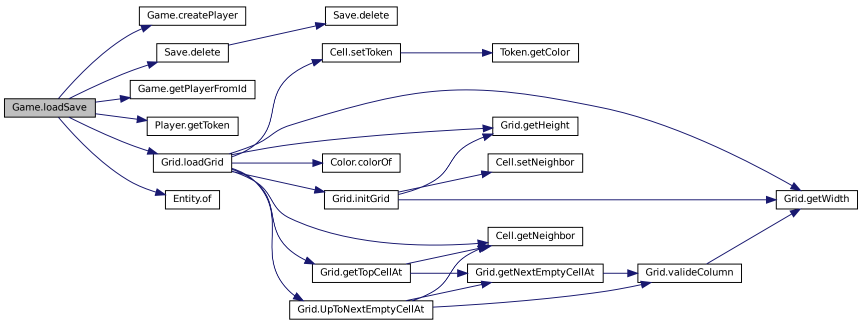
Initialisation of the game by an existing save

Parameters

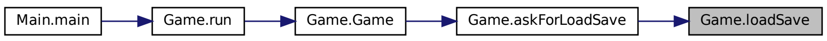
saveText	string representing a save

Definition at line 105 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



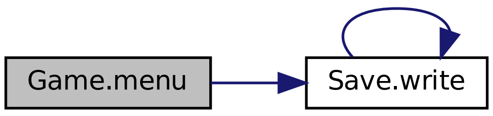
4.6.3.13 menu()

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

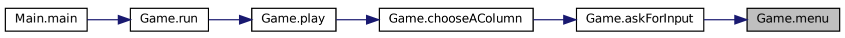
Game menu (play, save, quit) loop.

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

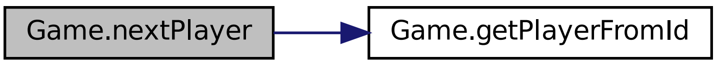
current	the last Player who have played

Returns

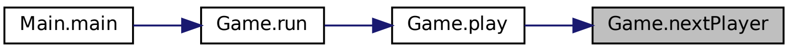
the next player to play

Definition at line 253 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



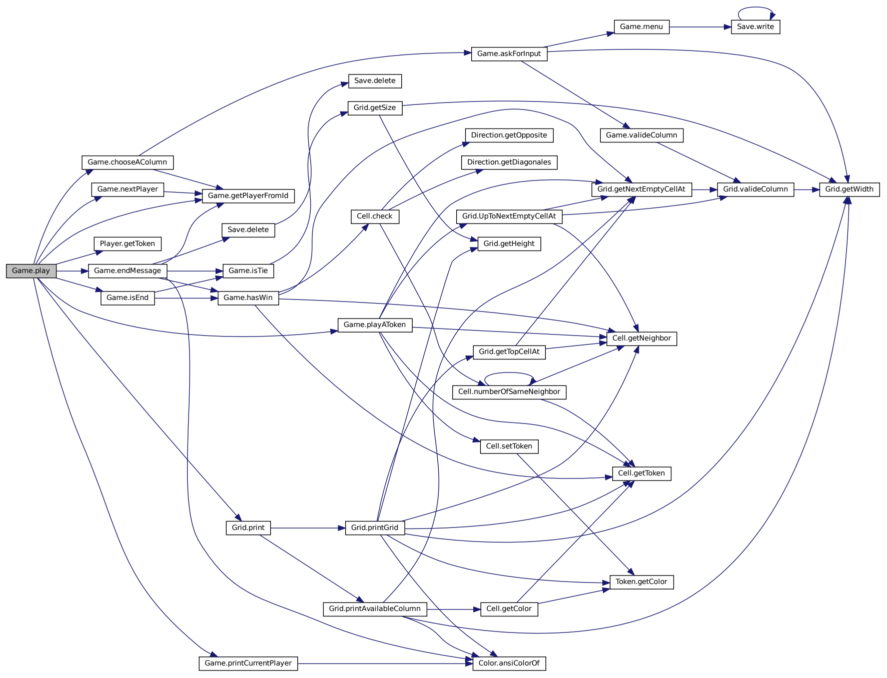
4.6.3.15 play()

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

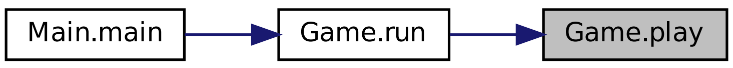
Game loop, end when game finish (Tie or Win or Quit)

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

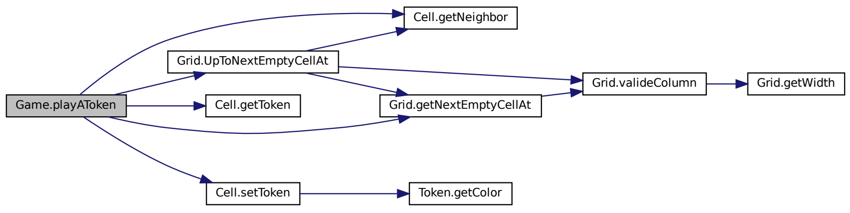
token	The token to play
column	The column where to play

Returns

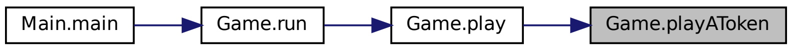
a boolean if the play is a success or not

Definition at line 393 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



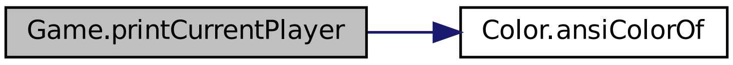
4.6.3.17 printCurrentPlayer()

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

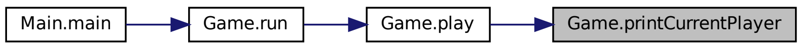
Print the toString() method in his personnal token color with ANSI

Definition at line 265 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



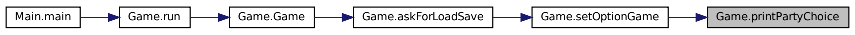
4.6.3.18 printPartyChoice()

`void Game.printPartyChoice ( ) [private]`

Print option game choice

Definition at line 188 of file Game.java.

Here is the caller graph for this function:



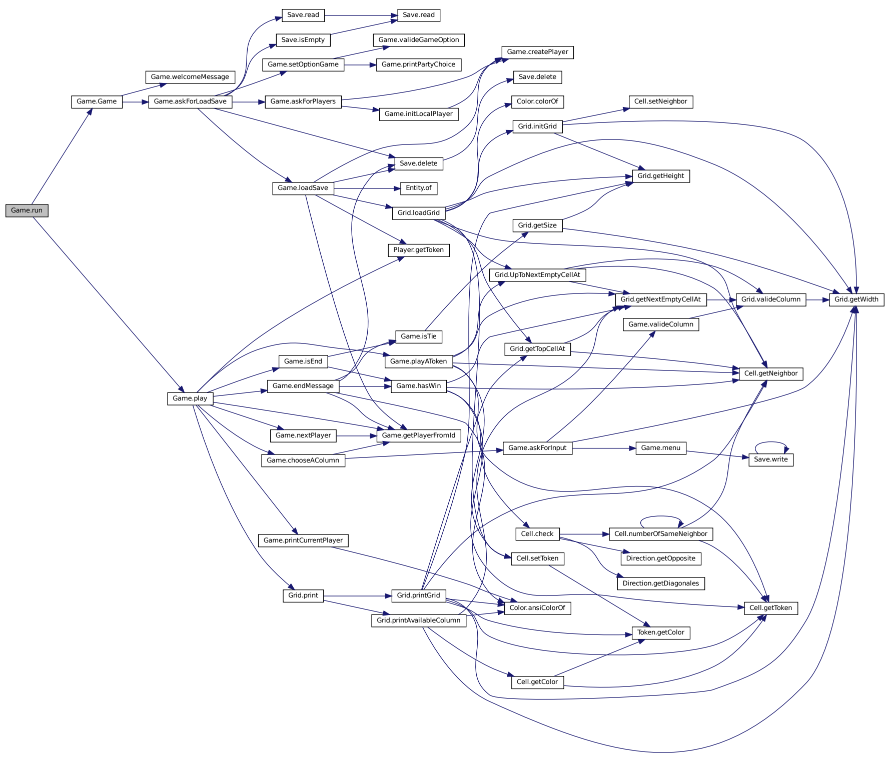
4.6.3.19 run()

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

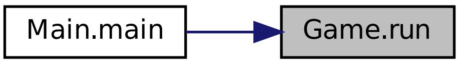
Run a game of puissance 4

Definition at line 232 of file Game.java.

Here is the call graph for this function:



Here is the caller graph for this function:



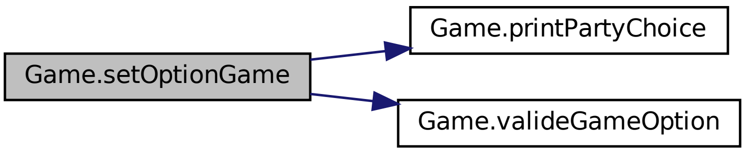
4.6.3.20 setOptionGame()

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

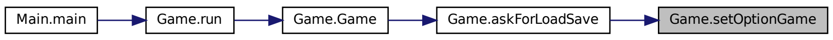
Ask the player to choose a Game Option (LOCAL or IA)

Definition at line 209 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 ( )
```

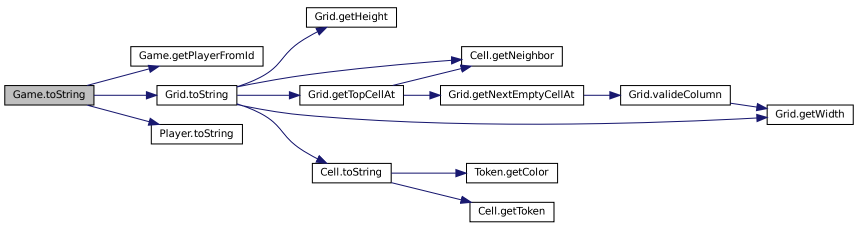
Representing color of grid cells, players informations and the next player who will play

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

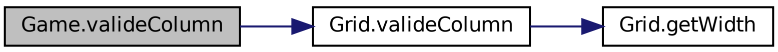
column	column to validate

Returns

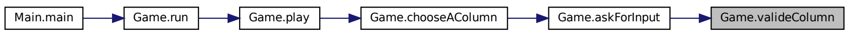
the valide column

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

option	game option to validate

Exceptions

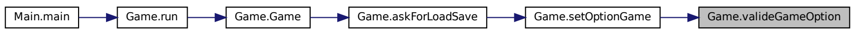
IllegalArgumentException	

Returns

valide game option

Definition at line 197 of file Game.java.

Here is the caller graph for this function:



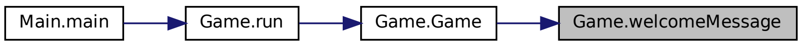
4.6.3.24 welcomeMessage()

```
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.

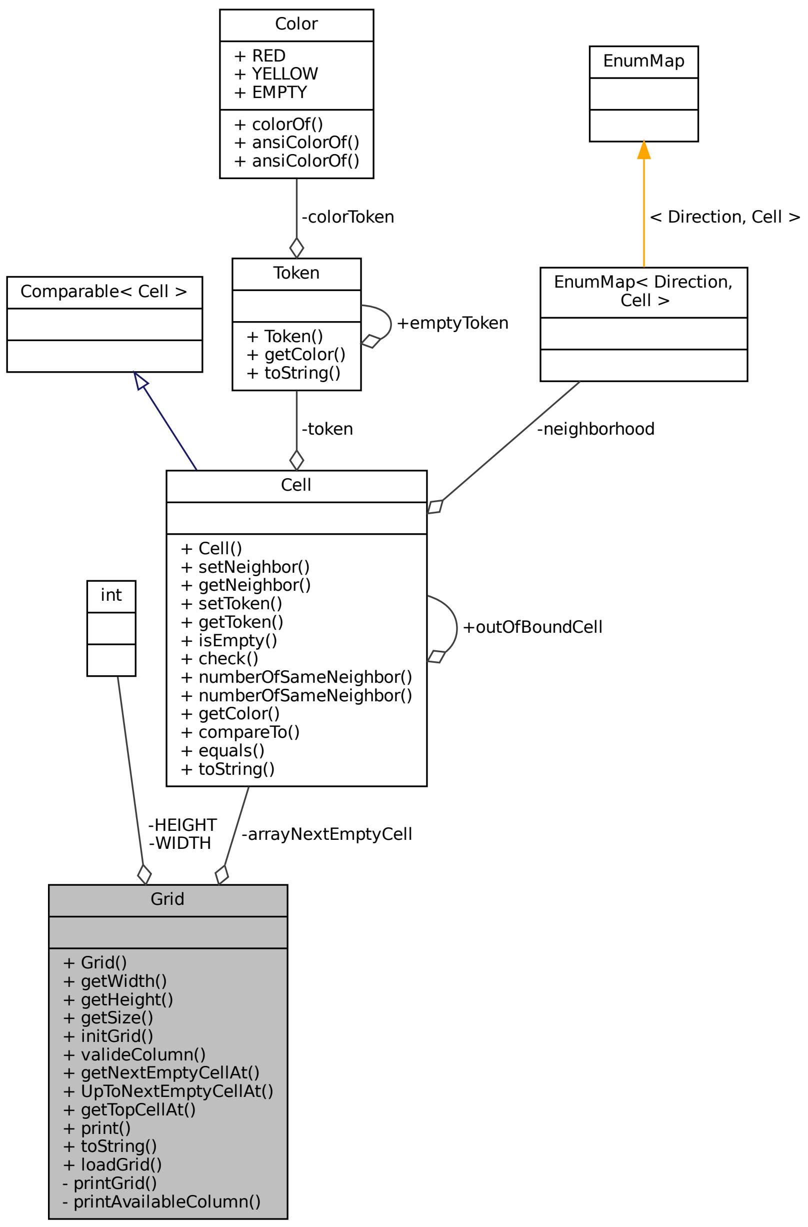
The documentation for this class was generated from the following file:

- /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

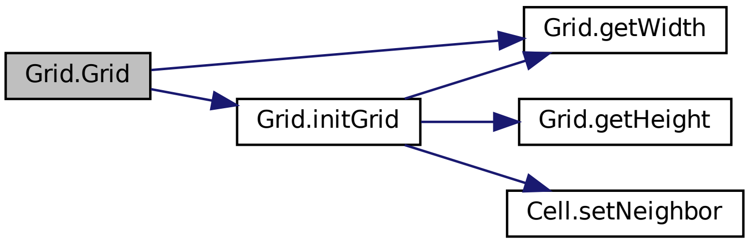
4.7.2.1 Grid()

```
Grid.Grid ( )
```

Grid constructor

Definition at line 19 of file Grid.java.

Here is the call graph for this function:



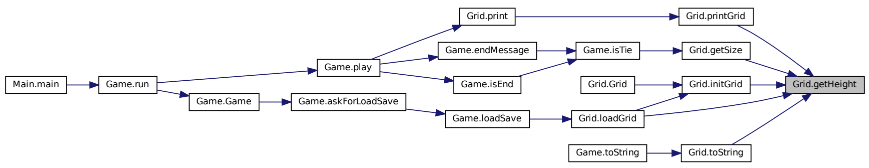
4.7.3 Member Function Documentation

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

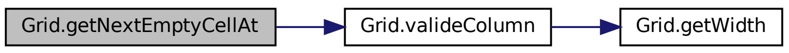
column	the column of the Grid where is the Cell

Returns

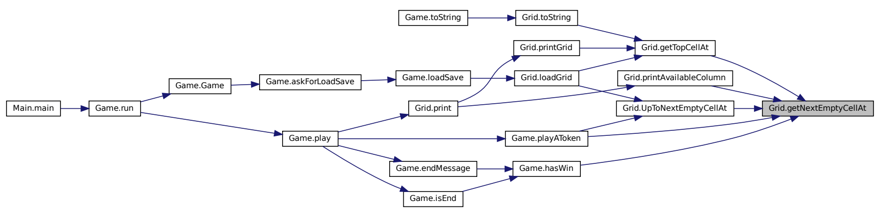
the Cell corresponding to the given column

Definition at line 78 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:

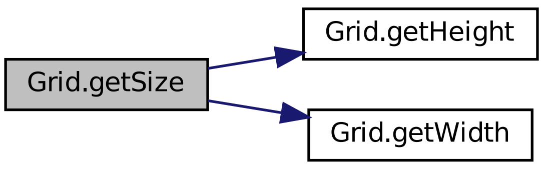


4.7.3.3 getSize()

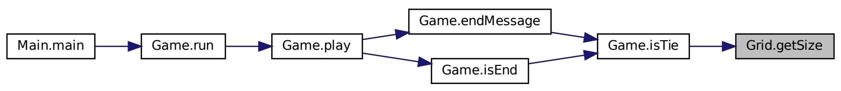
```
int Grid.getSize ( )
```

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

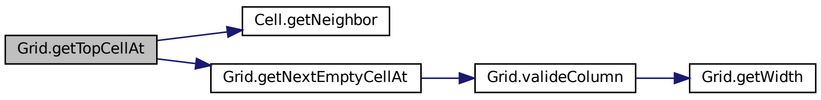
column	the column to get the top Cell

Returns

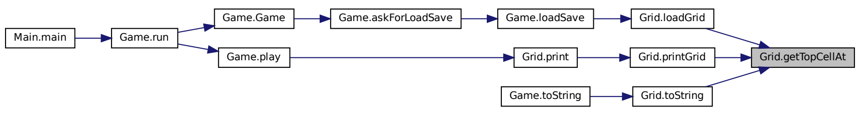
the top Cell of the column

Definition at line 99 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:

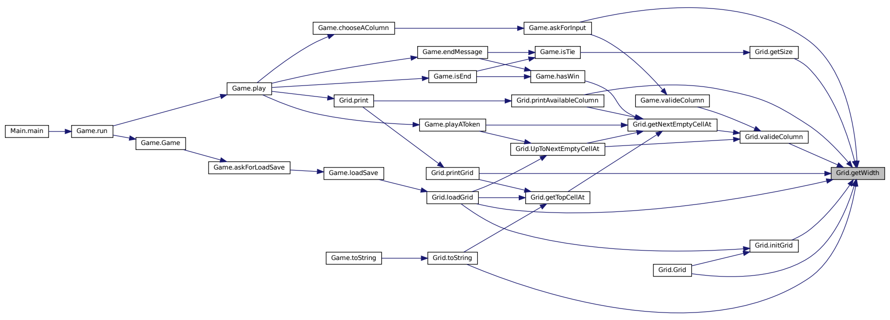


4.7.3.5 getWidth()

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

Definition at line 27 of file Grid.java.

Here is the caller graph for this function:



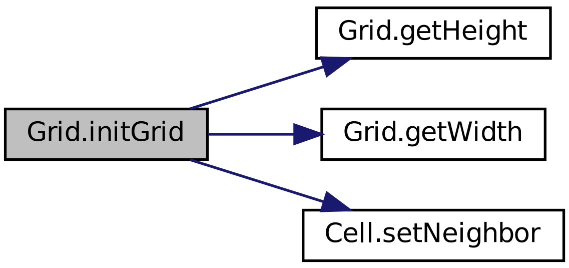
4.7.3.6 initGrid()

```
void Grid.initGrid ( )
```

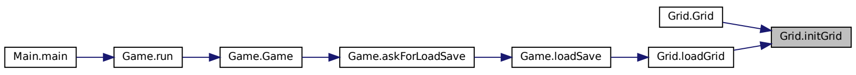
Initialisation of Cell of the Grid with their 4 neighbor

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 )
```

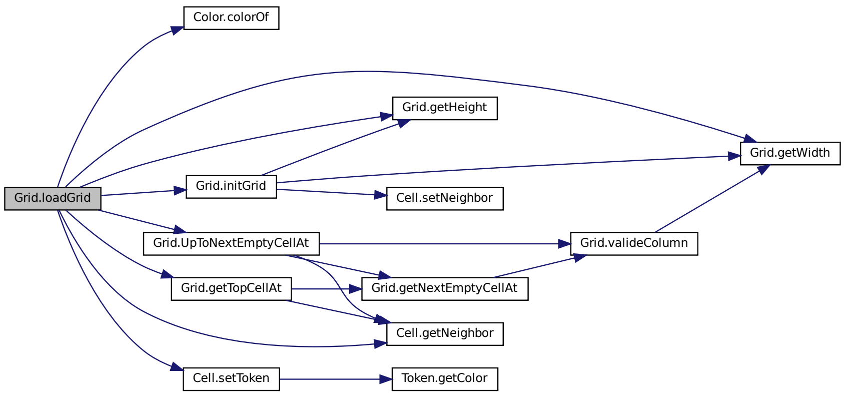
Load a Grid from a save (based on its own toString() method)

Parameters

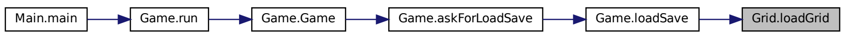
schema	the grid representation
tokenOfPlayers	a tab of Player Token

Definition at line 231 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:



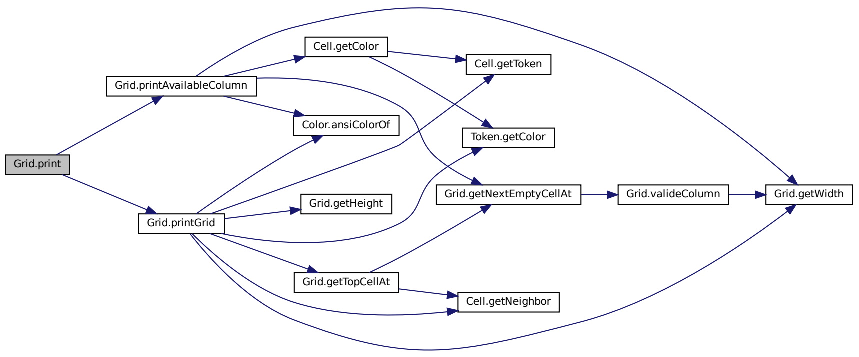
4.7.3.8 print()

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

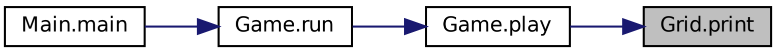
Print the pretty print of the grid and column available

Definition at line 199 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:



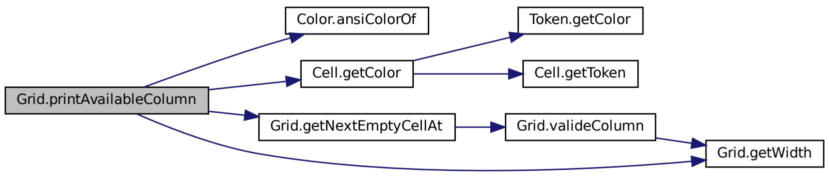
4.7.3.9 printAvailableColumn()

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

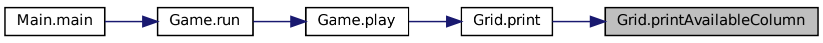
Pretty print of the column available to put a token in, print a 'X' when the column is full of Token

Definition at line 171 of file Grid.java.

Here is the call graph for this function:



Here is the caller graph for this function:



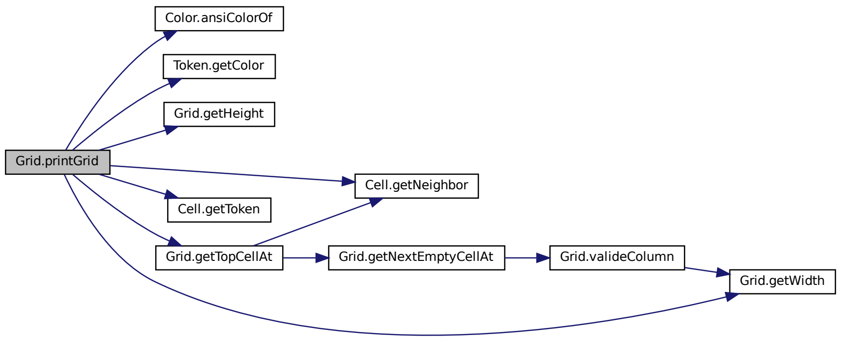
4.7.3.10 printGrid()

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

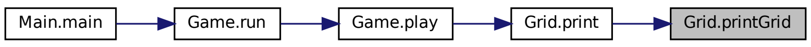
Pretty print of the grid with ansi color corresponding to the tokens in it

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 ( )
```

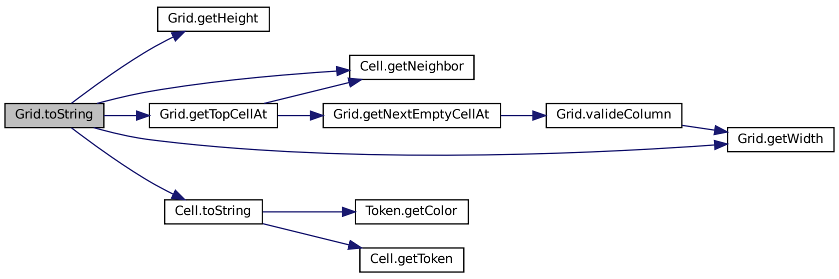
Return the String representation of the Grid : the color of each Token of each Cell

Returns

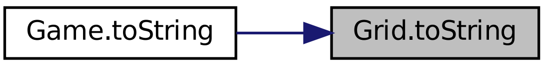
the String of Color separated by ';'

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 )
```

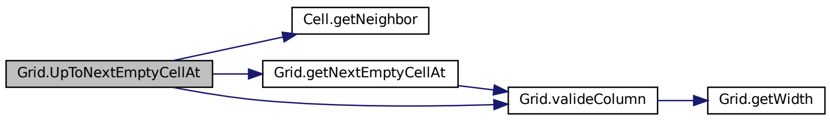
Update the next EMPTY Cell in the column

Parameters

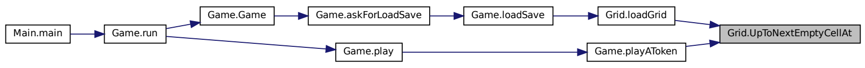
column	the column of the Grid where is the Cell

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

column	the column to verify

Exceptions

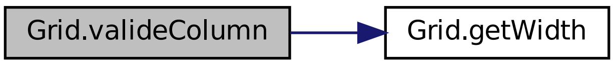
IllegalArgumentException	

Returns

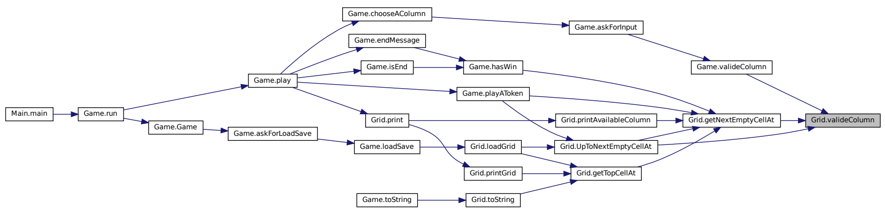
the valide column

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.

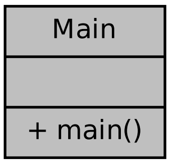
The documentation for this class was generated from the following file:

- /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.

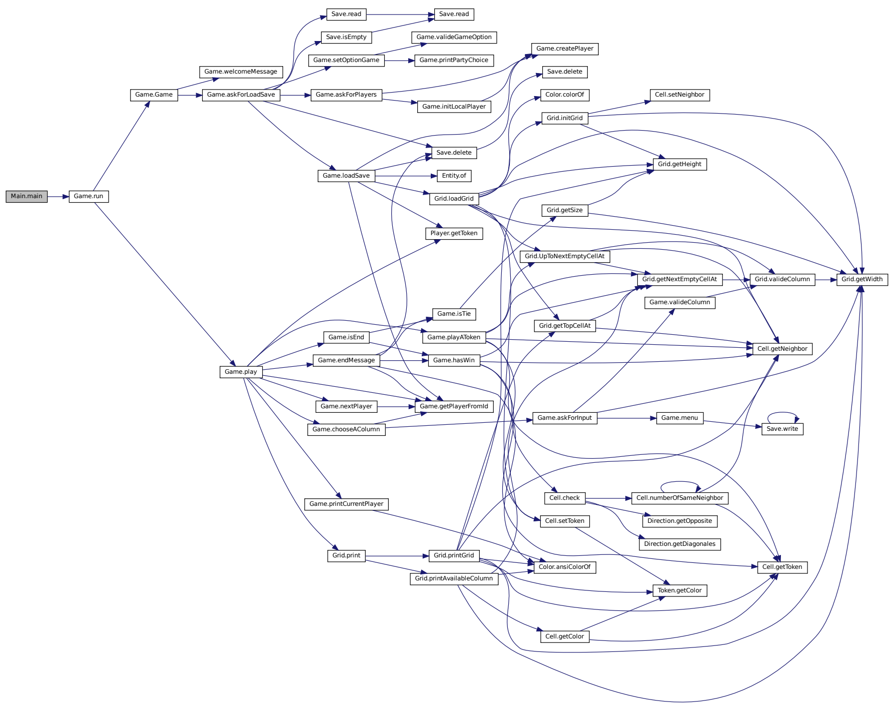
4.8.2 Member Function Documentation

4.8.2.1 main()

```
static void Main.main (
             String[] args )  [static]
```

Definition at line 8 of file Main.java.

Here is the call graph for this function:



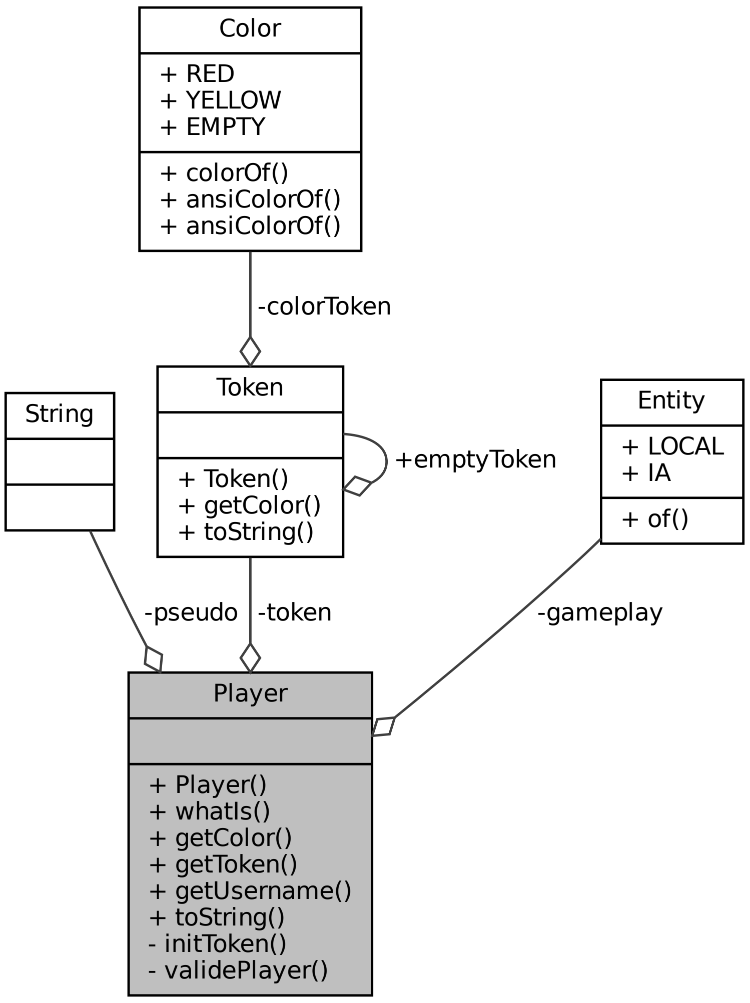
The documentation for this class was generated from the following file:

- /home/hozen/cur/projet-java/src/Main.java

4.9 Player Class Reference

Represent a Player of a Game using Token.

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 )
```

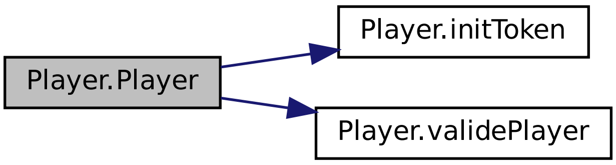
Player constructor

Parameters

pseudo	username of the player
indexPlayer	id of the player, decide its Token Color
e	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 ( )
```

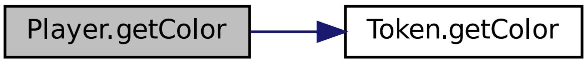
Give the Color of the Token of the Player (Delegation on Token getColor())

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 ( )
```

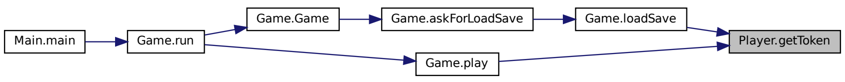
Give the reference of the Token of the Player

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

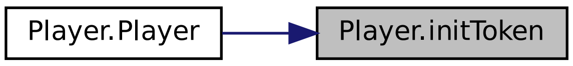
id	index of player

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 ( )
```

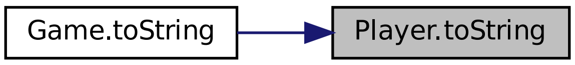
Give the String representation of the Player

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

id	index to validate

Exceptions

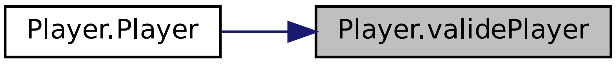
IllegalArgumentException	

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.

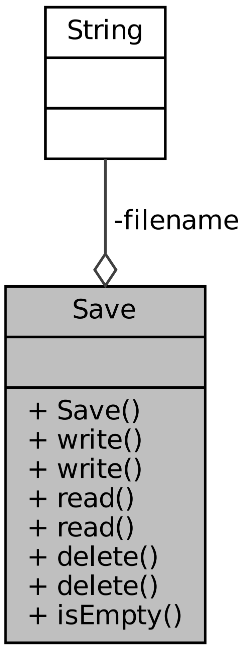
The documentation for this class was generated from the following file:

- /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

filename	filename to write, read, delete

Definition at line 17 of file Save.java.

4.10.3 Member Function Documentation

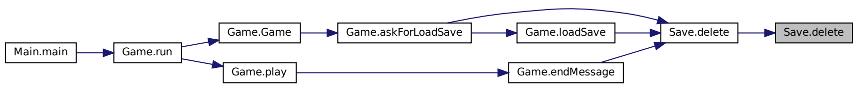
4.10.3.1 delete() `[1/2]`

```
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 )
```

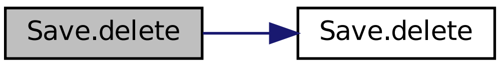
Verbose method of delete()

Parameters

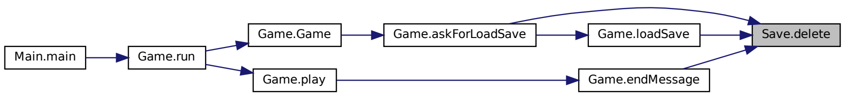
verbose	if true print a successful message

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 ( )`

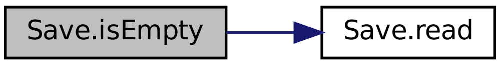
Response if the file content is empty

Returns

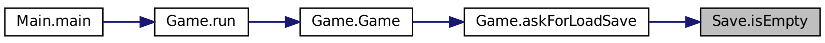
boolean true if file is empty, else false

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 ( )
```

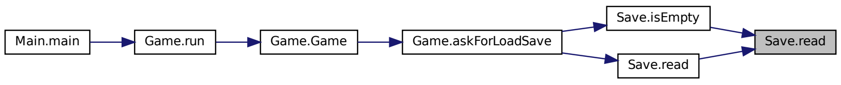
Read from a file and give its contents as a String

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

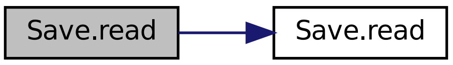
verbose	if true print a successful message

Returns

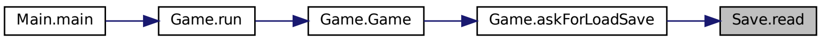
String representation of the content of the file

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 )
```

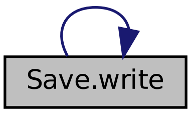
Verbose method of write(Object src)

Parameters

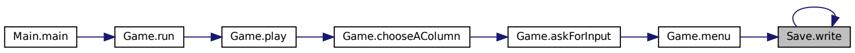
verbose	if true print a successful message
src	Object to write in the save

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.

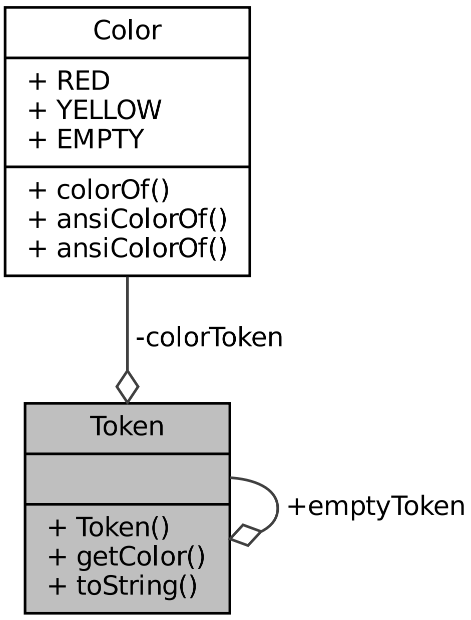
The documentation for this class was generated from the following file:

- /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 ( )`

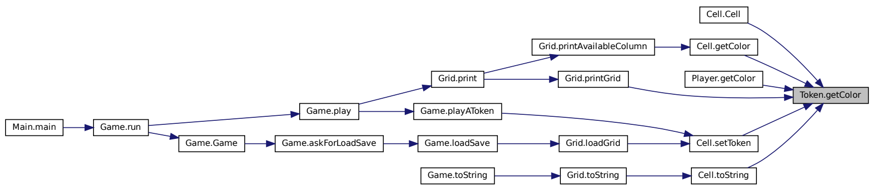
Give the Color of the Token

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 ( )`

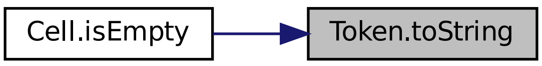
Give the String representation of a Token

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.

```
import static java.util.Objects.requireNonNull;
import java.util.EnumMap;

public class Cell implements Comparable<Cell> {

    public final static Cell outOfBoundCell = new Cell(Token.emptyToken);

    private Token token;
    private EnumMap<Direction, Cell> neighborhood;

    public Cell(Token token) {
    this.neighborhood = new EnumMap<>(Direction.class);
    if (token.getColor() == Color.EMPTY) this.token = Token.emptyToken;
    }

    public void setNeighbor(Cell c, Direction d) {
    this.neighborhood.put(requireNonNull(d), requireNonNull(c));
    }

    public Cell getNeighbor(Direction d) {
    Cell get;
    get = this.neighborhood.get(requireNonNull(d));
    if (get == null) {
        get = this.outOfBoundCell;
    }
    return get;
    }

    public void setToken(Token t) {
    if (this.token.getColor() != Color.EMPTY) {
        throw new IllegalArgumentException("There is already a token to this cell");
    }
    this.token = requireNonNull(t);
    }

    public Token getToken() {
    return this.token;
    }

    public boolean isEmpty() {
```

```
    return this.token.toString() == "EMPTY";
    }

    public boolean check() {
    EnumMap<Direction, Direction> diagonales = Direction.getDiagonales();
    int count;

    for (Direction d: Direction.values()) {
        /* Horizontal and vertical check */
        count = this.numberOfSameNeighbor(d);
        count += this.numberOfSameNeighbor(Direction.getOpposite(d));
        count ++; // Pour compter le pion actuel

        if (count >= Game.numberOfTokenToWin) return true;

        /* Diagonales check */
        count = this.numberOfSameNeighbor(d, diagonales.get(d));
        count += this.numberOfSameNeighbor(Direction.getOpposite(d),
                           Direction.getOpposite(diagonales.get(d)));
        count ++;

        if (count >= Game.numberOfTokenToWin) return true;
    }
    return false;
    }

    public int numberOfSameNeighbor(Direction d) {
    Cell next = this.getNeighbor(d);
    if (next == this.outOfBoundCell ||
        this.getToken() != next.getToken()) {
        return 0;
    }
    else return 1 + next.numberOfSameNeighbor(d);
    }

    public int numberOfSameNeighbor(Direction d1, Direction d2) {
    Cell next = this.getNeighbor(d1).getNeighbor(d2);
    if (next == this.outOfBoundCell ||
        this.getToken() != next.getToken()) {
        return 0;
    }
    else return 1 + next.numberOfSameNeighbor(d1, d2);
    }

    public Color getColor() {
    return this.getToken().getColor();
    }

    @Override
    public int compareTo(Cell other) {
    if (this.getToken() == other.getToken()) return 0;
    return 1;
    }

    @Override
    public boolean equals(Object other) {
    if (other == null) return false;
    if (other == this) return true;
    if (!(other instanceof Cell)) return false;
    return this.compareTo((Cell) other) == 0;
    }

    @Override
    public String toString() {
    return this.getToken().getColor().toString();
    }
}
```

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.

```
public enum Color {

    RED, YELLOW, EMPTY;

    public static Color colorOf(String colorString) {
    switch (colorString) {
    case "RED":
        return Color.RED;
    case "YELLOW":
        return Color.YELLOW;
    }
    return Color.EMPTY;
    }

    public static String ansiColorOf(Color c) {
    switch (c) {
    case RED :
        return ansiColorOf("RED");
    case YELLOW :
        return ansiColorOf("YELLOW");
    }
    return ansiColorOf("WHITE");
    }

    public static String ansiColorOf(String c) {
    switch (c) {
    case "RED":
        return "\u001B[31m"; // RED
    case "YELLOW":
        return "\u001B[33m"; // YELLOW
    case "BLUE":
        return "\u001B[36m"; // BLUE
    case "GREEN":
        return "\u001B[32m"; // GREEN
    case "WHITE":
        return "\u001B[37m"; // WHITE
    }
    System.out.println("There is no color corresponding to the argument: white is returning");
    return ansiColorOf("WHITE");
    }
}
```

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.

```
import java.util.EnumMap;

public enum Direction {

    UP, DOWN, RIGHT, LEFT;

    public static EnumMap<Direction, Direction> getDiagonales() {
    EnumMap<Direction, Direction> diagonales = new EnumMap<>(Direction.class);
    diagonales.put(Direction.UP, Direction.LEFT);
    diagonales.put(Direction.DOWN, Direction.RIGHT);
    diagonales.put(Direction.LEFT, Direction.DOWN);
    diagonales.put(Direction.RIGHT, Direction.UP);
    return diagonales;
    }

    public static Direction getOpposite(Direction d) {
    switch (d) {
    case UP:
        return DOWN;
    case DOWN:
        return UP;
    case RIGHT:
        return LEFT;
    case LEFT:
        return RIGHT;
    }
    return UP;
    }
}
```

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.

```

public enum Entity {

    LOCAL, IA;

    public static Entity of(String e) {
    Entity option = LOCAL;
    switch (e) {
    case "LOCAL":
        option = LOCAL;
        break;
    case "IA":
        option = IA;
        break;
    }
    return option;
    }
}
```

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.

```
import java.util.Random;
import java.util.Scanner;
import java.util.InputMismatchException;
import static java.util.Objects.requireNonNull;
import java.util.EnumMap;
import java.lang.Math;

public class Game {
    // Const
    public final static int numberOfPlayers = 2;
    public final static int numberOfTokenToWin = 4;

    // Gameplay attributs
    private Player[] arrayPlayer;
    private Grid grid;

    // Save
    private final Save save;

    // Game state attributs
    private boolean end;
    private int iteration;
    private String gameplay; // IA or LOCAL

    private int currentPlayerId;
    private int lastMove; // represent the last column played

    private Game() {
        this.welcomeMessage();

        // Init Player physical representation
        this.arrayPlayer = new Player[numberOfPlayers];

        // Init Grid
        this.grid = new Grid();

        // Init random first Player
        this.currentPlayerId = (int) Math.round(Math.random());

        // Init Save
        this.save = new Save("../sauv");
        this.askForLoadSave();

        // Init Game State
        this.end = false;
        this.iteration = 0;
    }

    private void askForLoadSave() {
        if (!this.save.isEmpty()) {
            boolean done = false;
            while (!done) {
                System.out.println("Sauvegarde est existante, voulez-vous la charger ?");
                System.out.println("[1:Oui][2:Non][3:Supprimer]");
                Scanner input = (new Scanner(System.in));
                try {
                    switch (input.nextInt()) {
                    case 1:
                        this.loadSave(this.save.read());
                        break;
                    case 3:
                        this.save.delete();
                        this.setOptionGame();
                        this.askForPlayers();
                        break;
                    default:
                        this.setOptionGame();
                        this.askForPlayers();
                        break;
                    }
                    done = true;
                }
                catch (InputMismatchException e) {
                    System.out.println("Ce choix n'existe pas");
                }
            }
        }
        else {
            this.setOptionGame();
            this.askForPlayers();
        }
    }
```

```java

    private void askForPlayers() {
        this.initLocalPlayer(0) ;  // premier joueur toujours humain
        if(this.gameplay.equals("LOCAL")) initLocalPlayer(1);
        else createPlayer("Ordinateur", 1, Entity.IA);
    }

    private void loadSave(String saveText) {
        String[] saveSplit = saveText.split("&");
        if (saveSplit.length < this.numberOfPlayers+2) {
            this.save.delete();
            throw new IllegalArgumentException("Sauvegarde corrompue");
        }

        // Initialise the players //
        for (int i=0; i<this.numberOfPlayers; i++) {
            Entity og = Entity.of(saveSplit[i+1].split(" ")[0]);
            String playerName = saveSplit[i+1].split(" ")[1];
            this.createPlayer(playerName, i, og);
        }

        // Initialise the current player //
        this.currentPlayerId = Integer.parseInt(saveSplit[this.numberOfPlayers+1]);

        // Initialise the grid //
        Token[] tokenArray = new Token[Color.values().length];
        for (int i=0; i<this.numberOfPlayers; i++) {
            tokenArray[i] = this.getPlayerFromId(i).getToken();
        }
        tokenArray[Color.values().length-1] = Token.emptyToken;
        this.grid.loadGrid(saveSplit[0], tokenArray);
    }

    private void initLocalPlayer(int position) {
        System.out.println("Veuillez saisir le pseudo du joueur " + position + " :");
        String username = (new Scanner(System.in)).nextLine();
        this.createPlayer(username, position, Entity.LOCAL);
    }

    private void createPlayer(String name, int id, Entity og) {
        this.arrayPlayer[id] = new Player(name, id, og);
    }

    private void welcomeMessage() {
        System.out.println("Bienvenu au jeu du puissance 4.\n");
    }

    private void endMessage() {
        // Print game ended
        System.out.println("*****************************");
        System.out.println("*" + Color.ansiColorOf("RED") +
                           "         Game Ended        "
                           + Color.ansiColorOf("WHITE") + "*");
        System.out.println("*****************************");

        // Print the winner or if pat
        if (this.isTie()) {
            this.save.delete();
            System.out.println("Egalité");
        }
        else if (this.hasWin()) {
            this.save.delete();
            System.out.println("Le gagnant est : " +
                               this.getPlayerFromId(this.currentPlayerId));
        }
    }

    private void printPartyChoice() {
        System.out.println("Vous avec 2 options de jeu :");
        System.out.println("-[1] 1 VS 1 avec 2 joueurs en local");
        System.out.println("-[2] 1 VS 1 contre une IA\n");
    }

    private int valideGameOption(int option) {
        if (option <= 0 || option > 2) throw new IllegalArgumentException();
        return option;
    }

    private void setOptionGame() {
        while (true) {
            try {
                printPartyChoice();
                System.out.println("A quel mode de jeu voulez-vous jouer ? ");
                int option = valideGameOption((new Scanner(System.in)).nextInt());
                if (option == 1) this.gameplay = "LOCAL";
                else this.gameplay = "IA";
                break;
            }
            catch (IllegalArgumentException e) {
                System.out.println("Ce mode de jeu n’existe pas.\n");
            }
            catch (InputMismatchException e) {
                System.out.println("Erreur lors de la saisie," +
                                   "veuillez saisir l’entier ’1’ ou ’2’\n");
            }
        }
    }

    public static void run() {
        Game game = new Game();
        game.play();
    }

    private Player getPlayerFromId(int id) {
        if (id < 0 || id > this.numberOfPlayers) {
            throw new IllegalArgumentException("Bad player id");
        }
        return this.arrayPlayer[id];
    }

    private Player nextPlayer(Player current) {
        this.currentPlayerId = (this.currentPlayerId+1)%this.numberOfPlayers;
        return this.getPlayerFromId(this.currentPlayerId);
    }

    private void printCurrentPlayer() {
        System.out.println(
            Color.ansiColorOf(this.getPlayerFromId(this.currentPlayerId).getColor())
            + "["
            + this.getPlayerFromId(this.currentPlayerId) + "]"
            + Color.ansiColorOf("WHITE"));
    }

    private void play() {
        boolean playerHasPlay;
        Player currentPlayer = this.getPlayerFromId(this.currentPlayerId);

        while (!this.end) {

            this.iteration += 1;
            this.grid.print();
            playerHasPlay = false;

            while (!playerHasPlay) {
                this.printCurrentPlayer();
                System.out.println("[m: Menu]");
                int columnChosen = chooseAColumn();
                if (this.end) break;
                playerHasPlay = this.playAToken(currentPlayer.getToken(),
                                                columnChosen);
            }

            this.isEnd();

            if (!this.end) {
                currentPlayer = nextPlayer(currentPlayer);
            }
        }
        this.grid.print();
        this.endMessage();
    }

    private int chooseAColumn() {
        if (this.getPlayerFromId(this.numberOfPlayers-1).whatIs() == Entity.IA) {
            if (this.currentPlayerId == this.numberOfPlayers-1) {
                return (new Random()).nextInt(7);
            }
        }
        return this.askForInput();
    }

    private void menu() {
        boolean done = false;
        while (!done) {
            System.out.println("[1:Continuer][2:Sauvegarder][3:Quitter le jeu]");
            System.out.print("Choississez une option : ");
            Scanner input = new Scanner(System.in);
            try {
                switch (input.nextInt()) {
                case 1:
                    done = true;
                    break;
                case 2:
                    this.save.write(this);
                    break;
                case 3:
                    done = true;
                    this.end = true;
                    break;
                default:
                    done = true;
                    break;
                }
            }
            catch (InputMismatchException e) {
                System.out.println("Ce choix n'existe pas.");
            }
        }
    }

    private int askForInput() {
        int column = 0;
        while (true) {
            System.out.print("Choisissez une colonne : ");
            Scanner input = new Scanner(System.in);
            String sInput = input.nextLine();
            if (sInput.equals("m")) {
                this.menu();
                if (this.end) break;
            }
            else {
                try {
                    column = valideColumn(Integer.parseInt(sInput)-1);
                    break;
                }
                catch (IllegalArgumentException e) {
                    System.out.println("Colonne invalide, un entier en 1 et " +
                                       this.grid.getWidth());
                }
            }
        }
        return column;
    }

    private int valideColumn(int column) {
        return this.grid.valideColumn(column);
    }

    private boolean playAToken(Token token, int column) {
        if (token == Token.emptyToken) {
            throw new IllegalArgumentException("You can't play an empty Token");
        }

        Cell played = this.grid.getNextEmptyCellAt(column);
        if (played.getToken() != Token.emptyToken) {
            System.out.println("La colonne est pleine");
            return false;
        }
        played.setToken(requireNonNull(token));
        if (played.getNeighbor(Direction.UP) != Cell.outOfBoundCell) {
            this.grid.UpToNextEmptyCellAt(column);
        }
        this.lastMove = column;
        return true;
    }

    private boolean isTie() {
        return this.iteration == this.grid.getSize();
    }

    private boolean hasWin() {
        Cell lastCellPlayed = this.grid.getNextEmptyCellAt(this.lastMove);
        if (lastCellPlayed.getToken() != Token.emptyToken);
        else lastCellPlayed = lastCellPlayed.getNeighbor(Direction.DOWN);
        return lastCellPlayed.check();
    }

    private void isEnd() {
        if (this.isTie() || this.hasWin()) this.end = true;
    }

    @Override
    public String toString() {
        return this.grid.toString() + "&"
            + this.getPlayerFromId(0).toString() + "&"
            + this.getPlayerFromId(1).toString() + "&"
            + String.valueOf(this.currentPlayerId);
    }
}
```

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.

```
import static java.util.Objects.requireNonNull;
import java.util.EnumMap;

public class Grid {

    private static final int WIDTH = 7;
    private static final int HEIGHT = 6;

    private Cell[] arrayNextEmptyCell; /* contient la référance vers la prochaine cellule
                      vide de chaque colonne */
    /* Constructeurs */

    public Grid() {
        this.arrayNextEmptyCell = new Cell[this.getWidth()];
    this.initGrid();
    }

    public int getWidth() {return this.WIDTH;}

    public int getHeight() {return this.HEIGHT;}

    public int getSize() {return this.getWidth()*this.getHeight();}

    public void initGrid() {
    Cell[][] tempGrid2D = new Cell[this.getHeight()][this.getWidth()];
    for(int i = 0; i < this.getHeight(); i++) {
        for(int j = 0; j < this.getWidth(); ++j) {
        tempGrid2D[i][j] = new Cell(Token.emptyToken);
        }
    }
    for(int i = 0; i < this.getHeight(); i++) {
        for(int j = 0; j < this.getWidth(); ++j) {
        if (i > 0) {
            tempGrid2D[i][j].setNeighbor(tempGrid2D[i-1][j], Direction.UP);
        }
        if (i < this.getHeight()-1) {
            tempGrid2D[i][j].setNeighbor(tempGrid2D[i+1][j], Direction.DOWN);
        }
        if (j > 0) {
            tempGrid2D[i][j].setNeighbor(tempGrid2D[i][j-1], Direction.LEFT);
        }
        if (j < this.getWidth()-1) {
            tempGrid2D[i][j].setNeighbor(tempGrid2D[i][j+1], Direction.RIGHT);
        }
        }
    }
    for(int j = 0; j<this.getWidth() ; ++j) {
        this.arrayNextEmptyCell[j] = tempGrid2D[this.getHeight()-1][j] ;
    }
    }

    public int valideColumn(int column) {
    if (column < 0 || column >= this.getWidth()) {
        throw new IllegalArgumentException("column outOfBound");
    }
    return column;
    }

    public Cell getNextEmptyCellAt(int column) {
    return this.arrayNextEmptyCell[valideColumn(column)];
    }

    public void UpToNextEmptyCellAt(int column) {
    this.arrayNextEmptyCell[valideColumn(column)] =
        this.getNextEmptyCellAt(column).getNeighbor(Direction.UP);
    }

```

```
public Cell getTopCellAt(int column) {
// Parcours jusqu'à la dernière cellule haute de la colonne i
Cell top = this.getNextEmptyCellAt(column);
while (top.getNeighbor(Direction.UP) != Cell.outOfBoundCell){
    top = top.getNeighbor(Direction.UP);
}
return top;
}

private void printGrid() {
char[][] tempArray = new char[this.getHeight()][this.getWidth()];
for(int i=0; i < this.getWidth(); ++i) {
    Cell cellTemp = this.arrayNextEmptyCell[i];
    cellTemp = this.getTopCellAt(i);
    int j = 0;
    // Parcours jusqu'à la dernière cellule basse de la colonne i
    while (cellTemp != Cell.outOfBoundCell) {
    if (cellTemp.getToken().getColor() == Color.RED) {
        tempArray[j][i] = 'r';
    }
    else if (cellTemp.getToken().getColor() == Color.YELLOW) {
        tempArray[j][i] = 'y';
    }
    else if (cellTemp.getToken().getColor() == Color.EMPTY) {
        tempArray[j][i] = ' ';
    }
    ++j;
    cellTemp = cellTemp.getNeighbor(Direction.DOWN);
    }
}

System.out.print(Color.ansiColorOf("BLUE"));

for(int i=0; i<this.getHeight(); ++i) {
    for(int j=0; j<this.getWidth(); ++j) {
    System.out.print("+---");
    }
    System.out.println("+");
    System.out.print("|");
    for(int j=0; j<this.getWidth(); ++j) {
    if(tempArray[i][j] == 'r') {
        System.out.print(Color.ansiColorOf("RED") +
                 " O " +
                 Color.ansiColorOf("BLUE"));
    }
    else if (tempArray[i][j] == 'y') {
        System.out.print(Color.ansiColorOf("YELLOW") +
                 " O " +
                 Color.ansiColorOf("BLUE"));
    }
    else {
        System.out.print("   ");
    }
    System.out.print("|");
    }
    System.out.println();
}
for(int j=0; j<this.getWidth(); ++j) {
    System.out.print("+---");
}
System.out.println("+" + Color.ansiColorOf("WHITE"));
}

private void printAvailableColumn() {
System.out.print(Color.ansiColorOf("BLUE"));
for(int j=0; j<this.getWidth(); ++j) {
    System.out.print("+-^-");
}
System.out.println("+");
for(int j=0; j<this.getWidth(); ++j) {
    if (this.getNextEmptyCellAt(j).getColor() != Color.EMPTY) {
    System.out.print("| X ");
    }
    else {
    System.out.print("| "
             + Color.ansiColorOf("GREEN")
             + (j+1) + Color.ansiColorOf("BLUE")
             + " ");
    }
}
System.out.println("|");
for(int j=0; j<this.getWidth(); ++j) {
    System.out.print("+---");
}
System.out.println("+" + Color.ansiColorOf("WHITE"));
}

```

```
    public void print() {
    System.out.println();
    this.printGrid();
    System.out.println();
    this.printAvailableColumn();
    System.out.println();

    }

    @Override
    public String toString() {
    StringBuilder s = new StringBuilder();
    Cell c;
    for (int i=0; i<this.getWidth(); i++) {
        c = this.getTopCellAt(i);
        for(int j=0; j<this.getHeight(); j++) {
        s.append(c.toString()+";");
        c = c.getNeighbor(Direction.DOWN);
        }
    }
    return s.toString();
    }

    public void loadGrid(String schema, Token[] tokenOfPlayers) {
    this.initGrid();
    Cell current;
    String[] cells = schema.split(";");
    for (int i=0; i<this.getWidth(); i++) {
        current = this.getTopCellAt(i);
        for (int j=0; j<this.getHeight(); j++) {
        for (int k=0; k<Color.values().length; k++) {
            if (tokenOfPlayers[k].getColor() ==
            Color.colorOf(cells[j+i*this.getHeight()]))
            {
            current.setToken(tokenOfPlayers[k]);
            if (tokenOfPlayers[k] != Token.emptyToken)
                this.UpToNextEmptyCellAt(i);
            break;
            }
        }
        current = current.getNeighbor(Direction.DOWN);
        }
    }
    }
}
```

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.

```

public class Main {
    public static void main(String[] args) {
    Game.run();
    }
}
```

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.

```
public class Player {

    private final String pseudo;
    private final Token token;
    private final Entity gameplay;

    public Player(String pseudo, int indexPlayer, Entity e) {
    this.token = initToken(validePlayer(indexPlayer));
        this.pseudo = pseudo;
    this.gameplay = e;
    }

    private Token initToken(int id) {
    switch (id) {
    case 0: return new Token(Color.RED);
    case 1: return new Token(Color.YELLOW);
    }
    return Token.emptyToken;
    }

    private int validePlayer(int id) {
    if (id < 0 || id >= Game.numberOfPlayers) {
        throw new IllegalArgumentException("indexPlayer must equal 0 if first  player else 1.");
    }
    return id;
    }

    public Entity whatIs() {
    return this.gameplay;
    }

    public Color getColor() {
    return this.token.getColor();
    }

    public Token getToken() {
    return this.token;
    }

    public String getUsername() {
    return this.pseudo;
    }

    @Override
    public String toString(){
    return String.format("%s %s (%s)",
                 this.gameplay.toString(),
                 this.pseudo,
                 this.getColor());
    }
}
```

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.

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors
```

```

public class Save {

    private String filename;

    public Save(String filename) {
    this.filename = filename;
    }

    public void write(boolean verbose, Object src) {
    this.write(src);
    if (verbose) System.out.println("Successfully wrote to the file.");
    }

    public void write(Object src) {
    try {
        FileWriter myWriter = new FileWriter(this.filename);
        myWriter.write(src.toString());
        myWriter.close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
    }

    public String read(boolean verbose) {
    String result = this.read();
    System.out.println("Succesfully read the file.");
    return result;
    }

    public String read() {
    StringBuilder s = new StringBuilder();
    try {

        File myObj = new File(this.filename);
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        // System.out.println(data);
        s.append(data);
        }
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
    return s.toString();
    }

    public void delete(boolean verbose) {
    this.delete();
    if (verbose) System.out.println("Successfully wrote to the file.");
    }

    public void delete() {
    try {
        new FileWriter(filename, false).close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
    }

    public boolean isEmpty() {
    return this.read().equals("");
    }
}
```

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.

```java

public class Token {

    public static final Token emptyToken = new Token(Color.EMPTY);

    private final Color colorToken;

    public Token(Color c) {
    this.colorToken = c;
    }

    public Color getColor(){
        return this.colorToken;
    }

    @Override
    public String toString(){
        return this.colorToken.toString();
    }
}
```

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