

# Tetris Mikoli

Version finale en réseau

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	mikoli::Block Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	Block() . . . . .	6
3.1.3	Member Function Documentation . . . . .	6
3.1.3.1	getPosition() . . . . .	6
3.1.3.2	getQColor() . . . . .	7
3.1.3.3	setPosition() . . . . .	7
3.2	mikoli::Board Class Reference . . . . .	8
3.2.1	Detailed Description . . . . .	9
3.2.2	Constructor & Destructor Documentation . . . . .	9
3.2.2.1	Board() . . . . .	9
3.2.3	Member Function Documentation . . . . .	10
3.2.3.1	addFigure() . . . . .	10
3.2.3.2	areBlocksAvailable() . . . . .	10
3.2.3.3	canGoLower() . . . . .	11
3.2.3.4	canMove() . . . . .	12

3.2.3.5	canPut()	12
3.2.3.6	canRotate()	13
3.2.3.7	checkLines()	13
3.2.3.8	entryPoint()	14
3.2.3.9	getBlocks()	14
3.2.3.10	getBoardSize()	15
3.2.3.11	move()	15
3.2.3.12	removeLine()	15
3.2.3.13	rotate()	16
3.2.3.14	setBlockDown()	16
3.2.3.15	validationHeight()	17
3.2.3.16	validationWidth()	17
3.3	mikoli::Buttons Class Reference	18
3.3.1	Detailed Description	18
3.3.2	Constructor & Destructor Documentation	19
3.3.2.1	Buttons() [1/2]	19
3.3.2.2	Buttons() [2/2]	19
3.3.3	Member Function Documentation	20
3.3.3.1	setVisibility()	20
3.4	mikoli::ChoiceBoard Class Reference	20
3.4.1	Detailed Description	21
3.4.2	Constructor & Destructor Documentation	22
3.4.2.1	ChoiceBoard()	22
3.4.3	Member Function Documentation	23
3.4.3.1	getModeValue()	24
3.5	mikoli::Deserializable Class Reference	24
3.5.1	Detailed Description	24
3.6	mikoli::Figure Class Reference	25
3.6.1	Detailed Description	25
3.6.2	Constructor & Destructor Documentation	25

3.6.2.1	Figure()	25
3.6.3	Member Function Documentation	26
3.6.3.1	getBlocks()	27
3.6.3.2	getPositions()	27
3.6.3.3	getTypeFigure()	27
3.6.3.4	move()	27
3.6.3.5	newPosition()	28
3.6.3.6	rotate()	29
3.7	mikoli::FiguresBag Class Reference	30
3.7.1	Detailed Description	31
3.7.2	Member Function Documentation	31
3.7.2.1	getNextFigure()	31
3.8	mikoli::GameMessage Class Reference	31
3.8.1	Detailed Description	32
3.8.2	Constructor & Destructor Documentation	32
3.8.2.1	GameMessage()	32
3.8.3	Member Function Documentation	33
3.8.3.1	getGameMode()	33
3.8.3.2	getGoalGameMode()	33
3.8.3.3	getLines()	34
3.8.3.4	getTypeMessage()	34
3.8.3.5	serialize_to_json()	34
3.9	mikoli::IpDialog Class Reference	35
3.9.1	Detailed Description	36
3.9.2	Constructor & Destructor Documentation	36
3.9.2.1	IpDialog()	36
3.10	mikoli::Mode Class Reference	37
3.10.1	Detailed Description	37
3.10.2	Member Function Documentation	37
3.10.2.1	getGameMode()	38

3.10.2.2	getGoal()	38
3.11	mikoli::MyServer Class Reference	38
3.11.1	Detailed Description	39
3.11.2	Constructor & Destructor Documentation	39
3.11.2.1	MyServer()	39
3.11.3	Member Function Documentation	40
3.11.3.1	sendToAllTread	40
3.11.3.2	sendToClientOne	40
3.11.3.3	sendToClientTwo	40
3.12	mikoli::MySuperThread Class Reference	41
3.12.1	Detailed Description	42
3.12.2	Member Function Documentation	42
3.12.2.1	calculIP()	42
3.12.2.2	calculPort()	42
3.12.2.3	getIP()	43
3.12.2.4	getPort()	43
3.12.2.5	IPFromGateway()	43
3.12.2.6	IPFromInternet()	44
3.12.2.7	IPFromLoops()	45
3.13	mikoli::MyThreadClient Class Reference	45
3.13.1	Detailed Description	46
3.13.2	Constructor & Destructor Documentation	46
3.13.2.1	MyThreadClient() [1/2]	46
3.13.2.2	MyThreadClient() [2/2]	47
3.13.3	Member Function Documentation	48
3.13.3.1	getIsConnected()	48
3.13.3.2	getSocket()	48
3.13.3.3	sendMessage()	48
3.13.3.4	setIsConnected()	49
3.14	mikoli::MyThreadServer Class Reference	49

3.14.1	Detailed Description . . . . .	50
3.14.2	Constructor & Destructor Documentation . . . . .	50
3.14.2.1	MyThreadServer() . . . . .	50
3.14.3	Member Function Documentation . . . . .	51
3.14.3.1	getSocket() . . . . .	51
3.14.3.2	messageToAllClient . . . . .	51
3.14.3.3	sendToClientOne . . . . .	52
3.14.3.4	sendToClientTwo . . . . .	52
3.14.3.5	setId() . . . . .	53
3.15	mikoli::Observable Class Reference . . . . .	53
3.15.1	Detailed Description . . . . .	54
3.16	mikoli::ObservableWidget Class Reference . . . . .	54
3.16.1	Detailed Description . . . . .	54
3.17	mikoli::Observer Class Reference . . . . .	55
3.17.1	Detailed Description . . . . .	55
3.18	mikoli::ObserverWidget Class Reference . . . . .	55
3.18.1	Detailed Description . . . . .	56
3.19	mikoli::Position Class Reference . . . . .	56
3.19.1	Detailed Description . . . . .	57
3.19.2	Constructor & Destructor Documentation . . . . .	57
3.19.2.1	Position() . . . . .	57
3.19.3	Member Function Documentation . . . . .	57
3.19.3.1	getX() . . . . .	57
3.19.3.2	getY() . . . . .	58
3.19.3.3	isSame() . . . . .	58
3.19.3.4	setX() . . . . .	58
3.19.3.5	setY() . . . . .	59
3.20	mikoli::Score Class Reference . . . . .	59
3.20.1	Detailed Description . . . . .	60
3.20.2	Member Function Documentation . . . . .	60

3.20.2.1	calculScore()	60
3.20.2.2	getLevel()	61
3.20.2.3	getNbLines()	61
3.20.2.4	getScore()	62
3.20.2.5	updateScore()	62
3.21	mikoli::Serializable Class Reference	63
3.21.1	Detailed Description	63
3.22	mikoli::SideBoard Class Reference	63
3.22.1	Detailed Description	64
3.22.2	Constructor & Destructor Documentation	64
3.22.2.1	SideBoard()	64
3.23	mikoli::SoundPlayer Class Reference	65
3.23.1	Detailed Description	66
3.24	mikoli::TetrisException Class Reference	66
3.24.1	Detailed Description	66
3.25	mikoli::TetrisGame Class Reference	66
3.25.1	Detailed Description	68
3.25.2	Member Function Documentation	68
3.25.2.1	checkIfIsBlocked()	69
3.25.2.2	endMove()	69
3.25.2.3	getBoard()	70
3.25.2.4	getCurrentFig()	70
3.25.2.5	getIsFalling()	71
3.25.2.6	getLevel()	71
3.25.2.7	getMode()	71
3.25.2.8	getNbLines()	72
3.25.2.9	getNextFig()	72
3.25.2.10	getScore()	72
3.25.2.11	getShadowCF()	73
3.25.2.12	getSpeed()	73



3.25.2.13	getTimer()	73
3.25.2.14	isBegin()	74
3.25.2.15	isGameOver()	74
3.25.2.16	isPaused()	74
3.25.2.17	isSinglePlayer()	75
3.25.2.18	isWon()	75
3.25.2.19	move()	75
3.25.2.20	rotate()	76
3.25.2.21	setAutoDown()	76
3.25.2.22	setIsBegin()	77
3.25.2.23	setIsFalling()	77
3.25.2.24	upCurrentFigure()	77
3.25.2.25	updateGameFromOpponent()	78
3.26	TimeController Class Reference	79
3.26.1	Detailed Description	79
3.27	mikoli::Timer Class Reference	79
3.27.1	Detailed Description	80
3.27.2	Member Function Documentation	80
3.27.2.1	getHours()	80
3.27.2.2	getMinutes()	81
3.27.2.3	getSeconds()	81
3.27.2.4	getTotalTime()	81
3.27.2.5	statutTimeGame()	82
3.28	Ui_Widget Class Reference	82
3.28.1	Detailed Description	82
3.29	mikoli::ViewBoard Class Reference	83
3.29.1	Detailed Description	83
3.29.2	Constructor & Destructor Documentation	83
3.29.2.1	ViewBoard()	83
3.29.3	Member Function Documentation	84
3.29.3.1	paint()	84
3.30	Ui::Widget Class Reference	85
3.30.1	Detailed Description	85
3.31	Widget Class Reference	85
3.31.1	Detailed Description	86



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

mikoli::Block . . . . .	5
mikoli::Board . . . . .	8
mikoli::Buttons . . . . .	18
mikoli::Deserializable . . . . .	24
mikoli::GameMessage . . . . .	31
exception	
mikoli::TetrisException . . . . .	66
mikoli::Figure . . . . .	25
mikoli::FiguresBag . . . . .	30
mikoli::Mode . . . . .	37
mikoli::Observable . . . . .	53
mikoli::TetrisGame . . . . .	66
mikoli::Timer . . . . .	79
mikoli::ObservableWidget . . . . .	54
mikoli::TetrisGame . . . . .	66
mikoli::Observer . . . . .	55
mikoli::ChoiceBoard . . . . .	20
mikoli::SideBoard . . . . .	63
mikoli::ViewBoard . . . . .	83
TimeController . . . . .	79
mikoli::ObserverWidget . . . . .	55
Widget . . . . .	85
mikoli::Position . . . . .	56
QDialog	
mikoli::IpDialog . . . . .	35
QObject	
mikoli::TetrisGame . . . . .	66
mikoli::Timer . . . . .	79
QTcpServer	
mikoli::MyServer . . . . .	38
QThread	
mikoli::MySuperThread . . . . .	41
mikoli::MyThreadClient . . . . .	45
mikoli::MyThreadServer . . . . .	49

QWidget	
Widget . . . . .	85
mikoli::Score . . . . .	59
mikoli::Serializable . . . . .	63
mikoli::GameMessage . . . . .	31
mikoli::SoundPlayer . . . . .	65
Ui_Widget . . . . .	82
Ui::Widget . . . . .	85

## Chapter 2

# Class Index

### 2.1 Class List

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

<b>mikoli::Block</b>	
The <b>Block</b> (p. 5) class . . . . .	5
<b>mikoli::Board</b>	
The <b>Board</b> (p. 8) class . . . . .	8
<b>mikoli::Buttons</b>	
The <b>Buttons</b> (p. 18) class . . . . .	18
<b>mikoli::ChoiceBoard</b>	20
<b>mikoli::Deserializable</b>	
The <b>Deserializable</b> (p. 24) class . . . . .	24
<b>mikoli::Figure</b>	
The <b>Figure</b> (p. 25) class . . . . .	25
<b>mikoli::FiguresBag</b>	
The <b>FiguresBag</b> (p. 30) class . . . . .	30
<b>mikoli::GameMessage</b>	
The <b>GameMessage</b> (p. 31) class This is the message each player can send to each other. This class is serializable and deserializable in order to be send through the network via TCP . . . .	31
<b>mikoli::IpDialog</b>	
This class is use for creating a QDialog Box for asking host's ip and port to be connected in multiplayer game . . . . .	35
<b>mikoli::Mode</b>	
The <b>Mode</b> (p. 37) class Manage the game mode of the game. By this class, it's possible to switch between different game modes . . . . .	37
<b>mikoli::MyServer</b>	38
<b>mikoli::MySuperThread</b>	
<b>MySuperThread</b> (p. 41) class is a Thread created for the Server . . . . .	41
<b>mikoli::MyThreadClient</b>	45
<b>mikoli::MyThreadServer</b>	
The <b>MyThreadServer</b> (p. 49) class An instance of this class is a thread connected to a specific client. When the server receives a client, it creates an instance of this class to communicate with him . . . . .	49
<b>mikoli::Observable</b>	
The <b>Observable</b> (p. 53) class Interface implemented by the classes that have to be observed .	53
<b>mikoli::ObservableWidget</b>	
The <b>Observable</b> (p. 53) class Interface implemented by the classes that have to be observed .	54

<b>mikoli::Observer</b>	
The <b>Observer</b> (p.55) class Implemented by the class that have to Observe another class(observable) . . . . .	55
<b>mikoli::ObserverWidget</b>	
The <b>Observer</b> (p.55) class Implemented by the class that have to Observe another class(observable) . . . . .	55
<b>mikoli::Position</b>	
The <b>Position</b> (p.56) class . . . . .	56
<b>mikoli::Score</b>	
The <b>Score</b> (p.59) class This class will inform the score of the user and the number of lines he's done. Also used buy the GUI . . . . .	59
<b>mikoli::Serializable</b>	
The <b>Serializable</b> (p.63) class . . . . .	63
<b>mikoli::SideBoard</b>	
The SideBoardd class . . . . .	63
<b>mikoli::SoundPlayer</b>	
The <b>SoundPlayer</b> (p.65) class Each instance of this class is a sound. Through this class, we can handle the sound : play, pause, .. . . . .	65
<b>mikoli::TetrisException</b>	
The <b>TetrisException</b> (p.66) class This is the exception class used for the game . . . . .	66
<b>mikoli::TetrisGame</b>	
The <b>TetrisGame</b> (p.66) class . . . . .	66
<b>TimeController</b> . . . . .	79
<b>mikoli::Timer</b> . . . . .	79
<b>Ui_Widget</b> . . . . .	82
<b>mikoli::ViewBoard</b> . . . . .	83
<b>Ui::Widget</b> . . . . .	85
<b>Widget</b>	
<b>Widget</b> (p.85) class used to display the Game . . . . .	85

## Chapter 3

# Class Documentation

### 3.1 mikoli::Block Class Reference

The **Block** (p. 5) class.

```
#include <block.h>
```

#### Public Member Functions

- **Block** ()  
*Block* (p. 5)'s Constructor without parameters. This constructor will use the default constructor of **Position** (p. 56) for *\_position* and set the color to red.
- **Block** (int x, int y, Color color)  
*Block* (p. 5)'s Constructor with parameters.
- **~Block** ()  
*Block* (p. 5)'s destructor.
- **Position getPosition** () const  
*getPosition*
- **QColor getQColor** ()  
*getColor*
- **Color getColor** ()
- **void setPosition** (int x, int y)  
*setPosition*
- **void move** (Direction direction)

#### 3.1.1 Detailed Description

The **Block** (p. 5) class.

This class will be used for building blocks that are part of a figure. A standard figure is composed with 4 blocks. *\_position* the position of the block in the board. *\_color* the color of the block.

Definition at line 20 of file block.h.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 Block()

```
mikoli::Block::Block (
    int x,
    int y,
    Color color )
```

**Block** (p. 5)'s Constructor with parameters.

##### Parameters

<i>x</i>	The value for the horizontal axis of the block.
<i>y</i>	The value for the vertical axis of the block.
<i>color</i>	The color of the block.

Definition at line 8 of file block.cpp.

```
8 :_position{Position(x, y)}, _color{color}{} 
```

### 3.1.3 Member Function Documentation

#### 3.1.3.1 getPosition()

```
Position mikoli::Block::getPosition ( ) const
```

getPosition

##### Returns

return The position of the block.

Definition at line 14 of file block.cpp.

```
14         {
15     return _position;
16 }
```



### 3.1.3.2 getQColor()

```
QColor mikoli::Block::getQColor ( )
```

getColor

#### Returns

The QColor of the block.

Definition at line 18 of file block.cpp.

```
18         {
19     QColor color;
20     switch(this->_color){
21     case Color::RED:
22         color = Qt::red;
23         break;
24     case Color::GREEN:
25         color = QColor(50,205, 50, 255);
26         break;
27     case Color::ORANGE:
28         color = QColor(238, 154, 0, 255);
29         break;
30     case Color::BLUE:
31         color = QColor(72, 118, 255, 255);
32         break;
33     case Color::PURPLE:
34         color = QColor(89, 51, 204, 255);
35         break;
36     case Color::DEEPPINK:
37         color = QColor(255, 52, 179, 255);
38         break;
39     case Color::CYAN:
40         color = Qt::cyan;
41         break;
42     case Color::YELLOW:
43         color = Qt::yellow;
44         break;
45     case Color::GREY:
46         color = QColor(204,204,204,255);
47         break;
48     }
49     return color;
50 }
51 }
```

### 3.1.3.3 setPosition()

```
void mikoli::Block::setPosition (
    int x,
    int y )
```

setPosition

#### Parameters

<i>x</i>	The new value for the horizontal axis.
<i>y</i>	The new value for the ordinate axis.

Definition at line 58 of file block.cpp.

```

58                                     {
59     _position.setX(x);
60     _position.setY(y);
61 }

```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/block.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/block.cpp

## 3.2 mikoli::Board Class Reference

The **Board** (p. 8) class.

```
#include <board.h>
```

### Public Member Functions

- **Board** ()  
*Board* (p. 8)'s constructor without parameters. This constructor uses the constructor with parameters by setting the height to 20 and the width to 10.
- **Board** (int height, int width)  
*Board* (p. 8)'s constructor with parameters.
- **~Board** ()  
*~Board*'s inline destructor.
- std::list< **Block** > **getBlocks** () const  
*getBlocks*
- void **addFigure** ( **Figure** &cF)  
*addFigure*
- bool **canGoLower** ( **Figure** &cF)  
*canGoLower*
- int **validationHeight** (int nb)  
*validationHeight* This method check if the number received is  $\geq 10$ .
- int **validationWidth** (int nb)  
*validationWidth* This method check if the number received is  $\geq 5$ .
- void **move** ( **Figure** &cF, Direction direction)  
*move* This method uses **canMove()** (p. 11) to check if it's possible to move the figure in this direction, and if so, will actually move the figure by calling her method **move()** (p. 15).
- void **rotate** ( **Figure** &cF, Direction direction)  
*rotate*
- bool **canMove** ( **Figure** cF, Direction direction)  
*canMove* This method check if the current figure can move in the direction wished. To do this, she calculates the new positions of the current figure and then check if theses positions are available in the board. So, she compares theses positions with the positions of the blocks in *\_listBlocks*.
- bool **canRotate** ( **Figure** cF, Direction direction)  
*canRotate* This method check if the current figure can rotate in the direction wished. To do this, she calculates the new positions of the current figure and then check if theses positions are available in the board. So, she compares theses positions with the positions of the blocks in *\_listBlocks*.
- bool **canPut** ( **Figure** cF)  
*canPut* This method check if the position of the *\_entryPoint* is available. To do this, she calculates the positions the current figure need and check if theses positions are available at the *\_entryPoint*.

- bool **areBlocksAvailable** ( **Figure** cF)  
*areBlocksAvailable* This method check is the new positions of the current figure are available in the board.
- int **checkLines** ( **Figure** cF, std::vector< std::vector< int >> &)  
*checkLines* This method check if there are lines in the board to delete. It checks each lines of the board and for each x, check in the `_listBlocks` if a block has this position. It calculates the number of blocks for each line and if there are as much blocks as the width of the board, it adds the y in a List to send to `removeLines()`.
- void **removeLine** (int line)  
*removeLines* This method receives a list of all the y in the board to delete and then, will just delete them.
- void **reorganize** (int line)  
*reorganize* This method will get down all the blocks in the middle of the board after a suppression of lines. She's called after `isFragmented()` in the case she returns true.
- **Position** **entryPoint** ()  
*entryPoint* This method calculate the position of the `_entryPoint`. To do this, it calculates it from the width and the height of the board.
- int **fall** ( **Figure** &cF)  
*fall* This method will get the current figure straight to the bottom of the board. It is called after the player pressed a certain button.
- void **setBlockDown** ( **Block** &bl)  
*setBlockDown*
- void **reset** ()  
*reset* This method cleans the board's list of blocks.
- std::pair< int, int > **getBoardSize** ()  
*getBoardSize*
- void **upLines** (int)
- void **addLines** (std::vector< std::vector< int >>)

### 3.2.1 Detailed Description

The **Board** (p. 8) class.

This class is used to represent a fictive board in which the user will play. Here will stand the blocks and the current figure of the player. A "fictive board" because it's not really a board. It is a list of **Block** (p. 5). This list contains only the blocks placed in the graphical board. There are methods that allow to handle the moves of the current figure, to rotate the current figure and the suppression of lines, the reorganization of the blocks in the "board".

`_listBlocks` The list of the blocks inside the board. `_width` The width of the board. `_height` The height of the board. `_entryPoint` The position where the current figure appears when she arrives in the board.

Definition at line 30 of file `board.h`.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 Board()

```
mikoli::Board::Board (
    int height,
    int width )
```

**Board** (p. 8)'s constructor with parameters.

**Parameters**

<i>height</i>	The height of the board.
<i>width</i>	The width of the board.

Definition at line 9 of file board.cpp.

```

9      : _height{validationHeight(height)}, _width{
      validationWidth(width)}{
10      _entryPoint = entryPoint();
11      _rotationSound = new SoundPlayer("sounds/figRotation.mp3", false);
12      _rotationFailSound = new SoundPlayer("sounds/figRotationFail.mp3", false);
13 }
```

**3.2.3 Member Function Documentation****3.2.3.1 addFigure()**

```

void mikoli::Board::addFigure (
    Figure & cF )
```

addFigure

**Parameters**

<i>cF</i>	The figure to add in the board. Add the current figure to the board's list of blocks.
-----------	---

Definition at line 23 of file board.cpp.

```

23      {
24      for(Block bl: cF.getBlocks()){
25          _listBlocks.push_back(bl);
26      }
27 }
```

**3.2.3.2 areBlocksAvailable()**

```

bool mikoli::Board::areBlocksAvailable (
    Figure cF )
```

areBlocksAvailable This method check is the new positions of the current figure are available in the board.

**Parameters**

<i>cF</i>	The current figure
-----------	--------------------

**Returns**

true if the positions are available, false otherwise.

Definition at line 101 of file board.cpp.

```

101         {
102             std::list<Position> listPosBoard;
103
104             for(Block bl : _listBlocks){
105                 listPosBoard.push_back(bl.getPosition());
106             }
107
108             for(Position posCF : cF.getPositions()){
109                 for(Position posBoard : listPosBoard){
110                     if(posCF.isSame(posBoard)){
111                         return false;
112                     }
113                 }
114                 if(posCF.getX() <= 0 || posCF.getX() > _width){
115                     return false;
116                 }
117                 if(posCF.getY() <= 0 || posCF.getY() > _height){
118                     return false;
119                 }
120             }
121             return true;
122 }

```

**3.2.3.3 canGoLower()**

```

bool mikoli::Board::canGoLower (
    Figure & cF )

```

canGoLower

**Parameters**

<i>cF</i>	The current figure to test.
-----------	-----------------------------

**Returns**

True if the figure can go once down, false otherwise.

Definition at line 30 of file board.cpp.

```

30         {
31             for(Block bl : cF.getBlocks()) {
32                 for(Block blTab : _listBlocks) {
33                     if((bl.getPosition().getX() == blTab.getPosition().getX()) && (bl.getPosition().getY() -1 ==
34                     blTab.getPosition().getY())) {
35                         return false;
36                     }
37                 }
38                 if(bl.getPosition().getY() - 1 <= 0) {
39                     return false;
40                 }
41             }
42             return true;

```

### 3.2.3.4 canMove()

```
bool mikoli::Board::canMove (
    Figure cF,
    Direction direction )
```

canMove This method check if the current figure can move in the direction wished. To do this, she calculates the new positions of the current figure and then check if theses positions are available in the board. So, she compares theses positions with the positions of the blocks in `_listBlocks`.

#### Parameters

<i>cF</i>	The current figure.
<i>direction</i>	The direction we want to move.

#### Returns

true if the positions are available, false otherwise.

Definition at line 82 of file board.cpp.

```
82                                     {
83     cF.move(direction);
84     return areBlocksAvailable(cF);
85 }
```

### 3.2.3.5 canPut()

```
bool mikoli::Board::canPut (
    Figure cF )
```

canPut This method check if the position of the `_entryPoint` is available. To do this, she calculates the positions the current figure need and check if theses positions are available at the `_entryPoint`.

#### Parameters

<i>cF</i>	The figure we want to put in the board.
-----------	---

#### Returns

true if the positions are available, false otherwise.

Definition at line 94 of file board.cpp.

```
94                                     {
95     cF.newPosition(entryPoint());
96     return areBlocksAvailable(cF);
97 }
```

## 3.2.3.6 canRotate()

```
bool mikoli::Board::canRotate (
    Figure cF,
    Direction direction )
```

canRotate This method check if the current figure can rotate in the direction wished. To do this, she calculates the new positions of the current figure and then check if theses positions are available in the board. So, she compares theses positions with the positions of the blocks in `_listBlocks`.

## Parameters

<i>cF</i>	The current figure.
<i>direction</i>	The direction we want to move.

## Returns

true if the positions are available, false otherwise.

Definition at line 88 of file board.cpp.

```
88                                     {
89     cF.rotate(direction);
90     return areBlocksAvailable(cF);
91 }
```

## 3.2.3.7 checkLines()

```
int mikoli::Board::checkLines (
    Figure cF,
    std::vector< std::vector< int >> & lts )
```

checkLines This method check if there are lines in the board to delete. It checks each lines of the board and for each x, check in the `_listBlocks` if a block has this position. It calculates the number of blocks for each line and if there are as much blocks as the width of the board, it adds the y in a List to send to `removeLines()`.

## Returns

The list with all the lines to delete. If there isn't any lines to delete, this list is empty.

Definition at line 125 of file board.cpp.

```
125                                     {
126     std::unordered_map<int, int> lines;
127     int lineToRemove = 0;
128
129     for(Block b1 : _listBlocks){
130         Position posB = b1.getPosition();
131         auto it = lines.find(posB.getY());
132         if(it == lines.end()){
133             lines.insert({posB.getY(), 1});
134         } else {
135             lines.at(posB.getY()) += 1;
136         }
137     }
```

```

138
139     for(auto it = lines.begin(); it != lines.end(); ++it){
140         if(it->second == _width){
141             lineToRemove = it->first;
142             break;
143         }
144     }
145
146     if (lineToRemove == 0) {
147         return lineToRemove;
148     }
149
150     std::vector<int> currentLine;
151     for (Position pos : cF.getPositions()) {
152         if (pos.getY() == lineToRemove) {
153             currentLine.push_back(pos.getX());
154         }
155     }
156     lts.push_back(currentLine);
157
158     return lineToRemove;
159 }

```

### 3.2.3.8 entryPoint()

**Position** mikoli::Board::entryPoint ( )

entryPoint This method calculate the position of the \_entryPoint. To do this, it calculates it from the width and the height of the board.

#### Returns

The position of the \_entryPoint.

Definition at line 182 of file board.cpp.

```

182     {
183         int midWidth = 0;
184         int height = 0;
185
186         if((_width % 2) == 1){
187             midWidth = round(_width/2);
188         } else {
189             midWidth = _width/2;
190         }
191
192         height = _height -1;
193
194         return Position(midWidth, height);
195     }
196 }

```

### 3.2.3.9 getBlocks()

std::list< **Block** > mikoli::Board::getBlocks ( ) const

#### getBlocks

#### Returns

The list of the blocks in the board.

Definition at line 18 of file board.cpp.

```

18     {
19         return _listBlocks;
20     }

```



## 3.2.3.10 getBoardSize()

```
std::pair< int, int > mikoli::Board::getBoardSize ( )
```

getBoardSize

## Returns

A pair with two integers, the width and height of the board.

Definition at line 215 of file board.cpp.

```
215                                     {
216     std::pair<int,int> boardSize;
217     boardSize.first = _width;
218     boardSize.second = _height;
219     return boardSize;
220 }
```

## 3.2.3.11 move()

```
void mikoli::Board::move (
    Figure & cF,
    Direction direction )
```

move This method uses **canMove()** (p. 11) to check if it's possible to move the figure in this direction, and if so, will actually move the figure by calling her method **move()** (p. 15).

If it's not possible to move the figure, the method does nothing.

## Parameters

<i>cF</i>	The current figure.
<i>direction</i>	The direction into move the figure.

Definition at line 61 of file board.cpp.

```
61                                     {
62     if (canMove(cF, direction)){
63         cF.move(direction);
64     }
65 }
```

## 3.2.3.12 removeLine()

```
void mikoli::Board::removeLine (
    int line )
```

removeLines This method receives a list of all the y in the board to delete and then, will just delete them.

## Parameters

<i>line</i>	The line where all the blocks will be removed from the board's list.
-------------	--

Definition at line 162 of file board.cpp.

```

162         {
163     _listBlocks.remove_if([line](Block bl){
164         return bl.getPosition().getY() == line;
165     });
166 }
```

## 3.2.3.13 rotate()

```

void mikoli::Board::rotate (
    Figure & cF,
    Direction direction )
```

rotate

## Parameters

<i>cF</i>	The current figure to rotate
<i>direction</i>	The direction we want to rotate

Definition at line 68 of file board.cpp.

```

68         {
69     if (canRotate(cF, direction)){
70         if (cF.getTypeFigure() != TypeShape::O){
71             _rotationSound->play();
72         } else {
73             _rotationFailSound->play();
74         }
75         cF.rotate(direction);
76     } else {
77         _rotationFailSound->play();
78     }
79 }
```

## 3.2.3.14 setBlockDown()

```

void mikoli::Board::setBlockDown (
    Block & bl )
```

setBlockDown

## Parameters

<i>bl</i>	The block to move This method is used in <b>reorganize()</b> (p. 9). It changes the position of the block to y - 1.
-----------	---

Definition at line 210 of file board.cpp.

```
210                                     {
211     bl.setPosition(bl.getPosition().getX(), bl.getPosition().getY() - 1);
212 }
```

### 3.2.3.15 validationHeight()

```
int mikoli::Board::validationHeight (
    int nb )
```

validationHeight This method check if the number received is  $\geq 10$ .

#### Parameters

<i>nb</i>	The number to validate.
-----------	-------------------------

#### Returns

The number if he's  $\geq 10$ , an exception otherwise.

#### Exceptions

<b><i>TetrisException</i></b> (p. 66)	if $nb < 10$ .
---------------------------------------	----------------

Definition at line 45 of file board.cpp.

```
45                                     {
46     if (nb < 10) {
47         throw TetrisException {"Height invalid for the board, minimum 10."};
48     }
49     return nb;
50 }
```

### 3.2.3.16 validationWidth()

```
int mikoli::Board::validationWidth (
    int nb )
```

validationWidth This method check if the number received is  $\geq 5$ .

#### Parameters

<i>nb</i>	The number to validate.
-----------	-------------------------

### Returns

The number if he's  $\geq 5$ , an exception otherwise.

### Exceptions

<b><i>TetrisException</i></b> (p. 66)	if nb < 5.
---------------------------------------	------------

Definition at line 52 of file board.cpp.

```

52                                     {
53     if (nb < 5) {
54         throw TetrisException {"Parameter invalid for the board, minimum 5."};
55     }
56     return nb;
57 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/board.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/board.cpp

## 3.3 mikoli::Buttons Class Reference

The **Buttons** (p. 18) class.

```
#include <buttons.h>
```

### Public Member Functions

- **Buttons** ()  
***Buttons** (p. 18)'s constructor without parameters. This constructor uses the constructor with parameters by setting the height to 20 and the width to 10.*
- **Buttons** (QWidget &fenetre)  
***Buttons** (p. 18)'s constructor with QWidget as parameter. This constructor is used to create a QPushButton for the "Quick Game".*
- **Buttons** (QWidget &fenetre, bool b)
- **Buttons** (QWidget &fenetre, int x, int y, int width, int height, const QString title)  
***Buttons** (p. 18)'s constructor with parameters. This constructor is used to create a **Buttons** (p. 18) configured.*
- void **setVisibility** (bool visibility)  
*To set the visibility of the QPushButton.*
- QPushButton \* **getButton** ()  
*To get the QPushButton.*

#### 3.3.1 Detailed Description

The **Buttons** (p. 18) class.

This class construct an Object that contains a QPushButton configured

Definition at line 13 of file buttons.h.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 Buttons() [1/2]

```
mikoli::Buttons::Buttons (
    QWidget & fenetre )
```

**Buttons** (p. 18)'s constructor with QWidget as parameter. This constructor is used to create a QPushButton for the "Quick Game".

##### Parameters

<i>fenetre</i>	the <b>Widget</b> (p. 85) in which the Button has to appear
----------------	---

Definition at line 8 of file buttons.cpp.

```
8         {
9         //création du bouton start new game
10        _button = new QPushButton("Quick Game", &fenetre);
11        _button->setGeometry(QRect(QPoint(80, 300), QSize(200, 50)));
12        _button->setFocusPolicy(Qt::NoFocus);
13        _button->setStyleSheet(
14            "background-color: darkRed;"
15            "border: 1px solid black;"
16            "border-radius: 15px;"
17            "color: lightGray;"
18            "font-size: 25px;"
19        );
20        QObject::connect(_button, SIGNAL(clicked()), &fenetre, SLOT(start()));
21    }
```

#### 3.3.2.2 Buttons() [2/2]

```
mikoli::Buttons::Buttons (
    QWidget & fenetre,
    int x,
    int y,
    int width,
    int height,
    const QString title )
```

**Buttons** (p. 18)'s constructor with parameters. This constructor is used to create a **Buttons** (p. 18) configured.

##### Parameters

<i>fenetre</i>	the <b>Widget</b> (p. 85) in which the Button has to appear
<i>x</i>	the x coordonate of the QPushButton
<i>y</i>	the y coordonate of the QPushButton
<i>width</i>	the QPushButton's with
<i>height</i>	the QPushButton's height
<i>title</i>	the QPushButton's label

Definition at line 51 of file buttons.cpp.

```

51                                     {
52
53     _button = new QPushButton(title, &fenetre);
54     _button->setGeometry(QRect(QPoint(x, y), QSize(width, height)));
55     _button->setFocusPolicy(Qt::NoFocus);
56     _button->setStyleSheet(
57         "background-color: darkRed;"
58         "border: 1px solid black;"
59         "border-radius: 15px;"
60         "color: lightGray;"
61         "font-size: 25px;"
62     );
63 }
```

### 3.3.3 Member Function Documentation

#### 3.3.3.1 setVisibility()

```

void mikoli::Buttons::setVisibility (
    bool visibility )
```

To set the visibility of the QPushButton.

##### Parameters

<i>visibility</i>	QPushButton visible or not
-------------------	----------------------------

Definition at line 69 of file buttons.cpp.

```

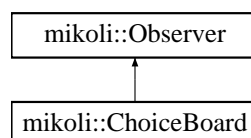
69                                     {
70     _button->setVisible(visibility);
71
72 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/buttons.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/buttons.cpp

## 3.4 mikoli::ChoiceBoard Class Reference

Inheritance diagram for mikoli::ChoiceBoard:



## Public Member Functions

- **ChoiceBoard** ()  
*The constructor of **ChoiceBoard** (p. 20) without parameters.*
- **ChoiceBoard** (QWidget &fenetre, **TetrisGame** \*game)  
*The constructor of **ChoiceBoard** (p. 20) with parameters.*
- Gamemode **getMode** ()  
*To get the GameMode.*
- QPushButton \* **getButtonRestart** ()  
*To get the restart Button.*
- QPushButton \* **getButtonGodMode** ()  
*To get the cheat code Button (GodMode)*
- QPushButton \* **getButtonNewGame** ()  
*To get the NewGame Button.*
- QPushButton \* **getButtonQuickGame** ()  
*To get the QuickGame Button.*
- QPushButton \* **getButtonTwoPlayers** ()  
*To get the 2 players Button.*
- QPushButton \* **getButtonSinglePlayer** ()  
*To get the 1 player Button.*
- QPushButton \* **getButtonHome** ()  
*To get the 1 player Button.*
- QPushButton \* **getButtonQuit** ()  
*To get the cancel multi players Button.*
- QPushButton \* **getButtonServerOn** ()  
*To get the server ON Button.*
- QPushButton \* **getButtonJoin** ()  
*To get the join Button.*
- QPushButton \* **getButtonServerOff** ()  
*To get the server OFF Button.*
- int **getModeValue** (Gamemode mode)  
*To get the Game mode value (which is the goal)*
- void **hide** ()  
*To hide the **ChoiceBoard** (p. 20).*
- void **show** ()  
*To show the **ChoiceBoard** (p. 20) with the **Buttons** (p. 18).*
- void **showChooseMenu** ()  
*To show the **ChoiceBoard** (p. 20) (only the radiobuttons and spin boxes)*
- void **Update** ()  
*The method executed when the observable changed.*
- IpDialog \* **getIpDialog** ()  
*To get the Dialog Box Ip Used for asking the ip and port for connecting to the host in multiplayer game.*

## Additional Inherited Members

### 3.4.1 Detailed Description

Definition at line 20 of file choiceboard.h.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 ChoiceBoard()

```
mikoli::ChoiceBoard::ChoiceBoard (
    QWidget & fenetre,
    TetrisGame * game )
```

The constructor of **ChoiceBoard** (p. 20) with parameters.

##### Parameters

<i>fenetre</i>	the <b>Widget</b> (p. 85) in which the <b>ChoiceBoard</b> (p. 20) has to appear
<i>game</i>	the <b>TetrisGame</b> (p. 66)

Definition at line 7 of file choiceboard.cpp.

```
7
8                                     {
9     _game = game;
10
11     //QPalette blanche utilisée pour l'affichage dans les menus latéraux
12     QPalette* palette = new QPalette();
13     palette->setColor(QPalette::Foreground,Qt::white);
14
15     //Groupbox contenant les "QRadioButton" pour le choix des modes de jeux
16     _groupbox = new QGroupBox("Playing modes",&fenetre);
17     _groupbox->setPalette(*palette);
18
19     //Radios buttons
20     _radio1 = new QRadioButton("Score to Reach");
21     _radio2 = new QRadioButton("Number of Lines");
22     _radio3 = new QRadioButton("Survival time");
23
24     //Affecte la palette au QRadioButtons
25     _radio1->setPalette(*palette);
26     _radio2->setPalette(*palette);
27     _radio3->setPalette(*palette);
28
29     //Connexion des boutons radios à la méthode qui set le mode choisit
30     QObject::connect(_radio1,SIGNAL(clicked()),&fenetre,SLOT(setMode()));
31     QObject::connect(_radio2,SIGNAL(clicked()),&fenetre,SLOT(setMode()));
32     QObject::connect(_radio3,SIGNAL(clicked()),&fenetre,SLOT(setMode()));
33
34     //Spin boxes (choix de la valeur de l'attribut goal)
35     _scoreToReach = new QSpinBox;
36     _scoreToReach->setRange(10000, 100000);
37     _scoreToReach->setSingleStep(1);
38     _scoreToReach->setPrefix("Pts ");
39     _scoreToReach->setValue(5000);
40
41     _nbLines = new QSpinBox;
42     _nbLines->setRange(10, 250);
43     _nbLines->setSingleStep(1);
44     _nbLines->setPrefix("Lines ");
45     _nbLines->setValue(75);
46
47     _timeUp = new QSpinBox;
48     _timeUp->setRange(1,20);
49     _timeUp->setSingleStep(1);
50     _timeUp->setPrefix("Min ");
51     _timeUp->setValue(3);
52
53     //Ajout des QRadioButtons et spinboxes à une VBox
54     QVBoxLayout *vbox = new QVBoxLayout;
55     vbox->addWidget(_radio1);
56     vbox->addWidget(_scoreToReach);
57     vbox->addWidget(_radio2);
58     vbox->addWidget(_nbLines);
```



```

59     vbox->addWidget(_radio3);
60     vbox->addWidget(_timeUp);
61
62     _groupbox->setLayout(vbox);
63     _groupbox->move(QPoint(365,350));
64
65     //Création d'Objets Buttons contenant un bouton configuré
66     Buttons butStart = Buttons(fenetre,526,475,100,50,QString("Start"));
67     Buttons butRestart = Buttons(fenetre,526,475,100,50,QString("Restart"));
68     Buttons butNewGame = Buttons(fenetre,526,357,150,50,QString("New Game"));
69     Buttons butQuickGame = Buttons(fenetre,526,300,150,50,QString("Quick Game"));
70     Buttons butSingle = Buttons(fenetre,95,300,170,50,QString("1 Player"));
71     Buttons butTwoPlayers = Buttons(fenetre,95,365,170,50,QString("2 Players"));
72     Buttons butQuit = Buttons(fenetre,526,415,100,50,QString("Quit"));
73     Buttons butHome = Buttons(fenetre,526,415,100,50,QString("Home"));
74     Buttons butJoin = Buttons(fenetre,526,300,150,50,QString("Join Game"));
75
76     Buttons butServerOn = Buttons(fenetre,true);
77     Buttons butServerOff = Buttons(fenetre,false);
78
79     //Initialisation des boutons
80     _buttonStart = butStart.getButton();
81     _buttonRestart = butRestart.getButton();
82     _buttonNewGame = butNewGame.getButton();
83     _buttonQuickGame = butQuickGame.getButton();
84     _buttonTwoPlayers = butTwoPlayers.getButton();
85     _buttonQuit = butQuit.getButton();
86     _buttonSinglePlayer = butSingle.getButton();
87     _buttonHome = butHome.getButton();
88     _buttonJoin = butJoin.getButton();
89
90     _buttonServerOn = butServerOn.getButton();
91     _buttonServerOff = butServerOff.getButton();
92
93     //Set la visibilité requise des boutons
94     butRestart.setVisibility(false);
95     butNewGame.setVisibility(false);
96     butQuickGame.setVisibility(false);
97     butQuit.setVisibility(false);
98     butRestart.setVisibility(false);
99     butHome.setVisibility(false);
100    butServerOn.setVisibility(false);
101    butJoin.setVisibility(false);
102    hide();
103
104
105    //GodMode
106
107    _buttonGodMode = new QPushButton("", &fenetre);
108    _buttonGodMode->setGeometry(QRect(QPoint(622, 573), QSize(40, 47)));
109    _buttonGodMode->setFocusPolicy(Qt::NoFocus);
110    _buttonGodMode->setStyleSheet(
111        "background-color: transparent;"
112    );
113
114    //Connexion des boutons au slots
115    QObject::connect(_buttonStart,SIGNAL(clicked()),&fenetre,SLOT(startMode()));
116    QObject::connect(_buttonRestart,SIGNAL(clicked()),&fenetre,SLOT(restart()));
117    QObject::connect(_buttonNewGame,SIGNAL(clicked()),&fenetre,SLOT(startNewGame()));
118    QObject::connect(_buttonQuickGame,SIGNAL(clicked()),&fenetre,SLOT(quickGame()));
119    QObject::connect(_buttonTwoPlayers,SIGNAL(clicked()),&fenetre,SLOT(twoPlayers()));
120    QObject::connect(_buttonSinglePlayer,SIGNAL(clicked()),&fenetre,SLOT(singlePlayer()));
121    QObject::connect(_buttonQuit,SIGNAL(clicked()),&fenetre,SLOT(quit()));
122    QObject::connect(_buttonHome,SIGNAL(clicked()),&fenetre,SLOT(home()));
123    QObject::connect(_buttonServerOn,SIGNAL(clicked()),&fenetre,SLOT(serverOn()));
124    QObject::connect(_buttonServerOff,SIGNAL(clicked()),&fenetre,SLOT(serverOff()));
125    QObject::connect(_buttonJoin,SIGNAL(clicked()),&fenetre,SLOT(quickGame()));
126    QObject::connect(_buttonGodMode,SIGNAL(clicked()),&fenetre,SLOT(godMode()));
127
128    _ipDialog = new IpDialog(fenetre);
129
130    // QObject::connect(_buttonTwoPlayers, SIGNAL(clicked()), _ipDialog, SLOT(showIpDialog()));
131
132 }

```

### 3.4.3 Member Function Documentation

### 3.4.3.1 getModeValue()

```
int mikoli::ChoiceBoard::getModeValue (
    Gamemode mode )
```

To get the Game mode value (which is the goal)

#### Parameters

<i>mode</i>	the Gamemode
-------------	--------------

Definition at line 173 of file choiceboard.cpp.

```
173         {
174     if(mode==Gamemode::SCORE) return _scoreToReach->value();
175     if(mode==Gamemode::NBLINES) return _nbLines->value();
176     if(mode==Gamemode::TIME) return _timeUp->value();
177     else return 0;
178 }
```

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

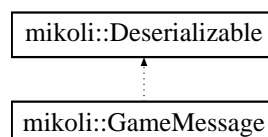
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/choiceboard.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/choiceboard.cpp

## 3.5 mikoli::Deserializable Class Reference

The **Deserializable** (p. 24) class.

```
#include <deserializable.h>
```

Inheritance diagram for mikoli::Deserializable:



### Public Member Functions

- virtual void **deserialize\_from\_json** (const QJsonObject &)=0

### 3.5.1 Detailed Description

The **Deserializable** (p. 24) class.

Definition at line 12 of file deserializable.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/deserializable.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/deserializable.cpp

## 3.6 mikoli::Figure Class Reference

The **Figure** (p. 25) class.

```
#include <figure.h>
```

### Public Member Functions

- **Figure** ()  
*Figures constructor without parameters. This constructor will use the constructor without parameters of **Position** (p. 56) for initializing the attribute `_position` and settings the `_color` attribute to red.*
- **Figure** (TypeShape typeShape)  
*Figures Constructor with parameters.*
- **~Figure** ()  
*Figures destructor.*
- std::vector< **Position** > **getPositions** ()  
*getPositions*
- std::vector< **Block** > **getBlocks** ()  
*getBlocks*
- TypeShape **getTypeFigure** ()  
*getTypeFigure*
- void **setBlocks** (std::vector< **Block** >)
- void **move** (Direction direction)  
*move Makes it possible to move a figure to the left, right or down.*
- void **rotate** (Direction direction)  
*rotate Makes it possible to rotate a figure to the left or to the right.*
- void **newPosition** ( **Position** position)  
*newPosition Makes it possible to displace a figure by modifying the location of its central point.*

### 3.6.1 Detailed Description

The **Figure** (p. 25) class.

This class is used to construct a figure. A figure is composed with 4 blocks and a center point around which the figure can rotate. `_Blocks` An array of 4 blocks . `_axePoint` The point around which the figure can rotate also used to put a figure in the board at a certain position.

Definition at line 21 of file figure.h.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Figure()

```
mikoli::Figure::Figure (
    TypeShape typeShape )
```

Figures Constructor with parameters.

## Parameters

<i>typeShape</i>	The type of the <b>Figure</b> (p.25).
------------------	---------------------------------------

Definition at line 11 of file figure.cpp.

```

11         {
12
13         _typeFigure = typeShape;
14         int x = 0;
15         int y = 0;
16
17         switch (typeShape) {
18         case TypeShape::I: {
19             _Blocks[0] = Block(x, y, Color::CYAN);
20             _Blocks[1] = Block(x-1, y, Color::CYAN);
21             _Blocks[2] = Block(x+1, y, Color::CYAN);
22             _Blocks[3] = Block(x+2, y, Color::CYAN);
23         }break;
24         case TypeShape::O: {
25             _Blocks[0] = Block(x, y, Color::GREEN);
26             _Blocks[1] = Block(x, y-1, Color::GREEN);
27             _Blocks[2] = Block(x+1, y, Color::GREEN);
28             _Blocks[3] = Block(x+1, y-1, Color::GREEN);
29
30         }break;
31         case TypeShape::T: {
32             _Blocks[0] = Block(x, y, Color::YELLOW);
33             _Blocks[1] = Block(x-1, y, Color::YELLOW);
34             _Blocks[2] = Block(x+1, y, Color::YELLOW);
35             _Blocks[3] = Block(x, y-1, Color::YELLOW);
36
37         }break;
38         case TypeShape::J: {
39             _Blocks[0] = Block(x, y, Color::BLUE);
40             _Blocks[1] = Block(x-1, y, Color::BLUE);
41             _Blocks[2] = Block(x+1, y, Color::BLUE);
42             _Blocks[3] = Block(x+1, y-1, Color::BLUE);
43
44         }break;
45         case TypeShape::L: {
46             _Blocks[0] = Block(x, y, Color::RED);
47             _Blocks[1] = Block(x+1, y, Color::RED);
48             _Blocks[2] = Block(x-1, y, Color::RED);
49             _Blocks[3] = Block(x-1, y-1, Color::RED);
50
51         }break;
52         case TypeShape::Z: {
53             _Blocks[0] = Block(x, y, Color::PURPLE);
54             _Blocks[1] = Block(x-1, y, Color::PURPLE);
55             _Blocks[2] = Block(x, y-1, Color::PURPLE);
56             _Blocks[3] = Block(x+1, y-1, Color::PURPLE);
57
58         }break;
59         case TypeShape::S: {
60             _Blocks[0] = Block(x, y, Color::DEEPPINK);
61             _Blocks[1] = Block(x+1, y, Color::DEEPPINK);
62             _Blocks[2] = Block(x, y-1, Color::DEEPPINK);
63             _Blocks[3] = Block(x-1, y-1, Color::DEEPPINK);
64
65         }break;
66         case TypeShape::C: {
67
68         }break;
69
70     }
71 }
```

### 3.6.3 Member Function Documentation

### 3.6.3.1 getBlocks()

```
std::vector< Block > mikoli::Figure::getBlocks ( )
```

getBlocks

#### Returns

return a list of Blocks which forms the figure.

Definition at line 89 of file figure.cpp.

```
89                                     {  
90     std::vector<Block> list;  
91     list.push_back(_Blocks[0]);  
92     list.push_back(_Blocks[1]);  
93     list.push_back(_Blocks[2]);  
94     list.push_back(_Blocks[3]);  
95     return list;  
96 }
```

### 3.6.3.2 getPositions()

```
std::vector< Position > mikoli::Figure::getPositions ( )
```

getPositions

#### Returns

return The list of positions which forms the figure.

Definition at line 75 of file figure.cpp.

```
75                                     {  
76     std::vector<Position> list;  
77     for(Block bl : _Blocks){  
78         list.push_back(bl.getPosition());  
79     }  
80     return list;  
81 }
```

### 3.6.3.3 getTypeFigure()

```
TypeShape mikoli::Figure::getTypeFigure ( )
```

getTypeFigure

#### Returns

The type of the figure

Definition at line 84 of file figure.cpp.

```
84                                     {  
85     return _typeFigure;  
86 }
```

### 3.6.3.4 move()

```
void mikoli::Figure::move (  
    Direction direction )
```

move Makes it possible to move a figure to the left, right or down.

## Parameters

<i>direction</i>	The direction in which the figure should be moved.
------------------	--

Definition at line 100 of file figure.cpp.

```

100                                     {
101     switch(direction){
102     case Direction::LEFT:
103         for(int i = 0; i < 4; i++){
104             _Blocks[i].setPosition(_Blocks[i].getPosition().getX() - 1,
105                                     _Blocks[i].getPosition().getY());
106         }
107         break;
108     case Direction::RIGHT:
109         for(int i = 0; i < 4; i++){
110             _Blocks[i].setPosition(_Blocks[i].getPosition().getX() + 1,
111                                     _Blocks[i].getPosition().getY());
112         }
113         break;
114     case Direction::DOWN:
115         for(int i = 0; i < 4; i++){
116             _Blocks[i].setPosition(_Blocks[i].getPosition().getX(),
117                                     _Blocks[i].getPosition().getY()-1);
118         }
119         break;
120     }
121 }
122 }
```

## 3.6.3.5 newPosition()

```

void mikoli::Figure::newPosition (
    Position position )
```

newPosition Makes it possible to displace a figure by modifying the location of its central point.

## Parameters

<i>position</i>	The new position of the figures's central point.
-----------------	--

Definition at line 162 of file figure.cpp.

```

162                                     {
163     int moveX = position.getX();
164     int moveY = position.getY();
165
166     for(int i = 0; i<4; i++){
167         _Blocks[i].setPosition(_Blocks[i].getPosition().getX() +
168                                 moveX, _Blocks[i].getPosition().getY() + moveY);
169     }
170 }
171 }
```

#### 3.6.3.6 rotate()

```
void mikoli::Figure::rotate (
    Direction direction )
```

rotate Makes it possible to rotate a figure to the left or to the right.

## Parameters

<i>direction</i>	The direction in which the figure should rotate.
------------------	--

Definition at line 125 of file figure.cpp.

```

125                                     {
126     if(direction==Direction::DOWN){
127         throw TetriseException{"Can't turn in this direction"};
128     }
129     if(_typeFigure != TypeShape::O){
130
131         switch(direction){
132             case Direction::LEFT:
133                 for(int i=0;i<4;i++){
134
135                     int x = - (_Blocks[i].getPosition().getY() - _Blocks[0].
getPosition().getY()) + _Blocks[0].getPosition().getX();
136
137                     int y = (_Blocks[i].getPosition().getX() - _Blocks[0].
getPosition().getX()) + _Blocks[0].getPosition().getY();
138
139                     _Blocks[i].setPosition(x,y);
140
141                 }
142                 break;
143             case Direction::RIGHT:
144                 for(int i=0;i<4;i++){
145
146                     int x = (_Blocks[i].getPosition().getY() - _Blocks[0].
getPosition().getY()) + _Blocks[0].getPosition().getX();
147
148                     int y = -(_Blocks[i].getPosition().getX() - _Blocks[0].
getPosition().getX()) + _Blocks[0].getPosition().getY();
149
150                     _Blocks[i].setPosition(x,y);
151
152                 }
153                 break;
154             case Direction::DOWN:
155                 break;
156         }
157     }
158 }
159 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/figure.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/figure.cpp

### 3.7 mikoli::FiguresBag Class Reference

The **FiguresBag** (p. 30) class.

```
#include <figuresbag.h>
```

#### Public Member Functions

- **FiguresBag** ()  
*FiguresBag* (p. 30) constructor without parameters. This constructor will initialize the **FiguresBag** (p. 30) with default parameters.
- **~FiguresBag** ()  
*FiguresBag* (p. 30) destructor. Deallocate the memory that was previously reserved for the **FiguresBag** (p. 30).
- **Figure getNextFigure** ()  
*getType()* Recover the type of the figure
- **void refresh** ()  
*refresh* Reinitialize the **FiguresBag** (p. 30) when the **FiguresBag** (p. 30) is empty.



### 3.7.1 Detailed Description

The **FiguresBag** (p. 30) class.

**\_nextFigure** The **Figure** (p. 25) that will become the current **Figure** (p. 25). **\_listFigures** The list of differents figures that will be played.

Definition at line 18 of file figuresbag.h.

### 3.7.2 Member Function Documentation

#### 3.7.2.1 getNextFigure()

**Figure** mikoli::FiguresBag::getNextFigure ( )

getType() Recover the type of the figure

#### Returns

The next **Figure** (p. 25) that will become the current **Figure** (p. 25).

Definition at line 14 of file figuresbag.cpp.

```

14         {
15     Figure fig = _nextFigure;
16     if(_listFigures.size() == 0){
17         refresh();
18     } else {
19         _nextFigure = _listFigures.back();
20         _listFigures.pop_back();
21     }
22     return fig;
23 }
```

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

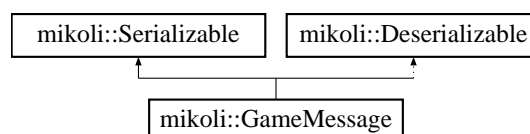
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/figuresbag.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/figuresbag.cpp

## 3.8 mikoli::GameMessage Class Reference

The **GameMessage** (p. 31) class This is the message each player can send to each other. This class is serializable and deserializable in order to be send through the network via TCP.

```
#include <gamemessage.h>
```

Inheritance diagram for mikoli::GameMessage:



## Public Member Functions

- **GameMessage** ()  
*GameMessage* (p. 31) Default's constructor.
- **GameMessage** (TypeMessage)  
*GameMessage* (p. 31) Constructor with only a message.
- **GameMessage** (TypeMessage, std::vector< std::vector< int >>)  
*GameMessage* (p. 31) Constructor with a message and the lines to send to the opponent.
- **GameMessage** (TypeMessage, Gamemode, int)  
*GameMessage* (p. 31) Constructor with a message, a gamemode and the goal for this gamemode.
- **GameMessage** (const **GameMessage** &copie)  
*GameMessage* (p. 31).
- ~**GameMessage** ()  
~GameMessage Default's destructor
- TypeMessage **getTypeMessage** ()  
*getTypeMessage*
- std::vector< std::vector< int > > **getLines** ()  
*getLines*
- Gamemode **getGameMode** ()  
*getGameMode*
- int **getGoalGameMode** ()  
*getGoalGameMode*
- QJsonObject **serialize\_to\_json** ()  
*serialize\_to\_json*
- void **deserialize\_from\_json** (const QJsonObject &)  
*deserialize\_from\_json* This method transform a QJsonObject into an instance of this class.

### 3.8.1 Detailed Description

The **GameMessage** (p. 31) class This is the message each player can send to each other. This class is serializable and deserializable in order to be send through the network via TCP.

Definition at line 25 of file gamemessage.h.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 GameMessage()

```
mikoli::GameMessage::GameMessage (
    const GameMessage & copie )
```

**GameMessage** (p. 31).

#### Parameters

<i>copie</i>	A Gamemessage Constructor by copy.
--------------	------------------------------------

Definition at line 26 of file gamemessage.cpp.

```
26                                     {
27     _message = copie._message;
28     _lines = copie._lines;
29     _gameMode = copie._gameMode;
30     _goalGameMode = copie._goalGameMode;
31 }
```

### 3.8.3 Member Function Documentation

#### 3.8.3.1 getGameMode()

```
Gamemode mikoli::GameMessage::getGameMode ( )
```

getGameMode

##### Returns

The gamemode

Definition at line 45 of file gamemessage.cpp.

```
45                                     {
46     return _gameMode;
47 }
```

#### 3.8.3.2 getGoalGameMode()

```
int mikoli::GameMessage::getGoalGameMode ( )
```

getGoalGameMode

##### Returns

The goal for the gamemode

Definition at line 49 of file gamemessage.cpp.

```
49                                     {
50     return _goalGameMode;
51 }
```

### 3.8.3.3 getLines()

```
std::vector< std::vector< int > > mikoli::GameMessage::getLines ( )
```

getLines

#### Returns

The lines from the opponent to add in the board

Definition at line 41 of file gamemessage.cpp.

```
41                                     {  
42     return _lines;  
43 }
```

### 3.8.3.4 getTypeMessage()

```
TypeMessage mikoli::GameMessage::getTypeMessage ( )
```

getTypeMessage

#### Returns

The message

Definition at line 37 of file gamemessage.cpp.

```
37                                     {  
38     return _message;  
39 }
```

### 3.8.3.5 serialize\_to\_json()

```
JsonObject mikoli::GameMessage::serialize_to_json ( ) [virtual]
```

serialize\_to\_json

**Returns**

An JSON object representation of the message. This method transform an instance of this class into a `QObject`. It's necessary because a `QObject` can be send through the network via TCP.

Implements **mikoli::Serializable** (p. 63).

Definition at line 55 of file gamemessage.cpp.

```

55                                     {
56     QObject obj;
57
58     // s rialisation de l' num ration
59     int enumMsgInt = enumToInt(_message);
60     obj.insert("message", enumMsgInt);
61
62     // s rialisation des lignes
63     QVector v1;
64     for (std::vector<int> l : _lines) {
65         QObject temp;
66         QVector v2;
67         for (int i : l) {
68             QObject hole;
69             hole.insert("h", i);
70             v2.append(hole);
71         }
72         temp.insert("l", v2);
73         v1.append(temp);
74     }
75     obj.insert("lignes", v1);
76
77     // s rialisation du gamemode
78     int enumGmInt = gmToInt(_gameMode);
79     obj.insert("gamemode", enumGmInt);
80
81     // s rialisation du goal du gamemode
82     obj.insert("goal", _goalGameMode);
83
84     return obj;
85 }
```

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

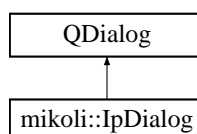
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/gamemessage.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/gamemessage.cpp

## 3.9 mikoli::IpDialog Class Reference

This class is use for creating a QDialog Box for asking host's ip and port to be connected in multiplayer game.

```
#include <ipdialog.h>
```

Inheritance diagram for mikoli::IpDialog:



## Public Member Functions

- **IpDialog** (QWidget &fenetre, QWidget \*parent=0)  
The **IpDialog** (p. 35) constructor.
- QString **getIp** ()  
To get the ip from the Qdialog box.
- QString **getPort** ()  
To get the port from the QDialog box.

### 3.9.1 Detailed Description

This class is use for creating a QDialog Box for asking host's ip and port to be connected in multiplayer game.

Definition at line 17 of file ipdialog.h.

### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 IpDialog()

```
mikoli::IpDialog::IpDialog (
    QWidget & fenetre,
    QWidget * parent = 0 )
```

The **IpDialog** (p. 35) constructor.

#### Parameters

<i>fenetre</i>	the QWidget associated therewith
----------------	----------------------------------

Definition at line 11 of file ipdialog.cpp.

```
12     :QDialog(parent)
13 {
14
15     QGroupBox *connexionGroup = new QGroupBox(tr("Connexion"));
16
17     _confirm = new QPushButton("confirm");
18
19     _lineEditIp = new QLineEdit;
20     _lineEditIp->setPlaceholderText("Host IP");
21     _lineEditIp->setFocus();
22
23     _lineEditPort = new QLineEdit;
24     _lineEditPort->setPlaceholderText("Host Port");
25     _lineEditPort->setFocus();
26
27     QGridLayout *connexionLayout = new QGridLayout;
28     connexionLayout->addWidget(_lineEditIp, 0, 0);
29     connexionLayout->addWidget(_lineEditPort, 0, 1);
30     connexionLayout->addWidget(_confirm, 0, 2);
31
32     connexionGroup->setLayout(connexionLayout);
33
34     QGridLayout *layout = new QGridLayout;
35     layout->addWidget(connexionGroup, 0, 0);
```

```

36
37     setLayout(layout);
38
39
40
41     setWindowTitle(tr("Connexion to Host"));
42
43     QObject::connect(_confirm, SIGNAL(clicked()), &fenetre, SLOT(confirm()));
44
45
46
47 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/ipdialog.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/ipdialog.cpp

## 3.10 mikoli::Mode Class Reference

The **Mode** (p. 37) class Manage the game mode of the game. By this class, it's possible to switch between different game modes.

```
#include <mode.h>
```

### Public Member Functions

- **Mode** ()  
*Mode* (p. 37) *Constructor by default.*
- **Mode** (Gamemode, int)  
*Mode* (p. 37) *Constructor with parameters. Gamemode is the game mode of the game. int is the goal to reach.*
- Gamemode **getGameMode** ()  
*getGameMode*
- int **getGoal** ()  
*getGoal*
- void **setGameMode** (Gamemode)  
*setGameMode* *Set a new game mode to the game.*
- void **setGoal** (int)  
*setGoal* *Set a new goal to reach.*

### 3.10.1 Detailed Description

The **Mode** (p. 37) class Manage the game mode of the game. By this class, it's possible to switch between different game modes.

Definition at line 12 of file mode.h.

### 3.10.2 Member Function Documentation

### 3.10.2.1 getGameMode()

```
Gamemode mikoli::Mode::getGameMode ( )
```

getGameMode

#### Returns

The current game mode of the game.

Definition at line 17 of file mode.cpp.

```
17 {  
18     return _gameMode;  
19 }
```

### 3.10.2.2 getGoal()

```
int mikoli::Mode::getGoal ( )
```

getGoal

#### Returns

The goal to reach.

Definition at line 22 of file mode.cpp.

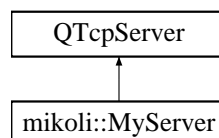
```
22 {  
23     return _goal;  
24 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/mode.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/mode.cpp

## 3.11 mikoli::MyServer Class Reference

Inheritance diagram for mikoli::MyServer:





## Public Slots

- void **receivingFromThread** ( **GameMessage** message, int id)

## Signals

- void **sendToAllTread** ( **GameMessage** message)  
*sendToAllTread send a message to all the client connected*
- void **sendToClientOne** ( **GameMessage** message)  
*sendToClientOne send a message to the client 1*
- void **sendToClientTwo** ( **GameMessage** message)  
*sendToClientTwo send a message to the client 2*

## Public Member Functions

- **MyServer** (ushort, QObject \*parent=0)  
***MyServer** (p. 38) constructor.*
- void **startServer** ()  
*to start the server*
- bool **isWaiting** ()  
*isWaiting to know if the server is waiting for a second player*
- void **setMode** (Gamemode)  
*to set the game mode value of the Server*

## Protected Member Functions

- void **incomingConnection** (qintptr socketDescriptor)

### 3.11.1 Detailed Description

Definition at line 12 of file myserver.h.

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 MyServer()

```
mikoli::MyServer::MyServer (  
    ushort port,  
    QObject * parent = 0 ) [explicit]
```

**MyServer** (p. 38) constructor.

#### Parameters

<i>ushort</i>	the port to create the TcpServerSocket
---------------	--

Definition at line 6 of file myserver.cpp.

```
6                                     : QTcpServer (parent) {  
7     _gameMode=Gamemode::NONE;  
8     _port = port;  
9 }
```

### 3.11.3 Member Function Documentation

#### 3.11.3.1 sendToAllTread

```
void mikoli::MyServer::sendToAllTread (  
    GameMessage message ) [signal]
```

sendToAllTread send a message to all the client connected

#### Parameters

<i>message</i>	the message to send
----------------	---------------------

#### 3.11.3.2 sendToClientOne

```
void mikoli::MyServer::sendToClientOne (  
    GameMessage message ) [signal]
```

sendToClientOne send a message to the client 1

#### Parameters

<i>message</i>	the message to send
----------------	---------------------

#### 3.11.3.3 sendToClientTwo

```
void mikoli::MyServer::sendToClientTwo (  
    GameMessage message ) [signal]
```

sendToClientTwo send a message to the client 2

## Parameters

<i>message</i>	the message to send
----------------	---------------------

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

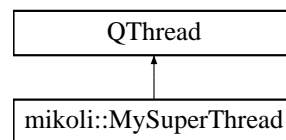
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/myserver.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/myserver.cpp

## 3.12 mikoli::MySuperThread Class Reference

**MySuperThread** (p. 41) class is a Thread created for the Server.

```
#include <mysuperthread.h>
```

Inheritance diagram for mikoli::MySuperThread:



### Public Member Functions

- **MySuperThread** (QObject \*parent=0)  
*MySuperThread* (p. 41) constructor.
- **~MySuperThread** ()  
*MySuperThread* (p. 41) destructor.
- void **run** ()  
*to run the Thread Server*
- **MyServer** \* **getServer** ()  
*to get the Server*
- QString **getIP** ()  
*getIP to get the IP from the server*
- ushort **getPort** ()  
*getPort to get the port from the server*
- QString **calculIP** ()  
*calculIP to find the local ip of the server*
- bool **IPFromInternet** (QString &)  
*return true if the local ip is found*
- bool **IPFromGateway** (QString &)  
*return true if the local ip is found*
- bool **IPFromLoops** (QString &)  
*return true if the local ip is found*
- ushort **calculPort** ()  
*to find the first available port that will be used by the server fot his socket*

### 3.12.1 Detailed Description

**MySuperThread** (p. 41) class is a Thread created for the Server.

Definition at line 16 of file mysuperthread.h.

### 3.12.2 Member Function Documentation

#### 3.12.2.1 calculIP()

```
QString mikoli::MySuperThread::calculIP ( )
```

calculIP to find the local ip of the server

##### Returns

an IP adress

Definition at line 34 of file mysuperthread.cpp.

```
34                                     {
35     QString ip;
36
37     if (IPFromInternet(ip)) return ip;
38     if (IPFromGateway(ip)) return ip;
39     if (IPFromLoops(ip)) return ip;
40 }
```

#### 3.12.2.2 calculPort()

```
ushort mikoli::MySuperThread::calculPort ( )
```

to find the first available port that will be used by the server fot his socket

##### Returns

the first port available

Definition at line 109 of file mysuperthread.cpp.

```
109                                     {
110     QTcpSocket socket;
111     int port = 1024;
112
113     while (port <= 65535) {
114         socket.connectToHost("127.0.0.1", port);
115
116         if (socket.waitForConnected(100)) {
117             socket.close();
118             ++port;
119         } else {
120             socket.close();
121             return static_cast<ushort> (port);
122         }
123     }
124 }
```

### 3.12.2.3 getIP()

```
QString mikoli::MySuperThread::getIP ( )
```

getIP to get the IP from the server

#### Returns

an IP adress

Definition at line 25 of file mysuperthread.cpp.

```
25                                     {  
26     return _ip;  
27 }
```

### 3.12.2.4 getPort()

```
ushort mikoli::MySuperThread::getPort ( )
```

getPort to get the port from the server

#### Returns

a port

Definition at line 29 of file mysuperthread.cpp.

```
29                                     {  
30     return _port;  
31 }
```

### 3.12.2.5 IPFromGateway()

```
bool mikoli::MySuperThread::IPFromGateway (   
        QString & ip )
```

return true if the local ip is found

#### Parameters

<i>QString</i>	this value is initialized vy the method
----------------	---

**Returns**

if the ip is found or not

Definition at line 59 of file mysuperthread.cpp.

```

59                                     {
60     bool found = false;
61
62     QTcpSocket socket;
63     int cpt = 0;
64
65     while (!found && cpt <= 255) {
66         QString buildIP = "192.168." + QString::number(cpt) + ".1";
67         socket.connectToHost(buildIP, 53);
68
69         if (socket.waitForConnected(100)) {
70             found = true;
71             ip = socket.localAddress().toString();
72         }
73
74         ++cpt;
75     }
76
77     socket.close();
78     return found;
79 }
```

**3.12.2.6 IPFromInternet()**

```

bool mikoli::MySuperThread::IPFromInternet (
    QString & ip )
```

return true if the local ip is found

**Parameters**

<i>QString</i>	this value is initialized vy the method
----------------	---

**Returns**

if the ip is found or not

Definition at line 43 of file mysuperthread.cpp.

```

43                                     {
44     bool found = false;
45
46     QTcpSocket socket;
47     socket.connectToHost("8.8.8.8", 53);
48     found = socket.waitForConnected(100);
49
50     if (found) {
51         ip = socket.localAddress().toString();
52     }
53
54     socket.close();
55     return found;
56 }
```

## 3.12.2.7 IPFromLoops()

```
bool mikoli::MySuperThread::IPFromLoops (
    QString & ip )
```

return true if the local ip is found

## Parameters

<i>QString</i>	this value is initialized vy the method
----------------	---

## Returns

if the ip is found or not

Definition at line 82 of file mysuperthread.cpp.

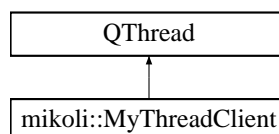
```
82                                     {
83     bool found = false;
84
85     QTcpSocket socket;
86     int cpt1 = 0;
87     int cpt2 = 1;
88
89     while (!found && cpt1 <= 255) {
90         while (!found && cpt2 <= 254) {
91             QString buildIP = "192.168." + QString::number(cpt1) + "." + QString::number(cpt2);
92             socket.connectToHost(buildIP, 53);
93
94             if (socket.waitForConnected(100)) {
95                 found = true;
96                 ip = buildIP;
97             }
98
99             ++cpt2;
100         }
101         ++cpt1;
102     }
103
104     socket.close();
105     return found;
106 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mysuperthread.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mysuperthread.cpp

## 3.13 mikoli::MyThreadClient Class Reference

Inheritance diagram for mikoli::MyThreadClient:



## Public Slots

- void **readyRead** ()  
*readyRead called when a message is received*
- void **disconnected** ()  
*disconnected called by signal emission disconnected*

## Signals

- void **endThread** ()  
*endThread emit when thread close*
- void **infoServerStatus** ()  
*infoServerStatus emit to inform that infoServerStatus changed; server mode can be on or off*

## Public Member Functions

- **MyThreadClient** ( **TetrisGame** \*, **SideBoard** \*, **ChoiceBoard** \*, int, QString, QObject \*parent=0)  
*MyThreadClient* (p. 45).
- **MyThreadClient** ( **TetrisGame** \*, **SideBoard** \*, **ChoiceBoard** \*, Gamemode, int, int, QString, QObject \*parent=0)  
*MyThreadClient* (p. 45).
- void **run** ()  
*run to run the Qthread*
- void **sendMessage** ( **GameMessage** message)  
*sendMessage to send a Game message to the server*
- void **handleQuit** ()  
*handleQuit to handle the end of a multiplayer game when receiving a message QUIT from the server*
- QTcpSocket \* **getSocket** ()  
*getSocket to get the socket*
- void **setIsConnected** (bool b)  
*setIsConnected to set the value isConnected*
- bool **getIsConnected** ()  
*getIsConnected to get the isConnected value*

### 3.13.1 Detailed Description

Definition at line 17 of file mythreadClient.h.

### 3.13.2 Constructor & Destructor Documentation

#### 3.13.2.1 MyThreadClient() [1/2]

```
mikoli::MyThreadClient::MyThreadClient (
    TetrisGame * game,
    SideBoard * sideboard,
    ChoiceBoard * choiceBoard,
    int port,
    QString ip,
    QObject * parent = 0 )
```

**MyThreadClient** (p. 45).



## Parameters

<b><i>TetrisGame</i></b> (p. 66)	the game
<b><i>SideBoard</i></b> (p. 63)	the sidebar
<b><i>ChoiceBoard</i></b> (p. 20)	the choiceboard
<i>int</i>	the port to connect to host
<i>QString</i>	the IP adress to connect to host
<i>parent</i>	

Definition at line 5 of file mythreadClient.cpp.

```

5                                     : QThread(parent) {
6     _socket = new QTcpSocket();
7
8     connect(_socket, SIGNAL(readyRead()), this, SLOT(readyRead()), Qt::DirectConnection);
9     connect(_socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
10
11     _socket->connectToHost(ip, port);
12
13     if(!_socket->waitForConnected(2000)) {
14         sidebar->setMessage(QString("Server unreachable"));
15         sidebar->visibleMessage(true);
16         sidebar->setDisplay();
17         return;
18     }
19
20     _game = game;
21     _sideBoard = sidebar;
22     _choiceBoard = choiceBoard;
23     _isConnected = true;
24
25     start();
26
27 }
```

## 3.13.2.2 MyThreadClient() [2/2]

```

mikoli::MyThreadClient::MyThreadClient (
    TetrisGame * game,
    SideBoard * sidebar,
    ChoiceBoard * choiceBoard,
    Gamemode mode,
    int goal,
    int port,
    QString ip,
    QObject * parent = 0 )
```

**MyThreadClient** (p. 45).

## Parameters

<b><i>TetrisGame</i></b> (p. 66)	the game
<b><i>SideBoard</i></b> (p. 63)	the sidebar
<b><i>ChoiceBoard</i></b> (p. 20)	the choiceboard
<i>GameMode</i>	the game mode
<i>int</i>	the value of the goal to reach to
<i>int</i>	the port to connect to host
<i>QString</i>	the IP adress to connect to host
<i>parent</i>	

Definition at line 28 of file mythreadClient.cpp.

```
28                                     :
29     MyThreadClient (game,sideboard,choiceBoard,port,ip) {
30
31     sendMessage (GameMessage (TypeMessage::PLAYPARAM,mode,goal));
32 }
```

### 3.13.3 Member Function Documentation

#### 3.13.3.1 getIsConnected()

```
bool mikoli::MyThreadClient::getIsConnected ( )
```

getIsConnected to get the isConnected value

##### Returns

isConnected value

Definition at line 113 of file mythreadClient.cpp.

```
113                                     {
114     return _isConnected;
115 }
```

#### 3.13.3.2 getSocket()

```
QTcpSocket * mikoli::MyThreadClient::getSocket ( )
```

getSocket to get the socket

##### Returns

the socket

Definition at line 109 of file mythreadClient.cpp.

```
109                                     {
110     return _socket;
111 }
```

#### 3.13.3.3 sendMessage()

```
void mikoli::MyThreadClient::sendMessage (
    GameMessage message )
```

sendMessage to send a Game message to the server

## Parameters

<i>message</i>	the message to send to the server
----------------	-----------------------------------

Definition at line 81 of file mythreadClient.cpp.

```

81                                     {
82     _socket->write(QJsonDocument(message.serialize_to_json()).toJson());
83     _socket->flush();
84 }
```

## 3.13.3.4 setIsConnected()

```

void mikoli::MyThreadClient::setIsConnected (
    bool b )
```

setIsConnected to set the value isConnected

## Parameters

<i>b</i>	the bool value to set
----------	-----------------------

Definition at line 117 of file mythreadClient.cpp.

```

117                                     {
118     _isConnected = b;
119 }
```

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

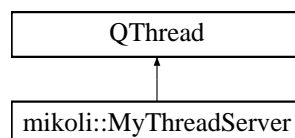
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mythreadClient.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mythreadClient.cpp

## 3.14 mikoli::MyThreadServer Class Reference

The **MyThreadServer** (p. 49) class An instance of this class is a thread connected to a specific client. When the server receives a client, it creates an instance of this class to communicate with him.

```
#include <mythreadserver.h>
```

Inheritance diagram for mikoli::MyThreadServer:



## Public Slots

- void **readyRead** ()  
*readyRead This slot handles a message received on the socket.*
- void **disconnected** ()  
*disconnected This slot handles a deconnexion of the socket.*
- void **messageToAllClient** ( **GameMessage** message)  
*messageToAllClient*
- void **sendToClientOne** ( **GameMessage** message)  
*sendToClientOne*
- void **sendToClientTwo** ( **GameMessage** message)  
*sendToClientTwo*

## Signals

- void **error** (QTcpSocket::SocketError socketerror)
- void **sendToServer** ( **GameMessage** message, int id)

## Public Member Functions

- **MyThreadServer** (qintptr ID, QObject \*parent=0)  
***MyThreadServer** (p. 49).*
- void **setId** (int id)  
*setId*
- void **run** ()  
*run*
- void **closeSocket** ()  
*closeSocket Close correctly the socket of this thread.*
- QTcpSocket \* **getSocket** ()  
*getSocket*

### 3.14.1 Detailed Description

The **MyThreadServer** (p. 49) class An instance of this class is a thread connected to a specific client. When the server receives a client, it creates an instance of this class to communicate with him.

Definition at line 20 of file mythreadserver.h.

### 3.14.2 Constructor & Destructor Documentation

#### 3.14.2.1 MyThreadServer()

```
mikoli::MyThreadServer::MyThreadServer (
    qintptr ID,
    QObject * parent = 0 ) [explicit]
```

**MyThreadServer** (p. 49).

## Parameters

<i>ID</i>	The socket descriptor of the client connected to this thread.
<i>parent</i>	Default's constructor for a thread server.

Definition at line 5 of file mythreadserver.cpp.

```

5                                     :
6     QThread(parent){
7         this->_socketDescriptor = ID;
8         _socket = new QTcpSocket();
9
10        // set the ID
11        if(!_socket->setSocketDescriptor(this->_socketDescriptor)){
12            //Si erreur -> signal
13            emit error(_socket->error());
14            return;
15        }
16
17        connect(_socket, SIGNAL(readyRead()), this, SLOT(readyRead()), Qt::DirectConnection);
18        connect(_socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
19 }
```

### 3.14.3 Member Function Documentation

#### 3.14.3.1 getSocket()

QTcpSocket \* mikoli::MyThreadServer::getSocket ( )

getSocket

## Returns

The socket of this thread.

Definition at line 71 of file mythreadserver.cpp.

```

71                                     {
72         return _socket;
73 }
```

#### 3.14.3.2 messageToAllClient

void mikoli::MyThreadServer::messageToAllClient (   
     **GameMessage** message ) [slot]

messageToAllClient

**Parameters**

<i>message</i>	The message to send to the two clients. This slots send a message to the two clients.
----------------	---

Definition at line 42 of file mythreadserver.cpp.

```
42                                     {
43     if (_id==1) {
44         emit sendToClientTwo(message);
45     }
46     else if (_id==2) {
47         emit sendToClientOne(message);
48     }
49 }
```

**3.14.3.3 sendToClientOne**

```
void mikoli::MyThreadServer::sendToClientOne (
    GameMessage message ) [slot]
```

sendToClientOne

**Parameters**

<i>message</i>	The message to send
----------------	---------------------

Definition at line 52 of file mythreadserver.cpp.

```
52                                     {
53     _socket->write(QJsonDocument(message.serialize_to_json()).toJson());
54     _socket->flush();
55 }
```

**3.14.3.4 sendToClientTwo**

```
void mikoli::MyThreadServer::sendToClientTwo (
    GameMessage message ) [slot]
```

sendToClientTwo

**Parameters**

<i>message</i>	The message to send
----------------	---------------------

Definition at line 58 of file mythreadserver.cpp.

```
58                                     {
```

```

59     _socket->write(QJsonDocument(message.serialize_to_json()).toJson());
60     _socket->flush();
61 }

```

### 3.14.3.5 setId()

```

void mikoli::MyThreadServer::setId (
    int id )

```

setId

#### Parameters

<i>id</i>	1 if this thread is connected to the first client, 2 if it's the second.
-----------	--

Definition at line 63 of file mythreadserver.cpp.

```

63                                     {
64     _id = id;
65 }

```

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

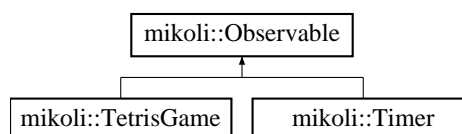
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mythreadserver.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/net/mythreadserver.cpp

## 3.15 mikoli::Observable Class Reference

The **Observable** (p. 53) class Interface implemented by the classes that have to be observed.

```
#include <observable.h>
```

Inheritance diagram for mikoli::Observable:



### Public Member Functions

- void **AddObs** ( **Observer** \*obs)
- void **DelObs** ( **Observer** \*obs)
- void **Notify** (void)

### 3.15.1 Detailed Description

The **Observable** (p. 53) class Interface implemented by the classes that have to be observed.

Definition at line 13 of file observable.h.

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

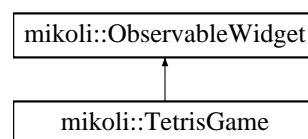
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observable.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observable.cpp

## 3.16 mikoli::ObservableWidget Class Reference

The **Observable** (p. 53) class Interface implemented by the classes that have to be observed.

```
#include <observablewidget.h>
```

Inheritance diagram for mikoli::ObservableWidget:



### Public Member Functions

- void **AddObsWidget** ( **ObserverWidget** \*obs)
- void **DelObsWidget** ( **ObserverWidget** \*obs)
- void **NotifyWidget** ( **GameMessage**)

### 3.16.1 Detailed Description

The **Observable** (p. 53) class Interface implemented by the classes that have to be observed.

Definition at line 14 of file observablewidget.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observablewidget.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observablewidget.cpp

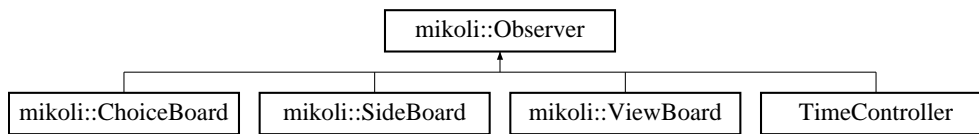


## 3.17 mikoli::Observer Class Reference

The **Observer** (p. 55) class Implemented by the class that have to Observe another class(observable)

```
#include <observer.h>
```

Inheritance diagram for mikoli::Observer:



### Public Member Functions

- virtual void **Update** ()=0
- void **AddObs** ( **Observable** \*obs)
- void **DelObs** ( **Observable** \*obs)

### Protected Types

- typedef std::list< **Observable** \* >::iterator **iterator**
- typedef std::list< **Observable** \* >::const\_iterator **const\_iterator**

### Protected Attributes

- std::list< **Observable** \* > **m\_list**

#### 3.17.1 Detailed Description

The **Observer** (p. 55) class Implemented by the class that have to Observe another class(observable)

Definition at line 22 of file observer.h.

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

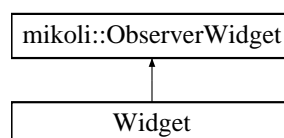
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observer.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observer.cpp

## 3.18 mikoli::ObserverWidget Class Reference

The **Observer** (p. 55) class Implemented by the class that have to Observe another class(observable)

```
#include <observerwidget.h>
```

Inheritance diagram for mikoli::ObserverWidget:



## Public Member Functions

- virtual void **UpdateWidget** ( **GameMessage**)=0
- void **AddObsWidget** ( **ObservableWidget** \*obs)
- void **DelObsWidget** ( **ObservableWidget** \*obs)

## Protected Types

- typedef std::list< **ObservableWidget** \* >::iterator **iterator**
- typedef std::list< **ObservableWidget** \* >::const\_iterator **const\_iterator**

## Protected Attributes

- std::list< **ObservableWidget** \* > **m\_list**

### 3.18.1 Detailed Description

The **Observer** (p. 55) class Implemented by the class that have to Observe another class(observable)

Definition at line 23 of file observerwidget.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observerwidget.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/observer/observerwidget.cpp

## 3.19 mikoli::Position Class Reference

The **Position** (p. 56) class.

```
#include <position.h>
```

## Public Member Functions

- **Position** ()  
***Position** (p. 56)'s constructor without parameters. This constructor will set the \_x attribute and Y to 0.*
- **Position** (int x, int y)  
***Position** (p. 56)'s constructor with 2 parameters.*
- **~Position** ()  
***Position** (p. 56)'s destructor.*
- int **getX** ()  
*getX*
- int **getY** ()  
*getY*
- void **setX** (int x)  
*setX*
- void **setY** (int y)  
*setY*
- bool **isSame** ( **Position** position)  
*isSame*

### 3.19.1 Detailed Description

The **Position** (p. 56) class.

This class will be used to determinate the **Block** (p. 5)'s **Position** (p. 56) in the board. `_x` The abscissa. `_y` The ordinate.

Definition at line 15 of file position.h.

### 3.19.2 Constructor & Destructor Documentation

#### 3.19.2.1 Position()

```
mikoli::Position::Position (
    int x,
    int y )
```

**Position** (p. 56)'s constructor with 2 parameters.

##### Parameters

<code>x</code>	the value for horizontal axis.
<code>y</code>	the value for vertical axis.

Definition at line 9 of file position.cpp.

```
9 :_x{x}, _y{y}{}
```

### 3.19.3 Member Function Documentation

#### 3.19.3.1 getX()

```
int mikoli::Position::getX ( )
```

**getX**

##### Returns

The value of `_x`.

Definition at line 16 of file position.cpp.

```
16 {
17     return _x;
18 }
```

### 3.19.3.2 getY()

```
int mikoli::Position::getY ( )
```

getY

#### Returns

The value of `_y`.

Definition at line 20 of file position.cpp.

```
20 {  
21     return _y;  
22 }
```

### 3.19.3.3 isSame()

```
bool mikoli::Position::isSame (   
    Position position )
```

isSame

#### Parameters

<i>position</i>	The position to compare to.
-----------------	-----------------------------

#### Returns

true if The position is the same, false otherwise.

Definition at line 35 of file position.cpp.

```
35 {  
36     return (_x == position.getX()) && (_y == position.getY());  
37 }
```

### 3.19.3.4 setX()

```
void mikoli::Position::setX (   
    int x )
```

setX

**Parameters**

<i>x</i>	The new value for <i>_x</i> .
----------	-------------------------------

Definition at line 25 of file position.cpp.

```
25                                     {  
26     _x = x;  
27 }
```

**3.19.3.5 setY()**

```
void mikoli::Position::setY (  
    int y )
```

setY

**Parameters**

<i>y</i>	The new value for <i>_y</i> .
----------	-------------------------------

Definition at line 29 of file position.cpp.

```
29                                     {  
30     _y = y;  
31 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/position.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/position.cpp

**3.20 mikoli::Score Class Reference**

The **Score** (p. 59) class This class will inform the score of the user and the number of lines he's done. Also used by the GUI.

```
#include <score.h>
```

## Public Member Functions

- **Score ()**  
*Score* (p. 59)'s constructor without parameters. This only constructor will set the score to 0 and the number of lines to 0 at the start of the game.
- **~Score ()**  
*Score* (p. 59)'s destructor.
- int **getNbLines ()** const  
*getNbLines*
- int **getScore ()** const  
*getScore*
- int **getLevel ()** const  
*getLevel*
- void **updateScore** (int nbL, int nbDrop)  
*updateScore*
- int **calculScore** (int nbL, int nbDrop)  
*calculScore*

### 3.20.1 Detailed Description

The **Score** (p. 59) class This class will inform the score of the user and the number of lines he's done. Also used by the GUI.

Definition at line 16 of file score.h.

### 3.20.2 Member Function Documentation

#### 3.20.2.1 calculScore()

```
int mikoli::Score::calculScore (
    int nbL,
    int nbDrop )
```

calculScore

#### Parameters

<i>nbL</i>	the number of lines the player made at the last move.
<i>nbDrop</i>	the number of lines the player cross during a fall.

#### Returns

the amount to add to the score.

Definition at line 49 of file score.cpp.

```
49                                     {
50     int result = 0;
51
52     switch(nbL) {
53     case 0: result += (40 * nbL) + nbDrop;
54         break;
55     case 1: result += (40 * nbL) + nbDrop;
56         break;
57     case 2 : result += (100 * nbL) + nbDrop;
58         break;
59     case 3: result += (300 * nbL) + nbDrop;
60         break;
61     case 4: result += (1200 * nbL) + nbDrop;
62         break;
63     default:
64         break;
65     }
66
67     return result;
68 }
```

### 3.20.2.2 getLevel()

```
int mikoli::Score::getLevel ( ) const
```

getLevel

#### Returns

The current level

Definition at line 26 of file score.cpp.

```
26                                     {
27     return _level;
28 }
```

### 3.20.2.3 getNbLines()

```
int mikoli::Score::getNbLines ( ) const
```

getNbLines

#### Returns

The number of lines made by the player from the start of the game.

Definition at line 16 of file score.cpp.

```
16                                     {
17     return _nbLines;
18 }
```

### 3.20.2.4 `getScore()`

```
int mikoli::Score::getScore ( ) const
```

`getScore`

#### Returns

The current score

Definition at line 21 of file `score.cpp`.

```
21                                     {
22     return _score;
23 }
```

### 3.20.2.5 `updateScore()`

```
void mikoli::Score::updateScore (
    int nbL,
    int nbDrop )
```

`updateScore`

#### Parameters

<i>the</i>	number of lines the player made at the last move. This number will be added to the previous score.
<i>nbDrop</i>	the number of lines the player cross during a fall.

#### Exceptions

<b><i>TetrisException</i></b> (p. 66)	if nb is negative.
---------------------------------------	--------------------

Definition at line 32 of file `score.cpp`.

```
32                                     {
33     if (nbL < 0) {
34         throw TetrisException {"Number of lines invalid, must be positive."};
35     }
36
37     _oldTen = _nbLines/10;
38     _nbLines += nbL;
39
40     if (( _nbLines/10 != _oldTen)) {
41         _level++;
42         _levelUpSound->play();
43     }
44
45     _score += calculScore(nbL, nbDrop);
46 }
```

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

- `C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/score.h`
- `C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/score.cpp`

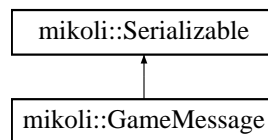


## 3.21 mikoli::Serializable Class Reference

The **Serializable** (p. 63) class.

```
#include <serializable.h>
```

Inheritance diagram for mikoli::Serializable:



### Public Member Functions

- virtual QObject **serialize\_to\_json** ()=0

#### 3.21.1 Detailed Description

The **Serializable** (p. 63) class.

Definition at line 12 of file `serializable.h`.

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

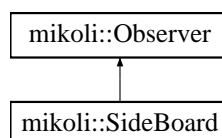
- `C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/serializable.h`
- `C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/serializable.cpp`

## 3.22 mikoli::SideBoard Class Reference

The SideBoard class.

```
#include <sideboard.h>
```

Inheritance diagram for mikoli::SideBoard:



## Public Member Functions

- **SideBoard** ()  
*Constructor of **SideBoard** (p. 63) without parameters.*
- **SideBoard** (QWidget &fenetre, **TetrisGame** \*game)  
*Constructor of **SideBoard** (p. 63) with parameters.*
- void **setDisplay** ()  
*To display the **SideBoard** (p. 63).*
- void **Update** ()  
*The method executed when the observable changed.*
- void **visibleMessage** (bool b)  
*to make visible or not the message label the one placed in the center of the playing board*
- bool **getWaiting** ()
- bool **getUnreachable** ()
- void **setUnreachable** (bool b)  
*to set the bool value used to display an unreachable message*
- void **setMessage** (QString)  
*to set The label message*
- void **setPort** (QString)  
*to set the port label*
- void **setIp** (QString)  
*to set the ip label*

## Public Attributes

- bool **\_isWaiting** = false  
*bool used to display or not a waiting label when multiplayer gaming*
- bool **\_isUnreachable** = false  
*bool used to display or not a "server unreachable" label when multiplayer gaming*
- bool **\_messageVisible**  
*bool used to display or not a message label in front of the playing board*

## Additional Inherited Members

### 3.22.1 Detailed Description

The SideBoard class.

Definition at line 15 of file sidebar.h.

### 3.22.2 Constructor & Destructor Documentation

#### 3.22.2.1 SideBoard()

```
mikoli::SideBoard::SideBoard (
    QWidget & fenetre,
    TetrisGame * game )
```

Constructor of **SideBoard** (p. 63) with parameters.

## Parameters

<i>fenetre</i>	the <b>Widget</b> (p. 85) in which the <b>SideBoard</b> (p. 63) has to appear
<i>game</i>	the <b>TetrisGame</b> (p. 66) (observable)

Definition at line 7 of file sidebar.cpp.

```

7                                     {
8
9     _game = game;
10    _width = this->_game->getBoard().getBoardSize().first;
11    _height = this->_game->getBoard().getBoardSize().second;
12    _ql = new QLabel(&fenetre);
13    setDisplay();
14 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/sidebar.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/sidebar.cpp

## 3.23 mikoli::SoundPlayer Class Reference

The **SoundPlayer** (p. 65) class Each instance of this class is a sound. Through this class, we can handle the sound : play, pause, ...

```
#include <soundplayer.h>
```

### Public Member Functions

- **SoundPlayer** ()  
*SoundPlayer* (p. 65) Constructor by default.
- **SoundPlayer** (std::string, bool)  
*SoundPlayer* (p. 65) Constructor with parameter: A string for sound's name A boolean set to true if the sound must play in loop, false otherwise.
- void **play** ()  
*play* Play the sound.
- void **setVolume** (int)  
*setVolume* Change the volume of the sound.
- void **stop** ()  
*stop* Stop the sound.
- void **switchMute** ()  
*switchMute* Mute the sound.
- void **reset** ()  
*reset* Replace the sound to the beginning.

### 3.23.1 Detailed Description

The **SoundPlayer** (p. 65) class Each instance of this class is a sound. Through this class, we can handle the sound : play, pause, ...

Definition at line 16 of file soundplayer.h.

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

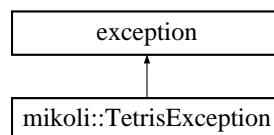
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/sounds/soundplayer.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/sounds/soundplayer.cpp

## 3.24 mikoli::TetrisException Class Reference

The **TetrisException** (p. 66) class This is the exception class used for the game .

```
#include <tetrisexception.h>
```

Inheritance diagram for mikoli::TetrisException:



### Public Member Functions

- **TetrisException** (const char \*Msg)
- const char \* **what** () const throw ()

### 3.24.1 Detailed Description

The **TetrisException** (p. 66) class This is the exception class used for the game .

Definition at line 14 of file tetrisexception.h.

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

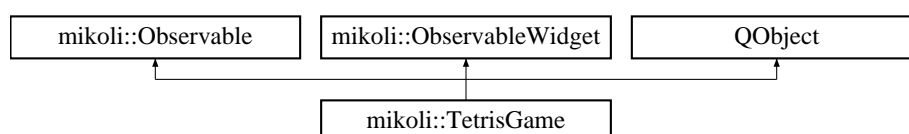
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/tetrisexception.h

## 3.25 mikoli::TetrisGame Class Reference

The **TetrisGame** (p. 66) class.

```
#include <tetrisgame.h>
```

Inheritance diagram for mikoli::TetrisGame:



## Public Member Functions

- **TetrisGame ()**  
*TetrisGame* (p. 66)'s constructor without parameter. Build a standard game with level mode "normal" and difficulty "normal".
- **TetrisGame** (int width, int height)  
*TetrisGame* (p. 66)'s constructor with size parameters. Build a standard game with level mode "normal" and difficulty "normal".
- **~TetrisGame ()**  
*TetrisGame* (p. 66) destructor. Deallocate the memory that was previously reserved for the Game.
- **Figure** **getCurrentFig ()**  
*getCurrentFig*
- **std::vector< Block >** **getShadowCF ()** **const**  
*getShadowCF*
- **Figure** **getNextFig ()**  
*getNextFig*
- **Board** **getBoard ()**  
*getBoard*
- **Board &** **getBoard2 ()**
- **bool** **isGameOver ()**  
*isGameOver*
- **bool** **isWon ()**  
*isWon*
- **int** **getScore ()**  
*getScore*
- **int** **getLevel ()**  
*getLevel*
- **int** **getNbLines ()**  
*getNbLines*
- **Mode** **getMode ()**  
*getMode*
- **bool** **isBegin ()**  
*isBegin*
- **bool** **isPaused ()**  
*isPaused*
- **int** **getSpeed ()**  
*statutSpeed*
- **bool** **getIsFalling ()**  
*statutIsFalling*
- **bool** **isSinglePlayer ()**  
*isSinglePlayer*
- **bool** **checkIfIsBlocked ()**  
*checkIfIsBlocked*
- **Timer \*** **getTimer ()**  
*getTimer*
- **void** **isWin ()**  
*isWin* Check if the conditions of win are completed.
- **void** **move** (Direction direction)  
*move*
- **void** **rotate** (Direction direction)  
*rotate*

- void **fall** ()  
*fall*
- void **fallSlow** ()  
*fallSlow*
- void **endMove** (int nbDrop)  
*endMove*
- void **endGame** ()  
*endGame* Stop the "tetris" music and play the "game over" sound, replace all the blocks from the board with grey blocks.
- void **calculateShadow** ()  
*calculateShadow* Calcul the positions of the shadow according to the positions of the current figure.
- void **switchPause** ()  
*switchPause* Switch the game into paused / not paused mode.
- void **setIsBegin** (bool **isBegin**)  
*setIsBegin*
- void **setIsFalling** (bool isFalling)  
*setIsFalling*
- void **setAutoDown** (bool autoDown)  
*setAutoDown*
- void **start** ()  
*start()* (p. 68) To start the game initialized. Activates the timer and make te "first current figure" moving down.
- void **startWithMode** (Gamemode, int, bool)  
*startWithMode* Do the same as **start()** (p. 68) but change the game mode and the goal.
- void **restart** ()  
*restart* Restart the game: reset the score, level, number of lines, board, figure's bag, reset the sounds, reset the attributes, ...
- void **upCurrentFigure** (int nb)  
*upCurrentFigure*
- void **updateGameFromOpponent** (std::vector< std::vector< int >> lines)  
*updateGameFromOpponent*
- void **endGameFromOpponent** ( **GameMessage**)  
*endGameFromOpponent* End the game according to the opponent status. If the opponent has won, this method calls the "game over" view. Otherwise, it calls the "win" view.

### 3.25.1 Detailed Description

The **TetrisGame** (p. 66) class.

This class will be used for build a new game.

**\_figuresBag** The list in which are each figure that will become the next figure and then the current **Figure** (p. 25).  
**\_currentFigure** The figure that can be rotated or moved during its descent. **\_nextFigure**

Definition at line 32 of file tetrisgame.h.

### 3.25.2 Member Function Documentation

## 3.25.2.1 checkIfIsBlocked()

```
bool mikoli::TetrisGame::checkIfIsBlocked ( )
```

checkIfIsBlocked

## Returns

True if the current figure can go lower, false otherwise.

Definition at line 331 of file tetrisgame.cpp.

```
331 {
332     return !_board.canGoLower(_currentFigure);
333 }
```

## 3.25.2.2 endMove()

```
void mikoli::TetrisGame::endMove (
    int nbDrop )
```

endMove

## Parameters

<i>nbDrop</i>	The number of lines crossed by a fall. Handle the end of a move: Add the current figure to the board, check and remove lines if necessary, update the score, change the current figure with the next one, check if the next current figure can be placed in the board, check the conditions of win, ..., call <b>endGame()</b> (p. 68) is necessary.
---------------	--

Definition at line 200 of file tetrisgame.cpp.

```
200 {
201     _fallSound->play();
202     _board.addFigure(_currentFigure);
203
204     std::vector<std::vector<int>> linesToSend;
205
206     int line = _board.checkLines(_currentFigure, linesToSend);
207     int nbLines = 0;
208
209     while(line != 0){
210         nbLines++;
211         _board.removeLine(line);
212         _board.reorganize(line);
213         line = _board.checkLines(_currentFigure, linesToSend);
214     }
215
216     if (linesToSend.size()>=2 && !_isSinglePlayer) {
217         NotifyWidget(GameMessage(TypeMessage::PARAM, linesToSend));
218     }
219
220     switch(nbLines){
221     case 1: _deleteOneLineSound->play();
222             break;
223     case 2: _deleteTwoLineSound->play();
224             break;
225     case 3: _deleteThreeLineSound->play();
226             break;
```

```

227     case 4: _deleteFourLineSound->play();
228         break;
229     }
230     _currentScore.updateScore(nbLines, nbDrop);
231     _currentFigure = _nextFigure;
232     _isFalling = false;
233     _nextFigure = _figuresBag.getNextFigure();
234     _currentFigure.newPosition(_board.entryPoint());
235     calculateShadow();
236     if(!_board.areBlocksAvailable(_currentFigure)){
237         _isGameOver = true;
238         if (!_isSinglePlayer) {NotifyWidget(GameMessage(TypeMessage::LOOSE));}
239         endGame();
240     }
241
242     isWin();
243     Notify();
244 }

```

### 3.25.2.3 getBoard()

**Board** mikoli::TetrisGame::getBoard ( )

getBoard

Returns

The board.

Definition at line 65 of file tetrisgame.cpp.

```

65     {
66     return _board;
67 }

```

### 3.25.2.4 getCurrentFig()

**Figure** mikoli::TetrisGame::getCurrentFig ( )

getCurrentFig

Returns

The current figure

Definition at line 46 of file tetrisgame.cpp.

```

46     {
47     return _currentFigure;
48 }

```



#### 3.25.2.5 getIsFalling()

```
bool mikoli::TetrisGame::getIsFalling ( )
```

statutIsFalling

##### Returns

True if the current figure is falling.

Definition at line 119 of file tetrisgame.cpp.

```
119                                     {  
120     return _isFalling;  
121 }
```

#### 3.25.2.6 getLevel()

```
int mikoli::TetrisGame::getLevel ( )
```

getLevel

##### Returns

The current level.

Definition at line 89 of file tetrisgame.cpp.

```
89                                     {  
90     return _currentScore.getLevel();  
91 }
```

#### 3.25.2.7 getMode()

```
Mode mikoli::TetrisGame::getMode ( )
```

getMode

##### Returns

The mode of the game.

Definition at line 96 of file tetrisgame.cpp.

```
96                                     {  
97     return _mode;  
98 }
```

### 3.25.2.8 getNbLines()

```
int mikoli::TetrisGame::getNbLines ( )
```

getNbLines

#### Returns

The current number of lines made.

Definition at line 92 of file tetrisgame.cpp.

```
92         {  
93     return _currentScore.getNbLines();  
94 }
```

### 3.25.2.9 getNextFig()

**Figure** mikoli::TetrisGame::getNextFig ( )

getNextFig

#### Returns

The next figure.

Definition at line 60 of file tetrisgame.cpp.

```
60         {  
61     return _nextFigure;  
62 }
```

### 3.25.2.10 getScore()

```
int mikoli::TetrisGame::getScore ( )
```

getScore

#### Returns

The current score.

Definition at line 86 of file tetrisgame.cpp.

```
86         {  
87     return _currentScore.getScore();  
88 }
```

#### 3.25.2.11 getShadowCF()

```
std::vector< Block > mikoli::TetrisGame::getShadowCF ( ) const
```

getShadowCF

##### Returns

A vector with the blocks of the shadow.

Definition at line 51 of file tetrisgame.cpp.

```
51                                     {
52     std::vector<Block> blocks;
53     for(int i=0; i<4; ++i) {
54         blocks.push_back(_shadowCF[i]);
55     }
56     return blocks;
57 }
```

#### 3.25.2.12 getSpeed()

```
int mikoli::TetrisGame::getSpeed ( )
```

statutSpeed

##### Returns

The actual speed of the game. It's a calcul made according of the level.

Definition at line 109 of file tetrisgame.cpp.

```
109                                     {
110     int speed;
111     if(_currentScore.getLevel() < 15){
112         speed = 1000 - (_currentScore.getLevel() * 62.5);
113     } else {
114         return 110;
115     }
116     return speed;
117 }
```

#### 3.25.2.13 getTimer()

```
Timer * mikoli::TetrisGame::getTimer ( )
```

getTimer

##### Returns

The timer instance with the elapsed time.

Definition at line 82 of file tetrisgame.cpp.

```
82                                     {
83     return _timerGame;
84 }
```

#### 3.25.2.14 isBegin()

```
bool mikoli::TetrisGame::isBegin ( )
```

isBegin

##### Returns

True if the game is began, false otherwise.

Definition at line 100 of file tetrisgame.cpp.

```
100 {  
101     return _isBegin;  
102 }
```

#### 3.25.2.15 isGameOver()

```
bool mikoli::TetrisGame::isGameOver ( )
```

isGameOver

##### Returns

True is the game is over, false otherwise.

Definition at line 74 of file tetrisgame.cpp.

```
74 {  
75     return _isGameOver;  
76 }
```

#### 3.25.2.16 isPaused()

```
bool mikoli::TetrisGame::isPaused ( )
```

isPaused

##### Returns

True if the game is paused, false otherwise.

Definition at line 103 of file tetrisgame.cpp.

```
103 {  
104     return _isPaused;  
105 }
```

### 3.25.2.17 isSinglePlayer()

```
bool mikoli::TetrisGame::isSinglePlayer ( )
```

isSinglePlayer

#### Returns

True if it's a single player game, false otherwise.

Definition at line 123 of file tetrisgame.cpp.

```
123 {  
124     return _isSinglePlayer;  
125 }
```

### 3.25.2.18 isWon()

```
bool mikoli::TetrisGame::isWon ( )
```

isWon

#### Returns

True if the player reached it's goal, false otherwise.

Definition at line 78 of file tetrisgame.cpp.

```
78 {  
79     return _isWon;  
80 }
```

### 3.25.2.19 move()

```
void mikoli::TetrisGame::move (  
    Direction direction )
```

move

#### Parameters

<i>direction</i>	The direction we want to move. Move the current figure in the direction "direction".
------------------	--

Definition at line 128 of file tetrisgame.cpp.

```

128                                     {
129     if(!_isPaused || !_isGameOver){
130         return;
131     }
132     if(!_isFalling){
133         direction=Direction::DOWN;
134     }
135     _board.move(_currentFigure, direction);
136     calculateShadow();
137     Notify();
138     if(!_board.canGoLower(_currentFigure) && _autoDown) {
139         endMove(0);
140     }
141     _autoDown = false;
142     isWin();
143 }

```

### 3.25.2.20 rotate()

```

void mikoli::TetrisGame::rotate (
    Direction direction )

```

rotate

#### Parameters

<i>direction</i>	The direction we want to rotate. Rotate the current figure in the direction "direction".
------------------	--

Definition at line 146 of file tetrisgame.cpp.

```

146                                     {
147     if(!_isPaused || !_isGameOver || !_isFalling){
148         return;
149     }
150     _board.rotate(_currentFigure, direction);
151     calculateShadow();
152     Notify();
153 }

```

### 3.25.2.21 setAutoDown()

```

void mikoli::TetrisGame::setAutoDown (
    bool autoDown )

```

setAutoDown

#### Parameters

<i>autoDown</i>	Set the attribute <code>_autoDown</code> with the parameter.
-----------------	--

Definition at line 289 of file tetrisgame.cpp.

```

289                                     {

```

```
290     _autoDown = autoDown;
291 }
```

#### 3.25.2.22 setIsBegin()

```
void mikoli::TetrisGame::setIsBegin (
    bool isBegin )
```

setIsBegin

##### Parameters

<i>isBegin</i>	Set the attribute <code>_isBegin</code> with the parameter.
----------------	---

Definition at line 281 of file `tetrisgame.cpp`.

```
281                                     {
282     _isBegin = isBegin;
283 }
```

#### 3.25.2.23 setIsFalling()

```
void mikoli::TetrisGame::setIsFalling (
    bool isFalling )
```

setIsFalling

##### Parameters

<i>isFalling</i>	Set the attribute <code>_isFalling</code> with the parameter.
------------------	---

Definition at line 285 of file `tetrisgame.cpp`.

```
285                                     {
286     _isFalling = isFalling;
287 }
```

#### 3.25.2.24 upCurrentFigure()

```
void mikoli::TetrisGame::upCurrentFigure (
    int nb )
```

upCurrentFigure

## Parameters

<i>nb</i>	Number of lines Move the current figure to nb lines higher.
-----------	---

Definition at line 336 of file tetrisgame.cpp.

```

336                                     {
337     int maxHeightCF = 0;
338
339     for(Block bl : _currentFigure.getBlocks()) {
340         if (bl.getPosition().getY() > maxHeightCF) {
341             maxHeightCF = bl.getPosition().getY();
342         }
343     }
344
345     if ((maxHeightCF + nbLines) < _board.getBoardSize().second) {
346         std::vector<Block> newBlocksCF;
347         for(Block bl : _currentFigure.getBlocks()) {
348             newBlocksCF.emplace_back(bl.getPosition().getX(), bl.getPosition().getY() + nbLines, bl.
getColor());
349         }
350         _currentFigure.setBlocks(newBlocksCF);
351     } else {
352         int maxPossible = _board.getBoardSize().second - maxHeightCF;
353         std::vector<Block> newBlocksCF;
354         for(Block bl : _currentFigure.getBlocks()) {
355             newBlocksCF.emplace_back(bl.getPosition().getX(), bl.getPosition().getY() + maxPossible, bl.
getColor());
356         }
357         _currentFigure.setBlocks(newBlocksCF);
358     }
359     calculateShadow();
360 }

```

### 3.25.2.25 updateGameFromOpponent()

```

void mikoli::TetrisGame::updateGameFromOpponent (
    std::vector< std::vector< int >> lines )

```

updateGameFromOpponent

## Parameters

<i>lines</i>	Each vector represents a line, each int represents an abscisse where there is a hole in this line. Add lines in the board. Theses lines come from the opponent.
--------------	---

Definition at line 363 of file tetrisgame.cpp.

```

363                                     {
364     upCurrentFigure(lines.size()-1);
365     _board.addLines(lines);
366     Notify();
367 }

```

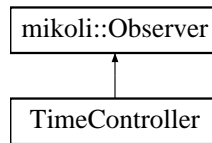
The documentation for this class was generated from the following files:

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/tetrisgame.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/tetrisgame.cpp



## 3.26 TimeController Class Reference

Inheritance diagram for TimeController:



### Public Member Functions

- **TimeController** ( **Widget** &w)
- void **Update** ()

### Friends

- class **Widget**

### Additional Inherited Members

#### 3.26.1 Detailed Description

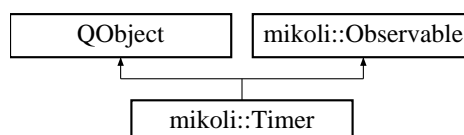
Definition at line 104 of file widget.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/widget.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/widget.cpp

## 3.27 mikoli::Timer Class Reference

Inheritance diagram for mikoli::Timer:



### Public Slots

- void **MySlot** ()  
*MySlot Inform the view to update the time elapsed view.*

## Public Member Functions

- **Timer** ()  
*Timer* (p. 79) Constructor by default.
- std::pair< int, int > **statutTimeGame** (void)  
*statutTimeGame*
- int **getSeconds** ()  
*getSeconds*
- int **getMinutes** ()  
*getMinutes*
- int **getHours** ()  
*getHours*
- int **getTotalTime** ()  
*getTotalTime*
- void **play** ()  
*play* Start the timer.
- void **pause** ()  
*pause* Pause the timer.
- void **updateDuration** ()  
*updateDuration* Update the duration attribute
- void **reset** ()  
*reset* Reset the timer.

## Public Attributes

- QTimer \* **\_timer**  
*\_timer* **Timer** (p. 79) that every second, inform the game to update it's view of the time elapsed.

### 3.27.1 Detailed Description

Definition at line 12 of file timer.h.

### 3.27.2 Member Function Documentation

#### 3.27.2.1 getHours()

```
int mikoli::Timer::getHours ( )
```

getHours

#### Returns

The number of hours elapsed.

Definition at line 60 of file timer.cpp.

```
60     {
61     updateDuration();
62     return std::chrono::duration_cast<std::chrono::hours>(_duration).count() % 24;
63 }
```

### 3.27.2.2 getMinutes()

```
int mikoli::Timer::getMinutes ( )
```

getMinutes

#### Returns

The number of minutes elapsed.

Definition at line 55 of file timer.cpp.

```
55         {
56     updateDuration();
57     return std::chrono::duration_cast<std::chrono::minutes>(_duration).count()%60;
58 }
```

### 3.27.2.3 getSeconds()

```
int mikoli::Timer::getSeconds ( )
```

getSeconds

#### Returns

The number of seconds elapsed.

Definition at line 50 of file timer.cpp.

```
50         {
51     updateDuration();
52     return std::chrono::duration_cast<std::chrono::seconds>(_duration).count()%60;
53 }
```

### 3.27.2.4 getTotalTime()

```
int mikoli::Timer::getTotalTime ( )
```

getTotalTime

#### Returns

The total time elapsed in seconds.

Definition at line 66 of file timer.cpp.

```
66         {
67     return std::chrono::duration_cast<std::chrono::seconds>(_duration).count();
68 }
```

### 3.27.2.5 statutTimeGame()

```
std::pair< int, int > mikoli::Timer::statutTimeGame (
    void )
```

statutTimeGame

#### Returns

A pair with the minutes and seconds elapsed since the start of the game.

Definition at line 23 of file timer.cpp.

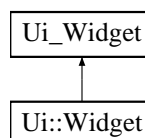
```
23                                     {
24     std::pair<int, int> temp;
25     temp.first = this->getMinutes();
26     temp.second = this->getSeconds();
27     return temp;
28 }
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/timer.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/model/timer.cpp

## 3.28 Ui\_Widget Class Reference

Inheritance diagram for Ui\_Widget:



### Public Member Functions

- void **setupUi** (QWidget \* **Widget**)
- void **retranslateUi** (QWidget \* **Widget**)

### 3.28.1 Detailed Description

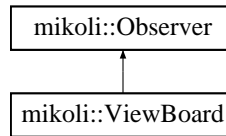
Definition at line 21 of file ui\_widget.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/ui\_widget.h

## 3.29 mikoli::ViewBoard Class Reference

Inheritance diagram for mikoli::ViewBoard:



### Public Member Functions

- **ViewBoard** ()  
The constructor of **ViewBoard** (p. 83) without parameter.
- **ViewBoard** (QWidget &fenetre, **TetrisGame** \*game)  
The constructor of **ViewBoard** (p. 83) with parameters.
- void **setDisplay** ()  
The method called to display the board.
- void **paint** ( **Block** bl, int blSize, QColor color, int a, int b, int c, int d, double opacity, bool grad)  
Method to paint blocks with parameters With a relief effect.
- void **Update** ()  
The method executed when the observable changed.

### Additional Inherited Members

#### 3.29.1 Detailed Description

Definition at line 9 of file viewboard.h.

#### 3.29.2 Constructor & Destructor Documentation

##### 3.29.2.1 ViewBoard()

```

mikoli::ViewBoard::ViewBoard (
    QWidget & fenetre,
    TetrisGame * game )
  
```

The constructor of **ViewBoard** (p. 83) with parameters.

##### Parameters

<i>fenetre</i>	the <b>Widget</b> (p. 85) in which the Viewboard has to appear
<i>game</i>	The <b>TetrisGame</b> (p. 66) (observed)

Definition at line 12 of file viewboard.cpp.

```

12         {
13
14         int width = 340;
15         int height = 640;
16
17         _game = game;
18         _width = game->getBoard().getBoardSize().first;
19         _height = game->getBoard().getBoardSize().second;
20         _ql = new QLabel(&fenetre);
21         _pixmap=QPixmap(width,height);
22         _pixmap.fill(QColor("transparent"));
23
24         setDisplay();
25
26     }
```

### 3.29.3 Member Function Documentation

#### 3.29.3.1 paint()

```

void mikoli::ViewBoard::paint (
    Block bl,
    int blSize,
    QColor color,
    int a,
    int b,
    int c,
    int d,
    double opacity,
    bool grad )
```

Method to paint blocks with parameters With a relief effect.

#### Parameters

<i>bl</i>	the block to paint
<i>blSize</i>	the block's width
<i>color</i>	the block's color
<i>a</i>	value used to print the first level of the block painting (position)
<i>b</i>	value used to print the first level of the block painting (width)
<i>c</i>	value used to print the second level of the block painting (position)
<i>d</i>	value used to print the second level of the block painting (width)
<i>opacity</i>	the opacity of the block painting

Definition at line 86 of file viewboard.cpp.

```

86         {
87
88         QPainter painter(&_pixmap);
89         painter.setPen(Qt::black);
90         painter.setBrush(color);
91         painter.setOpacity(opacity);
92         QRect myQRect= QRect((bl.getPosition().getX()*blSize)+a,
```

```

93         ((_height+1)*blSize-bl.getPosition().getY()*blSize)+b,blSize-c,blSize-d);
94     if (grad){
95         QLinearGradient gradient(myQRect.topLeft(), myQRect.bottomRight());
96         gradient.setColorAt(0, QColor(245, 184, 184,255));
97         gradient.setColorAt(0.5, color);
98         painter.fillRect(myQRect, gradient);
99     }else painter.drawRect(myQRect);
100 }

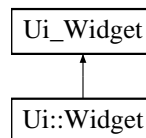
```

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/viewboard.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/view/viewboard.cpp

## 3.30 Ui::Widget Class Reference

Inheritance diagram for Ui::Widget:



### Additional Inherited Members

#### 3.30.1 Detailed Description

Definition at line 44 of file ui\_widget.h.

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

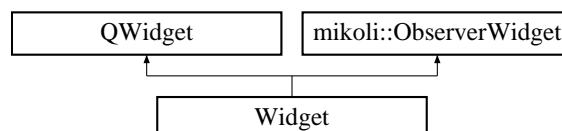
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/ui\_widget.h

## 3.31 Widget Class Reference

**Widget** (p. 85) class used to display the Game.

```
#include <widget.h>
```

Inheritance diagram for Widget:



## Public Member Functions

- **Widget** (QWidget \*parent=0)
- **TetrisGame** \* **getGame** ()
- void **UpdateWidget** ( **GameMessage**)
- void **closeEvent** (QCloseEvent \*event)
- void **homeWithOutGameReinit** ()
- void **sendLinesGodMode** ()
- void **setLinesToSend** (int)

## Public Attributes

- **SoundPlayer** \* **\_startSound**
- **SoundPlayer** \* **\_moveSound**

## Protected Member Functions

- void **timerEvent** (QTimerEvent \*event) override

## Friends

- class **TimeController**

## Additional Inherited Members

### 3.31.1 Detailed Description

**Widget** (p. 85) class used to display the Game.

Definition at line 35 of file widget.h.

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

- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/widget.h
- C:/Users/Olivier/Desktop/Mikoli\_T\_Final/Tetris\_Mikoli/widget.cpp



# Index

- addFigure
  - mikoli::Board, 10
- areBlocksAvailable
  - mikoli::Board, 10
- Block
  - mikoli::Block, 6
- Board
  - mikoli::Board, 9
- Buttons
  - mikoli::Buttons, 19
- calculIP
  - mikoli::MySuperThread, 42
- calculPort
  - mikoli::MySuperThread, 42
- calculScore
  - mikoli::Score, 60
- canGoLower
  - mikoli::Board, 11
- canMove
  - mikoli::Board, 11
- canPut
  - mikoli::Board, 12
- canRotate
  - mikoli::Board, 12
- checkIfIsBlocked
  - mikoli::TetrisGame, 68
- checkLines
  - mikoli::Board, 13
- ChoiceBoard
  - mikoli::ChoiceBoard, 22
- endMove
  - mikoli::TetrisGame, 69
- entryPoint
  - mikoli::Board, 14
- Figure
  - mikoli::Figure, 25
- GameMessage
  - mikoli::GameMessage, 32
- getBlocks
  - mikoli::Board, 14
  - mikoli::Figure, 26
- getBoard
  - mikoli::TetrisGame, 70
- getBoardSize
  - mikoli::Board, 14
- getCurrentFig
  - mikoli::TetrisGame, 70
- getGameMode
  - mikoli::GameMessage, 33
  - mikoli::Mode, 37
- getGoal
  - mikoli::Mode, 38
- getGoalGameMode
  - mikoli::GameMessage, 33
- getHours
  - mikoli::Timer, 80
- getIP
  - mikoli::MySuperThread, 42
- getIsConnected
  - mikoli::MyThreadClient, 48
- getIsFalling
  - mikoli::TetrisGame, 70
- getLevel
  - mikoli::Score, 61
  - mikoli::TetrisGame, 71
- getLines
  - mikoli::GameMessage, 33
- getMinutes
  - mikoli::Timer, 80
- getMode
  - mikoli::TetrisGame, 71
- getModeValue
  - mikoli::ChoiceBoard, 23
- getNbLines
  - mikoli::Score, 61
  - mikoli::TetrisGame, 71
- getNextFig
  - mikoli::TetrisGame, 72
- getNextFigure
  - mikoli::FiguresBag, 31
- getPort
  - mikoli::MySuperThread, 43
- getPosition
  - mikoli::Block, 6
- getPositions
  - mikoli::Figure, 27
- getQColor
  - mikoli::Block, 6
- getScore
  - mikoli::Score, 61
  - mikoli::TetrisGame, 72
- getSeconds
  - mikoli::Timer, 81
- getShadowCF
  - mikoli::TetrisGame, 72

- getSocket
  - mikoli::MyThreadClient, 48
  - mikoli::MyThreadServer, 51
- getSpeed
  - mikoli::TetrisGame, 73
- getTimer
  - mikoli::TetrisGame, 73
- getTotalTime
  - mikoli::Timer, 81
- getTypeFigure
  - mikoli::Figure, 27
- getTypeMessage
  - mikoli::GameMessage, 34
- getX
  - mikoli::Position, 57
- getY
  - mikoli::Position, 57
- IPFromGateway
  - mikoli::MySuperThread, 43
- IPFromInternet
  - mikoli::MySuperThread, 44
- IPFromLoops
  - mikoli::MySuperThread, 44
- IpDialog
  - mikoli::IpDialog, 36
- isBegin
  - mikoli::TetrisGame, 73
- isGameOver
  - mikoli::TetrisGame, 74
- isPaused
  - mikoli::TetrisGame, 74
- isSame
  - mikoli::Position, 58
- isSinglePlayer
  - mikoli::TetrisGame, 74
- isWon
  - mikoli::TetrisGame, 75
- messageToAllClient
  - mikoli::MyThreadServer, 51
- mikoli::Block, 5
  - Block, 6
  - getPosition, 6
  - getQColor, 6
  - setPosition, 7
- mikoli::Board, 8
  - addFigure, 10
  - areBlocksAvailable, 10
  - Board, 9
  - canGoLower, 11
  - canMove, 11
  - canPut, 12
  - canRotate, 12
  - checkLines, 13
  - entryPoint, 14
  - getBlocks, 14
  - getBoardSize, 14
  - move, 15
  - removeLine, 15
  - rotate, 16
  - setBlockDown, 16
  - validationHeight, 17
  - validationWidth, 17
- mikoli::Buttons, 18
  - Buttons, 19
  - setVisibility, 20
- mikoli::ChoiceBoard, 20
  - ChoiceBoard, 22
  - getModeValue, 23
- mikoli::Deserializable, 24
- mikoli::Figure, 25
  - Figure, 25
  - getBlocks, 26
  - getPositions, 27
  - getTypeFigure, 27
  - move, 27
  - newPosition, 28
  - rotate, 28
- mikoli::FiguresBag, 30
  - getNextFigure, 31
- mikoli::GameMessage, 31
  - GameMessage, 32
  - getGameMode, 33
  - getGoalGameMode, 33
  - getLines, 33
  - getTypeMessage, 34
  - serialize\_to\_json, 34
- mikoli::IpDialog, 35
  - IpDialog, 36
- mikoli::Mode, 37
  - getGameMode, 37
  - getGoal, 38
- mikoli::MyServer, 38
  - MyServer, 39
  - sendToAllTread, 40
  - sendToClientOne, 40
  - sendToClientTwo, 40
- mikoli::MySuperThread, 41
  - calculIP, 42
  - calculPort, 42
  - getIP, 42
  - getPort, 43
  - IPFromGateway, 43
  - IPFromInternet, 44
  - IPFromLoops, 44
- mikoli::MyThreadClient, 45
  - getIsConnected, 48
  - getSocket, 48
  - MyThreadClient, 46, 47
  - sendMessage, 48
  - setIsConnected, 49
- mikoli::MyThreadServer, 49
  - getSocket, 51
  - messageToAllClient, 51
  - MyThreadServer, 50
  - sendToClientOne, 52

- sendToClientTwo, 52
- setId, 53
- mikoli::Observable, 53
- mikoli::ObservableWidget, 54
- mikoli::Observer, 55
- mikoli::ObserverWidget, 55
- mikoli::Position, 56
  - getX, 57
  - getY, 57
  - isSame, 58
  - Position, 57
  - setX, 58
  - setY, 59
- mikoli::Score, 59
  - calculScore, 60
  - getLevel, 61
  - getNbLines, 61
  - getScore, 61
  - updateScore, 62
- mikoli::Serializable, 63
- mikoli::SideBoard, 63
  - SideBoard, 64
- mikoli::SoundPlayer, 65
- mikoli::TetrisException, 66
- mikoli::TetrisGame, 66
  - checkIfIsBlocked, 68
  - endMove, 69
  - getBoard, 70
  - getCurrentFig, 70
  - getIsFalling, 70
  - getLevel, 71
  - getMode, 71
  - getNbLines, 71
  - getNextFig, 72
  - getScore, 72
  - getShadowCF, 72
  - getSpeed, 73
  - getTimer, 73
  - isBegin, 73
  - isGameOver, 74
  - isPaused, 74
  - isSinglePlayer, 74
  - isWon, 75
  - move, 75
  - rotate, 76
  - setAutoDown, 76
  - setIsBegin, 77
  - setIsFalling, 77
  - upCurrentFigure, 77
  - updateGameFromOpponent, 78
- mikoli::Timer, 79
  - getHours, 80
  - getMinutes, 80
  - getSeconds, 81
  - getTotalTime, 81
  - statutTimeGame, 81
- mikoli::ViewBoard, 83
  - paint, 84
  - ViewBoard, 83
- move
  - mikoli::Board, 15
  - mikoli::Figure, 27
  - mikoli::TetrisGame, 75
- MyServer
  - mikoli::MyServer, 39
- MyThreadClient
  - mikoli::MyThreadClient, 46, 47
- MyThreadServer
  - mikoli::MyThreadServer, 50
- newPosition
  - mikoli::Figure, 28
- paint
  - mikoli::ViewBoard, 84
- Position
  - mikoli::Position, 57
- removeLine
  - mikoli::Board, 15
- rotate
  - mikoli::Board, 16
  - mikoli::Figure, 28
  - mikoli::TetrisGame, 76
- sendMessage
  - mikoli::MyThreadClient, 48
- sendToAllTread
  - mikoli::MyServer, 40
- sendToClientOne
  - mikoli::MyServer, 40
  - mikoli::MyThreadServer, 52
- sendToClientTwo
  - mikoli::MyServer, 40
  - mikoli::MyThreadServer, 52
- serialize\_to\_json
  - mikoli::GameMessage, 34
- setAutoDown
  - mikoli::TetrisGame, 76
- setBlockDown
  - mikoli::Board, 16
- setId
  - mikoli::MyThreadServer, 53
- setIsBegin
  - mikoli::TetrisGame, 77
- setIsConnected
  - mikoli::MyThreadClient, 49
- setIsFalling
  - mikoli::TetrisGame, 77
- setPosition
  - mikoli::Block, 7
- setVisibility
  - mikoli::Buttons, 20
- setX
  - mikoli::Position, 58
- setY
  - mikoli::Position, 59

SideBoard  
    mikoli::SideBoard, 64  
statutTimeGame  
    mikoli::Timer, 81  
  
TimeController, 79  
  
Ui::Widget, 85  
Ui\_Widget, 82  
upCurrentFigure  
    mikoli::TetrisGame, 77  
updateGameFromOpponent  
    mikoli::TetrisGame, 78  
updateScore  
    mikoli::Score, 62  
  
validationHeight  
    mikoli::Board, 17  
validationWidth  
    mikoli::Board, 17  
ViewBoard  
    mikoli::ViewBoard, 83  
  
Widget, 85