Tetris Mikoli

Version finale en réseau

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 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

x	The value for the horizontal axis of the block.
y	The value for the vertical axis of the block.
color	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.

```
                         {
    QColor color;
    switch(this->_color){
    case Color::RED:
        color = Qt::red;
        break;
    case Color::GREEN:
        color = QColor(50,205, 50, 255);
        break;
    case Color::ORANGE:
        color = QColor(238, 154, 0, 255);
        break;
    case Color::BLUE:
        color = QColor(72, 118, 255, 255);
        break;
    case Color::PURPLE:
        color =  QColor(89, 51, 204, 255);
        break;
    case Color::DEEPPINK:
        color = QColor(255, 52, 179, 255);
        break;
    case Color::CYAN:
        color = Qt::cyan;
        break;
    case Color::YELLOW:
        color = Qt::yellow;
        break;
    case Color::GREY:
        color = QColor(204,204,204,255);
        break;

    }
    return color;
}
```

3.1.3.3 setPosition()

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

setPosition

Parameters

x	The new value for the horizontal axis.
y	The new value for the ordinate axis.

Definition at line 58 of file block.cpp.

```
                                {
    _position.setX(x);
    _position.setY(y);
}
```

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 >= 10.
- int **validationWidth** (int nb)

  validationWidth This method check if the number received is >= 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

height	The height of the board.
width	The width of the board.

Definition at line 9 of file board.cpp.

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

3.2.3 Member Function Documentation

3.2.3.1 addFigure()

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

addFigure

Parameters

cF	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.

```
                                   {
     for(Block bl: cF.getBlocks()){
         _listBlocks.push_back(bl);
     }
 }
```

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

cF	The current figure

Returns

true if the positions are available, false otherwise.

Definition at line 101 of file board.cpp.

```
                                        {
    std::list<Position> listPosBoard;

    for(Block bl : _listBlocks){
        listPosBoard.push_back(bl.getPosition());
    }

    for(Position posCF : cF.getPositions()){
        for(Position posBoard : listPosBoard){
            if(posCF.isSame(posBoard)){
                return false;
            }
        }
        if(posCF.getX() <= 0 || posCF.getX() > _width){
            return false;
        }
        if(posCF.getY() <= 0 || posCF.getY() > _height){
            return false;
        }
    }
    return true;
}
```

3.2.3.3 canGoLower()

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

canGoLower

Parameters

cF	The current figure to test.

Returns

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

Definition at line 30 of file board.cpp.

```
                                    {
    for(Block bl : cF.getBlocks()) {
        for(Block blTab : _listBlocks) {
            if((bl.getPosition().getX() == blTab.getPosition().getX()) && (bl.getPosition().getY() -1 ==
    blTab.getPosition().getY())) {
                return false;
            }
        }
        if(bl.getPosition().getY() - 1 <= 0) {
            return false;
        }
    }
    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

cF	The current figure.
direction	The direction we want to move.

Returns

true if the positions are available, false otherwise.

Definition at line 82 of file board.cpp.

```
                                          {
    cF.move(direction);
    return areBlocksAvailable(cF);
}
```

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

cF	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.

```
                       {
    cF.newPosition(entryPoint());
    return areBlocksAvailable(cF);
}
```

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

cF	The current figure.
direction	The direction we want to move.

Returns

true if the positions are available, false otherwise.

Definition at line 88 of file board.cpp.

```
                                                  {
    cF.rotate(direction);
    return areBlocksAvailable(cF);
}
```

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.

```
                                                              {
    std::unordered_map<int, int> lines;
    int lineToRemove = 0;

    for(Block bl : _listBlocks){
        Position posB = bl.getPosition();
        auto it = lines.find(posB.getY());
        if(it == lines.end()){
            lines.insert({posB.getY(), 1});
        } else {
            lines.at(posB.getY()) += 1;
        }
    }

    for(auto it = lines.begin(); it != lines.end(); ++it){
        if(it->second == _width){
            lineToRemove = it->first;
            break;
        }
    }

    if (lineToRemove == 0) {
        return lineToRemove;
    }

    std::vector<int> currentLine;
    for (Position pos : cF.getPositions()) {
        if (pos.getY() == lineToRemove) {
            currentLine.push_back(pos.getX());
        }
    }
    lts.push_back(currentLine);

    return lineToRemove;
}
```

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.

```
                          {
    int midWidth = 0;
    int height = 0;

    if((_width % 2) == 1){
        midWidth = round(_width/2);
    } else {
        midWidth = _width/2;
    }

    height = _height -1;

    return Position(midWidth, height);

}
```

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.

```
                                     {
    return _listBlocks;
}
```

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.

```
                                    {
    std::pair<int,int> boardSize;
    boardSize.first = _width;
    boardSize.second = _height;
    return boardSize;
}
```

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 itsn't possible to move the figure, the method does nothing.

Parameters

cF	The current figure.
direction	The direction into move the figure.

Definition at line 61 of file board.cpp.

```
                                              {
    if(canMove(cF, direction)){
        cF.move(direction);
    }
}
```

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

line	The line where all the blocks will be removed from the board's list.

Definition at line 162 of file board.cpp.

```
                             {
    _listBlocks.remove_if([line](Block bl){
        return bl.getPosition().getY() == line;
    });
}
```

3.2.3.13 rotate()

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

rotate

Parameters

cF	The current figure to rotate
direction	The direction we want to rotate

Definition at line 68 of file board.cpp.

```
                                                  {
    if(canRotate(cF, direction)){
        if(cF.getTypeFigure() != TypeShape::O){
            _rotationSound->play();
        } else {
            _rotationFailSound->play();
        }
        cF.rotate(direction);
    } else {
        _rotationFailSound->play();
    }
}
```

3.2.3.14 setBlockDown()

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

setBlockDown

Parameters

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

Definition at line 210 of file board.cpp.

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

3.2.3.15 validationHeight()

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

validationHeight This method check if the number received is >= 10.

Parameters

nb	The number to validate.

Returns

The number if he's >= 10, an exception otherwise.

Exceptions

TetrisException *(p. 66)*	if nb < 10.

Definition at line 45 of file board.cpp.

```
                                 {
    if(nb < 10){
        throw TetrisException {"Height invalid for the board, minimum 10."};
    }
    return nb;
}
```

3.2.3.16 validationWidth()

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

validationWidth This method check if the number received is >= 5.

Parameters

nb	The number to validate.

Returns

The number if he's >= 5, an exception otherwise.

Exceptions

TetrisException *(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 QPushButtons 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

fenetre	the **Widget** (p. 85) in which the Button has to appear

Definition at line 8 of file buttons.cpp.

```
                              {
    //création du boutton start new game
     _button = new QPushButton("Quick Game", &fenetre);
     _button->setGeometry(QRect(QPoint(80, 300), QSize(200, 50)));
     _button->setFocusPolicy(Qt::NoFocus);
     _button->setStyleSheet(
                 "background-color: darkRed;"
                 "border: 1px solid black;"
                 "border-radius: 15px;"
                 "color: lightGray; "
                 "font-size: 25px;"
                 );
     QObject::connect(_button,SIGNAL(clicked()),&fenetre,SLOT(start()));
}
```

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

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

Definition at line 51 of file buttons.cpp.

```
                                                                                {

    _button = new QPushButton(title, &fenetre);
    _button->setGeometry(QRect(QPoint(x, y), QSize(width, height)));
    _button->setFocusPolicy(Qt::NoFocus);
    _button->setStyleSheet(
                "background-color: darkRed;"
                "border: 1px solid black;"
                "border-radius: 15px;"
                "color: lightGray; "
                "font-size: 25px;"
                );
}
```

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

visibility	QPushButton visible or not

Definition at line 69 of file buttons.cpp.

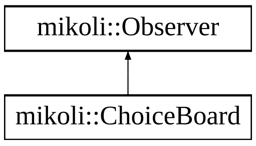
```
                                          {
    _button->setVisible(visibility);

}
```

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

fenetre	the **Widget** (p. 85) in which the **ChoiceBoard** (p. 20) has to appear
game	the **TetrisGame** (p. 66)

Definition at line 7 of file choiceboard.cpp.

```
                                                            {

     _game = game;

      //QPalette blanche utilisée pour l'affichage dans les menus lattéraux
      QPalette* palette = new QPalette();
      palette->setColor(QPalette::Foreground,Qt::white);

      //Groupbox contenant les "QradioButtons" pour le choix des modes de jeux
      _groupbox = new QGroupBox("Playing modes",&fenetre);
      _groupbox->setPalette(*palette);

      //Radios buttons
      _radio1 = new QRadioButton("Score to Reach");
      _radio2 = new QRadioButton("Number of Lines");
      _radio3 = new QRadioButton("Survival time");

      //Affecte la palette au QRadioButtons
      _radio1->setPalette(*palette);
      _radio2->setPalette(*palette);
      _radio3->setPalette(*palette);

      //Connexion des boutons radios à la méthode qui set le mode choisit
      QObject::connect(_radio1,SIGNAL(clicked()),&fenetre,SLOT(setMode()));
      QObject::connect(_radio2,SIGNAL(clicked()),&fenetre,SLOT(setMode()));
      QObject::connect(_radio3,SIGNAL(clicked()),&fenetre,SLOT(setMode()));

      //Spin boxes (choix de la valeur de l'attribut goal)
      _scoreToReach = new QSpinBox;
      _scoreToReach->setRange(10000, 100000);
      _scoreToReach->setSingleStep(1);
      _scoreToReach->setPrefix("Pts ");
      _scoreToReach->setValue(5000);

      _nbLines = new QSpinBox;
      _nbLines->setRange(10, 250);
      _nbLines->setSingleStep(1);
      _nbLines->setPrefix("Lines ");
      _nbLines->setValue(75);

      _timeUp = new QSpinBox;
      _timeUp->setRange(1,20);
      _timeUp->setSingleStep(1);
      _timeUp->setPrefix("Min ");
      _timeUp->setValue(3);

      //Ajout des QRadioButtons et spinboxes à une VBox
      QVBoxLayout *vbox = new QVBoxLayout;
      vbox->addWidget(_radio1);
      vbox->addWidget(_scoreToReach);
      vbox->addWidget(_radio2);
      vbox->addWidget(_nbLines);
```

```
    vbox->addWidget(_radio3);
    vbox->addWidget(_timeUp);

    _groupbox->setLayout(vbox);
    _groupbox->move(QPoint(365,350));

    //Création d'Objets Buttons contenant un bouton configuré
    Buttons butStart = Buttons(fenetre,526,475,100,50,QString("Start"));
    Buttons butRestart = Buttons(fenetre,526,475,100,50,QString("Restart"));
    Buttons butNewGame = Buttons(fenetre,526,357,150,50,QString("New Game"));
    Buttons butQuickGame = Buttons(fenetre,526,300,150,50,QString("Quick Game"));
    Buttons butSingle = Buttons(fenetre,95,300,170,50,QString("1 Player"));
    Buttons butTwoPlayers = Buttons(fenetre,95,365,170,50,QString("2 Players"));
    Buttons butQuit = Buttons(fenetre,526,415,100,50,QString("Quit"));
    Buttons butHome = Buttons(fenetre,526,415,100,50,QString("Home"));
    Buttons butJoin = Buttons(fenetre,526,300,150,50,QString("Join Game"));

    Buttons butServerOn = Buttons(fenetre,true);
    Buttons butServerOff = Buttons(fenetre,false);

    //Initialisation des boutons
    _buttonStart = butStart.getButton();
    _buttonRestart = butRestart.getButton();
    _buttonNewGame = butNewGame.getButton();
    _buttonQuickGame = butQuickGame.getButton();
    _buttonTwoPlayers = butTwoPlayers.getButton();
    _buttonQuit = butQuit.getButton();
    _buttonSinglePlayer = butSingle.getButton();
    _buttonHome = butHome.getButton();
    _buttonJoin = butJoin.getButton();

    _buttonServerOn = butServerOn.getButton();
    _buttonServerOff = butServerOff.getButton();


    //Set la visibilité requise des boutons
    butRestart.setVisibility(false);
    butNewGame.setVisibility(false);
    butQuickGame.setVisibility(false);
    butQuit.setVisibility(false);
    butRestart.setVisibility(false);
    butHome.setVisibility(false);
    butServerOn.setVisibility(false);
    butJoin.setVisibility(false);
    hide();

    //GodMode

    _buttonGodMode = new QPushButton("", &fenetre);
    _buttonGodMode->setGeometry(QRect(QPoint(622, 573), QSize(40, 47)));
    _buttonGodMode->setFocusPolicy(Qt::NoFocus);
    _buttonGodMode->setStyleSheet(
                "background-color: transparent;"
                );

    //Connexion des boutons au slots
    QObject::connect(_buttonStart,SIGNAL(clicked()),&fenetre,SLOT(startMode()));
    QObject::connect(_buttonRestart,SIGNAL(clicked()),&fenetre,SLOT(restart()));
    QObject::connect(_buttonNewGame,SIGNAL(clicked()),&fenetre,SLOT(startNewGame()));
    QObject::connect(_buttonQuickGame,SIGNAL(clicked()),&fenetre,SLOT(quickGame()));
    QObject::connect(_buttonTwoPlayers,SIGNAL(clicked()),&fenetre,SLOT(twoPlayers()));
    QObject::connect(_buttonSinglePlayer,SIGNAL(clicked()),&fenetre,SLOT(singlePlayer()));
    QObject::connect(_buttonQuit,SIGNAL(clicked()),&fenetre,SLOT(quit()));
    QObject::connect(_buttonHome,SIGNAL(clicked()),&fenetre,SLOT(home()));
    QObject::connect(_buttonServerOn,SIGNAL(clicked()),&fenetre,SLOT(serverOn()));
    QObject::connect(_buttonServerOff,SIGNAL(clicked()),&fenetre,SLOT(serverOff()));
    QObject::connect(_buttonJoin,SIGNAL(clicked()),&fenetre,SLOT(quickGame()));
    QObject::connect(_buttonGodMode,SIGNAL(clicked()),&fenetre,SLOT(godMode()));

    _ipDialog = new IpDialog(fenetre);

  //  QObject::connect(_buttonTwoPlayers, SIGNAL(clicked()), _ipDialog, SLOT(showIpDialog()));

}
```

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

mode	the Gamemode

Definition at line 173 of file choiceboard.cpp.

```
                                            {
    if(mode==Gamemode::SCORE)return _scoreToReach->value();
    if(mode==Gamemode::NBLINES)return _nbLines->value();
    if(mode==Gamemode::TIME)return _timeUp->value();
    else return 0;
}
```

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

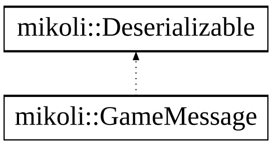
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/view/choiceboard.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/view/choiceboard.cpp

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 og **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

typeShape	The type of the **Figure** (p. 25).

Definition at line 11 of file figure.cpp.

```
                                 {

    _typeFigure = typeShape;
    int x = 0;
    int y = 0;

    switch(typeShape){
    case TypeShape::I:{
        _Blocks[0] = Block(x, y,Color::CYAN);
        _Blocks[1] = Block(x-1, y,Color::CYAN);
        _Blocks[2] = Block(x+1, y,Color::CYAN);
        _Blocks[3] = Block(x+2, y,Color::CYAN);
    }break;
    case TypeShape::O:{
        _Blocks[0] = Block(x, y,Color::GREEN);
        _Blocks[1] = Block(x, y-1,Color::GREEN);
        _Blocks[2] = Block(x+1, y,Color::GREEN);
        _Blocks[3] = Block(x+1, y-1,Color::GREEN);

    }break;
    case TypeShape::T:{
        _Blocks[0] = Block(x, y,Color::YELLOW);
        _Blocks[1] = Block(x-1, y,Color::YELLOW);
        _Blocks[2] = Block(x+1, y,Color::YELLOW);
        _Blocks[3] = Block(x, y-1,Color::YELLOW);

    }break;
    case TypeShape::J:{
        _Blocks[0] = Block(x, y,Color::BLUE);
        _Blocks[1] = Block(x-1, y,Color::BLUE);
        _Blocks[2] = Block(x+1, y,Color::BLUE);
        _Blocks[3] = Block(x+1, y-1,Color::BLUE);

    }break;
    case TypeShape::L:{
        _Blocks[0] = Block(x, y,Color::RED);
        _Blocks[1] = Block(x+1, y,Color::RED);
        _Blocks[2] = Block(x-1, y,Color::RED);
        _Blocks[3] = Block(x-1, y-1,Color::RED);

    }break;
    case TypeShape::Z:{
        _Blocks[0] = Block(x, y,Color::PURPLE);
        _Blocks[1] = Block(x-1, y,Color::PURPLE);
        _Blocks[2] = Block(x, y-1,Color::PURPLE);
        _Blocks[3] = Block(x+1, y-1,Color::PURPLE);

    }break;
    case TypeShape::S:{
        _Blocks[0] = Block(x, y,Color::DEEPPINK);
        _Blocks[1] = Block(x+1, y,Color::DEEPPINK);
        _Blocks[2] = Block(x, y-1,Color::DEEPPINK);
        _Blocks[3] = Block(x-1, y-1,Color::DEEPPINK);

    }break;
    case TypeShape::C:{

    }break;

    }
}
```

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.

```
                                 {
    std::vector<Block> list;
    list.push_back(_Blocks[0]);
    list.push_back(_Blocks[1]);
    list.push_back(_Blocks[2]);
    list.push_back(_Blocks[3]);
    return list;
}
```

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.

```
                                       {
    std::vector<Position> list;
    for(Block bl : _Blocks){
        list.push_back(bl.getPosition());
    }
    return list;
}
```

3.6.3.3 getTypeFigure()

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

getTypeFigure

Returns

The type of the figure

Definition at line 84 of file figure.cpp.

```
                              {
    return _typeFigure;
}
```

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

direction	The direction in which the figure should be moved.

Definition at line 100 of file figure.cpp.

```
                                     {
    switch(direction){
    case Direction::LEFT:
        for(int i = 0; i < 4; i++ ){
            _Blocks[i].setPosition(_Blocks[i].getPosition().getX()- 1,
                                   _Blocks[i].getPosition().getY());
        }
        break;
    case Direction::RIGHT:
        for(int i = 0; i < 4; i++ ){
            _Blocks[i].setPosition(_Blocks[i].getPosition().getX()+ 1,
                                   _Blocks[i].getPosition().getY());
        }
        break;
    case Direction::DOWN:
        for(int i = 0; i < 4; i++ ){
            _Blocks[i].setPosition(_Blocks[i].getPosition().getX(),
                                   _Blocks[i].getPosition().getY()-1);
        }
        break;
    }

}
```

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

position	The new position of the figures's central point.

Definition at line 162 of file figure.cpp.

```
                                         {
    int moveX = position.getX();
    int moveY = position.getY();

    for(int i = 0; i<4; i++){
        _Blocks[i].setPosition(_Blocks[i].getPosition().getX() +
                               moveX, _Blocks[i].getPosition().getY() + moveY);
    }

}
```

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

direction	The direction in which the figure should rotate.

Definition at line 125 of file figure.cpp.

```
                                        {
    if(direction==Direction::DOWN){
        throw TetrisException{"Can't turn in this direction"};
    }
    if(_typeFigure != TypeShape::O){

        switch(direction){
        case Direction::LEFT:
            for(int i=0;i<4;i++){

                int x = - (_Blocks[i].getPosition().getY() - _Blocks[0].
      getPosition().getY()) + _Blocks[0].getPosition().getX();

                int y = (_Blocks[i].getPosition().getX() - _Blocks[0].
      getPosition().getX()) + _Blocks[0].getPosition().getY();

                _Blocks[i].setPosition(x,y);

            }
            break;
        case Direction::RIGHT:
            for(int i=0;i<4;i++){

                int x = (_Blocks[i].getPosition().getY() - _Blocks[0].
      getPosition().getY()) + _Blocks[0].getPosition().getX();

                int y = -(_Blocks[i].getPosition().getX() - _Blocks[0].
      getPosition().getX()) + _Blocks[0].getPosition().getY();

                _Blocks[i].setPosition(x,y);

            }
            break;
        case Direction::DOWN:
            break;
        }
    }

}
```

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.

```
                                  {
    Figure fig = _nextFigure;
    if(_listFigures.size() == 0){
        refresh();
    } else {
        _nextFigure = _listFigures.back();
        _listFigures.pop_back();
    }
    return fig;
}
```

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

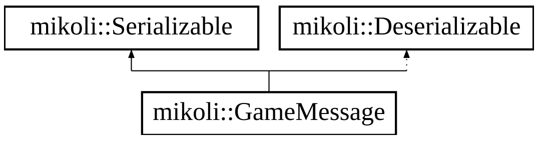
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/figuresbag.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/figuresbag.cpp

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

copie	A Gamemessage Constructor by copy.

Definition at line 26 of file gamemessage.cpp.

```
                                                    {
    _message = copie._message;
    _lines = copie._lines;
    _gameMode = copie._gameMode;
    _goalGameMode = copie._goalGameMode;
}
```

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.

```
                                   {
    return _gameMode;
}
```

3.8.3.2 getGoalGameMode()

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

getGoalGameMode

Returns

The goal for the gamemode

Definition at line 49 of file gamemessage.cpp.

```
                                  {
    return _goalGameMode;
}
```

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.

```
                                              {
    return _lines;
}
```

3.8.3.4 getTypeMessage()

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

getTypeMessage

Returns

The message

Definition at line 37 of file gamemessage.cpp.

```
                                      {
    return _message;
}
```

3.8.3.5 serialize_to_json()

```
QJsonObject 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 Q↩ JsonObject. It's necessary because a QJsonObject can be send through the network via TCP.

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

Definition at line 55 of file gamemessage.cpp.

```
                                             {
    QJsonObject obj;

    //  sérialisation de l'énumération
    int enumMsgInt = enumToInt(_message);
    obj.insert("message", enumMsgInt);

    //  sérialisation des lignes
    QJsonArray v1;
    for (std::vector<int> l : _lines) {
        QJsonObject temp;
        QJsonArray v2;
        for (int i : l) {
            QJsonObject hole;
            hole.insert("h", i);
            v2.append(hole);
        }
        temp.insert("l", v2);
        v1.append(temp);
    }
    obj.insert("lignes", v1);

    //  sérialisation du gamemode
    int enumGmInt = gmToInt(_gameMode);
    obj.insert("gamemode", enumGmInt);

    //  sérialisation du goal du gamemode
    obj.insert("goal", _goalGameMode);

    return obj;
}
```

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

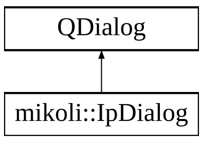
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/gamemessage.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/gamemessage.cpp

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

fenetre	the QWidget associated therewith

Definition at line 11 of file ipdialog.cpp.

```
    :QDialog(parent)
{

    QGroupBox *connexionGroup = new QGroupBox(tr("Connexion"));

    _confirm = new QPushButton("confirm");

    _lineEditIp = new QLineEdit;
    _lineEditIp->setPlaceholderText("Host IP");
    _lineEditIp->setFocus();

    _lineEditPort = new QLineEdit;
    _lineEditPort->setPlaceholderText("Host Port");
    _lineEditPort->setFocus();

    QGridLayout *connexionLayout = new QGridLayout;
    connexionLayout->addWidget(_lineEditIp, 0, 0);
    connexionLayout->addWidget(_lineEditPort, 0, 1);
    connexionLayout->addWidget(_confirm, 0, 2);

    connexionGroup->setLayout(connexionLayout);

    QGridLayout *layout = new QGridLayout;
    layout->addWidget(connexionGroup, 0, 0);

    setLayout(layout);



    setWindowTitle(tr("Connexion to Host"));

    QObject::connect(_confirm,SIGNAL(clicked()),&fenetre,SLOT(confirm()));



}
```

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.

```
                             {
    return _gameMode;
}
```

3.10.2.2 getGoal()

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

getGoal

Returns

The goal to reach.

Definition at line 22 of file mode.cpp.

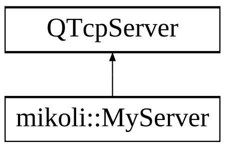
```
                 {
    return _goal;
}
```

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

ushort	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

message	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

message	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

message	the message to send

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

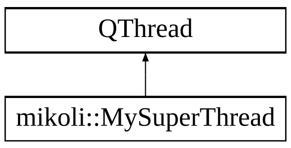
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/myserver.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/myserver.cpp

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.

```
                               {
    QString ip;

    if (IPFromInternet(ip)) return ip;
    if (IPFromGateway(ip)) return ip;
    if (IPFromLoops(ip)) return ip;
}
```

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.

```
                                 {
    QTcpSocket socket;
    int port = 1024;

    while (port <= 65535) {
        socket.connectToHost("127.0.0.1", port);

        if (socket.waitForConnected(100)) {
            socket.close();
            ++port;
        } else {
            socket.close();
            return static_cast<ushort> (port);
        }
    }
}
```

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.

```
                                  {
    return _ip;
}
```

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.

```
                                  {
    return _port;
}
```

3.12.2.5 IPFromGateway()

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

return true if the local ip is found

Parameters

QString	this value is initialized vy the method

Returns

if the ip is found or not

Definition at line 59 of file mysuperthread.cpp.

```
                                                   {
    bool found = false;

    QTcpSocket socket;
    int cpt = 0;

    while (!found && cpt <= 255) {
        QString buildIP = "192.168." + QString::number(cpt) + ".1";
        socket.connectToHost(buildIP, 53);

        if (socket.waitForConnected(100)) {
            found = true;
            ip = socket.localAddress().toString();
        }

        ++cpt;
    }

    socket.close();
    return found;
}
```

3.12.2.6 IPFromInternet()

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

return true if the local ip is found

Parameters

QString	this value is initialized vy the method

Returns

if the ip is found or not

Definition at line 43 of file mysuperthread.cpp.

```
                                                    {
    bool found = false;

    QTcpSocket socket;
    socket.connectToHost("8.8.8.8", 53);
    found = socket.waitForConnected(100);

    if (found) {
        ip = socket.localAddress().toString();
    }

    socket.close();
    return found;
}
```

3.12.2.7 IPFromLoops()

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

return true if the local ip is found

Parameters

QString	this value is initialized vy the method

Returns

if the ip is found or not

Definition at line 82 of file mysuperthread.cpp.

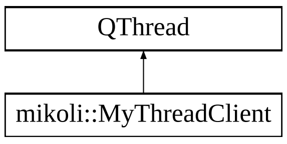
```
                                        {
    bool found = false;

    QTcpSocket socket;
    int cpt1 = 0;
    int cpt2 = 1;

    while (!found && cpt1 <= 255) {
        while (!found && cpt2 <= 254) {
            QString buildIP = "192.168." + QString::number(cpt1) + "." + QString::number(cpt2);
            socket.connectToHost(buildIP, 53);

            if (socket.waitForConnected(100)) {
                found = true;
                ip = buildIP;
            }

            ++cpt2;
        }
        ++cpt1;
    }

    socket.close();
    return found;
}
```

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 emition 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

TetrisGame *(p. 66)*	the game
SideBoard *(p. 63)*	the sideboard
ChoiceBoard *(p. 20)*	the choiceboard
int	the port to connect to host
QString	the IP adress to connect to host
parent	

Definition at line 5 of file mythreadClient.cpp.

```
                                 : QThread(parent) {
    _socket = new QTcpSocket();

    connect(_socket, SIGNAL(readyRead()), this, SLOT(readyRead()), Qt::DirectConnection);
    connect(_socket, SIGNAL(disconnected()), this, SLOT(disconnected()));

     _socket->connectToHost(ip, port);

     if(!_socket->waitForConnected(2000)) {
         sideboard->setMessage(QString("       Server unreachable"));
         sideboard->visibleMessage(true);
         sideboard->setDisplay();
         return;
     }

     _game = game;
     _sideBoard = sideboard;
     _choiceBoard = choiceBoard;
     _isConnected = true;

     start();

}
```

3.13.2.2 MyThreadClient() [2/2]

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

MyThreadClient (p. 45).

Parameters

TetrisGame *(p. 66)*	the game
SideBoard *(p. 63)*	the sideboard
ChoiceBoard *(p. 20)*	the choiceboard
GameMode	the game mode
int	the value of the goal to reach to
int	the port to connect to host
QString	the IP adress to connect to host
parent	

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

message	the message to send to the server

Definition at line 81 of file mythreadClient.cpp.

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

3.13.3.4 setIsConnected()

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

setIsConnected to set the value isConnected

Parameters

b	the bool value to set

Definition at line 117 of file mythreadClient.cpp.

```
                                      {
    _isConnected = b;
}
```

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

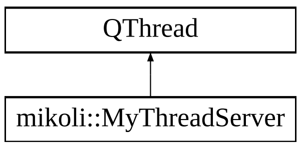
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/mythreadClient.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/mythreadClient.cpp

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

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

Definition at line 5 of file mythreadserver.cpp.

```
                                                          :
     QThread(parent){
     this->_socketDescriptor = ID;
     _socket = new QTcpSocket();

      // set the ID
      if(!_socket->setSocketDescriptor(this->_socketDescriptor)){
          //Si erreur -> signal
          emit error(_socket->error());
          return;
      }

      connect(_socket, SIGNAL(readyRead()), this, SLOT(readyRead()), Qt::DirectConnection);
      connect(_socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
  }
```

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.

```
                                        {
      return _socket;
  }
```

3.14.3.2 messageToAllClient

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

messageToAllClient

Parameters

message	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.

```
                                                            {
    if(_id==1){
        emit sendToClientTwo(message);
    }
    else if(_id==2){
        emit sendToClientOne(message);
    }
}
```

3.14.3.3 sendToClientOne

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

sendToClientOne

Parameters

message	The message to send

Definition at line 52 of file mythreadserver.cpp.

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

3.14.3.4 sendToClientTwo

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

sendToClientTwo

Parameters

message	The message to send

Definition at line 58 of file mythreadserver.cpp.

```
                                                         {
```

```
    _socket->write(QJsonDocument(message.serialize_to_json()).toJson());
    _socket->flush();
}
```

3.14.3.5 setId()

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

setId

Parameters

id	1 if this thread is connected to the first client, 2 if it's the second.

Definition at line 63 of file mythreadserver.cpp.

```
                            {
    _id = id;
}
```

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

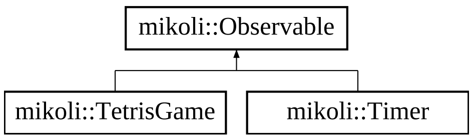
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/mythreadserver.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/mythreadserver.cpp

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:

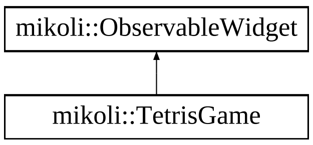
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observable.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observable.cpp

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:

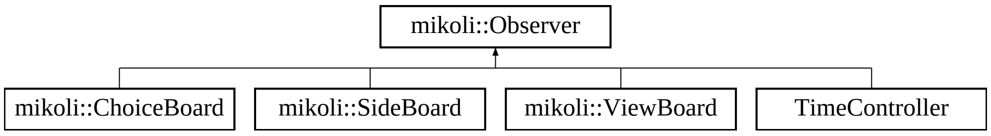
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observablewidget.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observablewidget.cpp

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:

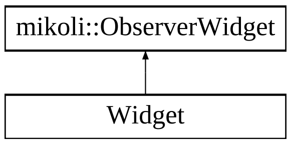
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observer.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/observer/observer.cpp

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

x	the value for horizontal axis.
y	the value for vertical axis.

Definition at line 9 of file position.cpp.

```
:_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.

```
                  {
    return _x;
}
```

3.19.3.2 getY()

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

getY

Returns

The value of _y.

Definition at line 20 of file position.cpp.

```
{
    return _y;
}
```

3.19.3.3 isSame()

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

isSame

Parameters

position	The position to compare to.

Returns

true if The position is the same, false otherwise.

Definition at line 35 of file position.cpp.

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

3.19.3.4 setX()

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

setX

Parameters

x	The new value for _x.

Definition at line 25 of file position.cpp.

```
                            {
    _x = x;
}
```

3.19.3.5 setY()

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

setY

Parameters

y	The new value for _y.

Definition at line 29 of file position.cpp.

```
                            {
    _y = y;
}
```

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 buy 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 buy 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

nbL	the number of lines the player made at the last move.
nbDrop	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.

```
                                        {
    int result = 0;

    switch(nbL){
    case 0: result += (40 * nbL) + nbDrop;
        break;
    case 1: result += (40 * nbL) + nbDrop;
        break;
    case 2 : result += (100 * nbL) + nbDrop;
        break;
    case 3: result += (300 * nbL) + nbDrop;
        break;
    case 4: result += (1200 * nbL) + nbDrop;
        break;
    default:
        break;
    }

    return result;
}
```

3.20.2.2 getLevel()

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

getLevel

Returns

The current level

Definition at line 26 of file score.cpp.

```
                        {
    return _level;
}
```

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.

```
                            {
    return _nbLines;
}
```

3.20.2.4 getScore()

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

getScore

Returns

The current score

Definition at line 21 of file score.cpp.

```
                         {
    return _score;
}
```

3.20.2.5 updateScore()

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

updateScore

Parameters

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

Exceptions

TetrisException *(p. 66)*	if nb is negative.

Definition at line 32 of file score.cpp.

```
                                        {
    if(nbL < 0){
        throw TetrisException {"Number of lines invalid, must be positive."};
    }

    _oldTen = _nbLines/10;
    _nbLines += nbL;

    if((_nbLines/10 != _oldTen)){
        _level++;
        _levelUpSound->play();
    }

    _score += calculScore(nbL, nbDrop);
}
```

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

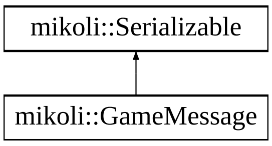
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/score.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/score.cpp

3.21 mikoli::Serializable Class Reference

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

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

Inheritance diagram for mikoli::Serializable:



Public Member Functions

- virtual QJsonObject **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:

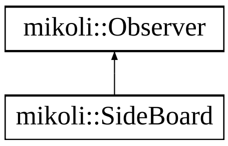
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/serializable.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/net/serializable.cpp

3.22 mikoli::SideBoard Class Reference

The SideBoartd 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 SideBoartd class.

Definition at line 15 of file sideboard.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

fenetre	the **Widget** (p. 85) in which the **SideBoard** (p. 63) has to appear
game	the **TetrisGame** (p. 66) (observable)

Definition at line 7 of file sideboard.cpp.

```
                                                          {

    _game = game;
     _width = this->_game->getBoard().getBoardSize().first;
     _height = this-> _game->getBoard().getBoardSize().second;
     _ql = new QLabel(&fenetre);
     setDisplay();
}
```

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

- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/view/sideboard.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/view/sideboard.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:

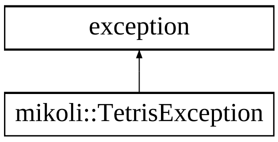
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/sounds/soundplayer.h
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/sounds/soundplayer.cpp

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:

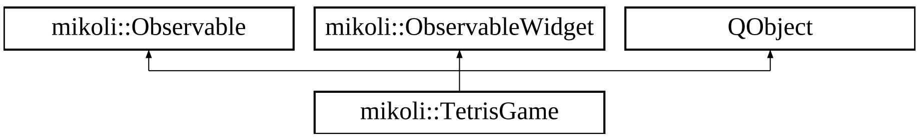
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/model/tetrisexception.h

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. Actives 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 attriubutes, ...
- 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.

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

3.25.2.2 endMove()

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

endMove

Parameters

nbDrop	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 **endGame()** (p. 68) is necessary.

Definition at line 200 of file tetrisgame.cpp.

```
                                      {
    _fallSound->play();
    _board.addFigure(_currentFigure);

    std::vector<std::vector<int>> linesToSend;

    int line = _board.checkLines(_currentFigure, linesToSend);
    int nbLines = 0;

    while(line != 0){
        nbLines++;
        _board.removeLine(line);
        _board.reorganize(line);
        line = _board.checkLines(_currentFigure, linesToSend);
    }

    if (linesToSend.size()>=2 && !_isSinglePlayer) {
        NotifyWidget(GameMessage(TypeMessage::PARAM, linesToSend));
    }

    switch(nbLines){
    case 1: _deleteOneLineSound->play();
        break;
    case 2: _deleteTwoLineSound->play();
        break;
    case 3: _deleteThreeLineSound->play();
        break;
    case 4: _deleteFourLineSound->play();
        break;
    }
    _currentScore.updateScore(nbLines, nbDrop);
    _currentFigure = _nextFigure;
    _isFalling = false;
    _nextFigure = _figuresBag.getNextFigure();
    _currentFigure.newPosition(_board.entryPoint());
    calculateShadow();
    if(!_board.areBlocksAvailable(_currentFigure)){
        _isGameOver = true;
        if (!_isSinglePlayer) {NotifyWidget(GameMessage(TypeMessage::LOOSE));}
        endGame();
    }

    isWin();
    Notify();
}
```

3.25.2.3 getBoard()

Board mikoli::TetrisGame::getBoard ( )

getBoard

Returns

The board.

Definition at line 65 of file tetrisgame.cpp.

```
                           {
    return _board;
}
```

3.25.2.4 getCurrentFig()

Figure mikoli::TetrisGame::getCurrentFig ( )

getCurrentFig

Returns

The current figure

Definition at line 46 of file tetrisgame.cpp.

```
                                 {
    return _currentFigure;
}
```

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.

```
                          {
    return _isFalling;
}
```

3.25.2.6 getLevel()

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

getLevel

Returns

The current level.

Definition at line 89 of file tetrisgame.cpp.

```
                      {
    return _currentScore.getLevel();
}
```

3.25.2.7 getMode()

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

getMode

Returns

The mode of the game.

Definition at line 96 of file tetrisgame.cpp.

```
                      {
    return _mode;
}
```

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.

```
                        {
    return _currentScore.getNbLines();
}
```

3.25.2.9 getNextFig()

```
Figure mikoli::TetrisGame::getNextFig ( )
```

getNextFig

Returns

The next figure.

Definition at line 60 of file tetrisgame.cpp.

```
                           {
    return _nextFigure;
}
```

3.25.2.10 getScore()

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

getScore

Returns

The current score.

Definition at line 86 of file tetrisgame.cpp.

```
                      {
    return _currentScore.getScore();
}
```

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.

```
                                                  {
    std::vector<Block> blocks;
    for(int i=0; i<4; ++i) {
        blocks.push_back(_shadowCF[i]);
    }
    return blocks;
}
```

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.

```
                         {
    int speed;
    if(_currentScore.getLevel() < 15){
        speed = 1000 - (_currentScore.getLevel() * 62.5);
    } else {
        return 110;
    }
    return speed;
}
```

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.

```
                           {
    return _timerGame;
}
```

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.

```
                         {
    return _isBegin;
}
```

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.

```
                            {
    return _isGameOver;
}
```

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.

```
                          {
    return _isPaused;
}
```

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.

```
                                {
    return _isSinglePlayer;
}
```

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.

```
                       {
    return _isWon;
}
```

3.25.2.19 move()

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

move

Parameters

direction	The direction we want to move. Move the current figure in the direction "direction".

Definition at line 128 of file tetrisgame.cpp.

```
                                        {
    if(_isPaused || _isGameOver){
        return;
    }
    if(_isFalling){
        direction=Direction::DOWN;
    }
    _board.move(_currentFigure, direction);
    calculateShadow();
    Notify();
    if(!_board.canGoLower(_currentFigure) && _autoDown) {
        endMove(0);
    }
    _autoDown = false;
    isWin();
}
```

3.25.2.20 rotate()

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

rotate

Parameters

direction	The direction we want to rotate. Rotate the current figure in the direction "direction".

Definition at line 146 of file tetrisgame.cpp.

```
                                          {
    if(_isPaused || _isGameOver || _isFalling){
        return;
    }
    _board.rotate(_currentFigure, direction);
    calculateShadow();
    Notify();
}
```

3.25.2.21 setAutoDown()

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

setAutoDown

Parameters

autoDown	Set the attribute _autoDown with the parameter.

Definition at line 289 of file tetrisgame.cpp.

```
                                         {
```

```
    _autoDown = autoDown;
}
```

3.25.2.22 setIsBegin()

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

setIsBegin

Parameters

isBegin	Set the attribute _isBegin with the parameter.

Definition at line 281 of file tetrisgame.cpp.

```
                                        {
    _isBegin = isBegin;
}
```

3.25.2.23 setIsFalling()

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

setIsFalling

Parameters

isFalling	Set the attribute _isFalling with the parameter.

Definition at line 285 of file tetrisgame.cpp.

```
                                            {
    _isFalling = isFalling;
}
```

3.25.2.24 upCurrentFigure()

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

upCurrentFigure

Parameters

nb	Number of lines Move the current figure to nb lines higher.

Definition at line 336 of file tetrisgame.cpp.

```
                                            {
    int maxHeightCF = 0;

    for(Block bl : _currentFigure.getBlocks()) {
        if (bl.getPosition().getY() > maxHeightCF) {
            maxHeightCF = bl.getPosition().getY();
        }
    }

    if ((maxHeightCF + nbLines) < _board.getBoardSize().second) {
        std::vector<Block> newBlocksCF;
        for(Block bl : _currentFigure.getBlocks()) {
            newBlocksCF.emplace_back(bl.getPosition().getX(), bl.getPosition().getY() + nbLines, bl.
      getColor());
        }
        _currentFigure.setBlocks(newBlocksCF);
    } else {
        int maxPossible = _board.getBoardSize().second - maxHeightCF;
        std::vector<Block> newBlocksCF;
        for(Block bl : _currentFigure.getBlocks()) {
            newBlocksCF.emplace_back(bl.getPosition().getX(), bl.getPosition().getY() + maxPossible, bl.
      getColor());
        }
        _currentFigure.setBlocks(newBlocksCF);
    }
    calculateShadow();
}
```

3.25.2.25 updateGameFromOpponent()

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

updateGameFromOpponent

Parameters

lines	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.

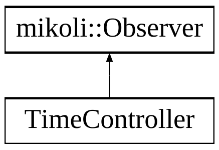
```
                                                                        {
    upCurrentFigure(lines.size()-1);
    _board.addLines(lines);
    Notify();
}
```

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

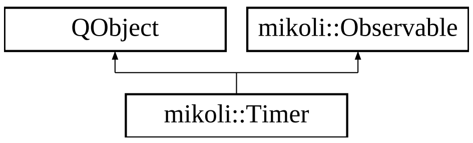
Definition at line 104 of file widget.h.

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.

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

3.27.2.2 getMinutes()

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

getMinutes

Returns

The number of minutes elapsed.

Definition at line 55 of file timer.cpp.

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

3.27.2.3 getSeconds()

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

getSeconds

Returns

The number of seconds elapsed.

Definition at line 50 of file timer.cpp.

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

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.

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

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.

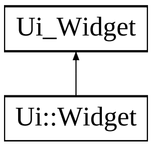
```
                                        {
    std::pair<int, int> temp;
    temp.first = this->getMinutes();
    temp.second = this->getSeconds();
    return temp;
}
```

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

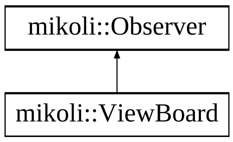
Definition at line 21 of file ui_widget.h.

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

fenetre	the **Widget** (p. 85) in which the Viewboard has to appear
game	The **TetrisGame** (p. 66) (observed)

Definition at line 12 of file viewboard.cpp.

```
                                                        {

    int width = 340;
    int height = 640;

    _game = game;
    _width = game->getBoard().getBoardSize().first;
    _height = game->getBoard().getBoardSize().second;
    _ql = new QLabel(&fenetre);
    _pixmap=QPixmap(width,height);
    _pixmap.fill(QColor("transparent"));

    setDisplay();

}
```

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

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

Definition at line 86 of file viewboard.cpp.

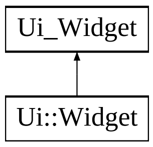
```
                                                                                                {

    QPainter painter(&_pixmap);
    painter.setPen(Qt::black);
    painter.setBrush(color);
    painter.setOpacity(opacity);
    QRect myQRect= QRect((bl.getPosition().getX()*blSize)+a,
                        ((_height+1)*blSize-bl.getPosition().getY()*blSize)+b,blSize-c,blSize-d);
    if(grad){
        QLinearGradient gradient(myQRect.topLeft(), myQRect.bottomRight());
        gradient.setColorAt(0, QColor(245, 184, 184,255));
        gradient.setColorAt(0.5, color);
        painter.fillRect(myQRect, gradient);
    }else painter.drawRect(myQRect);
}
```

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:

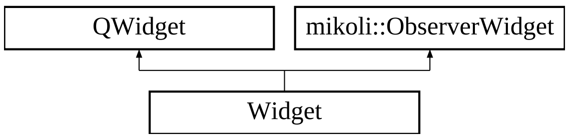
- C:/Users/Olivier/Desktop/Mikoli_T_Final/Tetris_Mikoli/ui_widget.h

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