

laying-grass-Project

Generated by Doxygen 1.15.0



<b>1 Directory Hierarchy</b>	<b>1</b>
1.1 Directories . . . . .	1
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List . . . . .	5
<b>4 File Index</b>	<b>7</b>
4.1 File List . . . . .	7
<b>5 Directory Documentation</b>	<b>9</b>
5.1 include/controllers Directory Reference . . . . .	9
5.2 src/controllers Directory Reference . . . . .	9
5.3 include Directory Reference . . . . .	9
5.4 include/models Directory Reference . . . . .	9
5.5 src/models Directory Reference . . . . .	10
5.6 src Directory Reference . . . . .	10
5.7 include/utils Directory Reference . . . . .	10
5.8 src/utils Directory Reference . . . . .	10
5.9 include/views Directory Reference . . . . .	11
5.10 src/views Directory Reference . . . . .	11
<b>6 Namespace Documentation</b>	<b>13</b>
6.1 Controllers Namespace Reference . . . . .	13
6.2 Models Namespace Reference . . . . .	13
6.2.1 Enumeration Type Documentation . . . . .	13
6.2.1.1 BonusType . . . . .	13
6.2.1.2 State . . . . .	14
6.3 Utils Namespace Reference . . . . .	14
6.3.1 Enumeration Type Documentation . . . . .	14
6.3.1.1 KeyCode . . . . .	14
6.4 Views Namespace Reference . . . . .	15
<b>7 Class Documentation</b>	<b>17</b>
7.1 Models::Board Class Reference . . . . .	17
7.1.1 Detailed Description . . . . .	17
7.1.2 Constructor & Destructor Documentation . . . . .	18
7.1.2.1 Board() . . . . .	18
7.1.3 Member Function Documentation . . . . .	18
7.1.3.1 canPlaceTile() . . . . .	18
7.1.3.2 checkBonusAcquisition() . . . . .	19
7.1.3.3 getCell() . . . . .	19

7.1.3.4 <a href="#">getGrid()</a>	19
7.1.3.5 <a href="#">getHeight()</a>	20
7.1.3.6 <a href="#">getPlayersNumber()</a>	20
7.1.3.7 <a href="#">getWidth()</a>	20
7.1.3.8 <a href="#">hasStoneAt()</a>	20
7.1.3.9 <a href="#">isCellTouchingSomething()</a>	20
7.1.3.10 <a href="#">isInsideBoard()</a>	21
7.1.3.11 <a href="#">isTileTouchingGrass()</a>	21
7.1.3.12 <a href="#">isTouchingWall()</a>	22
7.1.3.13 <a href="#">placeBonus()</a>	22
7.1.3.14 <a href="#">placeStone()</a>	22
7.1.3.15 <a href="#">placeTile()</a>	23
7.1.3.16 <a href="#">removeBonus()</a>	23
7.1.3.17 <a href="#">setGrid()</a>	23
7.1.3.18 <a href="#">setHeight()</a>	23
7.1.3.19 <a href="#">setWidth()</a>	24
7.2 <a href="#">Models::BonusSquare Class Reference</a>	24
7.2.1 <a href="#">Detailed Description</a>	24
7.2.2 <a href="#">Constructor &amp; Destructor Documentation</a>	24
7.2.2.1 <a href="#">BonusSquare()</a>	24
7.2.3 <a href="#">Member Function Documentation</a>	24
7.2.3.1 <a href="#">getBonusType()</a>	24
7.2.3.2 <a href="#">isExchange()</a>	25
7.2.3.3 <a href="#">isSteal()</a>	25
7.2.3.4 <a href="#">isStone()</a>	25
7.2.3.5 <a href="#">setBonusType()</a>	25
7.3 <a href="#">Models::Cell Class Reference</a>	25
7.3.1 <a href="#">Detailed Description</a>	26
7.3.2 <a href="#">Constructor &amp; Destructor Documentation</a>	26
7.3.2.1 <a href="#">Cell()</a>	26
7.3.3 <a href="#">Member Function Documentation</a>	26
7.3.3.1 <a href="#">getBonusType()</a>	26
7.3.3.2 <a href="#">getPlayerId()</a>	26
7.3.3.3 <a href="#">getState()</a>	26
7.3.3.4 <a href="#">isBonus()</a>	27
7.3.3.5 <a href="#">isEmpty()</a>	27
7.3.3.6 <a href="#">isGrass()</a>	27
7.3.3.7 <a href="#">setBonusType()</a>	27
7.3.3.8 <a href="#">setPlayerId()</a>	27
7.3.3.9 <a href="#">setState()</a>	27
7.4 <a href="#">Controllers::Game Class Reference</a>	28
7.4.1 <a href="#">Detailed Description</a>	28

7.4.2 Constructor & Destructor Documentation	28
7.4.2.1 Game()	28
7.4.2.2 ~Game()	28
7.4.3 Member Function Documentation	29
7.4.3.1 end()	29
7.4.3.2 getBoard()	29
7.4.3.3 getCurrentRound()	29
7.4.3.4 getPlayerById()	29
7.4.3.5 getPlayerCount()	29
7.4.3.6 getPlayers()	30
7.4.3.7 getTileQueue()	30
7.4.3.8 isGameOver()	30
7.4.3.9 run()	30
7.4.3.10 setCurrentRound()	30
7.4.3.11 start()	31
7.5 Utils::InputValidator Class Reference	31
7.5.1 Detailed Description	31
7.5.2 Member Function Documentation	31
7.5.2.1 getAvailableColors()	31
7.5.2.2 isValidNumberOfPlayers()	32
7.5.2.3 isValidPlayerColor()	32
7.5.2.4 isValidPlayerName()	32
7.5.2.5 selectColor()	32
7.6 Utils::KeyboardInput Class Reference	33
7.6.1 Detailed Description	33
7.6.2 Member Function Documentation	33
7.6.2.1 checkKeyPressed()	33
7.6.2.2 getKeyPressed()	33
7.7 Models::Player Class Reference	34
7.7.1 Detailed Description	34
7.7.2 Constructor & Destructor Documentation	34
7.7.2.1 Player()	34
7.7.3 Member Function Documentation	34
7.7.3.1 getColor()	34
7.7.3.2 getExchange()	35
7.7.3.3 getId()	35
7.7.3.4 getName()	35
7.7.3.5 getScore()	35
7.7.3.6 setColor()	35
7.7.3.7 setExchange()	35
7.7.3.8 setId()	36
7.7.3.9 setName()	36

7.7.3.10 setScore()	36
7.8 Utils::PlayerResult Struct Reference	36
7.8.1 Detailed Description	36
7.8.2 Member Data Documentation	37
7.8.2.1 playerGrass	37
7.8.2.2 playerId	37
7.8.2.3 playerScore	37
7.9 Models::Position Class Reference	37
7.9.1 Detailed Description	37
7.9.2 Constructor & Destructor Documentation	38
7.9.2.1 Position() [1/2]	38
7.9.2.2 Position() [2/2]	38
7.9.3 Member Function Documentation	38
7.9.3.1 getX()	38
7.9.3.2 getY()	38
7.9.3.3 operator!=(())	38
7.9.3.4 operator<()	39
7.9.3.5 operator==(())	39
7.9.3.6 setX()	39
7.9.3.7 setY()	39
7.10 Utils::Random Class Reference	39
7.10.1 Detailed Description	40
7.10.2 Member Function Documentation	40
7.10.2.1 getInt()	40
7.10.2.2 shuffle()	40
7.11 Utils::SquareCalculator Class Reference	40
7.11.1 Detailed Description	41
7.11.2 Constructor & Destructor Documentation	41
7.11.2.1 SquareCalculator()	41
7.11.3 Member Function Documentation	41
7.11.3.1 calculateGrass()	41
7.11.3.2 calculateSquare()	41
7.11.3.3 rankingPlayersByScore()	42
7.12 Models::Tile Class Reference	42
7.12.1 Detailed Description	43
7.12.2 Constructor & Destructor Documentation	43
7.12.2.1 Tile()	43
7.12.3 Member Function Documentation	43
7.12.3.1 convertJsonToTile()	43
7.12.3.2 createTile()	44
7.12.3.3 flipHorizontal()	44
7.12.3.4 getHeight()	44

7.12.3.5 getId()	44
7.12.3.6 getPattern()	44
7.12.3.7 getPlayerId()	45
7.12.3.8 getSize()	45
7.12.3.9 getWidth()	45
7.12.3.10 isPlaced()	45
7.12.3.11 rotate()	45
7.12.3.12 setHeight()	46
7.12.3.13 setId()	46
7.12.3.14 setPattern()	46
7.12.3.15 setPlaced()	46
7.12.3.16 setPlayerId()	46
7.12.3.17 setWidth()	46
7.13 Controllers::TilePlacer Class Reference	47
7.13.1 Detailed Description	47
7.13.2 Constructor & Destructor Documentation	47
7.13.2.1 TilePlacer()	47
7.13.3 Member Function Documentation	47
7.13.3.1 confirmPlacement()	47
7.13.3.2 getPosition()	48
7.13.3.3 getTile()	48
7.13.3.4 isValidPlacement()	48
7.13.3.5 moveDown()	48
7.13.3.6 moveLeft()	48
7.13.3.7 moveRight()	48
7.13.3.8 moveUp()	49
7.13.3.9 rotateTile()	49
7.13.3.10 setInitialPosition()	49
7.14 Models::TileQueue Class Reference	49
7.14.1 Detailed Description	50
7.14.2 Constructor & Destructor Documentation	50
7.14.2.1 TileQueue()	50
7.14.3 Member Function Documentation	50
7.14.3.1 addTile()	50
7.14.3.2 addUsedTile()	50
7.14.3.3 getCurrentTile()	50
7.14.3.4 getNextTiles()	51
7.14.3.5 getTileAt()	51
7.14.3.6 getTiles()	51
7.14.3.7 getUsedTiles()	51
7.14.3.8 isEmpty()	51
7.14.3.9 loadTiles()	52

7.14.3.10 recycleTiles()	52
7.14.3.11 removeTile()	52
7.14.3.12 removeUsedTile()	52
7.14.3.13 selectTileFromMarket()	53
7.14.3.14 shuffleTiles()	53
7.15 Views::UI_Cli Class Reference	53
7.15.1 Detailed Description	54
7.15.2 Member Function Documentation	54
7.15.2.1 askNumberOfPlayers()	54
7.15.2.2 askPlayerColor()	54
7.15.2.3 askPlayerName()	55
7.15.2.4 clearScreen()	55
7.15.2.5 displayBoard()	55
7.15.2.6 displayBoardWithTile()	56
7.15.2.7 displayMarket()	57
7.15.2.8 displayMessage()	57
7.15.2.9 displayPlayer()	58
7.15.2.10 displayTile()	58
7.15.2.11 displayWelcome()	58
7.15.2.12 displayWinner()	58
7.15.2.13 tilePlacement()	59
7.16 Views::UI_Qt Class Reference	60
7.16.1 Detailed Description	60
<b>8 File Documentation</b>	<b>61</b>
8.1 include/controllers/Game.h File Reference	61
8.2 Game.h	61
8.3 include/controllers/TilePlacer.h File Reference	62
8.4 TilePlacer.h	63
8.5 include/models/Board.h File Reference	63
8.6 Board.h	64
8.7 include/models/BonusSquare.h File Reference	64
8.8 BonusSquare.h	65
8.9 include/models/Cell.h File Reference	65
8.10 Cell.h	65
8.11 include/models/Player.h File Reference	66
8.12 Player.h	66
8.13 include/models/Position.h File Reference	67
8.14 Position.h	67
8.15 include/models/Tile.h File Reference	68
8.16 Tile.h	68
8.17 include/models/TileQueue.h File Reference	69

8.18 TileQueue.h . . . . .	69
8.19 include/models/Types.h File Reference . . . . .	70
8.20 Types.h . . . . .	70
8.21 include/utils/InputValidator.h File Reference . . . . .	70
8.22 InputValidator.h . . . . .	71
8.23 include/utils/KeyboardInput.h File Reference . . . . .	71
8.24 KeyboardInput.h . . . . .	72
8.25 include/utils/Random.h File Reference . . . . .	72
8.26 Random.h . . . . .	72
8.27 include/utils/SquareCalculator.h File Reference . . . . .	73
8.28 SquareCalculator.h . . . . .	73
8.29 include/views/UI_Cli.h File Reference . . . . .	74
8.30 UI_Cli.h . . . . .	74
8.31 include/views/UI_Qt.h File Reference . . . . .	75
8.32 UI_Qt.h . . . . .	75
8.33 src/controllers/Game.cpp File Reference . . . . .	75
8.34 Game.cpp . . . . .	75
8.35 src/controllers/TilePlacer.cpp File Reference . . . . .	80
8.36 TilePlacer.cpp . . . . .	80
8.37 src/main.cpp File Reference . . . . .	81
8.37.1 Function Documentation . . . . .	81
8.37.1.1 main() . . . . .	81
8.38 main.cpp . . . . .	81
8.39 src/models/Board.cpp File Reference . . . . .	82
8.40 Board.cpp . . . . .	82
8.41 src/models/BonusSquare.cpp File Reference . . . . .	85
8.42 BonusSquare.cpp . . . . .	85
8.43 src/models/Cell.cpp File Reference . . . . .	85
8.44 Cell.cpp . . . . .	86
8.45 src/models/Player.cpp File Reference . . . . .	86
8.46 Player.cpp . . . . .	86
8.47 src/models/Position.cpp File Reference . . . . .	87
8.48 Position.cpp . . . . .	87
8.49 src/models/Tile.cpp File Reference . . . . .	87
8.50 Tile.cpp . . . . .	88
8.51 src/models/TileQueue.cpp File Reference . . . . .	89
8.52 TileQueue.cpp . . . . .	89
8.53 src/utils/InputValidator.cpp File Reference . . . . .	90
8.54 InputValidator.cpp . . . . .	91
8.55 src/utils/KeyboardInput.cpp File Reference . . . . .	91
8.56 KeyboardInput.cpp . . . . .	92
8.57 src/utils/Random.cpp File Reference . . . . .	92

8.58 Random.cpp . . . . .	92
8.59 src/utls/SquareCalculator.cpp File Reference . . . . .	93
8.60 SquareCalculator.cpp . . . . .	93
8.61 src/views/UI_Cli.cpp File Reference . . . . .	94
8.62 UI_Cli.cpp . . . . .	94
8.63 src/views/UI_Qt.cpp File Reference . . . . .	99
8.64 UI_Qt.cpp . . . . .	99
<b>Index</b>	<b>101</b>

# Chapter 1

## Directory Hierarchy

### 1.1 Directories

controllers	9
Game.h	61
TilePlacer.h	62
controllers	9
Game.cpp	75
TilePlacer.cpp	80
include	9
controllers	9
Game.h	61
TilePlacer.h	62
models	9
Board.h	63
BonusSquare.h	64
Cell.h	65
Player.h	66
Position.h	67
Tile.h	68
TileQueue.h	69
Types.h	70
utils	10
InputValidator.h	70
KeyboardInput.h	71
Random.h	72
SquareCalculator.h	73
views	11
UI_Cli.h	74
UI_Qt.h	75
models	9
Board.h	63
BonusSquare.h	64
Cell.h	65
Player.h	66
Position.h	67
Tile.h	68
TileQueue.h	69
Types.h	70

models	10
Board.cpp	82
BonusSquare.cpp	85
Cell.cpp	85
Player.cpp	86
Position.cpp	87
Tile.cpp	87
TileQueue.cpp	89
src	10
controllers	9
Game.cpp	75
TilePlacer.cpp	80
models	10
Board.cpp	82
BonusSquare.cpp	85
Cell.cpp	85
Player.cpp	86
Position.cpp	87
Tile.cpp	87
TileQueue.cpp	89
utils	10
InputValidator.cpp	90
KeyboardInput.cpp	91
Random.cpp	92
SquareCalculator.cpp	93
views	11
UI_Cli.cpp	94
UI_Qt.cpp	99
main.cpp	81
utils	10
InputValidator.h	70
KeyboardInput.h	71
Random.h	72
SquareCalculator.h	73
utils	10
InputValidator.cpp	90
KeyboardInput.cpp	91
Random.cpp	92
SquareCalculator.cpp	93
views	11
UI_Cli.h	74
UI_Qt.h	75
views	11
UI_Cli.cpp	94
UI_Qt.cpp	99

## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">Controllers</a>	13
<a href="#">Models</a>	13
<a href="#">Utils</a>	14
<a href="#">Views</a>	15



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Models::Board</a>	17
<a href="#">Models::BonusSquare</a>	24
<a href="#">Models::Cell</a>	25
<a href="#">Controllers::Game</a>	28
<a href="#">Utils::InputValidator</a>	31
<a href="#">Utils::KeyboardInput</a>	33
<a href="#">Models::Player</a>	34
<a href="#">Utils::PlayerResult</a>	36
<a href="#">Models::Position</a>	37
<a href="#">Utils::Random</a>	39
<a href="#">Utils::SquareCalculator</a>	40
<a href="#">Models::Tile</a>	42
<a href="#">Controllers::TilePlacer</a>	47
<a href="#">Models::TileQueue</a>	49
<a href="#">Views::UI_Cli</a>	53
<a href="#">Views::UI_Qt</a>	60



# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/controllers/	Game.h	61
include/controllers/	TilePlacer.h	62
include/models/	Board.h	63
include/models/	BonusSquare.h	64
include/models/	Cell.h	65
include/models/	Player.h	66
include/models/	Position.h	67
include/models/	Tile.h	68
include/models/	TileQueue.h	69
include/models/	Types.h	70
include/utills/	InputValidator.h	70
include/utills/	KeyboardInput.h	71
include/utills/	Random.h	72
include/utills/	SquareCalculator.h	73
include/views/	UI_Cli.h	74
include/views/	UI_Qt.h	75
src/	main.cpp	81
src/controllers/	Game.cpp	75
src/controllers/	TilePlacer.cpp	80
src/models/	Board.cpp	82
src/models/	BonusSquare.cpp	85
src/models/	Cell.cpp	85
src/models/	Player.cpp	86
src/models/	Position.cpp	87
src/models/	Tile.cpp	87
src/models/	TileQueue.cpp	89
src/utills/	InputValidator.cpp	90
src/utills/	KeyboardInput.cpp	91
src/utills/	Random.cpp	92
src/utills/	SquareCalculator.cpp	93
src/views/	UI_Cli.cpp	94
src/views/	UI_Qt.cpp	99



## Chapter 5

# Directory Documentation

### 5.1 include/controllers Directory Reference

#### Files

- file [Game.h](#)
- file [TilePlacer.h](#)

### 5.2 src/controllers Directory Reference

#### Files

- file [Game.cpp](#)
- file [TilePlacer.cpp](#)

### 5.3 include Directory Reference

#### Directories

- directory [controllers](#)
- directory [models](#)
- directory [utils](#)
- directory [views](#)

### 5.4 include/models Directory Reference

#### Files

- file [Board.h](#)
- file [BonusSquare.h](#)
- file [Cell.h](#)
- file [Player.h](#)
- file [Position.h](#)
- file [Tile.h](#)
- file [TileQueue.h](#)
- file [Types.h](#)

## 5.5 src/models Directory Reference

### Files

- file [Board.cpp](#)
- file [BonusSquare.cpp](#)
- file [Cell.cpp](#)
- file [Player.cpp](#)
- file [Position.cpp](#)
- file [Tile.cpp](#)
- file [TileQueue.cpp](#)

## 5.6 src Directory Reference

### Directories

- directory [controllers](#)
- directory [models](#)
- directory [utils](#)
- directory [views](#)

### Files

- file [main.cpp](#)

## 5.7 include/utils Directory Reference

### Files

- file [InputValidator.h](#)
- file [KeyboardInput.h](#)
- file [Random.h](#)
- file [SquareCalculator.h](#)

## 5.8 src/utils Directory Reference

### Files

- file [InputValidator.cpp](#)
- file [KeyboardInput.cpp](#)
- file [Random.cpp](#)
- file [SquareCalculator.cpp](#)

## 5.9 include/views Directory Reference

### Files

- file [UI\\_Cli.h](#)
- file [UI\\_Qt.h](#)

## 5.10 src/views Directory Reference

### Files

- file [UI\\_Cli.cpp](#)
- file [UI\\_Qt.cpp](#)



## Chapter 6

# Namespace Documentation

### 6.1 Controllers Namespace Reference

#### Classes

- class [Game](#)
- class [TilePlacer](#)

### 6.2 Models Namespace Reference

#### Classes

- class [Board](#)
- class [BonusSquare](#)
- class [Cell](#)
- class [Player](#)
- class [Position](#)
- class [Tile](#)
- class [TileQueue](#)

#### Enumerations

- enum class [State](#) { [EMPTY](#) , [GRASS](#) , [BONUS](#) }
- enum class [BonusType](#) { [NONE](#) , [EXCHANGE](#) , [STONE](#) , [STEAL](#) }

#### 6.2.1 Enumeration Type Documentation

##### 6.2.1.1 BonusType

```
enum class Models::BonusType [strong]
```

**Enumerator**

NONE	
EXCHANGE	
STONE	
STEAL	

Definition at line 9 of file [Types.h](#).

```
00009 {NONE, EXCHANGE, STONE, STEAL};
```

**6.2.1.2 State**

```
enum class Models::State [strong]
```

**Enumerator**

EMPTY	
GRASS	
BONUS	

Definition at line 8 of file [Types.h](#).

```
00008 {EMPTY, GRASS, BONUS};
```

**6.3 Utils Namespace Reference****Classes**

- class [InputValidator](#)
- class [KeyboardInput](#)
- class [Random](#)
- struct [PlayerResult](#)
- class [SquareCalculator](#)

**Enumerations**

- enum class [KeyCode](#) {  
    [UP](#) , [DOWN](#) , [LEFT](#) , [RIGHT](#) ,  
    [ROTATE](#) , [CONFIRM](#) , [UNKNOWN](#) }

**6.3.1 Enumeration Type Documentation****6.3.1.1 KeyCode**

```
enum class Utils::KeyCode [strong]
```

### Enumerator

UP	
DOWN	
LEFT	
RIGHT	
ROTATE	
CONFIRM	
UNKNOWN	

Definition at line 11 of file [KeyboardInput.h](#).

```
00011                                     {
00012         UP,
00013         DOWN,
00014         LEFT,
00015         RIGHT,
00016         ROTATE,
00017         CONFIRM,
00018         UNKNOWN
00019     };
```

## 6.4 Views Namespace Reference

### Classes

- class [UI\\_Cli](#)
- class [UI\\_Qt](#)



# Chapter 7

## Class Documentation

### 7.1 Models::Board Class Reference

```
#include <Board.h>
```

#### Public Member Functions

- [Board](#) (int playersNumber)
- int const [getPlayersNumber](#) ()
- int const [getWidth](#) ()
- void [setWidth](#) (int w)
- int const [getHeight](#) ()
- void [setHeight](#) (int h)
- std::vector< std::vector< [Cell](#) > > const [getGrid](#) ()
- void [setGrid](#) (std::vector< std::vector< [Cell](#) > > g)
- [Cell](#) \* [getCell](#) ([Position](#) &pos)
- void [placeBonus](#) ()
- void [removeBonus](#) ([Position](#) &pos, int playerId)
- void [placeStone](#) ([Position](#) &pos)
- bool [hasStoneAt](#) ([Position](#) &pos) const
- bool [isCellTouchingSomething](#) ([Position](#) &pos, [State](#) state, int playerId)
- bool [isTileTouchingGrass](#) ([Tile](#) \*tile, [Position](#) &pos, int playerId)
- bool [isTouchingWall](#) ([Position](#) &pos)
- bool [isInsideBoard](#) ([Position](#) &pos)
- bool [canPlaceTile](#) ([Tile](#) \*tile, [Position](#) &pos, int playerId)
- void [placeTile](#) ([Tile](#) \*tile, [Position](#) &pos, int playerId)
- std::vector< [Position](#) > [checkBonusAcquisition](#) ([Tile](#) \*tile, [Position](#) &pos, int playerId)

#### 7.1.1 Detailed Description

Definition at line 14 of file [Board.h](#).

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 Board()

Models::Board::Board (  
     int *playersNumber*)

Definition at line 10 of file [Board.cpp](#).

```
00010         {
00011     this->playersNumber = playersNumber;
00012     this->exchangeCount = std::ceil(1.5 * playersNumber);
00013     this->stoneCount = std::ceil(0.5 * playersNumber);
00014     this->stealCount = playersNumber;
00015     if (playersNumber >= 2 && playersNumber <= 4) {
00016         this->width = 20;
00017         this->height = 20;
00018     } else if (playersNumber >= 5 && playersNumber <= 9) {
00019         this->width = 30;
00020         this->height = 30;
00021     }
00022     grid.resize(height, std::vector<Cell>(width, Cell()));
00023     placeBonus();
00024 };
```

## 7.1.3 Member Function Documentation

### 7.1.3.1 canPlaceTile()

bool Models::Board::canPlaceTile (  
     Tile \* *tile*,  
     Position & *pos*,  
     int *playerId*)

Definition at line 123 of file [Board.cpp](#).

```
00123         {
00124     const auto& pattern = tile->getPattern();
00125     int tileHeight = tile->getHeight();
00126     int tileWidth = tile->getWidth();
00127
00128     bool touchingOwnGrass = false;
00129
00130     for (int ty = 0; ty < tileHeight; ++ty) {
00131         for (int tx = 0; tx < tileWidth; ++tx) {
00132             if (pattern[ty][tx].getState() == State::GRASS) {
00133                 int boardX = pos.getX() + tx;
00134                 int boardY = pos.getY() + ty;
00135                 Position cellPos(boardX, boardY);
00136
00137                 if (!isInsideBoard(cellPos)) {
00138                     return false;
00139                 }
00140
00141                 if (grid[boardY][boardX].getState() != State::EMPTY) {
00142                     return false;
00143                 }
00144
00145                 if (isCellTouchingSomething(cellPos, State::GRASS, playerId)) {
00146                     return false;
00147                 }
00148
00149                 int directions[4][2] = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
00150                 for (auto& dir : directions) {
00151                     int checkX = boardX + dir[0];
00152                     int checkY = boardY + dir[1];
00153
00154                     if (checkX >= 0 && checkX < width && checkY >= 0 && checkY < height) {
00155                         if (grid[checkY][checkX].getState() == State::GRASS &&
00156                             grid[checkY][checkX].getPlayerId() == playerId) {
00157                             touchingOwnGrass = true;
00158                             break;
00159                         }
00160                     }
00161                 }
00162             }
00163         }
00164     }
00165 }
```

```

00161         }
00162     }
00163 }
00164 }
00165
00166 bool isFirstPlacement = true;
00167 for (int y = 0; y < height; ++y) {
00168     for (int x = 0; x < width; ++x) {
00169         if (grid[y][x].getState() == State::GRASS &&
00170             grid[y][x].getPlayerId() == playerId) {
00171             isFirstPlacement = false;
00172             break;
00173         }
00174     }
00175     if (!isFirstPlacement) break;
00176 }
00177 return isFirstPlacement || touchingOwnGrass;
00178 }

```

### 7.1.3.2 checkBonusAcquisition()

```

std::vector< Position > Models::Board::checkBonusAcquisition (
    Tile * tile,
    Position & pos,
    int playerId)

```

Definition at line 198 of file [Board.cpp](#).

```

00198 {
00199     std::vector<Position> acquiredBonuses;
00200
00201     for (int y = 1; y < height - 1; ++y) {
00202         for (int x = 1; x < width - 1; ++x) {
00203             if (grid[y][x].getState() == State::BONUS) {
00204                 bool surroundedTop = grid[y-1][x].getState() == State::GRASS &&
00205                     grid[y-1][x].getPlayerId() == playerId;
00206                 bool surroundedBottom = grid[y+1][x].getState() == State::GRASS &&
00207                     grid[y+1][x].getPlayerId() == playerId;
00208                 bool surroundedLeft = grid[y][x-1].getState() == State::GRASS &&
00209                     grid[y][x-1].getPlayerId() == playerId;
00210                 bool surroundedRight = grid[y][x+1].getState() == State::GRASS &&
00211                     grid[y][x+1].getPlayerId() == playerId;
00212
00213                 if (surroundedTop && surroundedBottom && surroundedLeft && surroundedRight) {
00214                     acquiredBonuses.push_back(Position(x, y));
00215                 }
00216             }
00217         }
00218     }
00219
00220     return acquiredBonuses;
00221 }

```

### 7.1.3.3 getCell()

```

Cell * Models::Board::getCell (
    Position & pos) [inline]

```

Definition at line 38 of file [Board.h](#).

```

00038 {
00039     if (!isInsideBoard(pos)) {
00040         return nullptr;
00041     }
00042     return &grid[pos.getY()][pos.getX()];
00043 };

```

### 7.1.3.4 getGrid()

```

std::vector< std::vector< Cell > > const Models::Board::getGrid () [inline]

```

Definition at line 35 of file [Board.h](#).

```

00035 { return grid; }

```

### 7.1.3.5 getHeight()

```
int const Models::Board::getHeight () [inline]
```

Definition at line 32 of file [Board.h](#).

```
00032 { return height; }
```

### 7.1.3.6 getPlayersNumber()

```
int const Models::Board::getPlayersNumber () [inline]
```

Definition at line 27 of file [Board.h](#).

```
00027 { return playersNumber; }
```

### 7.1.3.7 getWidth()

```
int const Models::Board::getWidth () [inline]
```

Definition at line 29 of file [Board.h](#).

```
00029 { return width; }
```

### 7.1.3.8 hasStoneAt()

```
bool Models::Board::hasStoneAt (
    Position & pos) const
```

Definition at line 246 of file [Board.cpp](#).

```
00246                                     {
00247     if (pos.getX() < 0 || pos.getX() >= width || pos.getY() < 0 || pos.getY() >= height) {
00248         return false;
00249     }
00250     return grid[pos.getY()][pos.getX()].getState() == State::BONUS &&
00251            grid[pos.getY()][pos.getX()].getBonusType() == BonusType::STONE;
00252 }
```

### 7.1.3.9 isCellTouchingSomething()

```
bool Models::Board::isCellTouchingSomething (
    Position & pos,
    State state,
    int playerId = -1)
```

Definition at line 50 of file [Board.cpp](#).

```
00050                                     {
00051     int x = pos.getX();
00052     int y = pos.getY();
00053
00054     int directions[4][2] = {
00055         {0, -1},
00056         {0, 1},
00057         {-1, 0},
00058         {1, 0}
00059     };
00060
00061     for (auto& dir : directions) {
00062         int checkX = x + dir[0];
00063         int checkY = y + dir[1];
00064     }
```

```

00065         if (checkX >= 0 && checkX < width && checkY >= 0 && checkY < height) {
00066             if (state == State::GRASS) {
00067                 if (grid[checkY][checkX].getState() == State::GRASS &&
00068                     grid[checkY][checkX].getPlayerId() != playerId &&
00069                     grid[checkY][checkX].getPlayerId() != -1) {
00070                     return true;
00071                 }
00072             } else {
00073                 if (grid[checkY][checkX].getState() == state) {
00074                     return true;
00075                 }
00076             }
00077         }
00078     }
00079     return false;
00080 }

```

### 7.1.3.10 isInsideBoard()

```

bool Models::Board::isInsideBoard (
    Position & pos)

```

Definition at line 116 of file [Board.cpp](#).

```

00116     {
00117         int x = pos.getX();
00118         int y = pos.getY();
00119         return (x >= 0 && x < width && y >= 0 && y < height);
00120     }
00121 }

```

### 7.1.3.11 isTileTouchingGrass()

```

bool Models::Board::isTileTouchingGrass (
    Tile * tile,
    Position & pos,
    int playerId)

```

Definition at line 83 of file [Board.cpp](#).

```

00083     {
00084         const auto& pattern = tile->getPattern();
00085         int tileHeight = tile->getHeight();
00086         int tileWidth = tile->getWidth();
00087         int tileSize = tile->getSize();
00088         int x = pos.getX();
00089         int y = pos.getY();
00090
00091         while (tileSize > 0) {
00092             for (int ty = 0; ty < tileHeight; ++ty) {
00093                 for (int tx = 0; tx < tileWidth; ++tx) {
00094                     if (pattern[ty][tx].getState() == State::GRASS) {
00095                         int boardX = x + tx;
00096                         int boardY = y + ty;
00097                         Position cellPos(boardX, boardY);
00098                         if (isCellTouchingSomething(cellPos, State::GRASS, playerId)) {
00099                             return true;
00100                         }
00101                         tileSize--;
00102                     }
00103                 }
00104             }
00105         }
00106         return false;
00107     }

```

### 7.1.3.12 isTouchingWall()

```
bool Models::Board::isTouchingWall (
    Position & pos)
```

Definition at line 109 of file [Board.cpp](#).

```
00109         {
00110             int x = pos.getX();
00111             int y = pos.getY();
00112
00113             return (x == 0 || x == width - 1 || y == 0 || y == height - 1);
00114         }
```

### 7.1.3.13 placeBonus()

```
void Models::Board::placeBonus ()
```

Definition at line 26 of file [Board.cpp](#).

```
00026         {
00027             while (exchangeCount != 0 || stoneCount != 0 || stealCount != 0) {
00028                 int x = Utils::Random::getInt(0, width - 1);
00029                 int y = Utils::Random::getInt(0, height - 1);
00030                 Position pos(x, y);
00031
00032                 if (grid[y][x].getState() == State::EMPTY && !isTouchingWall(pos) &&
!isCellTouchingSomething(pos, State::BONUS, -1)) {
00033                     if (exchangeCount > 0) {
00034                         grid[y][x].setState(State::BONUS);
00035                         grid[y][x].setBonusType(BonusType::EXCHANGE);
00036                         exchangeCount--;
00037                     } else if (stoneCount > 0) {
00038                         grid[y][x].setState(State::BONUS);
00039                         grid[y][x].setBonusType(BonusType::STONE);
00040                         stoneCount--;
00041                     } else if (stealCount > 0) {
00042                         grid[y][x].setState(State::BONUS);
00043                         grid[y][x].setBonusType(BonusType::STEAL);
00044                         stealCount--;
00045                     }
00046                 }
00047             }
00048         }
```

### 7.1.3.14 placeStone()

```
void Models::Board::placeStone (
    Position & pos)
```

Definition at line 235 of file [Board.cpp](#).

```
00235         {
00236             if (isInsideBoard(pos) && !isTouchingWall(pos)) {
00237                 int x = pos.getX();
00238                 int y = pos.getY();
00239                 if (grid[y][x].getState() == State::EMPTY) {
00240                     grid[y][x].setState(State::BONUS);
00241                     grid[y][x].setBonusType(BonusType::STONE);
00242                 }
00243             }
00244         }
```

### 7.1.3.15 placeTile()

```
void Models::Board::placeTile (
    Tile * tile,
    Position & pos,
    int playerId)
```

Definition at line 180 of file [Board.cpp](#).

```
00180
00181     const auto& pattern = tile->getPattern();
00182     int tileHeight = tile->getHeight();
00183     int tileWidth = tile->getWidth();
00184
00185     for (int ty = 0; ty < tileHeight; ++ty) {
00186         for (int tx = 0; tx < tileWidth; ++tx) {
00187             if (pattern[ty][tx].getState() == State::GRASS) {
00188                 int boardX = pos.getX() + tx;
00189                 int boardY = pos.getY() + ty;
00190
00191                 grid[boardY][boardX].setState(State::GRASS);
00192                 grid[boardY][boardX].setPlayerId(playerId);
00193             }
00194         }
00195     }
00196 }
```

### 7.1.3.16 removeBonus()

```
void Models::Board::removeBonus (
    Position & pos,
    int playerId)
```

Definition at line 223 of file [Board.cpp](#).

```
00223
00224     if (isInsideBoard(pos)) {
00225         int x = pos.getX();
00226         int y = pos.getY();
00227         if (grid[y][x].getState() == State::BONUS) {
00228             grid[y][x].setState(State::GRASS);
00229             grid[y][x].setPlayerId(playerId);
00230             grid[y][x].setBonusType(BonusType::NONE);
00231         }
00232     }
00233 }
```

### 7.1.3.17 setGrid()

```
void Models::Board::setGrid (
    std::vector< std::vector< Cell > > g) [inline]
```

Definition at line 36 of file [Board.h](#).

```
00036 { grid = g; }
```

### 7.1.3.18 setHeight()

```
void Models::Board::setHeight (
    int h) [inline]
```

Definition at line 33 of file [Board.h](#).

```
00033 { height = h; }
```

### 7.1.3.19 `setWidth()`

```
void Models::Board::setWidth (  
    int w)    [inline]
```

Definition at line 30 of file [Board.h](#).

```
00030 { width = w; }
```

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

- include/models/[Board.h](#)
- src/models/[Board.cpp](#)

## 7.2 `Models::BonusSquare` Class Reference

```
#include <BonusSquare.h>
```

### Public Member Functions

- [BonusSquare](#) ()
- [BonusType](#) const [getBonusType](#) ()
- void [setBonusType](#) ([BonusType](#) b)
- bool [isExchange](#) ()
- bool [isStone](#) ()
- bool [isSteal](#) ()

### 7.2.1 Detailed Description

Definition at line 9 of file [BonusSquare.h](#).

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 `BonusSquare()`

```
Models::BonusSquare::BonusSquare ()
```

### 7.2.3 Member Function Documentation

#### 7.2.3.1 `getBonusType()`

```
BonusType const Models::BonusSquare::getBonusType ()    [inline]
```

Definition at line 15 of file [BonusSquare.h](#).

```
00015 { return bonusType; }
```

### 7.2.3.2 isExchange()

```
bool Models::BonusSquare::isExchange ()
```

Definition at line 8 of file [BonusSquare.cpp](#).

```
00008      {
00009      return this->bonusType == BonusType::EXCHANGE;
00010      }
```

### 7.2.3.3 isSteal()

```
bool Models::BonusSquare::isSteal ()
```

Definition at line 16 of file [BonusSquare.cpp](#).

```
00016      {
00017      return this->bonusType == BonusType::STEAL;
00018      }
```

### 7.2.3.4 isStone()

```
bool Models::BonusSquare::isStone ()
```

Definition at line 12 of file [BonusSquare.cpp](#).

```
00012      {
00013      return this->bonusType == BonusType::STONE;
00014      }
```

### 7.2.3.5 setBonusType()

```
void Models::BonusSquare::setBonusType (
    BonusType b) [inline]
```

Definition at line 16 of file [BonusSquare.h](#).

```
00016 { bonusType = b; }
```

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

- include/models/[BonusSquare.h](#)
- src/models/[BonusSquare.cpp](#)

## 7.3 Models::Cell Class Reference

```
#include <Cell.h>
```

## Public Member Functions

- [Cell](#) ()
- [State](#) const [getState](#) () const
- void [setState](#) ([State](#) s)
- [BonusType](#) const [getBonusType](#) () const
- void [setBonusType](#) ([BonusType](#) b)
- int const [getPlayerId](#) () const
- void [setPlayerId](#) (int id)
- bool [isEmpty](#) ()
- bool [isGrass](#) ()
- bool [isBonus](#) ()

### 7.3.1 Detailed Description

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

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 Cell()

```
Models::Cell::Cell ()
```

Definition at line 9 of file [Cell.cpp](#).

```
00009 {}
```

### 7.3.3 Member Function Documentation

#### 7.3.3.1 getBonusType()

```
BonusType const Models::Cell::getBonusType () const [inline]
```

Definition at line 24 of file [Cell.h](#).

```
00024 { return bonusType; }
```

#### 7.3.3.2 getPlayerId()

```
int const Models::Cell::getPlayerId () const [inline]
```

Definition at line 27 of file [Cell.h](#).

```
00027 { return playerId; }
```

#### 7.3.3.3 getState()

```
State const Models::Cell::getState () const [inline]
```

Definition at line 21 of file [Cell.h](#).

```
00021 { return state ;}
```

#### 7.3.3.4 isBonus()

```
bool Models::Cell::isBonus ()
```

Definition at line 20 of file [Cell.cpp](#).

```
00020         {
00021         return this->state == State::BONUS;
00022     }
```

#### 7.3.3.5 isEmpty()

```
bool Models::Cell::isEmpty ()
```

Definition at line 12 of file [Cell.cpp](#).

```
00012         {
00013         return this->state == State::EMPTY;
00014     }
```

#### 7.3.3.6 isGrass()

```
bool Models::Cell::isGrass ()
```

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

```
00016         {
00017         return this->state == State::GRASS;
00018     }
```

#### 7.3.3.7 setBonusType()

```
void Models::Cell::setBonusType (
    BonusType b) [inline]
```

Definition at line 25 of file [Cell.h](#).

```
00025 { bonusType = b; }
```

#### 7.3.3.8 setPlayerId()

```
void Models::Cell::setPlayerId (
    int id) [inline]
```

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

```
00028 { playerId = id; }
```

#### 7.3.3.9 setState()

```
void Models::Cell::setState (
    State s) [inline]
```

Definition at line 22 of file [Cell.h](#).

```
00022 { state = s; }
```

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

- [include/models/Cell.h](#)
- [src/models/Cell.cpp](#)

## 7.4 Controllers::Game Class Reference

```
#include <Game.h>
```

### Public Member Functions

- [Game](#) ()
- [~Game](#) ()
- void [start](#) ()
- void [run](#) ()
- void [end](#) ()
- std::vector< [Models::Player](#) > const [getPlayers](#) ()
- [Models::Board](#) \* [getBoard](#) ()
- [Models::TileQueue](#) \* [getTileQueue](#) ()
- int [getCurrentRound](#) () const
- void [setCurrentRound](#) (int round)
- bool [isGameOver](#) () const
- int [getPlayerCount](#) () const
- [Models::Player](#) \* [getPlayerById](#) (int id)

### 7.4.1 Detailed Description

Definition at line 18 of file [Game.h](#).

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 Game()

```
Controllers::Game::Game ()
```

Definition at line 14 of file [Game.cpp](#).

```
00014         : board(nullptr), tileQueue(nullptr), ui(nullptr), currentRound(0), maxRounds(9) {
00015     }
```

#### 7.4.2.2 ~Game()

```
Controllers::Game::~~Game ()
```

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

```
00017     {
00018         delete board;
00019         delete tileQueue;
00020         delete ui;
00021     }
```

## 7.4.3 Member Function Documentation

### 7.4.3.1 end()

void Controllers::Game::end ()

Definition at line 45 of file [Game.cpp](#).

```
00045         {
00046
00047             calculateScores();
00048             Models::Player* winner = determineWinner();
00049
00050             if (winner != nullptr) {
00051                 std::cout << "=== WINNER ===" << std::endl;
00052                 std::cout << "Player: " << winner->getName() << " (ID: " << winner->getId() << ")" << std::endl;
00053                 std::cout << "Score: " << winner->getScore() << std::endl;
00054             }
00055     }
```

### 7.4.3.2 getBoard()

Models::Board \* Controllers::Game::getBoard () [inline]

Definition at line 55 of file [Game.h](#).

```
00055 { return board; }
```

### 7.4.3.3 getCurrentRound()

int Controllers::Game::getCurrentRound () const [inline]

Definition at line 57 of file [Game.h](#).

```
00057 { return currentRound; }
```

### 7.4.3.4 getPlayerById()

Models::Player \* Controllers::Game::getPlayerById (int id)

Definition at line 338 of file [Game.cpp](#).

```
00338         {
00339             for (auto& player : players) {
00340                 if (player.getId() == id) {
00341                     return &player;
00342                 }
00343             }
00344             return nullptr;
00345     }
```

### 7.4.3.5 getPlayerCount()

int Controllers::Game::getPlayerCount () const [inline]

Definition at line 62 of file [Game.h](#).

```
00062 { return players.size(); }
```

#### 7.4.3.6 getPlayers()

```
std::vector< Models::Player > const Controllers::Game::getPlayers () [inline]
```

Definition at line 54 of file [Game.h](#).

```
00054 { return players; }
```

#### 7.4.3.7 getTileQueue()

```
Models::TileQueue * Controllers::Game::getTileQueue () [inline]
```

Definition at line 56 of file [Game.h](#).

```
00056 { return tileQueue; }
```

#### 7.4.3.8 isGameOver()

```
bool Controllers::Game::isGameOver () const
```

Definition at line 334 of file [Game.cpp](#).

```
00334 {  
00335     return currentRound >= maxRounds;  
00336 }
```

#### 7.4.3.9 run()

```
void Controllers::Game::run ()
```

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

```
00034 {  
00035     while (!isGameOver()) {  
00036         playRound();  
00037         currentRound++;  
00038     }  
00039     finalPurchasePhase();  
00040     end();  
00041 }  
00042  
00043 }
```

#### 7.4.3.10 setCurrentRound()

```
void Controllers::Game::setCurrentRound (  
    int round) [inline]
```

Definition at line 59 of file [Game.h](#).

```
00059 { currentRound = round; }
```

### 7.4.3.11 start()

```
void Controllers::Game::start ()
```

Definition at line 23 of file [Game.cpp](#).

```
00023         {
00024
00025         ui = new Views::UI_Cli();
00026         ui->displayWelcome();
00027         initializePlayers();
00028         initializeBoard();
00029         initializeTileQueue();
00030         generateTurnOrder();
00031         placeStartingTiles();
00032     }
```

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

- include/controllers/[Game.h](#)
- src/controllers/[Game.cpp](#)

## 7.5 Utils::InputValidator Class Reference

```
#include <InputValidator.h>
```

### Public Member Functions

- `std::vector< std::string >` [getAvailableColors](#) ()
- `std::string` [selectColor](#) (int colorIndex)

### Static Public Member Functions

- static bool [isValidNumberOfPlayers](#) (int numPlayers)
- static bool [isValidPlayerName](#) (const std::string &name)
- static bool [isValidPlayerColor](#) (const std::string &color)

### 7.5.1 Detailed Description

Definition at line 12 of file [InputValidator.h](#).

### 7.5.2 Member Function Documentation

#### 7.5.2.1 getAvailableColors()

```
std::vector< std::string > Utils::InputValidator::getAvailableColors ()
```

Definition at line 26 of file [InputValidator.cpp](#).

```
00026                                     {
00027         return colors;
00028     }
```

### 7.5.2.2 isValidNumberOfPlayers()

```
bool Utils::InputValidator::isValidNumberOfPlayers (
    int numPlayers) [static]
```

Definition at line 9 of file [InputValidator.cpp](#).

```
00009
00010         return numPlayers >= 2 && numPlayers <= 9;
00011     }
```

### 7.5.2.3 isValidPlayerColor()

```
bool Utils::InputValidator::isValidPlayerColor (
    const std::string & color) [static]
```

Definition at line 17 of file [InputValidator.cpp](#).

```
00017
00018         for (int i = 0; i < color.length(); ++i) {
00019             if (!isalpha(color[i])) {
00020                 return false;
00021             }
00022         }
00023         return true;
00024     }
```

### 7.5.2.4 isValidPlayerName()

```
bool Utils::InputValidator::isValidPlayerName (
    const std::string & name) [static]
```

Definition at line 13 of file [InputValidator.cpp](#).

```
00013
00014         return !name.empty() && name.length() <= 20;
00015     }
```

### 7.5.2.5 selectColor()

```
std::string Utils::InputValidator::selectColor (
    int colorIndex)
```

Definition at line 30 of file [InputValidator.cpp](#).

```
00030
00031         if (colorIndex < 0 || colorIndex >= colors.size()) {
00032             return "";
00033         }
00034
00035         std::string selectedColor = colors[colorIndex];
00036         colors.erase(colors.begin() + colorIndex);
00037         takenColors.push_back(selectedColor);
00038
00039         return selectedColor;
00040     }
```

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

- [include/utlis/InputValidator.h](#)
- [src/utlis/InputValidator.cpp](#)

## 7.6 Utils::KeyboardInput Class Reference

```
#include <KeyboardInput.h>
```

### Static Public Member Functions

- static [KeyCode](#) [getKeyPressed\(\)](#)
- static [KeyCode](#) [checkKeyPressed\(\)](#)

### 7.6.1 Detailed Description

Definition at line 21 of file [KeyboardInput.h](#).

### 7.6.2 Member Function Documentation

#### 7.6.2.1 checkKeyPressed()

[KeyCode](#) [Utils::KeyboardInput::checkKeyPressed\(\)](#) [static]

Definition at line 23 of file [KeyboardInput.cpp](#).

```
00023                                     {
00024         if (!_kbhit()) {
00025             return KeyCode::UNKNOWN;
00026         }
00027         return getKeyPressed();
00028     }
```

#### 7.6.2.2 getKeyPressed()

[KeyCode](#) [Utils::KeyboardInput::getKeyPressed\(\)](#) [static]

Definition at line 6 of file [KeyboardInput.cpp](#).

```
00006                                     {
00007         int firstByte = _getch();
00008         if (firstByte == 224) {
00009             int secondByte = _getch();
00010             return mapSpecialKey(firstByte, secondByte);
00011         }
00012         switch (firstByte) {
00013             case 'R':
00014                 case 'r':
00015                     return KeyCode::ROTATE;
00016                 case ' ':
00017                     return KeyCode::CONFIRM;
00018             default:
00019                 return KeyCode::UNKNOWN;
00020         }
00021     }
```

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

- include/Utils/[KeyboardInput.h](#)
- src/Utils/[KeyboardInput.cpp](#)

## 7.7 Models::Player Class Reference

```
#include <Player.h>
```

### Public Member Functions

- [Player](#) (int id, std::string name, std::string color)
- int [getId](#) () const noexcept
- void [setId](#) (int newId) noexcept
- const std::string [getName](#) () const noexcept
- void [setName](#) (const std::string &newName)
- int [getScore](#) () const noexcept
- void [setScore](#) (int newScore) noexcept
- int [getExchange](#) () const noexcept
- void [setExchange](#) (int newExchange) noexcept
- const std::string & [getColor](#) () const noexcept
- void [setColor](#) (const std::string &newColor)

### 7.7.1 Detailed Description

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

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 Player()

```
Models::Player::Player (  
    int id,  
    std::string name,  
    std::string color)
```

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

```
00010                                     {  
00011     id = newId;  
00012     name = newName;  
00013     color = newColor;  
00014     exchange = 1;  
00015     score = 0;  
00016 }
```

### 7.7.3 Member Function Documentation

#### 7.7.3.1 getColor()

```
const std::string & Models::Player::getColor () const [inline], [noexcept]
```

Definition at line 35 of file [Player.h](#).

```
00035 { return color; }
```

### 7.7.3.2 getExchange()

```
int Models::Player::getExchange () const [inline], [noexcept]
```

Definition at line 32 of file [Player.h](#).

```
00032 { return exchange; }
```

### 7.7.3.3 getId()

```
int Models::Player::getId () const [inline], [noexcept]
```

Definition at line 23 of file [Player.h](#).

```
00023 { return id; }
```

### 7.7.3.4 getName()

```
const std::string Models::Player::getName () const [inline], [noexcept]
```

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

```
00026 { return name; }
```

### 7.7.3.5 getScore()

```
int Models::Player::getScore () const [inline], [noexcept]
```

Definition at line 29 of file [Player.h](#).

```
00029 { return score; }
```

### 7.7.3.6 setColor()

```
void Models::Player::setColor (  
    const std::string & newColor) [inline]
```

Definition at line 36 of file [Player.h](#).

```
00036 { color = newColor; }
```

### 7.7.3.7 setExchange()

```
void Models::Player::setExchange (  
    int newExchange) [inline], [noexcept]
```

Definition at line 33 of file [Player.h](#).

```
00033 { exchange = newExchange; }
```

#### 7.7.3.8 setId()

```
void Models::Player::setId (
    int newId) [inline], [noexcept]
```

Definition at line 24 of file [Player.h](#).

```
00024 { id = newId; }
```

#### 7.7.3.9 setName()

```
void Models::Player::setName (
    const std::string & newName) [inline]
```

Definition at line 27 of file [Player.h](#).

```
00027 { name = newName; }
```

#### 7.7.3.10 setScore()

```
void Models::Player::setScore (
    int newScore) [inline], [noexcept]
```

Definition at line 30 of file [Player.h](#).

```
00030 { score = newScore; }
```

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

- include/models/[Player.h](#)
- src/models/[Player.cpp](#)

## 7.8 Utils::PlayerResult Struct Reference

```
#include <SquareCalculator.h>
```

### Public Attributes

- int [playerID](#)
- int [playerScore](#)
- int [playerGrass](#)

### 7.8.1 Detailed Description

Definition at line 17 of file [SquareCalculator.h](#).

## 7.8.2 Member Data Documentation

### 7.8.2.1 playerGrass

```
int Utils::PlayerResult::playerGrass
```

Definition at line 20 of file [SquareCalculator.h](#).

### 7.8.2.2 playerID

```
int Utils::PlayerResult::playerID
```

Definition at line 18 of file [SquareCalculator.h](#).

### 7.8.2.3 playerScore

```
int Utils::PlayerResult::playerScore
```

Definition at line 19 of file [SquareCalculator.h](#).

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

- [include/utis/SquareCalculator.h](#)

## 7.9 Models::Position Class Reference

```
#include <Position.h>
```

### Public Member Functions

- [Position](#) ()
- [Position](#) (int x, int y)
- int const [getX](#) ()
- void [setX](#) (int x)
- int const [getY](#) ()
- void [setY](#) (int y)
- bool [operator==](#) (const [Position](#) &other) const
- bool [operator!=](#) (const [Position](#) &other) const
- bool [operator<](#) (const [Position](#) &other) const

### 7.9.1 Detailed Description

Definition at line 9 of file [Position.h](#).

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 Position() [1/2]

Models::Position::Position ()

Definition at line 8 of file [Position.cpp](#).

```
00008 : x(0), y(0) {}
```

### 7.9.2.2 Position() [2/2]

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

Definition at line 10 of file [Position.cpp](#).

```
00010 : x(x), y(y) {}
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 getX()

```
int const Models::Position::getX () [inline]
```

Definition at line 17 of file [Position.h](#).

```
00017 { return x; }
```

### 7.9.3.2 getY()

```
int const Models::Position::getY () [inline]
```

Definition at line 20 of file [Position.h](#).

```
00020 { return y; }
```

### 7.9.3.3 operator!=(())

```
bool Models::Position::operator!= (  
    const Position & other) const
```

Definition at line 16 of file [Position.cpp](#).

```
00016                                     {  
00017     return !(*this == other);  
00018 }
```

#### 7.9.3.4 operator<()

```
bool Models::Position::operator< (
    const Position & other) const
```

Definition at line 20 of file [Position.cpp](#).

```
00020                                     {
00021         if (this->x != other.x) {
00022             return this->x < other.x;
00023         }
00024         return this->y < other.y;
00025     }
```

#### 7.9.3.5 operator==(())

```
bool Models::Position::operator==(
    const Position & other) const
```

Definition at line 12 of file [Position.cpp](#).

```
00012                                     {
00013         return (this->x == other.x) && (this->y == other.y);
00014     }
```

#### 7.9.3.6 setX()

```
void Models::Position::setX (
    int x) [inline]
```

Definition at line 18 of file [Position.h](#).

```
00018 { this->x = x; }
```

#### 7.9.3.7 setY()

```
void Models::Position::setY (
    int y) [inline]
```

Definition at line 21 of file [Position.h](#).

```
00021 { this->y = y; }
```

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

- include/models/[Position.h](#)
- src/models/[Position.cpp](#)

## 7.10 Utils::Random Class Reference

```
#include <Random.h>
```

### Static Public Member Functions

- static int [getInt](#) (int min, int max)
- static int [shuffle](#) ()

#### 7.10.1 Detailed Description

Definition at line 9 of file [Random.h](#).

#### 7.10.2 Member Function Documentation

##### 7.10.2.1 [getInt\(\)](#)

```
int Utils::Random::getInt (
    int min,
    int max) [static]
```

Definition at line 9 of file [Random.cpp](#).

```
00009      {
00010          static std::random_device rd;
00011          static std::mt19937 gen(rd());
00012          std::uniform_int_distribution<> dis(min, max);
00013          return dis(gen);
00014      }
```

##### 7.10.2.2 [shuffle\(\)](#)

```
int Utils::Random::shuffle () [static]
```

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

- include/Utils/[Random.h](#)
- src/Utils/[Random.cpp](#)

## 7.11 Utils::SquareCalculator Class Reference

```
#include <SquareCalculator.h>
```

### Public Member Functions

- [SquareCalculator](#) ()=default

### Static Public Member Functions

- static std::vector< [PlayerResult](#) > [rankingPlayersByScore](#) ([Models::Board](#) &board)
- static int [calculateSquare](#) ([Models::Board](#) &board, int playerID)
- static int [calculateGrass](#) ([Models::Board](#) &board, int playerID)

## 7.11.1 Detailed Description

Definition at line 24 of file [SquareCalculator.h](#).

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 SquareCalculator()

Utils::SquareCalculator::SquareCalculator () [default]

## 7.11.3 Member Function Documentation

### 7.11.3.1 calculateGrass()

```
int Utils::SquareCalculator::calculateGrass (
    Models::Board & board,
    int playerId) [static]
```

Definition at line 66 of file [SquareCalculator.cpp](#).

```
00066
00067     std::vector<std::vector<Models::Cell>> grid = board.getGrid();
00068
00069     int height = static_cast<int>(grid.size());
00070     int width = static_cast<int>(grid[0].size());
00071     int grassCount = 0;
00072
00073     for (int row = 0; row < height; row++) {
00074         for (int col = 0; col < width; col++) {
00075             Models::Cell cell = grid[row][col];
00076
00077             if (cell.getPlayerId() == playerId) {
00078                 grassCount++;
00079             }
00080         }
00081     }
00082
00083     return grassCount;
00084 }
```

### 7.11.3.2 calculateSquare()

```
int Utils::SquareCalculator::calculateSquare (
    Models::Board & board,
    int playerId) [static]
```

Definition at line 38 of file [SquareCalculator.cpp](#).

```
00038
00039     std::vector<std::vector<Models::Cell>> grid = board.getGrid();
00040     if (grid.empty() || grid[0].empty()) return 0;
00041
00042     int height = static_cast<int>(grid.size());
00043     int width = static_cast<int>(grid[0].size());
00044
00045     std::vector<std::vector<int>> dp(height, std::vector<int>(width, 0));
00046     int maxSide = 0;
00047
00048     for (int r = 0; r < height; ++r) {
00049         for (int c = 0; c < width; ++c) {
00050             if (grid[r][c].getPlayerId() == playerId) {
00051                 if (r == 0 || c == 0) {
00052                     dp[r][c] = 1;
00053                 } else {
00054                     dp[r][c] = 1 + std::min({dp[r-1][c], dp[r][c-1], dp[r-1][c-1]});
00055                 }
00056                 if (dp[r][c] > maxSide) maxSide = dp[r][c];
00057             } else {
00058                 dp[r][c] = 0;
00059             }
00060         }
00061     }
00062
00063     return maxSide;
00064 }
```

### 7.11.3.3 rankingPlayersByScore()

```
std::vector< PlayerResult > Utils::SquareCalculator::rankingPlayersByScore (
    Models::Board & board) [static]
```

Definition at line 13 of file [SquareCalculator.cpp](#).

```
00013                                                                 {
00014         std::vector<PlayerResult> results;
00015
00016         int playerCount = board.getPlayerCount();
00017
00018         for (int playerID = 0; playerID < playerCount; playerID++) {
00019             PlayerResult result{};
00020             result.playerID = playerID;
00021             result.playerScore = calculateSquare(board, playerID);
00022             result.playerGrass = calculateGrass(board, playerID);
00023
00024             results.push_back(result);
00025         }
00026
00027         std::sort(results.begin(), results.end(),
00028             [](const PlayerResult& a, const PlayerResult& b) {
00029             if (a.playerScore != b.playerScore) {
00030                 return a.playerScore > b.playerScore;
00031             }
00032             return a.playerGrass > b.playerGrass;
00033             });
00034
00035         return results;
00036     }
```

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

- [include/Utils/SquareCalculator.h](#)
- [src/Utils/SquareCalculator.cpp](#)

## 7.12 Models::Tile Class Reference

```
#include <Tile.h>
```

### Public Member Functions

- [Tile](#) (int id, std::vector< std::vector< [Cell](#) > > &pattern)
- int const [getId](#) ()
- void [setId](#) (int id)
- int const [getWidth](#) ()
- void [setWidth](#) (int w)
- int const [getHeight](#) ()
- void [setHeight](#) (int h)
- int const [getSize](#) ()
- bool [isPlaced](#) ()
- void [setPlaced](#) (bool p)
- int const [getPlayerId](#) ()
- void [setPlayerId](#) (int id)
- std::vector< std::vector< [Cell](#) > > [getPattern](#) ()
- void [rotate](#) ()
- void [flipHorizontal](#) ()
- void [setPattern](#) (std::vector< std::vector< [Cell](#) > > p)

## Static Public Member Functions

- static [Tile convertJsonToTile](#) (const nlohmann::json &j)
- static [Tile createTile](#) (int id)

### 7.12.1 Detailed Description

Definition at line 16 of file [Tile.h](#).

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 Tile()

```
Models::Tile::Tile (
    int id,
    std::vector< std::vector< Cell > > & pattern)
```

Definition at line 10 of file [Tile.cpp](#).

```
00010                                     : id(id), pattern(pattern) {
00011     this->height = pattern.size();
00012     this->width = pattern.empty() ? 0 : pattern[0].size();
00013 }
```

### 7.12.3 Member Function Documentation

#### 7.12.3.1 convertJsonToTile()

```
Tile Models::Tile::convertJsonToTile (
    const nlohmann::json & j) [static]
```

Definition at line 27 of file [Tile.cpp](#).

```
00027                                     {
00028     int id = j["id"];
00029
00030     std::vector<std::vector<Cell>> pattern;
00031     for (auto& row : j["pattern"]) {
00032         std::vector<Cell> cellRow;
00033         for (int value : row) {
00034             Cell cell;
00035             if (value == 1) {
00036                 cell.setState(State::GRASS);
00037             }
00038             cellRow.push_back(cell);
00039         }
00040         pattern.push_back(cellRow);
00041     }
00042
00043     return Tile(id, pattern);
00044 }
```

### 7.12.3.2 createTile()

```
Tile Models::Tile::createTile (
    int id) [static]
```

Definition at line 46 of file [Tile.cpp](#).

```
00046         {
00047             std::ifstream file("Tiles.json");
00048             if (!file.is_open()) {
00049                 throw std::runtime_error("Cannot open Tiles.json file");
00050             }
00051
00052             nlohmann::json data = nlohmann::json::parse(file);
00053             file.close();
00054
00055             if (data.is_null() || !data.contains("tiles")) {
00056                 throw std::runtime_error("Invalid JSON format: missing 'tiles' key");
00057             }
00058
00059             for (const auto& tileJson : data["tiles"]) {
00060                 if (tileJson["id"] == id) {
00061                     return convertJsonToTile(tileJson);
00062                 }
00063             }
00064
00065             throw std::out_of_range("Tile with id " + std::to_string(id) + " not found");
00066         }
```

### 7.12.3.3 flipHorizontal()

```
void Models::Tile::flipHorizontal ()
```

Definition at line 85 of file [Tile.cpp](#).

```
00085         {
00086             for (int y = 0; y < height; ++y) {
00087                 std::reverse(pattern[y].begin(), pattern[y].end());
00088             }
00089         }
```

### 7.12.3.4 getHeight()

```
int const Models::Tile::getHeight () [inline]
```

Definition at line 34 of file [Tile.h](#).

```
00034 { return height; }
```

### 7.12.3.5 getId()

```
int const Models::Tile::getId () [inline]
```

Definition at line 28 of file [Tile.h](#).

```
00028 { return id; }
```

### 7.12.3.6 getPattern()

```
std::vector< std::vector< Cell > > Models::Tile::getPattern () [inline]
```

Definition at line 47 of file [Tile.h](#).

```
00047 {return pattern;;}
```

### 7.12.3.7 getPlayerId()

```
int const Models::Tile::getPlayerId () [inline]
```

Definition at line 42 of file [Tile.h](#).

```
00042 { return playerId; }
```

### 7.12.3.8 getSize()

```
int const Models::Tile::getSize ()
```

Definition at line 15 of file [Tile.cpp](#).

```
00015     {
00016         for (const auto& row : pattern) {
00017             for (const auto& cell : row) {
00018                 if (cell.getState() == State::GRASS) {
00019                     size++;
00020                 }
00021             }
00022         }
00023     }
00024     return size;
00025 }
```

### 7.12.3.9 getWidth()

```
int const Models::Tile::getWidth () [inline]
```

Definition at line 31 of file [Tile.h](#).

```
00031 { return width; }
```

### 7.12.3.10 isPlaced()

```
bool Models::Tile::isPlaced () [inline]
```

Definition at line 39 of file [Tile.h](#).

```
00039 { return placed; }
```

### 7.12.3.11 rotate()

```
void Models::Tile::rotate ()
```

Definition at line 68 of file [Tile.cpp](#).

```
00068     {
00069         int oldHeight = height;
00070         int oldWidth = width;
00071
00072         std::vector<std::vector<Cell>> rotatedPattern(oldWidth, std::vector<Cell>(oldHeight));
00073
00074         for (int y = 0; y < oldHeight; ++y) {
00075             for (int x = 0; x < oldWidth; ++x) {
00076                 rotatedPattern[x][oldHeight - 1 - y] = pattern[y][x];
00077             }
00078         }
00079
00080         pattern = rotatedPattern;
00081         width = oldHeight;
00082         height = oldWidth;
00083     }
```

### 7.12.3.12 setHeight()

```
void Models::Tile::setHeight (
    int h) [inline]
```

Definition at line 35 of file [Tile.h](#).

```
00035 { height = h; }
```

### 7.12.3.13 setId()

```
void Models::Tile::setId (
    int id) [inline]
```

Definition at line 29 of file [Tile.h](#).

```
00029 { this->id = id; }
```

### 7.12.3.14 setPattern()

```
void Models::Tile::setPattern (
    std::vector< std::vector< Cell > > p) [inline]
```

Definition at line 51 of file [Tile.h](#).

```
00051 { pattern = p; }
```

### 7.12.3.15 setPlaced()

```
void Models::Tile::setPlaced (
    bool p) [inline]
```

Definition at line 40 of file [Tile.h](#).

```
00040 { placed = p; }
```

### 7.12.3.16 setPlayerId()

```
void Models::Tile::setPlayerId (
    int id) [inline]
```

Definition at line 43 of file [Tile.h](#).

```
00043 { playerId = id; }
```

### 7.12.3.17 setWidth()

```
void Models::Tile::setWidth (
    int w) [inline]
```

Definition at line 32 of file [Tile.h](#).

```
00032 { width = w; }
```

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

- [include/models/Tile.h](#)
- [src/models/Tile.cpp](#)

## 7.13 Controllers::TilePlacer Class Reference

```
#include <TilePlacer.h>
```

### Public Member Functions

- [TilePlacer](#) ([Models::Tile](#) \*tile, [Models::Board](#) \*board, int playerId)
- void [setInitialPosition](#) ()
- void [moveUp](#) ()
- void [moveDown](#) ()
- void [moveLeft](#) ()
- void [moveRight](#) ()
- void [rotateTile](#) ()
- bool [isValidPlacement](#) () const
- [Models::Position](#) [getPosition](#) () const
- [Models::Tile](#) \* [getTile](#) () const
- bool [confirmPlacement](#) ()

### 7.13.1 Detailed Description

Definition at line 13 of file [TilePlacer.h](#).

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 TilePlacer()

```
Controllers::TilePlacer::TilePlacer (
    Models::Tile * tile,
    Models::Board * board,
    int playerId)
```

Definition at line 9 of file [TilePlacer.cpp](#).

```
00010 : currentTile(tile), board(board), playerId(playerId), placementValid(false) {
00011     setInitialPosition();
00012     updateValidity();
00013 }
```

### 7.13.3 Member Function Documentation

#### 7.13.3.1 confirmPlacement()

```
bool Controllers::TilePlacer::confirmPlacement ()
```

Definition at line 65 of file [TilePlacer.cpp](#).

```
00065 {
00066     if (!placementValid) {
00067         return false;
00068     }
00069     if (board == nullptr || currentTile == nullptr) {
00070         return false;
00071     }
00072     board->placeTile(currentTile, currentPos, playerId);
00073     currentTile->setPlaced(true);
00074     currentTile->setPlayerId(playerId);
00075     return true;
00076 }
```

### 7.13.3.2 getPosition()

`Models::Position` Controllers::TilePlacer::getPosition () const [inline]

Definition at line 35 of file [TilePlacer.h](#).

```
00035 { return currentPos; }
```

### 7.13.3.3 getTile()

`Models::Tile *` Controllers::TilePlacer::getTile () const [inline]

Definition at line 37 of file [TilePlacer.h](#).

```
00037 { return currentTile; }
```

### 7.13.3.4 isValidPlacement()

`bool` Controllers::TilePlacer::isValidPlacement () const [inline]

Definition at line 33 of file [TilePlacer.h](#).

```
00033 { return placementValid; }
```

### 7.13.3.5 moveDown()

`void` Controllers::TilePlacer::moveDown ()

Definition at line 37 of file [TilePlacer.cpp](#).

```
00037 {
00038     if (currentPos.getY() < board->getHeight() - 1) {
00039         currentPos.setY(currentPos.getY() + 1);
00040         updateValidity();
00041     }
00042 }
```

### 7.13.3.6 moveLeft()

`void` Controllers::TilePlacer::moveLeft ()

Definition at line 44 of file [TilePlacer.cpp](#).

```
00044 {
00045     if (currentPos.getX() > 0) {
00046         currentPos.setX(currentPos.getX() - 1);
00047         updateValidity();
00048     }
00049 }
```

### 7.13.3.7 moveRight()

`void` Controllers::TilePlacer::moveRight ()

Definition at line 51 of file [TilePlacer.cpp](#).

```
00051 {
00052     if (currentPos.getX() < board->getWidth() - 1) {
00053         currentPos.setX(currentPos.getX() + 1);
00054         updateValidity();
00055     }
00056 }
```

### 7.13.3.8 moveUp()

void Controllers::TilePlacer::moveUp ()

Definition at line 30 of file [TilePlacer.cpp](#).

```

00030         {
00031             if (currentPos.getY() > 0) {
00032                 currentPos.setY(currentPos.getY() - 1);
00033                 updateValidity();
00034             }
00035     }

```

### 7.13.3.9 rotateTile()

void Controllers::TilePlacer::rotateTile ()

Definition at line 58 of file [TilePlacer.cpp](#).

```

00058         {
00059             if (currentTile != nullptr) {
00060                 currentTile->rotate();
00061                 updateValidity();
00062             }
00063     }

```

### 7.13.3.10 setInitialPosition()

void Controllers::TilePlacer::setInitialPosition ()

Definition at line 15 of file [TilePlacer.cpp](#).

```

00015         {
00016             int centerX = board->getWidth() / 2;
00017             int centerY = board->getHeight() / 2;
00018             currentPos.setX(centerX);
00019             currentPos.setY(centerY);
00020     }

```

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

- include/controllers/[TilePlacer.h](#)
- src/controllers/[TilePlacer.cpp](#)

## 7.14 Models::TileQueue Class Reference

```
#include <TileQueue.h>
```

### Public Member Functions

- [TileQueue](#) ()
- std::deque< [Tile](#) > & [getTiles](#) ()
- std::vector< [Tile](#) > & [getUsedTiles](#) ()
- void [loadTiles](#) ()
- void [shuffleTiles](#) ()
- void [addTile](#) ([Tile](#) t)
- void [addUsedTile](#) ([Tile](#) t)
- void [removeTile](#) ([Tile](#) t)
- void [removeUsedTile](#) ([Tile](#) t)
- [Tile](#) \* [getCurrentTile](#) ()
- [Tile](#) \* [getTileAt](#) (int index)
- std::vector< [Tile](#) > [getNextTiles](#) (int count)
- void [selectTileFromMarket](#) (int marketIndex)
- void [recycleTiles](#) ()
- bool [isEmpty](#) () const

### 7.14.1 Detailed Description

Definition at line 14 of file [TileQueue.h](#).

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 TileQueue()

```
Models::TileQueue::TileQueue ()
```

Definition at line 12 of file [TileQueue.cpp](#).

```
00012             : currentIndex(0) {  
00013     loadTiles();  
00014 }
```

### 7.14.3 Member Function Documentation

#### 7.14.3.1 addTile()

```
void Models::TileQueue::addTile (  
    Tile t)
```

Definition at line 40 of file [TileQueue.cpp](#).

```
00040             {  
00041     tiles.push_back(t);  
00042 }
```

#### 7.14.3.2 addUsedTile()

```
void Models::TileQueue::addUsedTile (  
    Tile t)
```

Definition at line 44 of file [TileQueue.cpp](#).

```
00044             {  
00045     usedTiles.push_back(t);  
00046 }
```

#### 7.14.3.3 getCurrentTile()

```
Tile * Models::TileQueue::getCurrentTile ()
```

Definition at line 66 of file [TileQueue.cpp](#).

```
00066             {  
00067     if (tiles.empty()) {  
00068         recycleTiles();  
00069         if (tiles.empty()) {  
00070             return nullptr;  
00071         }  
00072     }  
00073     return &tiles.front();  
00074 }
```

#### 7.14.3.4 getNextTiles()

```
std::vector< Tile > Models::TileQueue::getNextTiles (
    int count)
```

Definition at line 83 of file [TileQueue.cpp](#).

```
00083                                     {
00084         std::vector<Tile> nextTiles;
00085         int available = std::min(count, static_cast<int>(tiles.size()));
00086
00087         for (int i = 0; i < available; ++i) {
00088             nextTiles.push_back(tiles[i]);
00089         }
00090
00091         return nextTiles;
00092     }
```

#### 7.14.3.5 getTileAt()

```
Tile * Models::TileQueue::getTileAt (
    int index)
```

Definition at line 76 of file [TileQueue.cpp](#).

```
00076                                     {
00077         if (index < 0 || index >= tiles.size()) {
00078             return nullptr;
00079         }
00080         return &tiles[index];
00081     }
```

#### 7.14.3.6 getTiles()

```
std::deque< Tile > & Models::TileQueue::getTiles () [inline]
```

Definition at line 22 of file [TileQueue.h](#).

```
00022 { return tiles; }
```

#### 7.14.3.7 getUsedTiles()

```
std::vector< Tile > & Models::TileQueue::getUsedTiles () [inline]
```

Definition at line 23 of file [TileQueue.h](#).

```
00023 { return usedTiles; }
```

#### 7.14.3.8 isEmpty()

```
bool Models::TileQueue::isEmpty () const
```

Definition at line 114 of file [TileQueue.cpp](#).

```
00114                                     {
00115         return tiles.empty() && usedTiles.empty();
00116     }
```

### 7.14.3.9 loadTiles()

void Models::TileQueue::loadTiles ()

Definition at line 16 of file [TileQueue.cpp](#).

```

00016         {
00017             int tileId = 1;
00018             while (true) {
00019                 try {
00020                     Tile t = Tile::createTile(tileId);
00021                     addTile(t);
00022                     tileId++;
00023                 } catch (const std::out_of_range& e) {
00024                     break;
00025                 } catch (const std::runtime_error& e) {
00026                     std::cerr << "Error loading tiles: " << e.what() << std::endl;
00027                     break;
00028                 }
00029             }
00030             shuffleTiles();
00031         }

```

### 7.14.3.10 recycleTiles()

void Models::TileQueue::recycleTiles ()

Definition at line 104 of file [TileQueue.cpp](#).

```

00104         {
00105             if (!usedTiles.empty()) {
00106                 for (auto& tile : usedTiles) {
00107                     tiles.push_back(tile);
00108                 }
00109                 usedTiles.clear();
00110                 shuffleTiles();
00111             }
00112         }

```

### 7.14.3.11 removeTile()

void Models::TileQueue::removeTile (  
     Tile t)

Definition at line 48 of file [TileQueue.cpp](#).

```

00048         {
00049             for (size_t i = 0; i < tiles.size(); ++i) {
00050                 if (tiles[i].getId() == t.getId()) {
00051                     tiles.erase(tiles.begin() + i);
00052                     break;
00053                 }
00054             }
00055         }

```

### 7.14.3.12 removeUsedTile()

void Models::TileQueue::removeUsedTile (  
     Tile t)

Definition at line 57 of file [TileQueue.cpp](#).

```

00057         {
00058             for (auto it = usedTiles.begin(); it != usedTiles.end(); ++it) {
00059                 if (it->getId() == t.getId()) {
00060                     usedTiles.erase(it);
00061                     break;
00062                 }
00063             }
00064         }

```

### 7.14.3.13 selectTileFromMarket()

```
void Models::TileQueue::selectTileFromMarket (
    int marketIndex)
```

Definition at line 94 of file [TileQueue.cpp](#).

```
00094                                     {
00095         if (marketIndex < 0 || marketIndex >= tiles.size()) {
00096             return;
00097         }
00098
00099         Tile selectedTile = tiles[marketIndex];
00100         tiles.erase(tiles.begin() + marketIndex);
00101         tiles.push_front(selectedTile);
00102     }
```

### 7.14.3.14 shuffleTiles()

```
void Models::TileQueue::shuffleTiles ()
```

Definition at line 33 of file [TileQueue.cpp](#).

```
00033     {
00034         for (size_t i = tiles.size() - 1; i > 0; --i) {
00035             int j = Utils::Random::getInt(0, i);
00036             std::swap(tiles[i], tiles[j]);
00037         }
00038     }
```

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

- [include/models/TileQueue.h](#)
- [src/models/TileQueue.cpp](#)

## 7.15 Views::UI\_Cli Class Reference

```
#include <UI_Cli.h>
```

### Public Member Functions

- void [clearScreen](#) ()
- void [displayWelcome](#) ()
- void [displayBoard](#) (Models::Board &board, std::vector< Models::Player > &players)
- void [displayTile](#) (Models::Tile &tile)
- int [displayMarket](#) (std::vector< Models::Tile > &marketTiles)
- void [displayPlayer](#) (Models::Player &player)
- void [displayMessage](#) (std::string &message)
- void [displayBoardWithTile](#) (Models::Board &board, Models::Tile &tile, Models::Position &pos, int playerId, std::vector< Models::Player > &players)
- void [displayWinner](#) (std::vector< Models::Player > &players, int winnerId)
- int [askNumberOfPlayers](#) ()
- std::string [askPlayerName](#) (std::string playerName)
- std::string [askPlayerColor](#) (std::vector< std::string > &availableColors)
- void [tilePlacement](#) (Models::Tile &tile, Models::Board &board, int playerId, std::vector< Models::Player > &players)

## 7.15.1 Detailed Description

Definition at line 17 of file [UI\\_Cli.h](#).

## 7.15.2 Member Function Documentation

### 7.15.2.1 askNumberOfPlayers()

```
int Views::UI_Cli::askNumberOfPlayers ()
```

Definition at line 268 of file [UI\\_Cli.cpp](#).

```
00268         {
00269             int playerNumber = 0;
00270             while (playerNumber < 2 || playerNumber > 9) {
00271                 std::cout << "Enter number of players (2-9): ";
00272                 if (!(std::cin >> playerNumber)) {
00273                     std::cin.clear();
00274                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00275                     std::cout << "Invalid input. Please enter a number between 2 and 9 : ";
00276                     playerNumber = 0;
00277                     continue;
00278                 }
00279                 if (playerNumber < 2 || playerNumber > 9) {
00280                     std::cout << "Invalid number of players. Please enter a number between 2 and 9.";
00281                 }
00282             }
00283             return playerNumber;
00284         }
```

### 7.15.2.2 askPlayerColor()

```
std::string Views::UI_Cli::askPlayerColor (
    std::vector< std::string > & availableColors)
```

Definition at line 297 of file [UI\\_Cli.cpp](#).

```
00297         {
00298             int choice = -1;
00299
00300             while (choice < 1 || choice > availableColors.size()) {
00301                 std::cout << "\nAvailable colors:" << std::endl;
00302
00303                 #if defined(_WIN32) || defined(_WIN64)
00304                     HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00305                     CONSOLE_SCREEN_BUFFER_INFO csbi;
00306                     WORD defaultAttr = 7;
00307                     if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00308                         defaultAttr = csbi.wAttributes & (FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE
00309 | FOREGROUND_INTENSITY);
00309                     }
00310                 #endif
00311
00312                 for (size_t i = 0; i < availableColors.size(); ++i) {
00313                     #if defined(_WIN32) || defined(_WIN64)
00314                         WORD attr = mapColorStringToAttr(availableColors[i]);
00315                         SetConsoleTextAttribute(hConsole, attr);
00316                         std::cout << (i + 1) << ". " << availableColors[i];
00317                         SetConsoleTextAttribute(hConsole, defaultAttr);
00318                         std::cout << std::endl;
00319                     #else
00320                         std::cout << (i + 1) << ". " << availableColors[i] << std::endl;
00321                     #endif
00322                 }
00323
00324                 std::cout << "Choose a color (1-" << availableColors.size() << "): ";
00325                 if (!(std::cin >> choice)) {
00326                     std::cin.clear();
00327                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00328                     std::cout << "Invalid input. Please enter a number between 1 and " <<
00329 availableColors.size() << ": ";
00329                     choice = -1;
00330                     continue;
00330                 }
```

```

00331         }
00332         if (choice < 1 || choice > static_cast<int>(availableColors.size())) {
00333             std::cout << "Invalid choice. Please enter a number between 1 and " <<
availableColors.size() << ": ";
00334         }
00335     }
00336
00337     std::string selectedColor = availableColors[choice - 1];
00338     availableColors.erase(availableColors.begin() + choice - 1);
00339     return selectedColor;
00340 }

```

### 7.15.2.3 askPlayerName()

```

std::string Views::UI_Cli::askPlayerName (
    std::string playerName)

```

Definition at line 286 of file [UI\\_Cli.cpp](#).

```

00286         {
00287             std::cout << "Enter name for " << playerName << ": ";
00288             std::string name;
00289             while (!(std::cin > name)) {
00290                 std::cin.clear();
00291                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00292                 std::cout << "Invalid input. Enter name for " << playerName << ": ";
00293             }
00294             return name;
00295         }

```

### 7.15.2.4 clearScreen()

```

void Views::UI_Cli::clearScreen ()

```

Definition at line 28 of file [UI\\_Cli.cpp](#).

```

00028         {
00029             std::cout << "\033[H\033[J";
00030             std::cout << std::endl;
00031         }

```

### 7.15.2.5 displayBoard()

```

void Views::UI_Cli::displayBoard (
    Models::Board & board,
    std::vector< Models::Player > & players)

```

Definition at line 86 of file [UI\\_Cli.cpp](#).

```

00086         {
00087
00088             int width = board.getWidth();
00089             int height = board.getHeight();
00090
00091
00092             #if defined(_WIN32) || defined(_WIN64)
00093                 HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00094                 CONSOLE_SCREEN_BUFFER_INFO csbi;
00095                 WORD defaultAttr = 7;
00096                 if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00097                     defaultAttr = csbi.wAttributes & (FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE |
FOREGROUND_INTENSITY);
00098                 }
00099             #endif
00100
00101             for (int y = 0; y < height; ++y) {
00102                 for (int x = 0; x < width; ++x) {
00103                     Models::Cell cell = board.getGrid()[y][x];
00104                     if (cell.getState() == Models::State::GRASS) {
00105                         int pid = cell.getPlayerId();

```

```

00106 #if defined(_WIN32) || defined(_WIN64)
00107     if (pid >= 0 && pid < static_cast<int>(players.size())) {
00108         WORD attr = mapColorStringToAttr(players[pid].getColor());
00109         SetConsoleTextAttribute(hConsole, attr);
00110     }
00111 #endif
00112     }
00113
00114     std::cout << renderCell(cell);
00115
00116 #if defined(_WIN32) || defined(_WIN64)
00117     //reset color après affichage d'une cellule d'herbe
00118     if (cell.getState() == Models::State::GRASS) {
00119         SetConsoleTextAttribute(hConsole, defaultAttr);
00120     }
00121 #endif
00122     }
00123     std::cout << std::endl;
00124 }
00125 }

```

### 7.15.2.6 displayBoardWithTile()

```

void Views::UI_Cli::displayBoardWithTile (
    Models::Board & board,
    Models::Tile & tile,
    Models::Position & pos,
    int playerId,
    std::vector< Models::Player > & players)

```

Definition at line 140 of file [UI\\_Cli.cpp](#).

```

00140
00141 {
00142     int boardWidth = board.getWidth();
00143     int boardHeight = board.getHeight();
00144     int tileWidth = tile.getWidth();
00145     int tileHeight = tile.getHeight();
00146     int tileX = pos.getX();
00147     int tileY = pos.getY();
00148
00149     std::vector<std::vector<Models::Cell>> tempGrid = board.getGrid();
00150
00151     for (int ty = 0; ty < tileHeight; ++ty) {
00152         for (int tx = 0; tx < tileWidth; ++tx) {
00153             int boardX = tileX + tx;
00154             int boardY = tileY + ty;
00155
00156             if (boardX >= 0 && boardX < boardWidth && boardY >= 0 && boardY < boardHeight) {
00157                 if (tile.getPattern()[ty][tx].getState() == Models::State::GRASS) {
00158                     tempGrid[boardY][boardX] = tile.getPattern()[ty][tx];
00159                     tempGrid[boardY][boardX].setPlayerId(playerId);
00160                 }
00161             }
00162         }
00163     }
00164 #if defined(_WIN32) || defined(_WIN64)
00165     HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00166     CONSOLE_SCREEN_BUFFER_INFO csbi;
00167     WORD defaultAttr = 7;
00168     if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00169         defaultAttr = csbi.wAttributes & (BACKGROUND_RED | BACKGROUND_GREEN | BACKGROUND_BLUE |
00170         BACKGROUND_INTENSITY);
00171     }
00172 #endif
00173     for (int y = 0; y < boardHeight; ++y) {
00174         for (int x = 0; x < boardWidth; ++x) {
00175             Models::Cell cell = tempGrid[y][x];
00176
00177             bool isTempTile = false;
00178             if (x >= tileX && x < tileX + tileWidth && y >= tileY && y < tileY + tileHeight) {
00179                 int tx = x - tileX;
00180                 int ty = y - tileY;
00181                 if (tile.getPattern()[ty][tx].getState() == Models::State::GRASS) {
00182                     isTempTile = true;
00183                 }
00184             }
00185         }
00186     }
00187 }

```

```

00186             //colorie la cellule si c'est de l'herbe en fonction de l'id du joueur
00187             if (cell.getState() == Models::State::GRASS) {
00188                 int pid = cell.getPlayerId();
00189 #if defined(_WIN32) || defined(_WIN64)
00190                 if (pid >= 0 && pid < static_cast<int>(players.size())) {
00191                     WORD attr = mapColorStringToAttr(players[pid].getColor());
00192                     SetConsoleTextAttribute(hConsole, attr);
00193                 }
00194 #endif
00195             }
00196
00197             std::cout << renderCell(cell, isTempTile);
00198
00199 #if defined(_WIN32) || defined(_WIN64)
00200             if (cell.getState() == Models::State::GRASS) {
00201                 SetConsoleTextAttribute(hConsole, defaultAttr);
00202             }
00203 #endif
00204         }
00205         std::cout << std::endl;
00206     }
00207 }

```

### 7.15.2.7 displayMarket()

```

int Views::UI_Cli::displayMarket (
    std::vector< Models::Tile > & marketTiles)

```

Definition at line 217 of file [UI\\_Cli.cpp](#).

```

00217                                     {
00218         int selectedIndex = 0;
00219
00220         while (true) {
00221             clearScreen();
00222             std::cout << "=== TILE MARKET ===" << std::endl;
00223             std::cout << "Choose a tile (UP/DOWN to navigate, SPACE to select)" << std::endl;
00224             std::cout << std::endl;
00225
00226             for (size_t i = 0; i < marketTiles.size(); ++i) {
00227                 if (i == selectedIndex) {
00228                     std::cout << ">> [" << (i + 1) << "]" << std::endl;
00229                 } else {
00230                     std::cout << "    [" << (i + 1) << "]" << std::endl;
00231                 }
00232                 displayTile(marketTiles[i]);
00233                 std::cout << std::endl;
00234             }
00235
00236             Utils::KeyCode key = Utils::KeyboardInput::getKeyPressed();
00237
00238             switch (key) {
00239                 case Utils::KeyCode::UP:
00240                     if (selectedIndex > 0) selectedIndex--;
00241                     break;
00242                 case Utils::KeyCode::DOWN:
00243                     if (selectedIndex < marketTiles.size() - 1) selectedIndex++;
00244                     break;
00245                 case Utils::KeyCode::CONFIRM:
00246                     return selectedIndex;
00247                 default:
00248                     break;
00249             }
00250         }
00251     }

```

### 7.15.2.8 displayMessage()

```

void Views::UI_Cli::displayMessage (
    std::string & message)

```

Definition at line 254 of file [UI\\_Cli.cpp](#).

```

00254                                     {
00255         std::cout << message << std::endl;
00256     }

```

### 7.15.2.9 displayPlayer()

```
void Views::UI_Cli::displayPlayer (
    Models::Player & player)
```

Definition at line 209 of file [UI\\_Cli.cpp](#).

```
00209
00210         std::cout << "Player " << player.getId() << ": " << player.getName()
00211             << " | Color: " << player.getColor()
00212             << " | Score: " << player.getScore()
00213             << " | Exchanges: " << player.getExchange()
00214             << std::endl;
00215     }
```

### 7.15.2.10 displayTile()

```
void Views::UI_Cli::displayTile (
    Models::Tile & tile)
```

Definition at line 127 of file [UI\\_Cli.cpp](#).

```
00127
00128         int height = tile.getHeight();
00129         int width = tile.getWidth();
00130
00131         for (int y = 0; y < height; ++y) {
00132             for (int x = 0; x < width; ++x) {
00133                 Models::Cell cell = tile.getPattern()[y][x];
00134                 std::cout << renderCell(cell);
00135             }
00136             std::cout << std::endl;
00137         }
00138     }
```

### 7.15.2.11 displayWelcome()

```
void Views::UI_Cli::displayWelcome ()
```

Definition at line 33 of file [UI\\_Cli.cpp](#).

```
00033
00034         std::cout << "
00035         std::cout << "=====
00036         std::cout << "      Welcome to the Laying Grass
00037         std::cout << "=====
00038         std::cout << "
00039     }
```

### 7.15.2.12 displayWinner()

```
void Views::UI_Cli::displayWinner (
    std::vector< Models::Player > & players,
    int winnerId)
```

Definition at line 258 of file [UI\\_Cli.cpp](#).

```
00258
00259         for (auto& player : players) {
00260             if (player.getId() == winnerId) {
00261                 std::cout << "Congratulations " << player.getName() << "! You are the winner with a score
of " << player.getScore() << "!" << std::endl;
00262                 return;
00263             }
00264         }
00265     }
```

## 7.15.2.13 tilePlacement()

```
void Views::UI_Cli::tilePlacement (
    Models::Tile & tile,
    Models::Board & board,
    int playerId,
    std::vector< Models::Player > & players)
```

Definition at line 342 of file [UI\\_Cli.cpp](#).

```
00342 {
00343     Controllers::TilePlacer placer(&tile, &board, playerId);
00344     bool placementConfirmed = false;
00345
00346     while (!placementConfirmed) {
00347         clearScreen();
00348         std::cout << std::endl;
00349
00350         Models::Position pos = placer.getPosition();
00351         bool isValid = placer.isValidPlacement();
00352
00353         // Display status
00354         if (isValid) {
00355             std::cout << "=== PLACEMENT VALID ===" << std::endl;
00356         } else {
00357             std::cout << "=== PLACEMENT INVALID ===" << std::endl;
00358         }
00359         std::cout << "Position: (" << pos.getX() << ", " << pos.getY() << ")" << std::endl;
00360         std::cout << std::endl;
00361
00362         displayBoardWithTile(board, tile, pos, playerId, players);
00363
00364         std::cout << std::endl;
00365         std::cout << "Controls:" << std::endl;
00366         std::cout << "  Arrow Keys: Move tile" << std::endl;
00367         std::cout << "    R: Rotate" << std::endl;
00368         std::cout << "  SPACE: Confirm" << std::endl;
00369         std::cout << std::endl;
00370         std::cout << "Press a key..." << std::endl;
00371
00372         Utils::KeyCode key = Utils::KeyboardInput::getKeyPressed();
00373
00374         switch (key) {
00375             case Utils::KeyCode::UP:
00376                 placer.moveUp();
00377                 break;
00378             case Utils::KeyCode::DOWN:
00379                 placer.moveDown();
00380                 break;
00381             case Utils::KeyCode::LEFT:
00382                 placer.moveLeft();
00383                 break;
00384             case Utils::KeyCode::RIGHT:
00385                 placer.moveRight();
00386                 break;
00387             case Utils::KeyCode::ROTATE:
00388                 placer.rotateTile();
00389                 break;
00390             case Utils::KeyCode::CONFIRM:
00391                 if (placer.isValidPlacement()) {
00392                     placer.confirmPlacement();
00393                     placementConfirmed = true;
00394                     clearScreen();
00395                     std::cout << "Tile placed successfully!" << std::endl;
00396                 } else {
00397                     clearScreen();
00398                     std::cout << "[ERROR] Cannot place tile at this position!" << std::endl;
00399                     std::cout << "Press any key to continue..." << std::endl;
00400                     _getch();
00401                 }
00402                 break;
00403             default:
00404                 break;
00405         }
00406     }
00407 }
00408 }
```

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

- [include/views/UI\\_Cli.h](#)
- [src/views/UI\\_Cli.cpp](#)

## 7.16 Views::UI\_Qt Class Reference

```
#include <UI_Qt.h>
```

### 7.16.1 Detailed Description

Definition at line 9 of file [UI\\_Qt.h](#).

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

- [include/views/UI\\_Qt.h](#)

# Chapter 8

## File Documentation

### 8.1 include/controllers/Game.h File Reference

```
#include <vector>
#include <memory>
#include "../models/Board.h"
#include "../models/Player.h"
#include "../models/Tile.h"
#include "../models/TileQueue.h"
#include "../models/Position.h"
#include "../views/UI_Cli.h"
```

#### Classes

- class [Controllers::Game](#)

#### Namespaces

- namespace [Controllers](#)

### 8.2 Game.h

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_GAME_H
00006 #define LAYING_GRASS_PROJECT_GAME_H
00007
00008 #include <vector>
00009 #include <memory>
00010 #include "../models/Board.h"
00011 #include "../models/Player.h"
00012 #include "../models/Tile.h"
00013 #include "../models/TileQueue.h"
00014 #include "../models/Position.h"
00015 #include "../views/UI_Cli.h"
00016
00017 namespace Controllers {
```

```

00018     class Game {
00019     private:
00020         std::vector<Models::Player> players;
00021         Models::Board* board;
00022         Models::TileQueue* tileQueue;
00023         Views::UI_Cli* ui;
00024         int currentRound;
00025         int maxRounds;
00026         std::vector<int> turnOrder;
00027
00028         void initializePlayers();
00029         void initializeBoard();
00030         void initializeTileQueue();
00031         void generateTurnOrder();
00032
00033         void placeStartingTiles();
00034         void playRound();
00035         void handlePlayerTurn(Models::Player& player);
00036         bool tryPlaceTile(Models::Player& player, Models::Tile* tile);
00037
00038         void applyExchangeBonus(Models::Player& player);
00039         void applyStoneBonus(Models::Player& player);
00040         void applyRobberyBonus(Models::Player& player);
00041
00042         void finalPurchasePhase();
00043         void calculateScores();
00044         Models::Player* determineWinner();
00045
00046     public:
00047         Game();
00048         ~Game();
00049
00050         void start();
00051         void run();
00052         void end();
00053
00054         std::vector<Models::Player> const getPlayers() { return players; }
00055         Models::Board* getBoard() { return board; }
00056         Models::TileQueue* getTileQueue() { return tileQueue; }
00057         int getCurrentRound() const { return currentRound; }
00058
00059         void setCurrentRound(int round) { currentRound = round; }
00060
00061         bool isGameOver() const;
00062         int getPlayerCount() const { return players.size(); }
00063         Models::Player* getPlayerById(int id);
00064     };
00065 } // Controllers
00066
00067 #endif //LAYING_GRASS_PROJECT_GAME_H

```

## 8.3 include/controllers/TilePlacer.h File Reference

```

#include "../models/Tile.h"
#include "../models/Board.h"
#include "../models/Position.h"

```

### Classes

- class [Controllers::TilePlacer](#)

### Namespaces

- namespace [Controllers](#)

## 8.4 TilePlacer.h

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

```

00001 //
00002 // Created by antoi 09/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_TILEPLACER_H
00006 #define LAYING_GRASS_PROJECT_TILEPLACER_H
00007
00008 #include "../models/Tile.h"
00009 #include "../models/Board.h"
00010 #include "../models/Position.h"
00011
00012 namespace Controllers {
00013     class TilePlacer {
00014     private:
00015         Models::Tile* currentTile;
00016         Models::Board* board;
00017         Models::Position currentPos;
00018         int playerId;
00019         bool placementValid;
00020         void updateValidity();
00021
00022     public:
00023         TilePlacer(Models::Tile* tile, Models::Board* board, int playerId);
00024         void setInitialPosition();
00025
00026         void moveUp();
00027         void moveDown();
00028         void moveLeft();
00029         void moveRight();
00030
00031         void rotateTile();
00032
00033         bool isValidPlacement() const { return placementValid; }
00034
00035         Models::Position getPosition() const { return currentPos; }
00036
00037         Models::Tile* getTile() const { return currentTile; }
00038
00039         bool confirmPlacement();
00040     };
00041 } // Controllers
00042
00043 #endif //LAYING_GRASS_PROJECT_TILEPLACER_H

```

## 8.5 include/models/Board.h File Reference

```

#include <vector>
#include <cmath>
#include "Cell.h"
#include "Position.h"
#include "Tile.h"

```

### Classes

- class [Models::Board](#)

### Namespaces

- namespace [Models](#)

## 8.6 Board.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_BOARD_H
00006 #define LAYING_GRASS_PROJECT_BOARD_H
00007 #include <vector>
00008 #include <cmath>
00009 #include "Cell.h"
00010 #include "Position.h"
00011 #include "Tile.h"
00012
00013 namespace Models {
00014     class Board {
00015     private:
00016         std::vector<std::vector<Cell>> grid;
00017         int width;
00018         int height;
00019         int playersNumber;
00020
00021         int exchangeCount;
00022         int stoneCount;
00023         int stealCount;
00024
00025     public:
00026         Board(int playersNumber);
00027         int const getPlayersNumber(){ return playersNumber; }
00028
00029         int const getWidth(){ return width; }
00030         void setWidth(int w){ width = w; }
00031
00032         int const getHeight(){ return height; }
00033         void setHeight(int h){ height = h; }
00034
00035         std::vector<std::vector<Cell>> const getGrid(){ return grid; }
00036         void setGrid(std::vector<std::vector<Cell>> g){ grid = g; }
00037
00038         Cell* getCell(Position& pos) {
00039             if (!isInsideBoard(pos)) {
00040                 return nullptr;
00041             }
00042             return &grid[pos.getY()][pos.getX()];
00043         };
00044
00045         void placeBonus();
00046         void removeBonus(Position& pos, int playerId);
00047         void placeStone(Position& pos);
00048         bool hasStoneAt(Position& pos) const;
00049
00050         bool isCellTouchingSomething(Position& pos, State state, int playerId);
00051         bool isTileTouchingGrass(Tile* tile, Position& pos, int playerId);
00052         bool isTouchingWall(Position& pos);
00053         bool isInsideBoard(Position& pos);
00054
00055         bool canPlaceTile(Tile* tile, Position& pos, int playerId);
00056         void placeTile(Tile* tile, Position& pos, int playerId);
00057
00058         std::vector<Position> checkBonusAcquisition(Tile* tile, Position& pos, int playerId);
00059
00060     };
00061 } // Models
00062
00063 #endif //LAYING_GRASS_PROJECT_BOARD_H

```

## 8.7 include/models/BonusSquare.h File Reference

```
#include "Types.h"
```

### Classes

- class [Models::BonusSquare](#)

## Namespaces

- namespace [Models](#)

## 8.8 BonusSquare.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004 #include "Types.h"
00005 #ifndef LAYING_GRASS_PROJECT_BONUSQUARE_H
00006 #define LAYING_GRASS_PROJECT_BONUSQUARE_H
00007
00008 namespace Models {
00009     class BonusSquare {
00010     private:
00011         BonusType bonusType;
00012     public:
00013         BonusSquare();
00014
00015         BonusType const getBonusType(){ return bonusType;}
00016         void setBonusType(BonusType b){ bonusType = b; }
00017
00018         bool isExchange();
00019         bool isStone();
00020         bool isSteal();
00021
00022     };
00023 } // Models
00024
00025 #endif //LAYING_GRASS_PROJECT_BONUSQUARE_H

```

## 8.9 include/models/Cell.h File Reference

```
#include "Types.h"
```

## Classes

- class [Models::Cell](#)

## Namespaces

- namespace [Models](#)

## 8.10 Cell.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "Types.h"
00006
00007 #ifndef LAYING_GRASS_PROJECT_CELL_H
00008 #define LAYING_GRASS_PROJECT_CELL_H
00009
00010 namespace Models {

```

```

00011     class Cell {
00012     private:
00013         State state = State::EMPTY;
00014         BonusType bonusType = BonusType::NONE;
00015         int playerId = -1;
00016
00017     public:
00018         Cell();
00019
00020         State const getState() const { return state; }
00021         void setState(State s){ state = s; }
00022
00023         BonusType const getBonusType() const { return bonusType; }
00024         void setBonusType(BonusType b) { bonusType = b; }
00025
00026         int const getPlayerId() const { return playerId; }
00027         void setPlayerId(int id) { playerId = id; }
00028
00029         bool isEmpty();
00030         bool isGrass();
00031         bool isBonus();
00032
00033     };
00034
00035 };
00036 } // Models
00037
00038 #endif //LAYING_GRASS_PROJECT_CELL_H

```

## 8.11 include/models/Player.h File Reference

```
#include <string>
```

### Classes

- class [Models::Player](#)

### Namespaces

- namespace [Models](#)

## 8.12 Player.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_PLAYER_H
00006 #define LAYING_GRASS_PROJECT_PLAYER_H
00007 #include <string>
00008
00009 namespace Models {
00010     class Board;
00011     class Player {
00012     private :
00013         int id;
00014         std::string name;
00015         int score=0;
00016         int exchange = 1;
00017         std::string color;
00018
00019     public:
00020         Player(int id,std::string name, std::string color);
00021
00022

```

```

00023     int getId() const noexcept { return id; }
00024     void setId(int newId) noexcept { id = newId; }
00025
00026     const std::string getName() const noexcept { return name; }
00027     void setName(const std::string &newName) { name = newName; }
00028
00029     int getScore() const noexcept { return score; }
00030     void setScore(int newScore) noexcept { score = newScore; }
00031
00032     int getExchange() const noexcept { return exchange; }
00033     void setExchange(int newExchange) noexcept { exchange = newExchange; }
00034
00035     const std::string &getColor() const noexcept { return color; }
00036     void setColor(const std::string &newColor) { color = newColor; }
00037
00038
00039
00040         //destructeur !!!!
00041     };
00042 } // Models
00043 #endif //LAYING_GRASS_PROJECT_PLAYER_H

```

## 8.13 include/models/Position.h File Reference

### Classes

- class [Models::Position](#)

### Namespaces

- namespace [Models](#)

## 8.14 Position.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_POSITION_H
00006 #define LAYING_GRASS_PROJECT_POSITION_H
00007
00008 namespace Models {
00009     class Position {
00010     private:
00011         int x = 0;
00012         int y = 0;
00013     public:
00014         Position();
00015         Position(int x, int y);
00016
00017         int const getX(){ return x; }
00018         void setX(int x){ this->x = x; }
00019
00020         int const getY(){ return y; }
00021         void setY(int y){ this->y = y; }
00022
00023         bool operator==(const Position& other) const;
00024         bool operator!=(const Position& other) const;
00025         bool operator<(const Position& other) const;
00026     };
00027 } // Models
00028
00029 #endif //LAYING_GRASS_PROJECT_POSITION_H

```

## 8.15 include/models/Tile.h File Reference

```
#include <vector>
#include <nlohmann/json.hpp>
#include <fstream>
#include "Position.h"
#include "Cell.h"
```

### Classes

- class [Models::Tile](#)

### Namespaces

- namespace [Models](#)

## 8.16 Tile.h

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include <vector>
00006 #include <nlohmann/json.hpp>
00007 #include <fstream>
00008
00009 #include "Position.h"
00010 #include "Cell.h"
00011
00012 #ifndef LAYING_GRASS_PROJECT_TILE_H
00013 #define LAYING_GRASS_PROJECT_TILE_H
00014
00015 namespace Models {
00016     class Tile {
00017     private:
00018         int id;
00019         std::vector<std::vector<Cell>> pattern;
00020         int width;
00021         int height;
00022         int size = 0;
00023         bool placed = false;
00024         int playerId = -1;
00025     public:
00026         Tile(int id, std::vector<std::vector<Cell>>& pattern);
00027
00028         int const getId(){ return id; }
00029         void setId(int id){ this->id = id; }
00030
00031         int const getWidth(){ return width; }
00032         void setWidth(int w){ width = w; }
00033
00034         int const getHeight(){ return height; }
00035         void setHeight(int h){ height = h; }
00036
00037         int const getSize();
00038
00039         bool isPlaced(){ return placed; }
00040         void setPlaced(bool p){ placed = p; }
00041
00042         int const getPlayerId(){ return playerId; }
00043         void setPlayerId(int id){ playerId = id; }
00044
00045         static Tile convertJsonToTile(const nlohmann::json& j);
00046         static Tile createTile(int id);
00047         std::vector<std::vector<Cell>> getPattern(){return pattern;};
00048
00049         void rotate();
00050         void flipHorizontal();
00051         void setPattern(std::vector<std::vector<Cell>> p) { pattern = p; }
00052     };
00053 } // Models
00054
00055 #endif //LAYING_GRASS_PROJECT_TILE_H
```

## 8.17 include/models/TileQueue.h File Reference

```
#include "Tile.h"
#include <vector>
#include <iostream>
#include <deque>
```

### Classes

- class [Models::TileQueue](#)

### Namespaces

- namespace [Models](#)

## 8.18 TileQueue.h

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "Tile.h"
00006 #include <vector>
00007 #include <iostream>
00008 #include <deque>
00009
00010 #ifndef LAYING_GRASS_PROJECT_TILEQUEUE_H
00011 #define LAYING_GRASS_PROJECT_TILEQUEUE_H
00012
00013 namespace Models {
00014     class TileQueue {
00015     private:
00016         std::deque<Tile> tiles;
00017         std::vector<Tile> usedTiles;
00018         int currentIndex;
00019     public:
00020         TileQueue();
00021
00022         std::deque<Tile>& getTiles() { return tiles; }
00023         std::vector<Tile>& getUsedTiles() { return usedTiles; }
00024
00025         void loadTiles();
00026         void shuffleTiles();
00027         void addTile(Tile t);
00028         void addUsedTile(Tile t);
00029
00030         void removeTile(Tile t);
00031         void removeUsedTile(Tile t);
00032
00033         Tile* getCurrentTile();
00034         Tile* getTileAt(int index);
00035         std::vector<Tile> getNextTiles(int count);
00036
00037         void selectTileFromMarket(int marketIndex);
00038         void recycleTiles();
00039         bool isEmpty() const;
00040     };
00041 } // Models
00042
00043 #endif //LAYING_GRASS_PROJECT_TILEQUEUE_H
```

## 8.19 include/models/Types.h File Reference

### Namespaces

- namespace [Models](#)

### Enumerations

- enum class [Models::State](#) { [Models::EMPTY](#) , [Models::GRASS](#) , [Models::BONUS](#) }
- enum class [Models::BonusType](#) { [Models::NONE](#) , [Models::EXCHANGE](#) , [Models::STONE](#) , [Models::STEAL](#) }

## 8.20 Types.h

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

```
00001 //
00002 // Created by antoi on 03/11/2025.
00003 //
00004 #ifndef LAYING_GRASS_PROJECT_TYPES_H
00005 #define LAYING_GRASS_PROJECT_TYPES_H
00006
00007 namespace Models {
00008     enum class State {EMPTY, GRASS, BONUS};
00009     enum class BonusType {NONE, EXCHANGE, STONE, STEAL};
00010 }
00011
00012 #endif //LAYING_GRASS_PROJECT_TYPES_H
```

## 8.21 include/utils/InputValidator.h File Reference

```
#include <iostream>
#include <vector>
```

### Classes

- class [Utils::InputValidator](#)

### Namespaces

- namespace [Utils](#)

## 8.22 InputValidator.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_INPUTVALIDATOR_H
00006 #define LAYING_GRASS_PROJECT_INPUTVALIDATOR_H
00007
00008 #include <iostream>
00009 #include <vector>
00010
00011 namespace Utils {
00012     class InputValidator {
00013     private:
00014         std::vector<std::string> colors = {"white", "light_blue", "dark_blue", "yellow", "red",
"purple", "pink", "brown", "green" };
00015         std::vector<std::string> takenColors;
00016     public:
00017         static bool isValidNumberOfPlayers(int numPlayers);
00018         static bool isValidPlayerName(const std::string& name);
00019         static bool isValidPlayerColor(const std::string& color);
00020
00021         std::vector<std::string> getAvailableColors();
00022         std::string selectColor(int colorIndex);
00023     };
00024 } // Utils
00025
00026 #endif //LAYING_GRASS_PROJECT_INPUTVALIDATOR_H

```

## 8.23 include/Utils/KeyboardInput.h File Reference

```
#include <conio.h>
```

### Classes

- class [Utils::KeyboardInput](#)

### Namespaces

- namespace [Utils](#)

### Enumerations

- enum class [Utils::KeyCode](#) {  
[Utils::UP](#) , [Utils::DOWN](#) , [Utils::LEFT](#) , [Utils::RIGHT](#) ,  
[Utils::ROTATE](#) , [Utils::CONFIRM](#) , [Utils::UNKNOWN](#) }

## 8.24 KeyboardInput.h

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

```

00001 //
00002 // Created by antoi on 09/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_KEYBOARDINPUT_H
00006 #define LAYING_GRASS_PROJECT_KEYBOARDINPUT_H
00007
00008 #include <conio.h>
00009
00010 namespace Utils {
00011     enum class KeyCode {
00012         UP,
00013         DOWN,
00014         LEFT,
00015         RIGHT,
00016         ROTATE,
00017         CONFIRM,
00018         UNKNOWN
00019     };
00020
00021     class KeyboardInput {
00022     public:
00023         static KeyCode getKeyPressed();
00024         static KeyCode checkKeyPressed();
00025
00026     private:
00027         static KeyCode mapSpecialKey(int firstByte, int secondByte);
00028     };
00029 } // Utils
00030
00031 #endif //LAYING_GRASS_PROJECT_KEYBOARDINPUT_H
00032
00033

```

## 8.25 include/utils/Random.h File Reference

### Classes

- class [Utils::Random](#)

### Namespaces

- namespace [Utils](#)

## 8.26 Random.h

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_RANDOM_H
00006 #define LAYING_GRASS_PROJECT_RANDOM_H
00007
00008 namespace Utils {
00009     class Random {
00010     public:
00011         static int getInt(int min, int max);
00012         static int shuffle();
00013     };
00014 } // Utils
00015
00016 #endif //LAYING_GRASS_PROJECT_RANDOM_H

```

## 8.27 include/utils/SquareCalculator.h File Reference

```
#include <optional>
#include <vector>
```

### Classes

- struct [Utils::PlayerResult](#)
- class [Utils::SquareCalculator](#)

### Namespaces

- namespace [Models](#)
- namespace [Utils](#)

## 8.28 SquareCalculator.h

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_SQUARECALCULATOR_H
00006 #define LAYING_GRASS_PROJECT_SQUARECALCULATOR_H
00007
00008 #include <optional>
00009 #include <vector>
00010
00011 namespace Models {
00012     class Board;
00013 }
00014
00015 namespace Utils {
00016
00017     struct PlayerResult {
00018         int playerId;
00019         int playerScore;
00020         int playerGrass;
00021     };
00022
00023
00024     class SquareCalculator {
00025     public:
00026         SquareCalculator() = default;
00027
00028         static std::vector<PlayerResult> rankingPlayersByScore(Models::Board& board);
00029         // on va get playerNuber depuis board et l utiliser pour classer les joueurs
00030
00031         static int calculateSquare(Models::Board& board, int playerId);
00032         // on va get playerNuber depuis board effectuer cette methode pour chaque player
00033
00034         static int calculateGrass(Models::Board& board, int playerId);
00035         // si score egal on compare le nombre de grass
00036
00037
00038     };
00039
00040 } // namespace Utils
00041
00042 #endif //LAYING_GRASS_PROJECT_SQUARECALCULATOR_H
```

## 8.29 include/views/UI\_Cli.h File Reference

```
#include <iostream>
#include <vector>
#include "models/Board.h"
#include "models/Player.h"
#include "models/Position.h"
#include <conio.h>
#include "models/TileQueue.h"
```

### Classes

- class [Views::UI\\_Cli](#)

### Namespaces

- namespace [Views](#)

## 8.30 UI\_Cli.h

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

```
00001 //
00002 // Created by antoi on 03/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_UI_CLI_H
00006 #define LAYING_GRASS_PROJECT_UI_CLI_H
00007 #include <iostream>
00008 #include <vector>
00009 #include "models/Board.h"
00010 #include "models/Player.h"
00011 #include "models/Position.h"
00012 #include <conio.h>
00013
00014 #include "models/TileQueue.h"
00015
00016 namespace Views {
00017     class UI_Cli {
00018     public:
00019         //display
00020         void clearScreen();
00021         void displayWelcome();
00022         void displayBoard(models::Board& board, std::vector<models::Player>& players);
00023         void displayTile(models::Tile& tile);
00024         int displayMarket(std::vector<models::Tile>& marketTiles);
00025         void displayPlayer(models::Player& player);
00026         void displayMessage(std::string& message);
00027         void displayBoardWithTile(models::Board& board, models::Tile& tile, models::Position& pos, int
playerId, std::vector<models::Player>& players);
00028         void displayWinner(std::vector<models::Player>& players, int winnerId);
00029
00030         //inputs
00031         int askNumberOfPlayers();
00032         std::string askPlayerName(std::string playerName);
00033         std::string askPlayerColor(std::vector<std::string>& availableColors);
00034         void tilePlacement(models::Tile& tile, models::Board& board, int playerId,
std::vector<models::Player>& players);
00035
00036     private:
00037         std::string renderCell(models::Cell& cell, bool isTempTile = false);
00038     };
00039 } // Views
00040
00041 #endif //LAYING_GRASS_PROJECT_UI_CLI_H
```

## 8.31 include/views/UI\_Qt.h File Reference

### Classes

- class [Views::UI\\_Qt](#)

### Namespaces

- namespace [Views](#)

## 8.32 UI\_Qt.h

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

```
00001 //
00002 // Created by antoi on 03/11/2025.
00003 //
00004
00005 #ifndef LAYING_GRASS_PROJECT_UI_QT_H
00006 #define LAYING_GRASS_PROJECT_UI_QT_H
00007
00008 namespace Views {
00009     class UI_Qt {
00010     };
00011 } // Views
00012
00013 #endif //LAYING_GRASS_PROJECT_UI_QT_H
```

## 8.33 src/controllers/Game.cpp File Reference

```
#include "../include/controllers/Game.h"
#include "../include/utils/Random.h"
#include "../include/utils/InputValidator.h"
#include "../include/utils/SquareCalculator.h"
#include <iostream>
#include <algorithm>
```

### Namespaces

- namespace [Controllers](#)

## 8.34 Game.cpp

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/controllers/Game.h"
00006 #include "../include/utils/Random.h"
00007 #include "../include/utils/InputValidator.h"
00008 #include "../include/utils/SquareCalculator.h"
00009 #include <iostream>
00010 #include <algorithm>
```

```

00011
00012 namespace Controllers {
00013
00014     Game::Game() : board(nullptr), tileQueue(nullptr), ui(nullptr), currentRound(0), maxRounds(9) {
00015     }
00016
00017     Game::~Game() {
00018         delete board;
00019         delete tileQueue;
00020         delete ui;
00021     }
00022
00023     void Game::start() {
00024
00025         ui = new Views::UI_Cli();
00026         ui->displayWelcome();
00027         initializePlayers();
00028         initializeBoard();
00029         initializeTileQueue();
00030         generateTurnOrder();
00031         placeStartingTiles();
00032     }
00033
00034     void Game::run() {
00035
00036         while (!isGameOver()) {
00037             playRound();
00038             currentRound++;
00039         }
00040
00041         finalPurchasePhase();
00042         end();
00043     }
00044
00045     void Game::end() {
00046
00047         calculateScores();
00048         Models::Player* winner = determineWinner();
00049
00050         if (winner != nullptr) {
00051             std::cout << "=== WINNER ===" << std::endl;
00052             std::cout << "Player: " << winner->getName() << " (ID: " << winner->getId() << ")" << std::endl;
00053             std::cout << "Score: " << winner->getScore() << std::endl;
00054         }
00055     }
00056
00057     void Game::initializePlayers() {
00058
00059         int playerCount = ui->askNumberOfPlayers();
00060
00061         Utils::InputValidator validator;
00062         std::vector<std::string> availableColors = validator.getAvailableColors();
00063
00064         for (int i = 0; i < playerCount; i++) {
00065             std::string name = ui->askPlayerName("Player " + std::to_string(i + 1));
00066             std::string color = ui->askPlayerColor(availableColors);
00067
00068             Models::Player player(i, name, color);
00069             players.push_back(player);
00070         }
00071     }
00072
00073     void Game::initializeBoard() {
00074
00075         int playerCount = players.size();
00076         board = new Models::Board(playerCount);
00077     }
00078
00079     void Game::initializeTileQueue() {
00080
00081         tileQueue = new Models::TileQueue();
00082         tileQueue->loadTiles();
00083     }
00084
00085     void Game::generateTurnOrder() {
00086
00087         turnOrder.clear();
00088         for (size_t i = 0; i < players.size(); i++) {
00089             turnOrder.push_back(static_cast<int>(i));
00090         }
00091
00092         for (int i = turnOrder.size() - 1; i > 0; --i) {
00093             int j = Utils::Random::getInt(0, i);
00094             std::swap(turnOrder[i], turnOrder[j]);
00095         }
00096     }
00097

```

```

00098
00099     std::cout << "Turn order: ";
00100     for (int id : turnOrder) {
00101         std::cout << id << " ";
00102     }
00103     std::cout << std::endl;
00104 }
00105
00106 void Game::placeStartingTiles() {
00107     for (auto& player : players) {
00108
00109         std::vector<std::vector<Models::Cell>> pattern(1, std::vector<Models::Cell>(1));
00110         pattern[0][0].setState(Models::State::GRASS);
00111         pattern[0][0].setPlayerId(player.getId());
00112
00113         Models::Tile startTile(0, pattern);
00114         startTile.setPlayerId(player.getId());
00115
00116         ui->tilePlacement(startTile, *board, player.getId(), players);
00117     }
00118 }
00119
00120
00121 void Game::playRound() {
00122     std::cout << "\n=== ROUND " << (currentRound + 1) << " / " << maxRounds << " ===" << std::endl;
00123
00124     for (int playerId : turnOrder) {
00125         Models::Player& player = players[playerId];
00126         handlePlayerTurn(player);
00127     }
00128 }
00129
00130 void Game::handlePlayerTurn(Models::Player& player) {
00131     std::cout << "\n--- Turn of " << player.getName() << " ---" << std::endl;
00132     std::cout << "Exchange coupons: " << player.getExchange() << std::endl;
00133
00134     Models::Tile* currentTile = tileQueue->getCurrentTile();
00135     if (currentTile == nullptr) {
00136         std::cout << "No more tiles available!" << std::endl;
00137         return;
00138     }
00139
00140     std::cout << "Current tile:" << std::endl;
00141     ui->displayTile(*currentTile);
00142
00143     std::cout << "Use exchange coupon to choose from market? (y/n): ";
00144     char choice;
00145     std::cin >> choice;
00146
00147     if ((choice == 'y' || choice == 'Y') && player.getExchange() > 0) {
00148         std::vector<Models::Tile> marketTiles = tileQueue->getNextTiles(5);
00149         if (!marketTiles.empty()) {
00150             int selectedIndex = ui->displayMarket(marketTiles);
00151             tileQueue->selectTileFromMarket(selectedIndex);
00152             currentTile = tileQueue->getCurrentTile();
00153             player.setExchange(player.getExchange() - 1);
00154         }
00155     }
00156 }
00157
00158 bool placed = tryPlaceTile(player, currentTile);
00159
00160 if (placed) {
00161     Models::Position lastPos(0, 0);
00162     std::vector<Models::Position> acquiredBonuses = board->checkBonusAcquisition(currentTile,
lastPos, player.getId());
00163
00164     for (auto& bonusPos : acquiredBonuses) {
00165         Models::Cell* bonusCell = board->getCell(bonusPos);
00166         if (bonusCell != nullptr) {
00167             Models::BonusType bonusType = bonusCell->getBonusType();
00168             board->removeBonus(bonusPos, player.getId());
00169
00170             switch (bonusType) {
00171                 case Models::BonusType::EXCHANGE:
00172                     applyExchangeBonus(player);
00173                     break;
00174                 case Models::BonusType::STONE:
00175                     applyStoneBonus(player);
00176                     break;
00177                 case Models::BonusType::STEAL:
00178                     applyRobberyBonus(player);
00179                     break;
00180                 default:
00181                     break;
00182             }
00183         }
00184     }
00185 }

```

```

00184         }
00185
00186         tileQueue->addUsedTile(*currentTile);
00187         tileQueue->removeTile(*currentTile);
00188     } else {
00189         std::cout << "Cannot place tile! Turn lost." << std::endl;
00190         tileQueue->addUsedTile(*currentTile);
00191         tileQueue->removeTile(*currentTile);
00192     }
00193 }
00194
00195 bool Game::tryPlaceTile(Models::Player& player, Models::Tile* tile) {
00196
00197     if (tile == nullptr || board == nullptr) {
00198         return false;
00199     }
00200     ui->displayBoard(*board, players);
00201     ui->tilePlacement(*tile, *board, player.getId(), players);
00202     return tile->isPlaced();
00203 }
00204
00205
00206 void Game::applyExchangeBonus(Models::Player& player) {
00207     player.setExchange(player.getExchange() + 1);
00208     std::cout << "Player " << player.getName() << " received 1 exchange coupon!" << std::endl;
00209 }
00210
00211 void Game::applyStoneBonus(Models::Player& player) {
00212     std::cout << "Player " << player.getName() << " can place a stone tile!" << std::endl;
00213     std::cout << "Enter stone position (x y): ";
00214     int x, y;
00215     std::cin >> x >> y;
00216
00217     Models::Position stonePos(x, y);
00218
00219     if (board->isInsideBoard(stonePos) && !board->isTouchingWall(stonePos)) {
00220         Models::Cell* cell = board->getCell(stonePos);
00221         if (cell != nullptr && cell->getState() == Models::State::EMPTY) {
00222             board->placeStone(stonePos);
00223             std::cout << "Stone placed at (" << x << ", " << y << ")!" << std::endl;
00224         } else {
00225             std::cout << "Invalid position for stone placement!" << std::endl;
00226         }
00227     } else {
00228         std::cout << "Cannot place stone at that position!" << std::endl;
00229     }
00230 }
00231
00232 void Game::applyRobberyBonus(Models::Player& player) {
00233     std::cout << "Player " << player.getName() << " can steal a tile from another player!" <<
std::endl;
00234     std::cout << "Available players to steal from:" << std::endl;
00235     for (auto& p : players) {
00236         if (p.getId() != player.getId()) {
00237             std::cout << "  Player " << p.getId() << ": " << p.getName() << std::endl;
00238         }
00239     }
00240
00241     std::cout << "Choose player ID to steal from: ";
00242     int targetPlayerId;
00243     std::cin >> targetPlayerId;
00244
00245     if (targetPlayerId == player.getId()) {
00246         std::cout << "Cannot steal from yourself!" << std::endl;
00247         return;
00248     }
00249
00250     Models::Player* targetPlayer = getPlayerById(targetPlayerId);
00251     if (targetPlayer == nullptr) {
00252         std::cout << "Invalid player ID!" << std::endl;
00253         return;
00254     }
00255     ui->displayBoard(*board, players);
00256     std::cout << "Enter position of tile to steal (x y): ";
00257     int x, y;
00258     std::cin >> x >> y;
00259
00260     Models::Position targetPos(x, y);
00261     Models::Cell* cell = board->getCell(targetPos);
00262
00263     if (cell != nullptr && cell->getState() == Models::State::GRASS &&
00264         cell->getPlayerId() == targetPlayerId) {
00265         cell->setPlayerId(player.getId());
00266         std::cout << "Tile stolen successfully!" << std::endl;
00267     } else {
00268         std::cout << "Invalid position or tile doesn't belong to target player!" << std::endl;
00269     }

```

```

00270     }
00271
00272     void Game::finalPurchasePhase() {
00273         std::cout << "\n=== FINAL PURCHASE PHASE ===" << std::endl;
00274
00275         for (auto& player : players) {
00276             std::cout << "\nPlayer " << player.getName() << " has " << player.getExchange() << " coupons."
<< std::endl;
00277
00278             while (player.getExchange() > 0) {
00279                 std::cout << "Buy a 1x1 tile? (y/n): ";
00280                 char choice;
00281                 std::cin >> choice;
00282
00283                 if (choice == 'y' || choice == 'Y') {
00284                     std::vector<std::vector<Models::Cell> pattern(1, std::vector<Models::Cell>(1));
00285                     pattern[0][0].setState(Models::State::GRASS);
00286                     pattern[0][0].setPlayerId(player.getId());
00287
00288                     Models::Tile purchasedTile(-1, pattern); // ID -1 for purchased tiles
00289                     purchasedTile.setPlayerId(player.getId());
00290                     ui->tilePlacement(purchasedTile, *board, player.getId(), players);
00291
00292                     player.setExchange(player.getExchange() - 1);
00293                     std::cout << "1x1 tile purchased and placed!" << std::endl;
00294                 } else {
00295                     break;
00296                 }
00297             }
00298         }
00299     }
00300
00301     void Game::calculateScores() {
00302
00303         std::vector<Utils::PlayerResult> results =
00304         Utils::SquareCalculator::rankingPlayersByScore(*board);
00305
00306         for (auto& result : results) {
00307             Models::Player* player = getPlayerById(result.playerID);
00308             if (player != nullptr) {
00309                 player->setScore(result.playerScore);
00310                 std::cout << "Player " << player->getName()
00311                     << " - Largest square: " << result.playerScore
00312                     << " - Grass count: " << result.playerGrass << std::endl;
00313             }
00314         }
00315
00316         Models::Player* Game::determineWinner() {
00317
00318             if (players.empty()) {
00319                 return nullptr;
00320             }
00321
00322             Models::Player* winner = &players[0];
00323
00324             for (auto& player : players) {
00325                 if (player.getScore() > winner->getScore()) {
00326                     winner = &player;
00327                 }
00328             }
00329
00330             return winner;
00331         }
00332
00333         // Utility methods
00334         bool Game::isGameOver() const {
00335             return currentRound >= maxRounds;
00336         }
00337
00338         Models::Player* Game::getPlayerById(int id) {
00339             for (auto& player : players) {
00340                 if (player.getId() == id) {
00341                     return &player;
00342                 }
00343             }
00344             return nullptr;
00345         }
00346     }
00347 } // Controllers

```

## 8.35 src/controllers/TilePlacer.cpp File Reference

```
#include "../..../include/controllers/TilePlacer.h"
```

### Namespaces

- namespace [Controllers](#)

## 8.36 TilePlacer.cpp

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

```
00001 //
00002 // Created by antoi 09/11/2025.
00003 //
00004
00005 #include "../..../include/controllers/TilePlacer.h"
00006
00007 namespace Controllers {
00008
00009     TilePlacer::TilePlacer(Models::Tile* tile, Models::Board* board, int playerId)
00010         : currentTile(tile), board(board), playerId(playerId), placementValid(false) {
00011         setInitialPosition();
00012         updateValidity();
00013     }
00014
00015     void TilePlacer::setInitialPosition() {
00016         int centerX = board->getWidth() / 2;
00017         int centerY = board->getHeight() / 2;
00018         currentPos.setX(centerX);
00019         currentPos.setY(centerY);
00020     }
00021
00022     void TilePlacer::updateValidity() {
00023         if (currentTile == nullptr || board == nullptr) {
00024             placementValid = false;
00025             return;
00026         }
00027         placementValid = board->canPlaceTile(currentTile, currentPos, playerId);
00028     }
00029
00030     void TilePlacer::moveUp() {
00031         if (currentPos.getY() > 0) {
00032             currentPos.setY(currentPos.getY() - 1);
00033             updateValidity();
00034         }
00035     }
00036
00037     void TilePlacer::moveDown() {
00038         if (currentPos.getY() < board->getHeight() - 1) {
00039             currentPos.setY(currentPos.getY() + 1);
00040             updateValidity();
00041         }
00042     }
00043
00044     void TilePlacer::moveLeft() {
00045         if (currentPos.getX() > 0) {
00046             currentPos.setX(currentPos.getX() - 1);
00047             updateValidity();
00048         }
00049     }
00050
00051     void TilePlacer::moveRight() {
00052         if (currentPos.getX() < board->getWidth() - 1) {
00053             currentPos.setX(currentPos.getX() + 1);
00054             updateValidity();
00055         }
00056     }
00057
00058     void TilePlacer::rotateTile() {
00059         if (currentTile != nullptr) {
00060             currentTile->rotate();
00061             updateValidity();
00062         }
00063     }
00064 }
```

```

00063     }
00064
00065     bool TilePlacer::confirmPlacement() {
00066         if (!placementValid) {
00067             return false;
00068         }
00069
00070         if (board == nullptr || currentTile == nullptr) {
00071             return false;
00072         }
00073
00074         board->placeTile(currentTile, currentPos, playerId);
00075         currentTile->setPlaced(true);
00076         currentTile->setPlayerId(playerId);
00077
00078         return true;
00079     }
00080
00081 } // Controllers
00082

```

## 8.37 src/main.cpp File Reference

```

#include <iostream>
#include "controllers/Game.h"

```

### Functions

- int [main](#) ()

### 8.37.1 Function Documentation

#### 8.37.1.1 main()

```
int main ()
```

Definition at line 4 of file [main.cpp](#).

```

00004     {
00005         Controllers::Game game;
00006         game.start();
00007         game.run();
00008         game.end();
00009
00010         return 0;
00011     }

```

## 8.38 main.cpp

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

```

00001 #include <iostream>
00002 #include "controllers/Game.h"
00003
00004 int main() {
00005     Controllers::Game game;
00006     game.start();
00007     game.run();
00008     game.end();
00009
00010     return 0;
00011 }

```

## 8.39 src/models/Board.cpp File Reference

```
#include "../include/models/Board.h"
#include "../include/utils/Random.h"
#include <cmath>
```

### Namespaces

- namespace [Models](#)

## 8.40 Board.cpp

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/models/Board.h"
00006 #include "../include/utils/Random.h"
00007 #include <cmath>
00008
00009 namespace Models {
00010     Board::Board(int playersNumber) {
00011         this->playersNumber = playersNumber;
00012         this->exchangeCount = std::ceil(1.5 * playersNumber);
00013         this->stoneCount = std::ceil(0.5 * playersNumber);
00014         this->stealCount = playersNumber;
00015         if (playersNumber >= 2 && playersNumber <= 4) {
00016             this->width = 20;
00017             this->height = 20;
00018         } else if (playersNumber >= 5 && playersNumber <= 9) {
00019             this->width = 30;
00020             this->height = 30;
00021         }
00022         grid.resize(height, std::vector<Cell>(width, Cell()));
00023         placeBonus();
00024     };
00025
00026     void Board::placeBonus() {
00027         while (exchangeCount != 0 || stoneCount != 0 || stealCount != 0) {
00028             int x = Utils::Random::getInt(0, width - 1);
00029             int y = Utils::Random::getInt(0, height - 1);
00030             Position pos(x, y);
00031
00032             if (grid[y][x].getState() == State::EMPTY && !isTouchingWall(pos) &&
!isCellTouchingSomething(pos, State::BONUS, -1)) {
00033                 if (exchangeCount > 0) {
00034                     grid[y][x].setState(State::BONUS);
00035                     grid[y][x].setBonusType(BonusType::EXCHANGE);
00036                     exchangeCount--;
00037                 } else if (stoneCount > 0) {
00038                     grid[y][x].setState(State::BONUS);
00039                     grid[y][x].setBonusType(BonusType::STONE);
00040                     stoneCount--;
00041                 } else if (stealCount > 0) {
00042                     grid[y][x].setState(State::BONUS);
00043                     grid[y][x].setBonusType(BonusType::STEAL);
00044                     stealCount--;
00045                 }
00046             }
00047         }
00048     }
00049
00050     bool Board::isCellTouchingSomething(Position& pos, State state, int playerId = -1) {
00051         int x = pos.getX();
00052         int y = pos.getY();
00053
00054         int directions[4][2] = {
00055             {0, -1},
00056             {0, 1},
00057             {-1, 0},
00058             {1, 0}
```

```

00059     };
00060
00061     for (auto& dir : directions) {
00062         int checkX = x + dir[0];
00063         int checkY = y + dir[1];
00064
00065         if (checkX >= 0 && checkX < width && checkY >= 0 && checkY < height) {
00066             if (state == State::GRASS) {
00067                 if (grid[checkY][checkX].getState() == State::GRASS &&
00068                     grid[checkY][checkX].getPlayerId() != playerId &&
00069                     grid[checkY][checkX].getPlayerId() != -1) {
00070                     return true;
00071                 }
00072             } else {
00073                 if (grid[checkY][checkX].getState() == state) {
00074                     return true;
00075                 }
00076             }
00077         }
00078     }
00079     return false;
00080 }
00081
00082
00083 bool Board::isTileTouchingGrass(Tile* tile, Position& pos, int playerId) {
00084     const auto& pattern = tile->getPattern();
00085     int tileHeight = tile->getHeight();
00086     int tileWidth = tile->getWidth();
00087     int tileSize = tile->getSize();
00088     int x = pos.getX();
00089     int y = pos.getY();
00090
00091     while (tileSize > 0) {
00092         for (int ty = 0; ty < tileHeight; ++ty) {
00093             for (int tx = 0; tx < tileWidth; ++tx) {
00094                 if (pattern[ty][tx].getState() == State::GRASS) {
00095                     int boardX = x + tx;
00096                     int boardY = y + ty;
00097                     Position cellPos(boardX, boardY);
00098                     if (isCellTouchingSomething(cellPos, State::GRASS, playerId)) {
00099                         return true;
00100                     }
00101                     tileSize--;
00102                 }
00103             }
00104         }
00105     }
00106     return false;
00107 }
00108
00109 bool Board::isTouchingWall(Position& pos) {
00110     int x = pos.getX();
00111     int y = pos.getY();
00112
00113     return (x == 0 || x == width - 1 || y == 0 || y == height - 1);
00114 }
00115
00116 bool Board::isInsideBoard(Position& pos) {
00117     int x = pos.getX();
00118     int y = pos.getY();
00119     return (x >= 0 && x < width && y >= 0 && y < height);
00120 }
00121
00122
00123 bool Board::canPlaceTile(Tile* tile, Position& pos, int playerId) {
00124     const auto& pattern = tile->getPattern();
00125     int tileHeight = tile->getHeight();
00126     int tileWidth = tile->getWidth();
00127
00128     bool touchingOwnGrass = false;
00129
00130     for (int ty = 0; ty < tileHeight; ++ty) {
00131         for (int tx = 0; tx < tileWidth; ++tx) {
00132             if (pattern[ty][tx].getState() == State::GRASS) {
00133                 int boardX = pos.getX() + tx;
00134                 int boardY = pos.getY() + ty;
00135                 Position cellPos(boardX, boardY);
00136
00137                 if (!isInsideBoard(cellPos)) {
00138                     return false;
00139                 }
00140
00141                 if (grid[boardY][boardX].getState() != State::EMPTY) {
00142                     return false;
00143                 }
00144
00145                 if (isCellTouchingSomething(cellPos, State::GRASS, playerId)) {

```

```

00146         return false;
00147     }
00148
00149     int directions[4][2] = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
00150     for (auto& dir : directions) {
00151         int checkX = boardX + dir[0];
00152         int checkY = boardY + dir[1];
00153
00154         if (checkX >= 0 && checkX < width && checkY >= 0 && checkY < height) {
00155             if (grid[checkY][checkX].getState() == State::GRASS &&
00156                 grid[checkY][checkX].getPlayerId() == playerId) {
00157                 touchingOwnGrass = true;
00158                 break;
00159             }
00160         }
00161     }
00162 }
00163 }
00164 }
00165
00166 bool isFirstPlacement = true;
00167 for (int y = 0; y < height; ++y) {
00168     for (int x = 0; x < width; ++x) {
00169         if (grid[y][x].getState() == State::GRASS &&
00170             grid[y][x].getPlayerId() == playerId) {
00171             isFirstPlacement = false;
00172             break;
00173         }
00174     }
00175     if (!isFirstPlacement) break;
00176 }
00177 return isFirstPlacement || touchingOwnGrass;
00178 }
00179
00180 void Board::placeTile(Tile* tile, Position& pos, int playerId) {
00181     const auto& pattern = tile->getPattern();
00182     int tileHeight = tile->getHeight();
00183     int tileWidth = tile->getWidth();
00184
00185     for (int ty = 0; ty < tileHeight; ++ty) {
00186         for (int tx = 0; tx < tileWidth; ++tx) {
00187             if (pattern[ty][tx].getState() == State::GRASS) {
00188                 int boardX = pos.getX() + tx;
00189                 int boardY = pos.getY() + ty;
00190
00191                 grid[boardY][boardX].setState(State::GRASS);
00192                 grid[boardY][boardX].setPlayerId(playerId);
00193             }
00194         }
00195     }
00196 }
00197
00198 std::vector<Position> Board::checkBonusAcquisition(Tile* tile, Position& pos, int playerId) {
00199     std::vector<Position> acquiredBonuses;
00200
00201     for (int y = 1; y < height - 1; ++y) {
00202         for (int x = 1; x < width - 1; ++x) {
00203             if (grid[y][x].getState() == State::BONUS) {
00204                 bool surroundedTop = grid[y-1][x].getState() == State::GRASS &&
00205                     grid[y-1][x].getPlayerId() == playerId;
00206                 bool surroundedBottom = grid[y+1][x].getState() == State::GRASS &&
00207                     grid[y+1][x].getPlayerId() == playerId;
00208                 bool surroundedLeft = grid[y][x-1].getState() == State::GRASS &&
00209                     grid[y][x-1].getPlayerId() == playerId;
00210                 bool surroundedRight = grid[y][x+1].getState() == State::GRASS &&
00211                     grid[y][x+1].getPlayerId() == playerId;
00212
00213                 if (surroundedTop && surroundedBottom && surroundedLeft && surroundedRight) {
00214                     acquiredBonuses.push_back(Position(x, y));
00215                 }
00216             }
00217         }
00218     }
00219
00220     return acquiredBonuses;
00221 }
00222
00223 void Board::removeBonus(Position& pos, int playerId) {
00224     if (isInsideBoard(pos)) {
00225         int x = pos.getX();
00226         int y = pos.getY();
00227         if (grid[y][x].getState() == State::BONUS) {
00228             grid[y][x].setState(State::GRASS);
00229             grid[y][x].setPlayerId(playerId);
00230             grid[y][x].setBonusType(BonusType::NONE);
00231         }
00232     }

```

```

00233     }
00234
00235     void Board::placeStone(Position& pos) {
00236         if (isInsideBoard(pos) && !isTouchingWall(pos)) {
00237             int x = pos.getX();
00238             int y = pos.getY();
00239             if (grid[y][x].getState() == State::EMPTY) {
00240                 grid[y][x].setState(State::BONUS);
00241                 grid[y][x].setBonusType(BonusType::STONE);
00242             }
00243         }
00244     }
00245
00246     bool Board::hasStoneAt(Position& pos) const {
00247         if (pos.getX() < 0 || pos.getX() >= width || pos.getY() < 0 || pos.getY() >= height) {
00248             return false;
00249         }
00250         return grid[pos.getY()][pos.getX()].getState() == State::BONUS &&
00251             grid[pos.getY()][pos.getX()].getBonusType() == BonusType::STONE;
00252     }
00253
00254 } // Models

```

## 8.41 src/models/BonusSquare.cpp File Reference

```
#include "../..//include/models/BonusSquare.h"
```

### Namespaces

- namespace [Models](#)

## 8.42 BonusSquare.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../..//include/models/BonusSquare.h"
00006
00007 namespace Models {
00008     bool BonusSquare::isExchange() {
00009         return this->bonusType == BonusType::EXCHANGE;
00010     }
00011
00012     bool BonusSquare::isStone() {
00013         return this->bonusType == BonusType::STONE;
00014     }
00015
00016     bool BonusSquare::isSteal() {
00017         return this->bonusType == BonusType::STEAL;
00018     }
00019 } // Models

```

## 8.43 src/models/Cell.cpp File Reference

```
#include "../..//include/models/Cell.h"
```

## Namespaces

- namespace [Models](#)

## 8.44 Cell.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/models/Cell.h"
00006
00007 namespace Models {
00008
00009     Cell::Cell() {}
00010
00011
00012     bool Cell::isEmpty() {
00013         return this->state == State::EMPTY;
00014     }
00015
00016     bool Cell::isGrass() {
00017         return this->state == State::GRASS;
00018     }
00019
00020     bool Cell::isBonus() {
00021         return this->state == State::BONUS;
00022     }
00023
00024 } // Models

```

## 8.45 src/models/Player.cpp File Reference

```

#include "../include/models/Player.h"
#include "../include/models/Board.h"
#include <vector>

```

## Namespaces

- namespace [Models](#)

## 8.46 Player.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/models/Player.h"
00006 #include "../include/models/Board.h"
00007 #include <vector>
00008
00009 namespace Models {
00010     Player::Player(int newId, std::string newName, std::string newColor) {
00011         id = newId;
00012         name = newName;
00013         color = newColor;
00014         exchange = 1;
00015         score = 0;
00016     }
00017 }

```

## 8.47 src/models/Position.cpp File Reference

```
#include "../..//include/models/Position.h"
```

### Namespaces

- namespace [Models](#)

## 8.48 Position.cpp

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

```
00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../..//include/models/Position.h"
00006
00007 namespace Models {
00008     Position::Position() : x(0), y(0) {}
00009
00010     Position::Position(int x, int y) : x(x), y(y) {}
00011
00012     bool Position::operator==(const Position &other) const {
00013         return (this->x == other.x) && (this->y == other.y);
00014     }
00015
00016     bool Position::operator!=(const Position &other) const {
00017         return !(*this == other);
00018     }
00019
00020     bool Position::operator<(const Position &other) const {
00021         if (this->x != other.x) {
00022             return this->x < other.x;
00023         }
00024         return this->y < other.y;
00025     }
00026
00027
00028 } // Models
```

## 8.49 src/models/Tile.cpp File Reference

```
#include "../..//include/models/Tile.h"
#include <stdexcept>
#include <algorithm>
```

### Namespaces

- namespace [Models](#)

## 8.50 Tile.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/models/Tile.h"
00006 #include <stdexcept>
00007 #include <algorithm>
00008
00009 namespace Models {
00010     Tile::Tile(int id, std::vector<std::vector<Cell>>& pattern) : id(id), pattern(pattern) {
00011         this->height = pattern.size();
00012         this->width = pattern.empty() ? 0 : pattern[0].size();
00013     }
00014
00015     int const Tile::getSize() {
00016         for (const auto& row : pattern) {
00017             for (const auto& cell : row) {
00018                 if (cell.getState() == State::GRASS) {
00019                     size++;
00020                 }
00021             }
00022         }
00023         return size;
00024     }
00025
00026     Tile Tile::convertJsonToTile(const nlohmann::json &j) {
00027         int id = j["id"];
00028
00029         std::vector<std::vector<Cell>> pattern;
00030         for (auto& row : j["pattern"]) {
00031             std::vector<Cell> cellRow;
00032             for (int value : row) {
00033                 Cell cell;
00034                 if (value == 1) {
00035                     cell.setState(State::GRASS);
00036                 }
00037                 cellRow.push_back(cell);
00038             }
00039             pattern.push_back(cellRow);
00040         }
00041         return Tile(id, pattern);
00042     }
00043
00044     Tile Tile::createTile(int id) {
00045         std::ifstream file("Tiles.json");
00046         if (!file.is_open()) {
00047             throw std::runtime_error("Cannot open Tiles.json file");
00048         }
00049
00050         nlohmann::json data = nlohmann::json::parse(file);
00051         file.close();
00052
00053         if (data.is_null() || !data.contains("tiles")) {
00054             throw std::runtime_error("Invalid JSON format: missing 'tiles' key");
00055         }
00056
00057         for (const auto& tileJson : data["tiles"]) {
00058             if (tileJson["id"] == id) {
00059                 return convertJsonToTile(tileJson);
00060             }
00061         }
00062
00063         throw std::out_of_range("Tile with id " + std::to_string(id) + " not found");
00064     }
00065
00066     void Tile::rotate() {
00067         int oldHeight = height;
00068         int oldWidth = width;
00069
00070         std::vector<std::vector<Cell>> rotatedPattern(oldWidth, std::vector<Cell>(oldHeight));
00071
00072         for (int y = 0; y < oldHeight; ++y) {
00073             for (int x = 0; x < oldWidth; ++x) {
00074                 rotatedPattern[x][oldHeight - 1 - y] = pattern[y][x];
00075             }
00076         }
00077
00078         pattern = rotatedPattern;
00079         width = oldHeight;
00080         height = oldWidth;
00081     }
00082

```

```

00083     }
00084
00085     void Tile::flipHorizontal() {
00086         for (int y = 0; y < height; ++y) {
00087             std::reverse(pattern[y].begin(), pattern[y].end());
00088         }
00089     }
00090
00091 } // Models

```

## 8.51 src/models/TileQueue.cpp File Reference

```

#include "../include/models/TileQueue.h"
#include "../include/utils/Random.h"
#include <iostream>
#include <stdexcept>
#include <algorithm>

```

### Namespaces

- namespace [Models](#)

## 8.52 TileQueue.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/models/TileQueue.h"
00006 #include "../include/utils/Random.h"
00007 #include <iostream>
00008 #include <stdexcept>
00009 #include <algorithm>
00010
00011 namespace Models {
00012     TileQueue::TileQueue() : currentIndex(0) {
00013         loadTiles();
00014     }
00015
00016     void TileQueue::loadTiles() {
00017         int tileId = 1;
00018         while (true) {
00019             try {
00020                 Tile t = Tile::createTile(tileId);
00021                 addTile(t);
00022                 tileId++;
00023             } catch (const std::out_of_range& e) {
00024                 break;
00025             } catch (const std::runtime_error& e) {
00026                 std::cerr << "Error loading tiles: " << e.what() << std::endl;
00027                 break;
00028             }
00029         }
00030         shuffleTiles();
00031     }
00032
00033     void TileQueue::shuffleTiles() {
00034         for (size_t i = tiles.size() - 1; i > 0; --i) {
00035             int j = Utils::Random::getInt(0, i);
00036             std::swap(tiles[i], tiles[j]);
00037         }
00038     }
00039
00040     void TileQueue::addTile(Tile t) {
00041         tiles.push_back(t);
00042     }

```

```

00043
00044 void TileQueue::addUsedTile(Tile t) {
00045     usedTiles.push_back(t);
00046 }
00047
00048 void TileQueue::removeTile(Tile t) {
00049     for (size_t i = 0; i < tiles.size(); ++i) {
00050         if (tiles[i].getId() == t.getId()) {
00051             tiles.erase(tiles.begin() + i);
00052             break;
00053         }
00054     }
00055 }
00056
00057 void TileQueue::removeUsedTile(Tile t) {
00058     for (auto it = usedTiles.begin(); it != usedTiles.end(); ++it) {
00059         if (it->getId() == t.getId()) {
00060             usedTiles.erase(it);
00061             break;
00062         }
00063     }
00064 }
00065
00066 Tile* TileQueue::getCurrentTile() {
00067     if (tiles.empty()) {
00068         recycleTiles();
00069         if (tiles.empty()) {
00070             return nullptr;
00071         }
00072     }
00073     return &tiles.front();
00074 }
00075
00076 Tile* TileQueue::getTileAt(int index) {
00077     if (index < 0 || index >= tiles.size()) {
00078         return nullptr;
00079     }
00080     return &tiles[index];
00081 }
00082
00083 std::vector<Tile> TileQueue::getNextTiles(int count) {
00084     std::vector<Tile> nextTiles;
00085     int available = std::min(count, static_cast<int>(tiles.size()));
00086
00087     for (int i = 0; i < available; ++i) {
00088         nextTiles.push_back(tiles[i]);
00089     }
00090
00091     return nextTiles;
00092 }
00093
00094 void TileQueue::selectTileFromMarket(int marketIndex) {
00095     if (marketIndex < 0 || marketIndex >= tiles.size()) {
00096         return;
00097     }
00098
00099     Tile selectedTile = tiles[marketIndex];
00100     tiles.erase(tiles.begin() + marketIndex);
00101     tiles.push_front(selectedTile);
00102 }
00103
00104 void TileQueue::recycleTiles() {
00105     if (!usedTiles.empty()) {
00106         for (auto& tile : usedTiles) {
00107             tiles.push_back(tile);
00108         }
00109         usedTiles.clear();
00110         shuffleTiles();
00111     }
00112 }
00113
00114 bool TileQueue::isEmpty() const {
00115     return tiles.empty() && usedTiles.empty();
00116 }
00117
00118 } // Models

```

## 8.53 src/utls/InputValidator.cpp File Reference

```

#include "../include/utls/InputValidator.h"
#include <vector>

```

## Namespaces

- namespace [Utils](#)

## 8.54 InputValidator.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/Utils/InputValidator.h"
00006 #include <vector>
00007
00008 namespace Utils {
00009     bool InputValidator::isValidNumberOfPlayers(int numPlayers) {
00010         return numPlayers >= 2 && numPlayers <= 9;
00011     }
00012
00013     bool InputValidator::isValidPlayerName(const std::string &name) {
00014         return !name.empty() && name.length() <= 20;
00015     }
00016
00017     bool InputValidator::isValidPlayerColor(const std::string &color) {
00018         for (int i = 0; i < color.length(); ++i) {
00019             if (!isalpha(color[i])) {
00020                 return false;
00021             }
00022         }
00023         return true;
00024     }
00025
00026     std::vector<std::string> InputValidator::getAvailableColors() {
00027         return colors;
00028     }
00029
00030     std::string InputValidator::selectColor(int colorIndex) {
00031         if (colorIndex < 0 || colorIndex >= colors.size()) {
00032             return "";
00033         }
00034
00035         std::string selectedColor = colors[colorIndex];
00036         colors.erase(colors.begin() + colorIndex);
00037         takenColors.push_back(selectedColor);
00038
00039         return selectedColor;
00040     }
00041 } // Utils

```

## 8.55 src/Utils/KeyboardInput.cpp File Reference

```

#include "../include/Utils/KeyboardInput.h"
#include <iostream>
#include <windows.h>

```

## Namespaces

- namespace [Utils](#)

## 8.56 KeyboardInput.cpp

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

```

00001 #include "../include/Utils/KeyboardInput.h"
00002 #include <iostream>
00003 #include <windows.h>
00004
00005 namespace Utils {
00006     KeyCode KeyboardInput::getKeyPressed() {
00007         int firstByte = _getch();
00008         if (firstByte == 224) {
00009             int secondByte = _getch();
00010             return mapSpecialKey(firstByte, secondByte);
00011         }
00012         switch (firstByte) {
00013             case 'R':
00014             case 'r':
00015                 return KeyCode::ROTATE;
00016             case ' ':
00017                 return KeyCode::CONFIRM;
00018             default:
00019                 return KeyCode::UNKNOWN;
00020         }
00021     }
00022
00023     KeyCode KeyboardInput::checkKeyPressed() {
00024         if (!_kbhit()) {
00025             return KeyCode::UNKNOWN;
00026         }
00027         return getKeyPressed();
00028     }
00029
00030     KeyCode KeyboardInput::mapSpecialKey(int firstByte, int secondByte) {
00031         if (firstByte == 224) {
00032             switch (secondByte) {
00033                 case 72: // UP arrow
00034                     return KeyCode::UP;
00035                 case 80: // DOWN arrow
00036                     return KeyCode::DOWN;
00037                 case 75: // LEFT arrow
00038                     return KeyCode::LEFT;
00039                 case 77: // RIGHT arrow
00040                     return KeyCode::RIGHT;
00041                 default:
00042                     return KeyCode::UNKNOWN;
00043             }
00044         }
00045     }
00046 } // Utils

```

## 8.57 src/Utils/Random.cpp File Reference

```

#include "../include/Utils/Random.h"
#include <random>

```

### Namespaces

- namespace [Utils](#)

## 8.58 Random.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004

```

```

00005 #include "../include/utls/Random.h"
00006 #include <random>
00007
00008 namespace Utlis {
00009     int Random::getInt(int min, int max) {
00010         static std::random_device rd;
00011         static std::mt19937 gen(rd());
00012         std::uniform_int_distribution<> dis(min, max);
00013         return dis(gen);
00014     }
00015 } // Utlis

```

## 8.59 src/utls/SquareCalculator.cpp File Reference

```

#include "../include/utls/SquareCalculator.h"
#include "../include/models/Board.h"
#include <algorithm>

```

### Namespaces

- namespace [Utlis](#)

## 8.60 SquareCalculator.cpp

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

```

00001 //
00002 // Created by antoi on 02/11/2025.
00003 //
00004
00005 #include "../include/utls/SquareCalculator.h"
00006 #include "../include/models/Board.h"
00007
00008 #include <algorithm>
00009
00010
00011 namespace Utlis {
00012
00013     std::vector<PlayerResult> SquareCalculator::rankingPlayersByScore(models::Board& board) {
00014         std::vector<PlayerResult> results;
00015
00016         int playerCount = board.getPlayersNumber();
00017
00018         for (int playerID = 0; playerID < playerCount; playerID++) {
00019             PlayerResult result{};
00020             result.playerID = playerID;
00021             result.playerScore = calculateSquare(board, playerID);
00022             result.playerGrass = calculateGrass(board, playerID);
00023
00024             results.push_back(result);
00025         }
00026
00027         std::sort(results.begin(), results.end(),
00028             [](const PlayerResult& a, const PlayerResult& b) {
00029                 if (a.playerScore != b.playerScore) {
00030                     return a.playerScore > b.playerScore;
00031                 }
00032                 return a.playerGrass > b.playerGrass;
00033             });
00034
00035         return results;
00036     }
00037
00038     int SquareCalculator::calculateSquare(models::Board& board, int playerID) {
00039         std::vector<std::vector<models::Cell>> grid = board.getGrid();
00040         if (grid.empty() || grid[0].empty()) return 0;
00041
00042         int height = static_cast<int>(grid.size());
00043         int width = static_cast<int>(grid[0].size());

```

```

00044
00045     std::vector<std::vector<int> dp(height, std::vector<int>(width, 0));
00046     int maxSide = 0;
00047
00048     for (int r = 0; r < height; ++r) {
00049         for (int c = 0; c < width; ++c) {
00050             if (grid[r][c].getPlayerId() == playerID) {
00051                 if (r == 0 || c == 0) {
00052                     dp[r][c] = 1;
00053                 } else {
00054                     dp[r][c] = 1 + std::min({dp[r-1][c], dp[r][c-1], dp[r-1][c-1]});
00055                 }
00056                 if (dp[r][c] > maxSide) maxSide = dp[r][c];
00057             } else {
00058                 dp[r][c] = 0;
00059             }
00060         }
00061     }
00062
00063     return maxSide;
00064 }
00065
00066 int SquareCalculator::calculateGrass(Models::Board& board, int playerID) {
00067     std::vector<std::vector<Models::Cell> grid = board.getGrid();
00068
00069     int height = static_cast<int>(grid.size());
00070     int width = static_cast<int>(grid[0].size());
00071     int grassCount = 0;
00072
00073     for (int row = 0; row < height; row++) {
00074         for (int col = 0; col < width; col++) {
00075             Models::Cell cell = grid[row][col];
00076
00077             if (cell.getPlayerId() == playerID) {
00078                 grassCount++;
00079             }
00080         }
00081     }
00082
00083     return grassCount;
00084 }
00085
00086 } // Utils

```

## 8.61 src/views/UI\_Cli.cpp File Reference

```

#include "../include/views/UI_Cli.h"
#include "../include/utils/KeyboardInput.h"
#include "../include/controllers/TilePlacer.h"
#include <iostream>

```

### Namespaces

- namespace [Views](#)

## 8.62 UI\_Cli.cpp

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

```

00001 //
00002 // Created by antoi on 03/11/2025.
00003 //
00004
00005 #include "../include/views/UI_Cli.h"
00006 #include "../include/utils/KeyboardInput.h"
00007 #include "../include/controllers/TilePlacer.h"
00008 #include <iostream>
00009 #if defined(_WIN32) || defined(_WIN64)
00010 #include <windows.h>

```

```

00011 #endif
00012
00013 #if defined(_WIN32) || defined(_WIN64)
00014 static constexpr WORD COL_GREEN = 2;
00015 static constexpr WORD COL_DARK_BLUE = 1;
00016 static constexpr WORD COL_DARK_RED = 4;
00017 static constexpr WORD COL_MAGENTA = 5;
00018 static constexpr WORD COL_BROWN = 6;
00019 static constexpr WORD COL_WHITE = 7;
00020 static constexpr WORD COL_LIGHT_BLUE = 9;
00021 static constexpr WORD COL_LIGHT_MAGENTA = 13;
00022 static constexpr WORD COL_LIGHT_YELLOW = 14;
00023 #endif
00024
00025 namespace Views {
00026
00027     // display
00028     void UI_Cli::clearScreen() {
00029         std::cout << "\033[H\033[J";
00030         std::cout << std::endl;
00031     }
00032
00033     void UI_Cli::displayWelcome() {
00034         std::cout << " " << std::endl;
00035         std::cout << "===== " << std::endl;
00036         std::cout << " Welcome to the Laying Grass " << std::endl;
00037         std::cout << "===== " << std::endl;
00038         std::cout << " " << std::endl;
00039     }
00040
00041     std::string UI_Cli::renderCell(const Models::Cell& cell, bool isTempTile) {
00042         switch (cell.getState()) {
00043             case Models::State::EMPTY:
00044                 return " . ";
00045             case Models::State::GRASS:
00046                 if (isTempTile) {
00047                     return "[#]";
00048                 } else {
00049                     return " # ";
00050                 }
00051             case Models::State::BONUS:
00052                 switch (cell.getBonusType()) {
00053                     case Models::BonusType::EXCHANGE:
00054                         return " E ";
00055                     case Models::BonusType::STONE:
00056                         return " S ";
00057                     case Models::BonusType::STEAL:
00058                         return " V ";
00059                     default:
00060                         return " B ";
00061                 }
00062             default:
00063                 return " ? ";
00064         }
00065     }
00066
00067     static WORD mapColorStringToAttr(const std::string& color) {
00068 #if defined(_WIN32) || defined(_WIN64)
00069         if (color == "white") return COL_WHITE;
00070         if (color == "light_blue") return COL_LIGHT_BLUE;
00071         if (color == "dark_blue") return COL_DARK_BLUE;
00072         if (color == "yellow") return COL_LIGHT_YELLOW;
00073         if (color == "red") return COL_DARK_RED;
00074         if (color == "purple") return COL_MAGENTA;
00075         if (color == "pink") return COL_LIGHT_MAGENTA;
00076         if (color == "brown") return COL_BROWN;
00077         if (color == "green") return COL_GREEN;
00078         return COL_WHITE;
00079     #else
00080         (void)color;
00081         return 0;
00082     #endif
00083     }
00084
00085     void UI_Cli::displayBoard(const Models::Board& board, std::vector<Models::Player>& players) {
00086
00087         int width = board.getWidth();
00088         int height = board.getHeight();
00089
00090 #if defined(_WIN32) || defined(_WIN64)
00091         HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00092         CONSOLE_SCREEN_BUFFER_INFO csbi;
00093         WORD defaultAttr = 7;
00094         if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00095             defaultAttr = csbi.wAttributes & (FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE |

```

```

        FOREGROUND_INTENSITY);
00098     }
00099 #endif
00100
00101     for (int y = 0; y < height; ++y) {
00102         for (int x = 0; x < width; ++x) {
00103             Models::Cell cell = board.getGrid()[y][x];
00104             if (cell.getState() == Models::State::GRASS) {
00105                 int pid = cell.getPlayerId();
00106 #if defined(_WIN32) || defined(_WIN64)
00107                 if (pid >= 0 && pid < static_cast<int>(players.size())) {
00108                     WORD attr = mapColorStringToAttr(players[pid].getColor());
00109                     SetConsoleTextAttribute(hConsole, attr);
00110                 }
00111 #endif
00112             }
00113
00114             std::cout << renderCell(cell);
00115
00116 #if defined(_WIN32) || defined(_WIN64)
00117             //reset color après affichage d'une cellule d'herbe
00118             if (cell.getState() == Models::State::GRASS) {
00119                 SetConsoleTextAttribute(hConsole, defaultAttr);
00120             }
00121 #endif
00122         }
00123         std::cout << std::endl;
00124     }
00125 }
00126
00127 void UI_Cli::displayTile(Models::Tile& tile) {
00128     int height = tile.getHeight();
00129     int width = tile.getWidth();
00130
00131     for (int y = 0; y < height; ++y) {
00132         for (int x = 0; x < width; ++x) {
00133             Models::Cell cell = tile.getPattern()[y][x];
00134             std::cout << renderCell(cell);
00135         }
00136         std::cout << std::endl;
00137     }
00138 }
00139
00140 void UI_Cli::displayBoardWithTile(Models::Board& board, Models::Tile& tile, Models::Position& pos,
int playerId, std::vector<Models::Player>& players) {
00141     int boardWidth = board.getWidth();
00142     int boardHeight = board.getHeight();
00143     int tileWidth = tile.getWidth();
00144     int tileHeight = tile.getHeight();
00145     int tileX = pos.getX();
00146     int tileY = pos.getY();
00147
00148     std::vector<std::vector<Models::Cell>> tempGrid = board.getGrid();
00149
00150     for (int ty = 0; ty < tileHeight; ++ty) {
00151         for (int tx = 0; tx < tileWidth; ++tx) {
00152             int boardX = tileX + tx;
00153             int boardY = tileY + ty;
00154
00155             if (boardX >= 0 && boardX < boardWidth && boardY >= 0 && boardY < boardHeight) {
00156                 if (tile.getPattern()[ty][tx].getState() == Models::State::GRASS) {
00157                     tempGrid[boardY][boardX] = tile.getPattern()[ty][tx];
00158                     tempGrid[boardY][boardX].setPlayerId(playerId);
00159                 }
00160             }
00161         }
00162     }
00163
00164 #if defined(_WIN32) || defined(_WIN64)
00165     HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00166     CONSOLE_SCREEN_BUFFER_INFO csbi;
00167     WORD defaultAttr = 7;
00168     if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00169         defaultAttr = csbi.wAttributes & (FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE |
        FOREGROUND_INTENSITY);
00170     }
00171 #endif
00172
00173     for (int y = 0; y < boardHeight; ++y) {
00174         for (int x = 0; x < boardWidth; ++x) {
00175             Models::Cell cell = tempGrid[y][x];
00176
00177             bool isTempTile = false;
00178             if (x >= tileX && x < tileX + tileWidth && y >= tileY && y < tileY + tileHeight) {
00179                 int tx = x - tileX;
00180                 int ty = y - tileY;
00181                 if (tile.getPattern()[ty][tx].getState() == Models::State::GRASS) {

```

```

00182         isTempTile = true;
00183     }
00184 }
00185
00186 //colorie la cellule si c'est de l'herbe en fonction de l'id du joueur
00187 if (cell.getState() == Models::State::GRASS) {
00188     int pid = cell.getPlayerId();
00189 #if defined(_WIN32) || defined(_WIN64)
00190     if (pid >= 0 && pid < static_cast<int>(players.size())) {
00191         WORD attr = mapColorStringToAttr(players[pid].getColor());
00192         SetConsoleTextAttribute(hConsole, attr);
00193     }
00194 #endif
00195 }
00196
00197 std::cout << renderCell(cell, isTempTile);
00198
00199 #if defined(_WIN32) || defined(_WIN64)
00200     if (cell.getState() == Models::State::GRASS) {
00201         SetConsoleTextAttribute(hConsole, defaultAttr);
00202     }
00203 #endif
00204 }
00205 std::cout << std::endl;
00206 }
00207 }
00208
00209 void UI_Cli::displayPlayer(Models::Player& player) {
00210     std::cout << "Player " << player.getId() << ": " << player.getName()
00211         << " | Color: " << player.getColor()
00212         << " | Score: " << player.getScore()
00213         << " | Exchanges: " << player.getExchange()
00214         << std::endl;
00215 }
00216
00217 int UI_Cli::displayMarket(std::vector<Models::Tile>& marketTiles) {
00218     int selectedIndex = 0;
00219
00220     while (true) {
00221         clearScreen();
00222         std::cout << "=== TILE MARKET ===" << std::endl;
00223         std::cout << "Choose a tile (UP/DOWN to navigate, SPACE to select)" << std::endl;
00224         std::cout << std::endl;
00225
00226         for (size_t i = 0; i < marketTiles.size(); ++i) {
00227             if (i == selectedIndex) {
00228                 std::cout << ">> [" << (i + 1) << "]" << std::endl;
00229             } else {
00230                 std::cout << " [" << (i + 1) << "]" << std::endl;
00231             }
00232             displayTile(marketTiles[i]);
00233             std::cout << std::endl;
00234         }
00235
00236         Utils::KeyCode key = Utils::KeyboardInput::getKeyPressed();
00237
00238         switch (key) {
00239             case Utils::KeyCode::UP:
00240                 if (selectedIndex > 0) selectedIndex--;
00241                 break;
00242             case Utils::KeyCode::DOWN:
00243                 if (selectedIndex < marketTiles.size() - 1) selectedIndex++;
00244                 break;
00245             case Utils::KeyCode::CONFIRM:
00246                 return selectedIndex;
00247             default:
00248                 break;
00249         }
00250     }
00251 }
00252
00253 void UI_Cli::displayMessage(std::string& message) {
00254     std::cout << message << std::endl;
00255 }
00256
00257 void UI_Cli::displayWinner(std::vector<Models::Player>& players, int winnerId) {
00258     for (auto& player : players) {
00259         if (player.getId() == winnerId) {
00260             std::cout << "Congratulations " << player.getName() << "! You are the winner with a score
00261 of " << player.getScore() << "!" << std::endl;
00262             return;
00263         }
00264     }
00265 }
00266
00267 //inputs

```

```

00268     int UI_Cli::askNumberOfPlayers() {
00269         int playerNumber = 0;
00270         while (playerNumber < 2 || playerNumber > 9) {
00271             std::cout << "Enter number of players (2-9): ";
00272             if (!(std::cin >> playerNumber)) {
00273                 std::cin.clear();
00274                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00275                 std::cout << "Invalid input. Please enter a number between 2 and 9 : ";
00276                 playerNumber = 0;
00277                 continue;
00278             }
00279             if (playerNumber < 2 || playerNumber > 9) {
00280                 std::cout << "Invalid number of players. Please enter a number between 2 and 9.";
00281             }
00282         }
00283         return playerNumber;
00284     }
00285
00286     std::string UI_Cli::askPlayerName(std::string playerName) {
00287         std::cout << "Enter name for " << playerName << ": ";
00288         std::string name;
00289         while (!(std::cin >> name)) {
00290             std::cin.clear();
00291             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00292             std::cout << "Invalid input. Enter name for " << playerName << ": ";
00293         }
00294         return name;
00295     }
00296
00297     std::string UI_Cli::askPlayerColor(std::vector<std::string>& availableColors) {
00298         int choice = -1;
00299
00300         while (choice < 1 || choice > availableColors.size()) {
00301             std::cout << "\nAvailable colors:" << std::endl;
00302
00303             #if defined(_WIN32) || defined(_WIN64)
00304                 HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00305                 CONSOLE_SCREEN_BUFFER_INFO csbi;
00306                 WORD defaultAttr = 7;
00307                 if (GetConsoleScreenBufferInfo(hConsole, &csbi)) {
00308                     defaultAttr = csbi.wAttributes & (FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE
00309 | FOREGROUND_INTENSITY);
00309                 }
00310             #endif
00311
00312             for (size_t i = 0; i < availableColors.size(); ++i) {
00313                 #if defined(_WIN32) || defined(_WIN64)
00314                     WORD attr = mapColorStringToAttr(availableColors[i]);
00315                     SetConsoleTextAttribute(hConsole, attr);
00316                     std::cout << (i + 1) << ". " << availableColors[i];
00317                     SetConsoleTextAttribute(hConsole, defaultAttr);
00318                     std::cout << std::endl;
00319                 #else
00320                     std::cout << (i + 1) << ". " << availableColors[i] << std::endl;
00321                 #endif
00322             }
00323
00324             std::cout << "Choose a color (1-" << availableColors.size() << "): ";
00325             if (!(std::cin >> choice)) {
00326                 std::cin.clear();
00327                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00328                 std::cout << "Invalid input. Please enter a number between 1 and " <<
availableColors.size() << ": ";
00329                 choice = -1;
00330                 continue;
00331             }
00332             if (choice < 1 || choice > static_cast<int>(availableColors.size())) {
00333                 std::cout << "Invalid choice. Please enter a number between 1 and " <<
availableColors.size() << ": ";
00334             }
00335         }
00336
00337         std::string selectedColor = availableColors[choice - 1];
00338         availableColors.erase(availableColors.begin() + choice - 1);
00339         return selectedColor;
00340     }
00341
00342     void UI_Cli::tilePlacement(models::Tile& tile, models::Board& board, int playerId,
std::vector<models::Player>& players) {
00343
00344         controllers::TilePlacer placer(&tile, &board, playerId);
00345         bool placementConfirmed = false;
00346
00347         while (!placementConfirmed) {
00348             clearScreen();
00349             std::cout << std::endl;
00350

```

```

00351         Models::Position pos = placer.getPosition();
00352         bool isValid = placer.isValidPlacement();
00353
00354         // Display status
00355         if (isValid) {
00356             std::cout << "=== PLACEMENT VALID ===" << std::endl;
00357         } else {
00358             std::cout << "=== PLACEMENT INVALID ===" << std::endl;
00359         }
00360         std::cout << "Position: (" << pos.getX() << ", " << pos.getY() << ")" << std::endl;
00361         std::cout << std::endl;
00362
00363         displayBoardWithTile(board, tile, pos, playerId, players);
00364
00365         std::cout << std::endl;
00366         std::cout << "Controls:" << std::endl;
00367         std::cout << "  Arrow Keys: Move tile" << std::endl;
00368         std::cout << "  R: Rotate" << std::endl;
00369         std::cout << "  SPACE: Confirm" << std::endl;
00370         std::cout << std::endl;
00371         std::cout << "Press a key..." << std::endl;
00372
00373         Utils::KeyCode key = Utils::KeyboardInput::getKeyPressed();
00374
00375         switch (key) {
00376             case Utils::KeyCode::UP:
00377                 placer.moveUp();
00378                 break;
00379             case Utils::KeyCode::DOWN:
00380                 placer.moveDown();
00381                 break;
00382             case Utils::KeyCode::LEFT:
00383                 placer.moveLeft();
00384                 break;
00385             case Utils::KeyCode::RIGHT:
00386                 placer.moveRight();
00387                 break;
00388             case Utils::KeyCode::ROTATE:
00389                 placer.rotateTile();
00390                 break;
00391             case Utils::KeyCode::CONFIRM:
00392                 if (placer.isValidPlacement()) {
00393                     placer.confirmPlacement();
00394                     placementConfirmed = true;
00395                     clearScreen();
00396                     std::cout << "Tile placed successfully!" << std::endl;
00397                 } else {
00398                     clearScreen();
00399                     std::cout << "[ERROR] Cannot place tile at this position!" << std::endl;
00400                     std::cout << "Press any key to continue..." << std::endl;
00401                     _getch();
00402                 }
00403                 break;
00404             default:
00405                 break;
00406         }
00407     }
00408 }
00409 } // Views

```

## 8.63 src/views/UI\_Qt.cpp File Reference

```
#include "../include/views/UI_Qt.h"
```

### Namespaces

- namespace [Views](#)

## 8.64 UI\_Qt.cpp

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

```
00001 //  
00002 // Created by antoi on 03/11/2025.  
00003 //  
00004  
00005 #include "../../include/views/UI_Qt.h"  
00006  
00007 namespace Views {  
00008 } // Views
```

# Index

- ~Game
  - Controllers::Game, 28
- addTile
  - Models::TileQueue, 50
- addUsedTile
  - Models::TileQueue, 50
- askNumberOfPlayers
  - Views::UI\_Cli, 54
- askPlayerColor
  - Views::UI\_Cli, 54
- askPlayerName
  - Views::UI\_Cli, 55
- Board
  - Models::Board, 18
- BONUS
  - Models, 14
- BonusSquare
  - Models::BonusSquare, 24
- BonusType
  - Models, 13
- calculateGrass
  - Utils::SquareCalculator, 41
- calculateSquare
  - Utils::SquareCalculator, 41
- canPlaceTile
  - Models::Board, 18
- Cell
  - Models::Cell, 26
- checkBonusAcquisition
  - Models::Board, 19
- checkKeyPressed
  - Utils::KeyboardInput, 33
- clearScreen
  - Views::UI\_Cli, 55
- CONFIRM
  - Utils, 15
- confirmPlacement
  - Controllers::TilePlacer, 47
- Controllers, 13
- Controllers::Game, 28
  - ~Game, 28
  - end, 29
  - Game, 28
  - getBoard, 29
  - getCurrentRound, 29
  - getPlayerById, 29
  - getPlayerCount, 29
  - getPlayers, 29
  - getTileQueue, 30
  - isGameOver, 30
  - run, 30
  - setCurrentRound, 30
  - start, 30
- Controllers::TilePlacer, 47
  - confirmPlacement, 47
  - getPosition, 47
  - getTile, 48
  - isValidPlacement, 48
  - moveDown, 48
  - moveLeft, 48
  - moveRight, 48
  - moveUp, 48
  - rotateTile, 49
  - setInitialPosition, 49
  - TilePlacer, 47
- convertJsonToTile
  - Models::Tile, 43
- createTile
  - Models::Tile, 43
- displayBoard
  - Views::UI\_Cli, 55
- displayBoardWithTile
  - Views::UI\_Cli, 56
- displayMarket
  - Views::UI\_Cli, 57
- displayMessage
  - Views::UI\_Cli, 57
- displayPlayer
  - Views::UI\_Cli, 57
- displayTile
  - Views::UI\_Cli, 58
- displayWelcome
  - Views::UI\_Cli, 58
- displayWinner
  - Views::UI\_Cli, 58
- DOWN
  - Utils, 15
- EMPTY
  - Models, 14
- end
  - Controllers::Game, 29
- EXCHANGE
  - Models, 14
- flipHorizontal

- Models::Tile, 44
- Game
  - Controllers::Game, 28
- getAvailableColors
  - Utils::InputValidator, 31
- getBoard
  - Controllers::Game, 29
- getBonusType
  - Models::BonusSquare, 24
  - Models::Cell, 26
- getCell
  - Models::Board, 19
- getColor
  - Models::Player, 34
- getCurrentRound
  - Controllers::Game, 29
- getCurrentTile
  - Models::TileQueue, 50
- getExchange
  - Models::Player, 34
- getGrid
  - Models::Board, 19
- getHeight
  - Models::Board, 19
  - Models::Tile, 44
- getId
  - Models::Player, 35
  - Models::Tile, 44
- getInt
  - Utils::Random, 40
- getKeyPressed
  - Utils::KeyboardInput, 33
- getName
  - Models::Player, 35
- getNextTiles
  - Models::TileQueue, 50
- getPattern
  - Models::Tile, 44
- getPlayerById
  - Controllers::Game, 29
- getPlayerCount
  - Controllers::Game, 29
- getPlayerId
  - Models::Cell, 26
  - Models::Tile, 44
- getPlayers
  - Controllers::Game, 29
- getPlayersNumber
  - Models::Board, 20
- getPosition
  - Controllers::TilePlacer, 47
- getScore
  - Models::Player, 35
- getSize
  - Models::Tile, 45
- getState
  - Models::Cell, 26
- getTile
  - Controllers::TilePlacer, 48
- getTileAt
  - Models::TileQueue, 51
- getTileQueue
  - Controllers::Game, 30
- getTiles
  - Models::TileQueue, 51
- getUsedTiles
  - Models::TileQueue, 51
- getWidth
  - Models::Board, 20
  - Models::Tile, 45
- getX
  - Models::Position, 38
- getY
  - Models::Position, 38
- GRASS
  - Models, 14
- hasStoneAt
  - Models::Board, 20
- include Directory Reference, 9
- include/controllers Directory Reference, 9
- include/controllers/Game.h, 61
- include/controllers/TilePlacer.h, 62, 63
- include/models Directory Reference, 9
- include/models/Board.h, 63, 64
- include/models/BonusSquare.h, 64, 65
- include/models/Cell.h, 65
- include/models/Player.h, 66
- include/models/Position.h, 67
- include/models/Tile.h, 68
- include/models/TileQueue.h, 69
- include/models/Types.h, 70
- include/Utils Directory Reference, 10
- include/Utils/InputValidator.h, 70, 71
- include/Utils/KeyboardInput.h, 71, 72
- include/Utils/Random.h, 72
- include/Utils/SquareCalculator.h, 73
- include/views Directory Reference, 11
- include/views/UI\_Cli.h, 74
- include/views/UI\_Qt.h, 75
- isBonus
  - Models::Cell, 26
- isCellTouchingSomething
  - Models::Board, 20
- isEmpty
  - Models::Cell, 27
  - Models::TileQueue, 51
- isExchange
  - Models::BonusSquare, 24
- isGameOver
  - Controllers::Game, 30
- isGrass
  - Models::Cell, 27
- isInsideBoard
  - Models::Board, 21
- isPlaced

- Models::Tile, 45
- isSteal
  - Models::BonusSquare, 25
- isStone
  - Models::BonusSquare, 25
- isTileTouchingGrass
  - Models::Board, 21
- isTouchingWall
  - Models::Board, 21
- isValidNumberOfPlayers
  - Utils::InputValidator, 31
- isValidPlacement
  - Controllers::TilePlacer, 48
- isValidPlayerColor
  - Utils::InputValidator, 32
- isValidPlayerName
  - Utils::InputValidator, 32
- KeyCode
  - Utils, 14
- LEFT
  - Utils, 15
- loadTiles
  - Models::TileQueue, 51
- main
  - main.cpp, 81
- main.cpp
  - main, 81
- Models, 13
  - BONUS, 14
  - BonusType, 13
  - EMPTY, 14
  - EXCHANGE, 14
  - GRASS, 14
  - NONE, 14
  - State, 14
  - STEAL, 14
  - STONE, 14
- Models::Board, 17
  - Board, 18
  - canPlaceTile, 18
  - checkBonusAcquisition, 19
  - getCell, 19
  - getGrid, 19
  - getHeight, 19
  - getPlayersNumber, 20
  - getWidth, 20
  - hasStoneAt, 20
  - isCellTouchingSomething, 20
  - isInsideBoard, 21
  - isTileTouchingGrass, 21
  - isTouchingWall, 21
  - placeBonus, 22
  - placeStone, 22
  - placeTile, 22
  - removeBonus, 23
  - setGrid, 23
  - setHeight, 23
  - setWidth, 23
- Models::BonusSquare, 24
  - BonusSquare, 24
  - getBonusType, 24
  - isExchange, 24
  - isSteal, 25
  - isStone, 25
  - setBonusType, 25
- Models::Cell, 25
  - Cell, 26
  - getBonusType, 26
  - getPlayerId, 26
  - getState, 26
  - isBonus, 26
  - isEmpty, 27
  - isGrass, 27
  - setBonusType, 27
  - setPlayerId, 27
  - setState, 27
- Models::Player, 34
  - getColor, 34
  - getExchange, 34
  - getId, 35
  - getName, 35
  - getScore, 35
  - Player, 34
  - setColor, 35
  - setExchange, 35
  - setId, 35
  - setName, 36
  - setScore, 36
- Models::Position, 37
  - getX, 38
  - getY, 38
  - operator!=, 38
  - operator<, 38
  - operator==, 39
  - Position, 38
  - setX, 39
  - setY, 39
- Models::Tile, 42
  - convertJsonToTile, 43
  - createTile, 43
  - flipHorizontal, 44
  - getHeight, 44
  - getId, 44
  - getPattern, 44
  - getPlayerId, 44
  - getSize, 45
  - getWidth, 45
  - isPlaced, 45
  - rotate, 45
  - setHeight, 45
  - setId, 46
  - setPattern, 46
  - setPlaced, 46
  - setPlayerId, 46

- setWidth, 46
  - Tile, 43
- Models::TileQueue, 49
  - addTile, 50
  - addUsedTile, 50
  - getCurrentTile, 50
  - getNextTiles, 50
  - getTileAt, 51
  - getTiles, 51
  - getUsedTiles, 51
  - isEmpty, 51
  - loadTiles, 51
  - recycleTiles, 52
  - removeTile, 52
  - removeUsedTile, 52
  - selectTileFromMarket, 52
  - shuffleTiles, 53
  - TileQueue, 50
- moveDown
  - Controllers::TilePlacer, 48
- moveLeft
  - Controllers::TilePlacer, 48
- moveRight
  - Controllers::TilePlacer, 48
- moveUp
  - Controllers::TilePlacer, 48
- NONE
  - Models, 14
- operator!=
  - Models::Position, 38
- operator<
  - Models::Position, 38
- operator==
  - Models::Position, 39
- placeBonus
  - Models::Board, 22
- placeStone
  - Models::Board, 22
- placeTile
  - Models::Board, 22
- Player
  - Models::Player, 34
- playerGrass
  - Utils::PlayerResult, 37
- playerID
  - Utils::PlayerResult, 37
- playerScore
  - Utils::PlayerResult, 37
- Position
  - Models::Position, 38
- rankingPlayersByScore
  - Utils::SquareCalculator, 41
- recycleTiles
  - Models::TileQueue, 52
- removeBonus
  - Models::Board, 23
- removeTile
  - Models::TileQueue, 52
- removeUsedTile
  - Models::TileQueue, 52
- RIGHT
  - Utils, 15
- ROTATE
  - Utils, 15
- rotate
  - Models::Tile, 45
- rotateTile
  - Controllers::TilePlacer, 49
- run
  - Controllers::Game, 30
- selectColor
  - Utils::InputValidator, 32
- selectTileFromMarket
  - Models::TileQueue, 52
- setBonusType
  - Models::BonusSquare, 25
  - Models::Cell, 27
- setColor
  - Models::Player, 35
- setCurrentRound
  - Controllers::Game, 30
- setExchange
  - Models::Player, 35
- setGrid
  - Models::Board, 23
- setHeight
  - Models::Board, 23
  - Models::Tile, 45
- setId
  - Models::Player, 35
  - Models::Tile, 46
- setInitialPosition
  - Controllers::TilePlacer, 49
- setName
  - Models::Player, 36
- setPattern
  - Models::Tile, 46
- setPlaced
  - Models::Tile, 46
- setPlayerId
  - Models::Cell, 27
  - Models::Tile, 46
- setScore
  - Models::Player, 36
- setState
  - Models::Cell, 27
- setWidth
  - Models::Board, 23
  - Models::Tile, 46
- setX
  - Models::Position, 39
- setY
  - Models::Position, 39

- shuffle
  - Utils::Random, 40
- shuffleTiles
  - Models::TileQueue, 53
- SquareCalculator
  - Utils::SquareCalculator, 41
- src Directory Reference, 10
- src/controllers Directory Reference, 9
- src/controllers/Game.cpp, 75
- src/controllers/TilePlacer.cpp, 80
- src/main.cpp, 81
- src/models Directory Reference, 10
- src/models/Board.cpp, 82
- src/models/BonusSquare.cpp, 85
- src/models/Cell.cpp, 85, 86
- src/models/Player.cpp, 86
- src/models/Position.cpp, 87
- src/models/Tile.cpp, 87, 88
- src/models/TileQueue.cpp, 89
- src/Utils Directory Reference, 10
- src/Utils/InputValidator.cpp, 90, 91
- src/Utils/KeyboardInput.cpp, 91, 92
- src/Utils/Random.cpp, 92
- src/Utils/SquareCalculator.cpp, 93
- src/views Directory Reference, 11
- src/views/UI\_Cli.cpp, 94
- src/views/UI\_Qt.cpp, 99
- start
  - Controllers::Game, 30
- State
  - Models, 14
- STEAL
  - Models, 14
- STONE
  - Models, 14
- Tile
  - Models::Tile, 43
- tilePlacement
  - Views::UI\_Cli, 58
- TilePlacer
  - Controllers::TilePlacer, 47
- TileQueue
  - Models::TileQueue, 50
- UNKNOWN
  - Utils, 15
- UP
  - Utils, 15
- Utils, 14
  - CONFIRM, 15
  - DOWN, 15
  - KeyCode, 14
  - LEFT, 15
  - RIGHT, 15
  - ROTATE, 15
  - UNKNOWN, 15
  - UP, 15
- Utils::InputValidator, 31
- getAvailableColors, 31
- isValidNumberOfPlayers, 31
- isValidPlayerColor, 32
- isValidPlayerName, 32
- selectColor, 32
- Utils::KeyboardInput, 33
  - checkKeyPressed, 33
  - getKeyPressed, 33
- Utils::PlayerResult, 36
  - playerGrass, 37
  - playerID, 37
  - playerScore, 37
- Utils::Random, 39
  - getInt, 40
  - shuffle, 40
- Utils::SquareCalculator, 40
  - calculateGrass, 41
  - calculateSquare, 41
  - rankingPlayersByScore, 41
  - SquareCalculator, 41
- Views, 15
- Views::UI\_Cli, 53
  - askNumberOfPlayers, 54
  - askPlayerColor, 54
  - askPlayerName, 55
  - clearScreen, 55
  - displayBoard, 55
  - displayBoardWithTile, 56
  - displayMarket, 57
  - displayMessage, 57
  - displayPlayer, 57
  - displayTile, 58
  - displayWelcome, 58
  - displayWinner, 58
  - tilePlacement, 58
- Views::UI\_Qt, 60