PG 1 - Praktikum 5

Generated by Doxygen 1.9.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Battleship Class Reference

The Battleship class.

`#include <battleship.h>`

Public Member Functions

- Battleship (const string &player1Name, const string &player2Name)

  Battleship constructor.
- void play ()

  Play one game of Battleships!

Private Attributes

- array< Player, 2 > **m_players**

  Array containing the two Player objects.
- array< GameBoard, 2 > **m_boards**

  Array containing the two GameBoard objects, one for each player.

3.1.1 Detailed Description

The Battleship class.

This is the main class of the game. An object is constructed by supplying two player names. One game round is then started by simply calling the play function.

The Players and their boards are stored in Arrays to make the implementation of the game logic easier.

Note: If you find that confusing, you are welcome the change the Arrays into something else, or have two attributes per player.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Battleship()

```
Battleship::Battleship (
            const string & player1Name,
            const string & player2Name )
```

Battleship constructor.

Parameters

`in`	*player1Name*	Name of the first player
`in`	*player2Name*	Name of the second player

3.1.3 Member Function Documentation

3.1.3.1 play()

```
void Battleship::play ( )
```

Play one game of Battleships!

This functions shall do the following:

- Initialize both players game boards:
  - The ships shall be placed at randomized locations
  - The 'cheat sheets' shall be empty
  - This can easily be achieved by creating new GameBoard objects and storing them in m_boards, overwriting the old boards which may be there already

Then, do the following in a loop:

1. Determine the active Player either by random coin toss or first player always gets to start
2. Print both the game board and the 'cheat sheet' for the active player
3. Ask the active player a location she wants to shoot at
4. Call the GameBoard::hit function for that location on the **inactive** Player's board
5. Inform the active player if she hit something and make the correct mark on the cheat sheet
6. Switch the players, e.g. active player becomes inactive and inactive Player becomes active
7. Repeat Steps 1 to 6 until one player has lost
8. Exit the function

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

- p5-stud/battleship.h

3.2 GameBoard Class Reference

The GameBoard class.

```
#include <gameboard.h>
```

Public Member Functions

- GameBoard ()

  GameBoard constructor.
- void printBoard ()

  Prints the player's board.
- void printEnemyBoard ()

  Prints the 'cheat sheet', containing markings where this player has hit or missed.
- bool hit (int row, int col)

  The enemy player's shot on our board.
- void mark (int row, int col, bool wasHit)

  Mark locations on the enemy board where we already shot at.
- void randomPlaceShips ()

  Randomly place ships.
- bool allShipsSunk ()

  Test if all ships are sunk.

Private Attributes

- array$<$ Ship, 10 $>$ m_ships

  The player's ships.
- array$<$ array$<$ char, 10 $>$, 10 $>$ m_enemyBoard

  The 'cheat sheet'.

3.2.1 Detailed Description

The GameBoard class.

This class contains everything that a single player needs in order to play. First of all, this class contains an array of one's own ships. Furthermore it contains a map of locations where Player A has tried to hit ships from Player B.

The primary functions of this board are hit which represents a shot from the enemy. The hit function handles the shot and returns true if any ship was actually hit by that shot. The other important function is mark which is used to mark positions where the player tried to hit enemy ships.

Every Player needs one instance of this in order to:

- Have a registry where her own ships are located
- Have an idea where she has already tried to hit her enemy's ships

Note that while the enemy board is represented as a two dimensional array of `char` values, the own registry exists simply as `vector` of Ship objects.

Also, this class has the neccessary functions functions to print both of those boards.

Note that Battleships is played on a 10 x 10 board, with valid rows and columns ranging from 0 to 9.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 GameBoard()

```
GameBoard::GameBoard ( )
```

GameBoard constructor.

This shall initialize the enemy board (attribute m_enemyBoard) with dot characters '.'. The player's ships can either be placed here also, or later by calling randomPlaceShips.

3.2.3 Member Function Documentation

3.2.3.1 allShipsSunk()

```
bool GameBoard::allShipsSunk ( )
```

Test if all ships are sunk.

Returns

True if all ships on this board are sunk

This function is used to determine if the player hast lost the game. As a reminder: The player has lost the game when she has no floating ship left.

3.2.3.2 hit()

```
bool GameBoard::hit (
             int row,
             int col )
```

The enemy player's shot on our board.

Parameters

in	*row*	The row where we are being shot
in	*col*	The column where we are being shot

Returns

True if the shot hit any of our ships, false otherwise.

This function is called when the enemy is trying to hit any of our ships. Generally a shot is directed at a specific location denoted by `row` and `col`.

If there is a ship part at that location, that part shall be damaged by that shot.

See also

Part::setDamaged
printBoard

3.2.3.3 mark()

```
void GameBoard::mark (
            int row,
            int col,
            bool wasHit )
```

Mark locations on the enemy board where we already shot at.

Parameters

in	*row*	The row where we shot
in	*col*	The column where we shot
in	*wasHit*	True if the shot was a hit (e.g. we hit a ship)

This function shall make a mark on the 'cheat sheet'. The mark shall be different depending on if we hit something there.

See also

hit
printEnemyBoard

3.2.3.4 printBoard()

```
void GameBoard::printBoard ( )
```

Prints the player's board.

This function prints the player's board to the console screen. You are relatively free to be creative here, but make sure that all ships are displayed properly.

Some suggestions are:

- A location containing water is represented by printing '.'
- A location containing an intact (e.g. undamaged) ship part is represented by its ship number (e.g. the ships index in the array)
- A location containing a damaged ship part of an **unsunken ship** is represented by an 'X'
- A location containing a part of a sunken ship is represented by an 'S'

You can, as mentioned before, use other characters, but those four types of locations should be distinguishable from another.

3.2.3.5 printEnemyBoard()

```
void GameBoard::printEnemyBoard ( )
```

Prints the 'cheat sheet', containing markings where this player has hit or missed.

When trying to hit the other player's ships, it does not make sense to shoot the same location more than once. In order to 'memorize' those locations, the two-dimensional array is used. In there, the characters have the following meanings:

- '.' represents a location which has not yet beens shot
- 'X' represents a hit ship at that locations
- 'O' represents a shot into open water

As with printBoard you are free to change the characters if you fancy something else.

3.2.3.6 randomPlaceShips()

```
void GameBoard::randomPlaceShips ( )
```

Randomly place ships.

This function randomly places ships on the board, e.g. it populates the m_ships vector.

The following ships shall be placed:

- 1 'Dreadnought' with 5 parts
- 2 'Cruisers' with 4 parts
- 3 'Destroyers' with 3 parts
- 4 'Submarines' with 2 parts

The ships shall be placed so that:

- No ships intersect each other
- No ship has parts outside the playing area
- When this is called for both players, the resulting placements shall be different

3.2.4 Member Data Documentation

3.2.4.1 m_enemyBoard

```
array<array<char, 10>, 10> GameBoard::m_enemyBoard  [private]
```

The 'cheat sheet'.

This is a two-dimensional Array of `char` values. What this means is that we use an `std::array<char, 10>` to represent a row. Then, a collection of 10 of such rows finally is a grid, our two-dimensional structure.

So the datatype for this is `array< array<char, 10>, 10 >`.

3.2.4.2 m_ships

```
array<Ship, 10> GameBoard::m_ships  [private]
```

The player's ships.

This is collection of the player's ships. Since there are always 10 ships on the board, we can use a simple `std::array`.

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

- p5-stud/gameboard.h

3.3 Part Class Reference

The Part class represents a part of a Ship in the context of the Battleship game.

```
#include <part.h>
```

Public Member Functions

- Part (int row, int col)

  Part constructor.
- bool isDamaged () const

  Returns wether or not this part is damaged.
- void setDamaged ()

  Sets the status of this part to a valued representing 'damaged'.
- int getRow () const

  Returns the row with which this part was constructed.
- int getCol () const

  Returns the column with which this part was constructed.

Private Attributes

- int **m_row**

  The row where this part is located on the grid.
- int **m_col**

  The column where this part is located on the grid.
- int m_status

  Status of the part as integer value.

3.3.1 Detailed Description

The Part class represents a part of a Ship in the context of the Battleship game.

One object of Part is located at a specific $*$ position denoted by row and column. Multiple Part objects make up one object of Ship, as a matter of fact there is a composition association between those two.

A ship's part has a status which indicates wether it is damaged or not. A part of a ship does neither 'know' to which ship it belongs neither if that ship is already sunk or not.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Part()

```
Part::Part (
            int row,
            int col )
```

Part constructor.

Parameters

in	*row*	The grid row on which this part is to be placed
in	*col*	The grid column on which this part is to be placed

This is supposed to be the only Part constructor. The values for m_row and m_col shall be assigned from `row` and `col` respectively. The status m_status shall be initialized with 0. indicating 'no damage'.

3.3.3 Member Function Documentation

3.3.3.1 getCol()

```
int Part::getCol ( ) const
```

Returns the column with which this part was constructed.

Returns

The column with which this part was constructed

3.3.3.2 getRow()

```
int Part::getRow ( ) const
```

Returns the row with which this part was constructed.

Returns

The row with which this part was constructed

3.3.3.3 isDamaged()

```
bool Part::isDamaged ( ) const
```

Returns wether or not this part is damaged.

Returns

True if damaged, False if undamaged

Note that there is no extra status for a sunken Ship. By definition, a ship is sunk when all parts were hit, e.g. are damaged.

3.3.3.4 setDamaged()

```
void Part::setDamaged ( )
```

Sets the status of this part to a valued representing 'damaged'.

The attribute m_status is supposed to be 0 if there is no damage, and 1 if this part was damaged. This method shall thereby set m_status to 1.

3.3.4 Member Data Documentation

3.3.4.1 m_status

```
int Part::m_status  [private]
```

Status of the part as integer value.

0 means 'No Damage' 1 means 'Damaged'

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

- p5-stud/part.h

3.4 Player Class Reference

The Player class represents a single player for the Battleship game.

```
#include <player.h>
```

Public Member Functions

- Player (const string &playerName)

  Player constructor.
- int getGamesWon () const

  Get games won.
- int getGamesLost () const

  Get games lost.
- int getGamesPlayed () const

  Get games played.
- void addGameWon ()

  Add another won game.
- void addGameLost ()

  Add another lost game.
- string getName ()

  Get the player's name.

Private Attributes

- int **m_gamesWon**

  Number of games won.
- int **m_gamesLost**

  Number of games lost.
- string **m_playerName**

  The player's name.

3.4.1 Detailed Description

The Player class represents a single player for the Battleship game.

Objects of this class store

- The player's name
- How many games where won
- How many games where lost

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Player()

```
Player::Player (
            const string & playerName )
```

Player constructor.

Parameters

`in`	*playerName*	The player's name.

The attribute m_playerName shall be initialized with the `playerName` parameter.

The attributes m_gamesWon and m_gamesLost shall be initialized with 0.

3.4.3 Member Function Documentation

3.4.3.1 addGameLost()

```
void Player::addGameLost ( )
```

Add another lost game.

This shall increase the number of lost games by 1

3.4.3.2 addGameWon()

```
void Player::addGameWon ( )
```

Add another won game.

This shall increase the number of won games by 1.

3.4.3.3 getGamesLost()

```
int Player::getGamesLost ( ) const
```

Get games lost.

Returns

The number of games which were lost by this player.

3.4.3.4 getGamesPlayed()

```
int Player::getGamesPlayed ( ) const
```

Get games played.

Returns

The total number of games this player has played.

See also

getGamesWon
getGamesLost

3.4.3.5 getGamesWon()

```
int Player::getGamesWon ( ) const
```

Get games won.

Returns

The number of games which were won by this player.

3.4.3.6 getName()

```
string Player::getName ( )
```

Get the player's name.

Returns

The player's name

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

- p5-stud/player.h

3.5 Ship Class Reference

The Ship class, a collection of Part objects.

```
#include <ship.h>
```

Public Member Functions

- Ship ()
  Ship Standard constructor.
- Ship (int row, int col, int lengthOfShip, Direction direction)
  Ship constructor.
- bool hasPartIn (int row, int col)
  Evaluates if this ship extends to the given row and col.
- Part & getPartIn (int row, int col)
  Returns the ship's part which is in the given row and col.
- bool isDamaged ()
  Returns wether or not the ship is damaged.
- bool isSunk ()
  Returns wether or not the ship is sunk.

Private Attributes

- vector< Part > m_parts
  The ship's parts.

3.5.1 Detailed Description

The Ship class, a collection of Part objects.

This class represents a ship for the Battleship game. A ship is made up of at parts, e.g. Part objects which are stored using a vector.

Ships of multiple sizes are represented by having different amount of parts:

- The 'Dreadnought' has 5 parts
- A 'Cruiser' has 4 parts
- A 'Destroyer' has 3 parts
- A 'Submarine' has 2 parts

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Ship() [1/2]

```
Ship::Ship ( )
```

Ship Standard constructor.

Should do nothing. This is just needed in order for the std::array in GameBoard to work.

3.5.2.2 Ship() [2/2]

```
Ship::Ship (
            int row,
            int col,
            int lengthOfShip,
            Direction direction )
```

Ship constructor.

Parameters

in	*row*	The row where the aft part of the ship shall be placed
in	*col*	The column where the aft part of the ship shall be placed
in	*lengthOfShip*	Which length the ship shall have (usually 2 to 5)
in	*direction*	Which direction the ship should point to with: • 0 meaning up • 1 meaning right • 2 meaning down • 3 meaning left

Exceptions

std::invalid_argument	if either the direction if the ship is an invalid value **or** if parts of the ship would be placed outside of the 10x10 grid.

This constructs a ship with the given parameters. The aft section of the ship is always placed at the given row and column. From there on, (lengthOfShip - 1) number of parts are placed in the given direction.

If, for example, a ship shall be constructed at `row = 3` and `col = 2`, a length of 3 and `direction = 1` this means the following part objects are created:

- Part at 3 / 2
- Part at 3 / 3
- Part at 3 / 4

3.5.3 Member Function Documentation

3.5.3.1 getPartIn()

```
Part & Ship::getPartIn (
            int row,
            int col )
```

Returns the ship's part which is in the given row and col.

Parameters

in	*row*	Row of the Part
in	*col*	Column of the Part

Returns

Reference to the ship part at the given location.

Exceptions

std::invalid_argument	if there is no such part at the given position.

It is advised to only use this function if you are sure that the given position really contains a part of this ship, e.g. calling hasPartIn first.

See also

hasPartIn

3.5.3.2 hasPartIn()

```
bool Ship::hasPartIn (
            int row,
            int col )
```

Evaluates if this ship extends to the given row and col.

Parameters

in	*row*	Row which is to be checked
in	*col*	Column which is to be checked

Returns

True if ship extends to the field row/col, false otherwise.

This function is used to determine wether or not a ship is present in the given row and col. This is easily answered by iterating over the ship's parts and checking if any of those are located on this position.

See also

getPartIn

3.5.3.3 isDamaged()

```
bool Ship::isDamaged ( )
```

Returns wether or not the ship is damaged.

Returns

True if the ship is damaged.

A ship is considered to be damaged of there is at least on of it's parts damaged. If **all** of the parts are damaged, the ship is considered to be sunk, but this also counts as damaged.

See also

isSunk

3.5.3.4 isSunk()

```
bool Ship::isSunk ( )
```

Returns wether or not the ship is sunk.

Returns

True if the ship is sunk.

A ship is considered to be sunk if **all** of it's parts are damaged.

See also

isDamaged

3.5.4 Member Data Documentation

3.5.4.1 m_parts

```
vector<Part> Ship::m_parts  [private]
```

The ship's parts.

This vector stores the Part objects which make up the ship.

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

- p5-stud/ship.h

Chapter 4

File Documentation

4.1 p5-stud/battleship.h File Reference

```
#include "gameboard.h"
#include "player.h"
#include <string>
#include <array>
```

Classes

- class Battleship

  The Battleship class.

4.1.1 Detailed Description

Date

08.12.2019

Author

Michael Roth This is the header file for the Battleship class, which is the main class for the game. This class holds two Player objects as well as their corresponding GameBoard objects.

4.2 battleship.h

Go to the documentation of this file.

```

#ifndef BATTLESHIP_H
#define BATTLESHIP_H
#include "gameboard.h"
#include "player.h"

#include <string>
using std::string;

#include <array>
using std::array;
```

```

class Battleship
{
  public:
  Battleship(const string& player1Name, const string& player2Name);

  void play();

  private:
  array<Player, 2> m_players;

  array<GameBoard, 2> m_boards;
};

#endif // BATTLESHIP_H
```

4.3 p5-stud/gameboard.h File Reference

```
#include "ship.h"
#include <array>
```

Classes

- class GameBoard

  The GameBoard class.

4.3.1 Detailed Description

Date

  08.12.2019

Author

  Michael Roth This is the header file for the GameBoard class which represents the game board(s) for a single player.

4.4 gameboard.h

Go to the documentation of this file.

```

#ifndef GAMEBOARD_H
#define GAMEBOARD_H
#include "ship.h"

#include <array>
using std::array;

class GameBoard
{
  public:
  GameBoard();

  void printBoard();

  void printEnemyBoard();

   bool hit(int row, int col);

   void mark(int row, int col, bool wasHit);

   void randomPlaceShips();

   bool allShipsSunk();

   private:
   array<Ship, 10> m_ships;

   array<array<char, 10>, 10> m_enemyBoard;
 };

 #endif // GAMEBOARD_H
```

4.5 p5-stud/part.h File Reference

Classes

- class Part

  The Part class represents a part of a Ship in the context of the Battleship game.

4.5.1 Detailed Description

Date

08.12.2019

Author

Michael Roth This is the header file for the Part class. The Part class is used to represent on piece (or part) of a Ship for the Battleship game

4.6 part.h

Go to the documentation of this file.

```
#ifndef PART_H
#define PART_H

class Part
{
  public:
  Part(int row, int col);

  bool isDamaged() const;

  void setDamaged();

  int getRow() const;

  int getCol() const;

  private:
  int m_row;

  int m_col;

  int m_status;
};

#endif // PART_H
```

4.7 p5-stud/player.h File Reference

`#include <string>`

Classes

- class Player

  The Player class represents a single player for the Battleship game.

4.7.1 Detailed Description

Date

08.12.2019

Author

Michael Roth This is the header file for the Player class. This class is used for storing player Names and some sort of highscore table.

4.8 player.h

Go to the documentation of this file.

```

#ifndef PLAYER_H
#define PLAYER_H

#include <string>
using std::string;

class Player
{
  public:
  Player(const string& playerName);

  int getGamesWon() const;

  int getGamesLost() const;

  int getGamesPlayed() const;

  void addGameWon();

  void addGameLost();

  string getName();

  private:
  int m_gamesWon;

  int m_gamesLost;

  string m_playerName;
};

#endif // PLAYER_H
```

4.9 p5-stud/ship.h File Reference

```
#include <vector>
#include "part.h"
```

Classes

- class Ship

  The Ship class, a collection of Part objects.

Enumerations

- enum class Direction { **north** , **east** , **south** , **west** }

  The Direction enum, indicating in which direction a ship is pointed.

4.9.1 Detailed Description

Date

08.12.2019

Author

Michael Roth This is the header file for the Ship class. A Ship is, in this context, a composite collection of Part objects, where the part objects make up the ship.

4.10 ship.h

Go to the documentation of this file.

```

#ifndef SHIP_H
#define SHIP_H
#include <vector>
using std::vector;

#include "part.h"

enum class Direction
{
  north,
  east,
  south,
  west
};

class Ship
{
  public:
  Ship();

  Ship(int row, int col, int lengthOfShip, Direction direction);

  bool hasPartIn(int row, int col);

   Part& getPartIn(int row, int col);

   bool isDamaged();

   bool isSunk();

   private:
   vector<Part> m_parts;
 };

 #endif // SHIP_H
```

Index