Project 2

Title

Battleship

Course

CIS-17A

Section

42474

Due Date

June 1, 2018

Author

Mariia Gregory

Table of Contents

Introduction

As a final project, I modified my midterm version of Battleship, a text-based board game written in C++, utilizing classes, templates, exception handling techniques, and binary file operations. The idea to develop a Battleship (also known as Sea Battle) in C++ came from my understanding of its rules and the Chapter 8 tutorial of JavaScript Head First textbook. I adapted the web-paged game, written in JavaScript, to a text-based one written in C++, and I also changed some game features an added some new functionality, such as pointers, error handling, and binary file operations.

Terms Used:

- *almanac* – a binary file storing game history (round number, winner, number of guesses)
- *profiles* - a binary file storing players' data
  (player name, number of victories, and % success rate)

1. Game Rules

A user can play for both players, switching roles, or two users can play on one computer, switching turns.

A player A fires ships of player B by guessing their coordinates. Each guess results in "Missed", "Hit", or "Sunk". Players fire at each other's ships until all ship of one of the players are sunk. The player who sunk all enemy's battleships is a winner. Players can play several rounds.

Every time the ship is sunk, the number of sunk ships is incremented and compared to the total number of ships. If the numbers are equal, the round is over.

Development Summary

	Total	**Code**	**Comments**	Empty	Includes
Main.cpp	264	205	41	18	14
Round.h	39	27	9	3	5
Round.cpp	30	17	10	3	3
Profile.h	80	59	12	9	3
Profile.cpp	45	35	8	2	5
Player.h	44	26	12	6	4
Player.cpp	73	56	13	4	4
Board.h	44	33	7	4	2
Board.cpp	226	199	22	5	7
Ship.h	46	36	7	3	1
Ship.cpp	55	42	11	2	4
Template.h	29	17	7	5	2
Streams.h	72	56	12	4	3
Total	**1047**	**808**	**171**	68	57
			21%		

The project is written in C++ using NetBeans IDE 8.2 and Dev-C++ 5.11.

2. Concepts Used

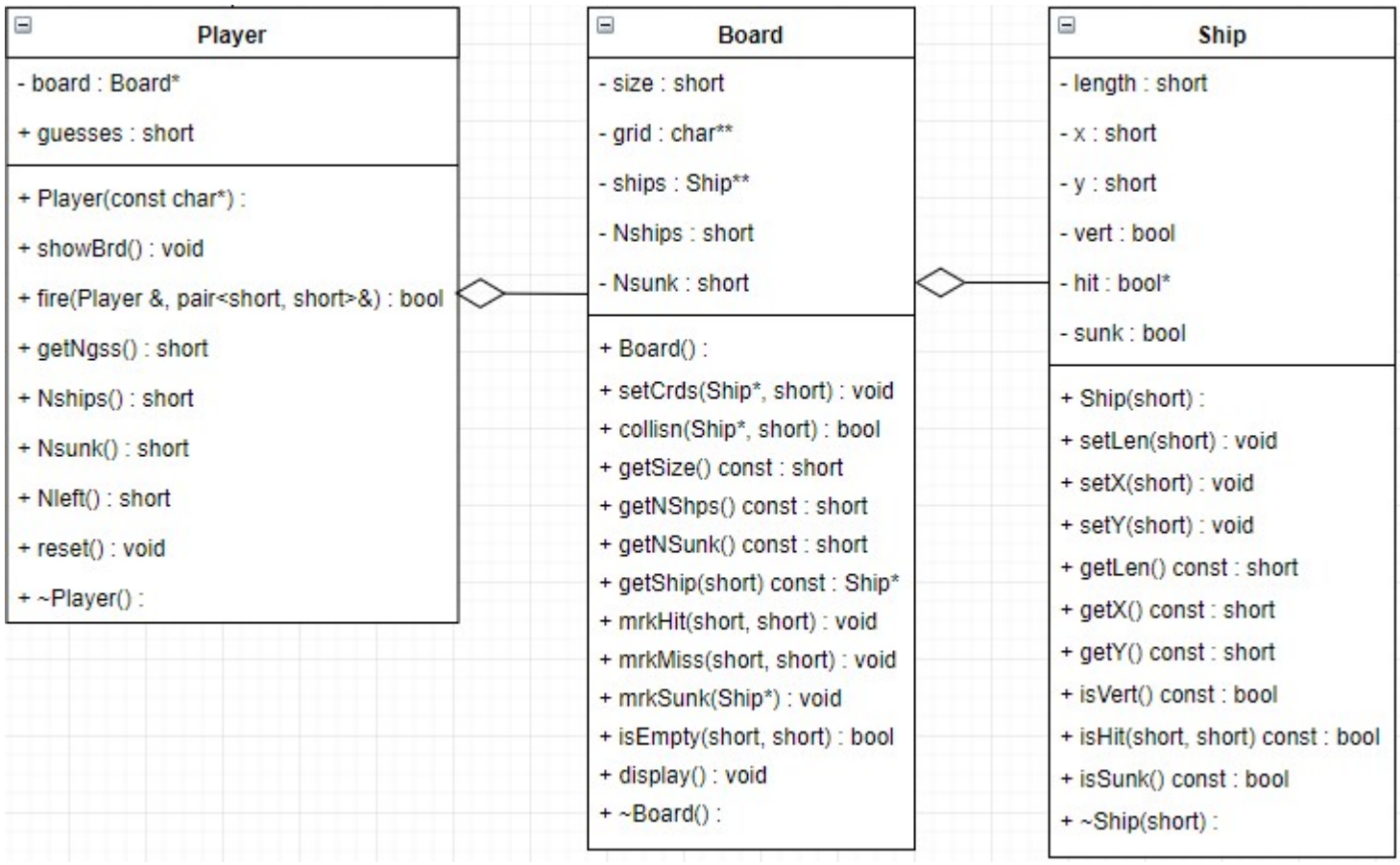
- Classes:
  - Profile (base for Player)
  - Player (derived from Profile)
  - Board (contains Ship array)
  - Ship (nested inside Board)
  - Round
- Arrays:
  - 2-D: pointers to Ship objects, inside a Board class,
  - 1-D: bool “hit” inside a Ship class,
  - 2-D: pointers to Player objects
  - 2-D: “grid” (of chars) inside a Board object
- Pointers and references:

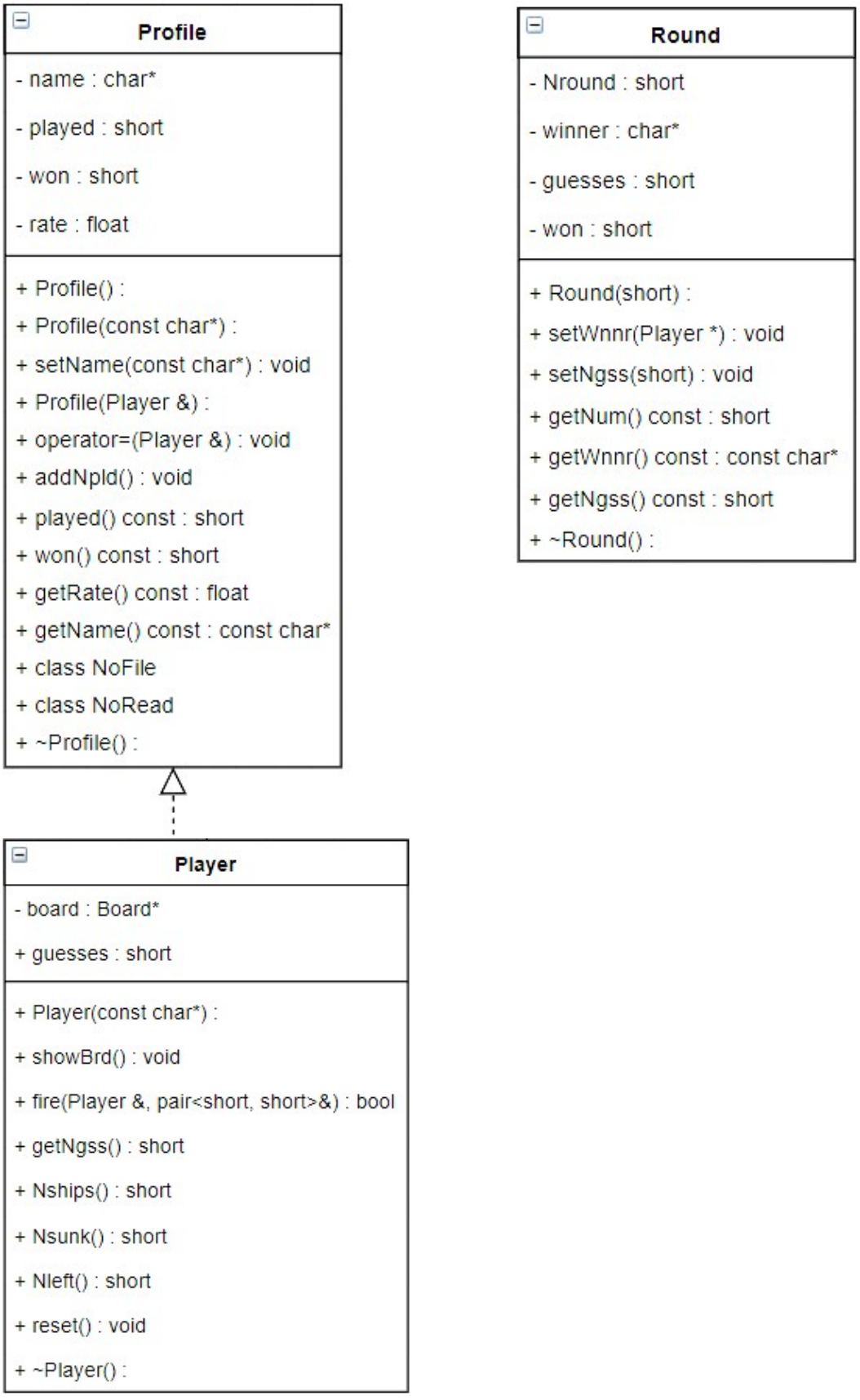
  - passing Round, Player, Board, and Ship objects to functions
  - passing a pair of values to and returning it from functions
- Exception Handling
  - Using try {} catch(bad_alloc) construct
    to check if the dynamic memory was successfully allocated
- Templates:
  - Dynamically allocating memory for any data type
- Dynamic memory allocation:
  - “grid” and “ships” inside Board objects,
  - “hit” inside Ship objects,
- Character arrays: c-strings, for “name” in a Profile class
- String objects: in takeGuess(), for guess input testing
- Binary files:
  - almanac – round results (round#, winner name, # of guesses it took to win),
  - profiles – player success rates (player name, # of games won, success rate %)
- Pairs: as a pair of ship XY coordinates on a grid

3. Specifications

3.1 Classes

In this program, I used five classes, with Ship as a nested structure inside Board.


Player
- board : Board*
+ guesses : short
+ Player(const char*) :
+ showBrd() : void
+ fire(Player &, pair<short, short>&) : bool
+ getNgss() : short
+ Nships() : short
+ Nsunk() : short
+ Nleft() : short
+ reset() : void
+ ~Player() :
Board
- size : short
- grid : char**
- ships : Ship**
- Nships : short
- Nsunk : short
+ Board() :
+ setCrds(Ship*, short) : void
+ collisn(Ship*, short) : bool
+ getSize() const : short
+ getNShps() const : short
+ getNSunk() const : short
+ getShip(short) const : Ship*
+ mrkHit(short, short) : void
+ mrkMiss(short, short) : void
+ mrkSunk(Ship*) : void
+ isEmpty(short, short) : bool
+ display() : void
+ ~Board() :
Ship
- length : short
- x : short
- y : short
- vert : bool
- hit : bool*
- sunk : bool
+ Ship(short) :
+ setLen(short) : void
+ setX(short) : void
+ setY(short) : void
+ getLen() const : short
+ getX() const : short
+ getY() const : short
+ isVert() const : bool
+ isHit(short, short) const : bool
+ isSunk() const : bool
+ ~Ship(short) :


Profile
- name : char*
- played : short
- won : short
- rate : float
+ Profile() :
+ Profile(const char*) :
+ setName(const char*) : void
+ Profile(Player &) :
+ operator=(Player &) : void
+ addNpld() : void
+ played() const : short
+ won() const : short
+ getRate() const : float
+ getName() const : const char*
+ class NoFile
+ class NoRead
+ ~Profile() :
Round
- Nround : short
- winner : char*
- guesses : short
- won : short
+ Round(short) :
+ setWnnr(Player *) : void
+ setNgss(short) : void
+ getNum() const : short
+ getWnnr() const : const char*
+ getNgss() const : short
+ ~Round() :
Player
- board : Board*
+ guesses : short
+ Player(const char*) :
+ showBrd() : void
+ fire(Player &, pair<short, short>&) : bool
+ getNgss() : short
+ Nships() : short
+ Nsunk() : short
+ Nleft() : short
+ reset() : void
+ ~Player() :

3.2 Inputs/Outputs

After the game welcome title appears, the user is asked to enter the first player's name. If the user just presses Enter, the default name, like "John" or "Mary," is assigned. Then, the user is asked to enter a name of the second player, and the same order repeats.

After the user introduces both players by name, in the beginning, the program pauses until the user presses Enter, to start a game.

A player 1 is prompted for a ship coordinate, in a format A0 (letter-number). If the user enters something inadequate, or a coordinate is overboard, or this spot has already been fired at, the user is re-prompted until the valid coordinate is entered.

```
ROUND 1!
3 ships are created for each player.
~   ~   ~   ~   ~   ~   ~   ~   ~   ~   ~
~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Press Enter to start a game.

John's turn:
        Firing on Mary's board:
        (3 ships left.)
      A   B   C   D   E   F
    +---+---+---+---+---+---+
 0  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 1  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 2  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 3  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 4  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 5  |   |   |   |   |   |   |
    +---+---+---+---+---+---+

        ~~~~ Fire! ~~~~
        Type a coordinate from A0 to F5:
```

```
John's turn:
        Firing on Mary's board:
        (3 ships left.)
      A   B   C   D   E   F
    +---+---+---+---+---+---+
 0  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 1  |   |   |   |   |   |   |
    +---+---+---+---+---+---+
 2  |   | - |   |   | - |   |
    +---+---+---+---+---+---+
 3  |   |   | - |   |   |   |
    +---+---+---+---+---+---+
 4  |   |   |   | - |   |   |
    +---+---+---+---+---+---+
 5  |   |   |   |   |   |   |
    +---+---+---+---+---+---+

        ~~~~ Fire! ~~~~
        Type a coordinate from A0 to F5: e2
        You already fired at this spot.
        Try another one.
        Type a coordinate from A0 to F5: s9
        Error!
        Type a coordinate from A0 to F5: ddd
        Error! No 2 chars were entered
        Type a coordinate from A0 to F5:
```

Then – “Fire!” If the ship is hit, the spot is marked ‘X’ on the enemy’s board. If all the spots on a single ship are hit, the ship is sunk, and all the sunk ship’s spots are marked ‘S’. If no enemy ship was hit, the spot is marked as missed.

```
      ~~~~~ Fire! ~~~~~
      Type a coordinate from A0 to F5: a2
        A   B   C   D   E   F
      +---+---+---+---+---+---+
  0   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  1   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  2   | - |   |   |   |   |   |
      +---+---+---+---+---+---+
  3   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  4   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  5   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
          Missed!
```

```
      ~~~~~ Fire! ~~~~~
      Type a coordinate from A0 to F5: f5
        A   B   C   D   E   F
      +---+---+---+---+---+---+
  0   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  1   | - |   |   |   |   |   |
      +---+---+---+---+---+---+
  2   |   |   |   |   |   | - |
      +---+---+---+---+---+---+
  3   | - |   |   |   |   |   |
      +---+---+---+---+---+---+
  4   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  5   | - |   |   |   |   | X |
      +---+---+---+---+---+---+
          The ship is hit!
```

After each guess, the enemy’s board with marks (hit/miss/sunk) is displayed, and the program pauses again, until the player plays Enter. This gives players time to evaluate the situation, plan the strategy, and switch turns.

```
      ~~~~~ Fire! ~~~~~
      Type a coordinate from A0 to F5: f3
        A   B   C   D   E   F
      +---+---+---+---+---+---+
  0   |   |   |   |   |   | - |
      +---+---+---+---+---+---+
  1   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  2   | - |   |   |   |   | - |
      +---+---+---+---+---+---+
  3   |   |   |   |   | - | S |
      +---+---+---+---+---+---+
  4   | - |   |   |   |   | S |
      +---+---+---+---+---+---+
  5   |   |   |   |   |   | S |
      +---+---+---+---+---+---+
          The ship is sunk!
~   ~   ~   ~   ~   ~   ~   ~   ~   ~   ~   ~
~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Press Enter to continue.
```

```
      ~~~~~ Fire! ~~~~~
      Type a coordinate from A0 to F5: b0
        A   B   C   D   E   F
      +---+---+---+---+---+---+
  0   |   | S |   |   |   |   |
      +---+---+---+---+---+---+
  1   | - | S |   | S |   |   |
      +---+---+---+---+---+---+
  2   |   | S |   | S |   | - |
      +---+---+---+---+---+---+
  3   |   | - |   | S |   |   |
      +---+---+---+---+---+---+
  4   |   |   |   |   |   |   |
      +---+---+---+---+---+---+
  5   |   | - |   | S | S | S |
      +---+---+---+---+---+---+
          The ship is sunk!
          All ships are sunk, so ----- GAME OVER!

And the WINNER is Mary
          Mary beat John in the round 1 with 13 guesses

Do you want to play again?
Yes (Y) or No (any other key):
```

After each round, the info on winner and number of guesses it took him or her to win, is written to *almanac*, a binary history file.

```
Do you want to play again?
Yes (Y) or No (any other key): q

  GAME HISTORY

Round # |        Winner        | Guesses
-------------------------------------------
     1    | Mary                |   18
     2    | Mary                |   16
-------------------------------------------

 PLAYERS INFO

Name                 | Games Won | Success Rate
-------------------------------------------------
John                 |     0     |       0%
Mary                 |     2     |     100%
-------------------------------------------------
Total games played: 2


That's it,
Bye.

RUN SUCCESSFUL (total time: 4m 15s)
```

After each round, players are given a choice to start a new round or to end the program. If players choose to continue, they type ‘y’ or ‘Y’, and the new round starts. If they do not want to continue, they press any other key, results are written to *almanac* and displayed on the screen:

- Round results – round number, winner name, number of guesses to win
- Player profiles – names, number of games won, success rate (100% * won / played.)
  Total row - number of rounds played.

3.3 Pseudocode

Simplified, the game is built in the following way:

1. Set the random number generator
2. Declare and initialize variables:

   a. Two file-stream objects: almanac and profiles
   b. Two instances of Player
      i. Each Player has one instance of Board,
      ii. Each Board has an array of Ship pointers
3. Create Round.
4. Guess and fire
   a. Hit ? Mark the hit spot 'X' on the grid.
      If hit, all spots of this ship are hit ? If yes, sunk!
      i. If sunk, mark the ship 'S'.
         If # sunk == # ships, – **Game over!** Go to #5.
   b. Missed ? Mark the missed spot "-" on the grid.
   c. Switch turns are repeat #4.

5. Write the round results to *almanac*.
6. Play again ?
   If yes, repeat from #3.
   If no, close *almanac*.
7. Reopen and read *almanac*.
   Display its contents as a table: round #, winner name, # guesses it took to win
8. Open *profiles* for output.
   Write each player profile: name, # games won, % success rate.
   Close *profiles*.
9. Reopen *profiles* for input.
   Read *profiles* twice, to a new instance of Profile
   Close *profiles*.
10. Display both players statistics as a table: name, # games won, % success rate.
11. Delete both Player objects.
12. End the program.

3.4 Function Descriptions

The program consists of _____ functions, including **main**. Here are some of them.

takeName prompts a player for a name and returns a name entered. If nothing was entered, a default name is assigned.

Constructors:

Profile assigns a name for a player. Nplayed = 0, Nwon = 0, rate = 0.0.

Player, in addition to Profile, allocates memory for Board*.

Each new round adds 1 to Nplayed, each won game adds 1 to Nwon.

Board allocates memory for the board grid (char**) and ships (Ship**).

Sets Nships. Nsunk = 0. Generates each ship coordinates by calling **setCrds** function.

Ship assigns a direction, length, starting XY coordinates,

"hit" and "sunk" boolean properties to a ship (all false, initially).

Round: sets Nround, winner (empty c-string), guesses = 0

Mutator function:

Board::setCrds generates each ship direction (vertical or horizontal) and its starting XY coordinates. Then, depending on a ship direction, it extends coordinates down or to the right. If the ship overflows the board, X in a horizontal ship is shifted to the left, and Y in a vertical ship is shifted up. After the ship is generated, **collisn** is called. If **collisn** returns true, setCrds is called recursively.

Board::collisn checks if a generated ship overlaps with any previously generated ships.

Accessor functions, inside each class, allow the program to access object properties without changing them.

Game flow:

takeGuess accepts player's guess as XY coordinates, X – letter, Y – number.

parseGuess represents a guess as a pair of numeric XY values, as indices: X – column number, and Y – row number.

Player::fire determined if a player hit or missed by calling **Ship::isHit**. If **isHit** returns true, **Ship::isSunk** is called. If isSunk returns true, **Board::mrkSunk** is called. The function returns *true* if all ships are sunk and *false* otherwise.

Ship::isHit determines if the ship is hit by comparing a player guess with the ship locations.

Ship::isSunk check if all spots on a single ship are hit.

Board::mrkSunk marks all the spots of a sunk ship on the board

Board::mrkHit marks a spot with 'X'

Board::mrkMiss marks a spot with '-'

Board::display shows the player's board, with ship marks: hit/miss/sunk, unharmed ships hidden with whitespaces,

Binary file operations with classes:

almanac* << Round writes the game round results to a binary file: round number, winner, number of guesses.

almanac* >> Round reads the game round results from a binary file.

***profiles* << Profile&** writes user info and results: names, number of games played, won, and a calculated success rate to a binary file. Then reads all the info back and displays it as a table.

readProf reads an input file contents and displays it as a table.

Destructors:

~Ship: releases memory for dynamic array "hit"

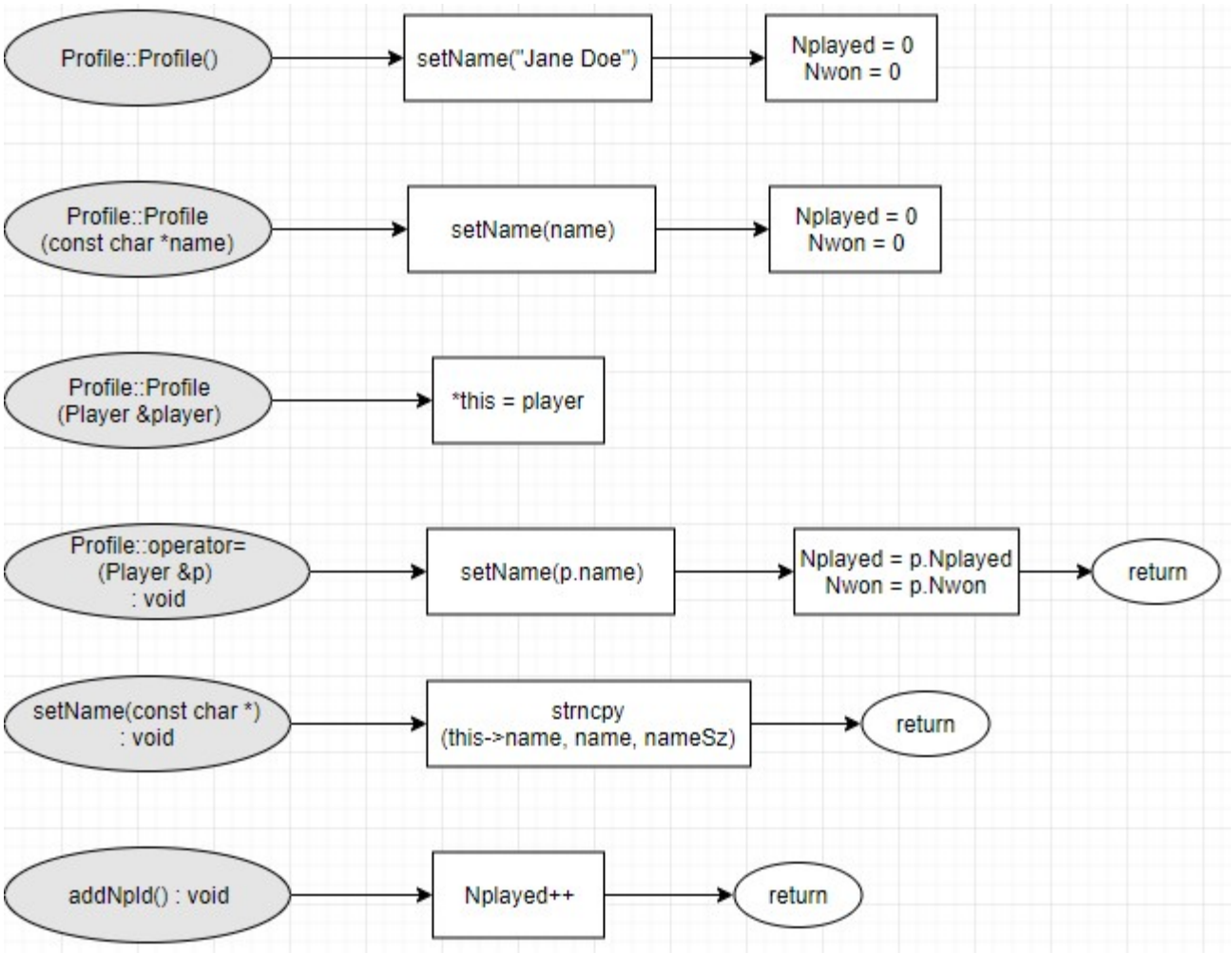
~Board: releases memory for board "grid" and Ship*

~Player: releases memory for Board*

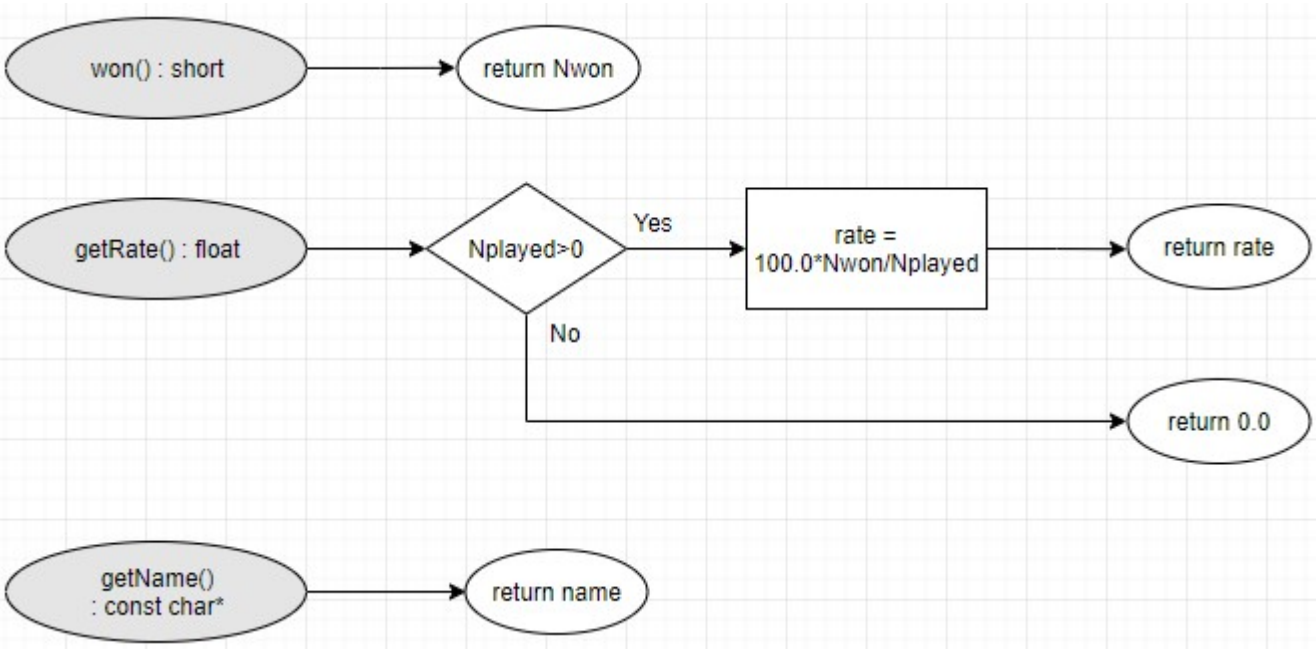
3.5 Flowcharts

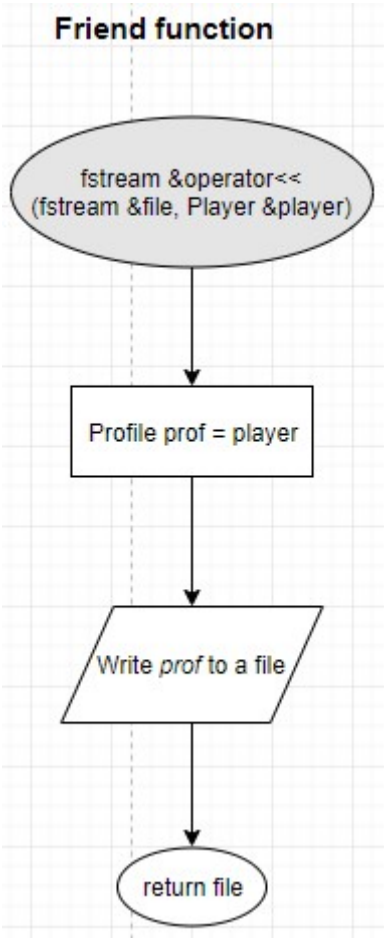
All flowcharts are available in .jpg format, attached to this assignment.

2.4.1 Profile Functions

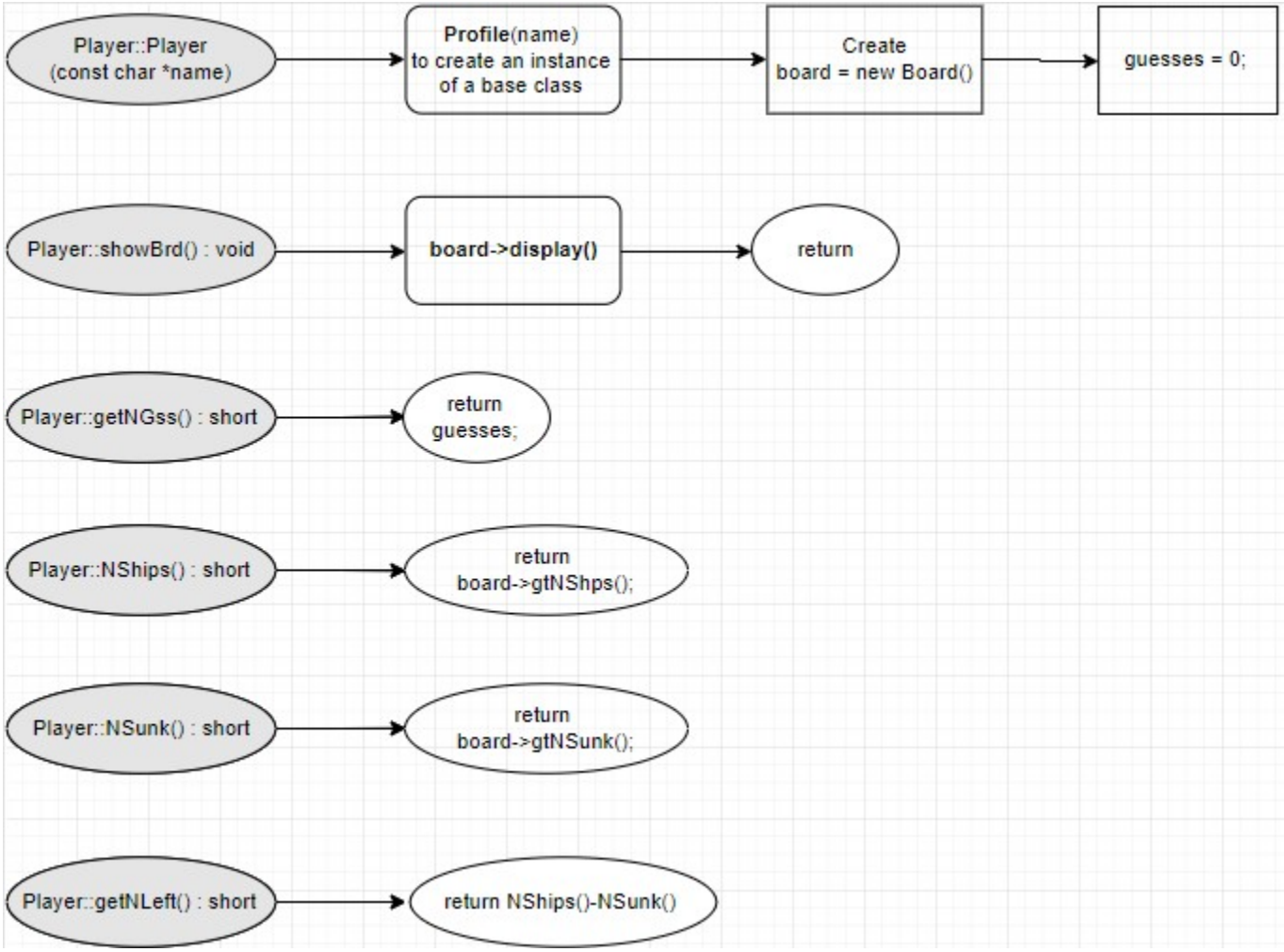


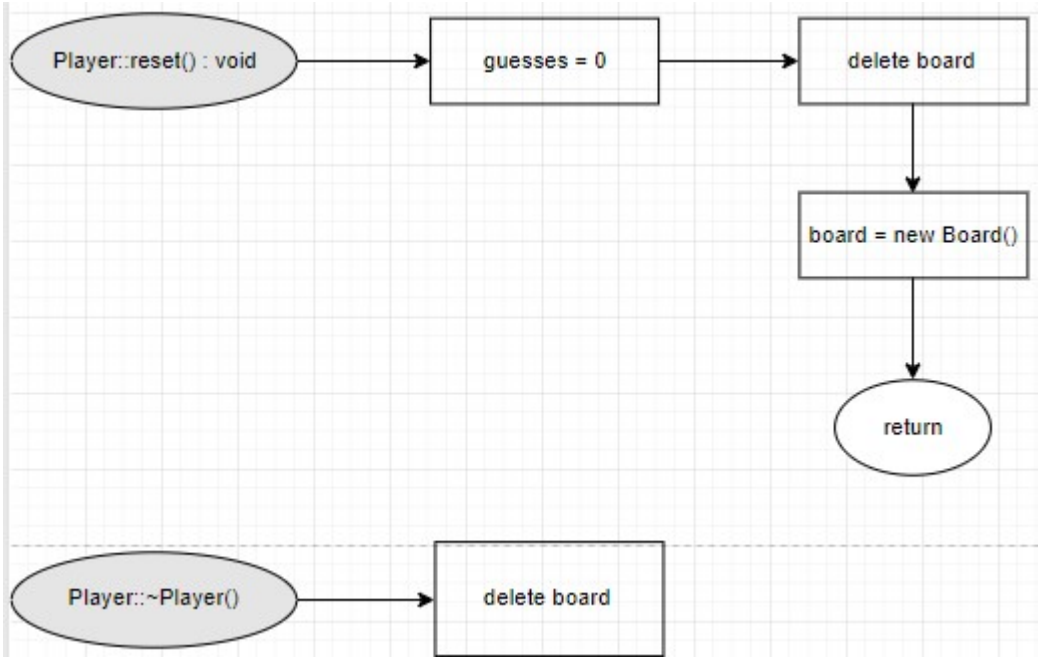
Profile Functions (cont)

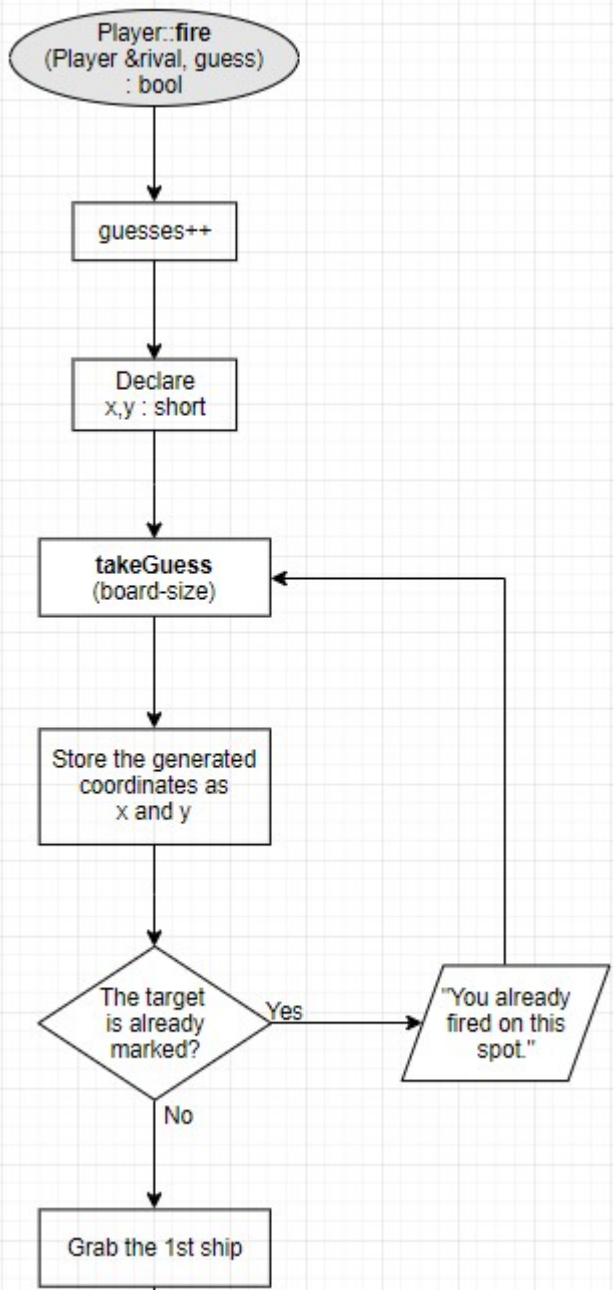




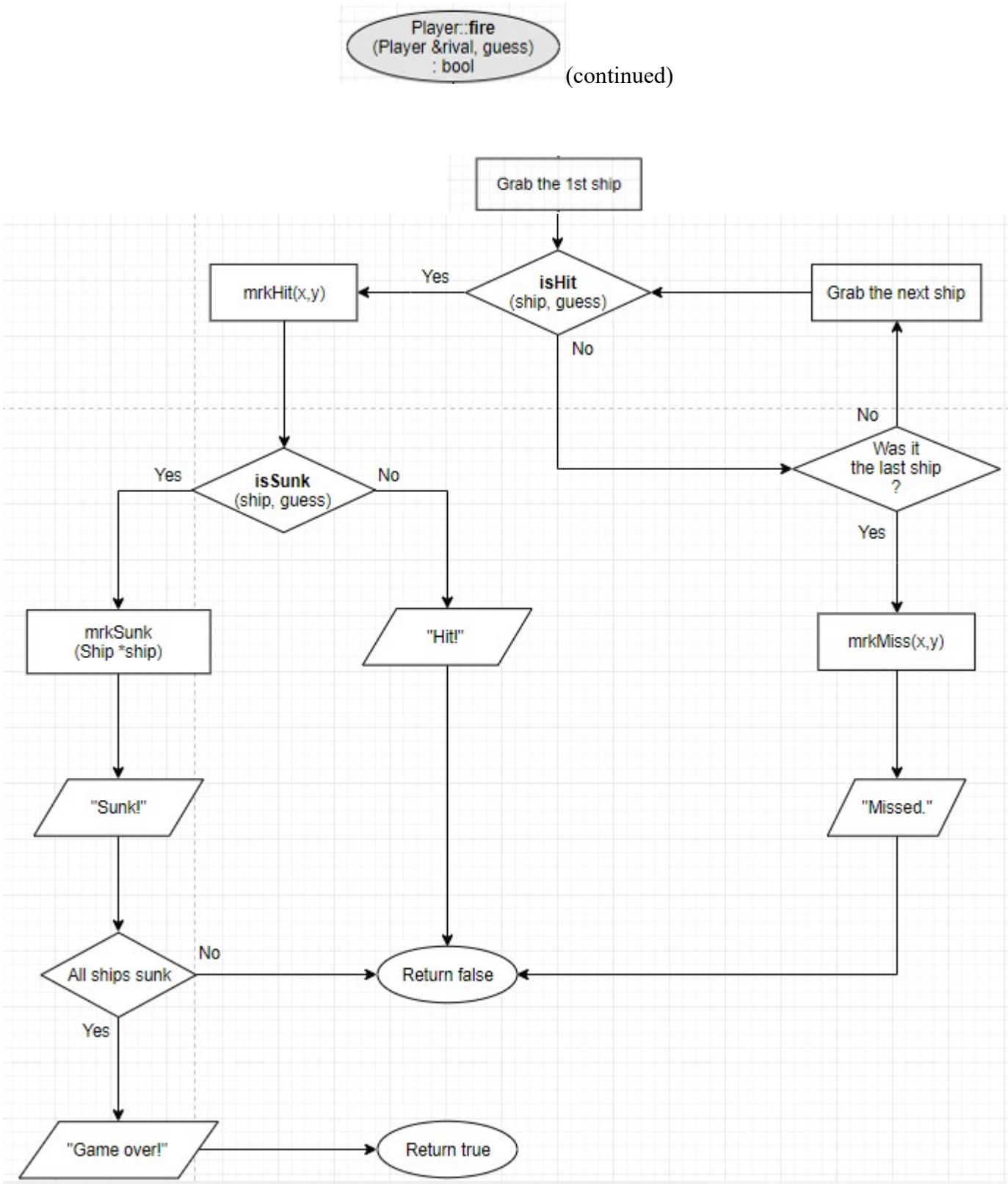
2.4.2 Player functions



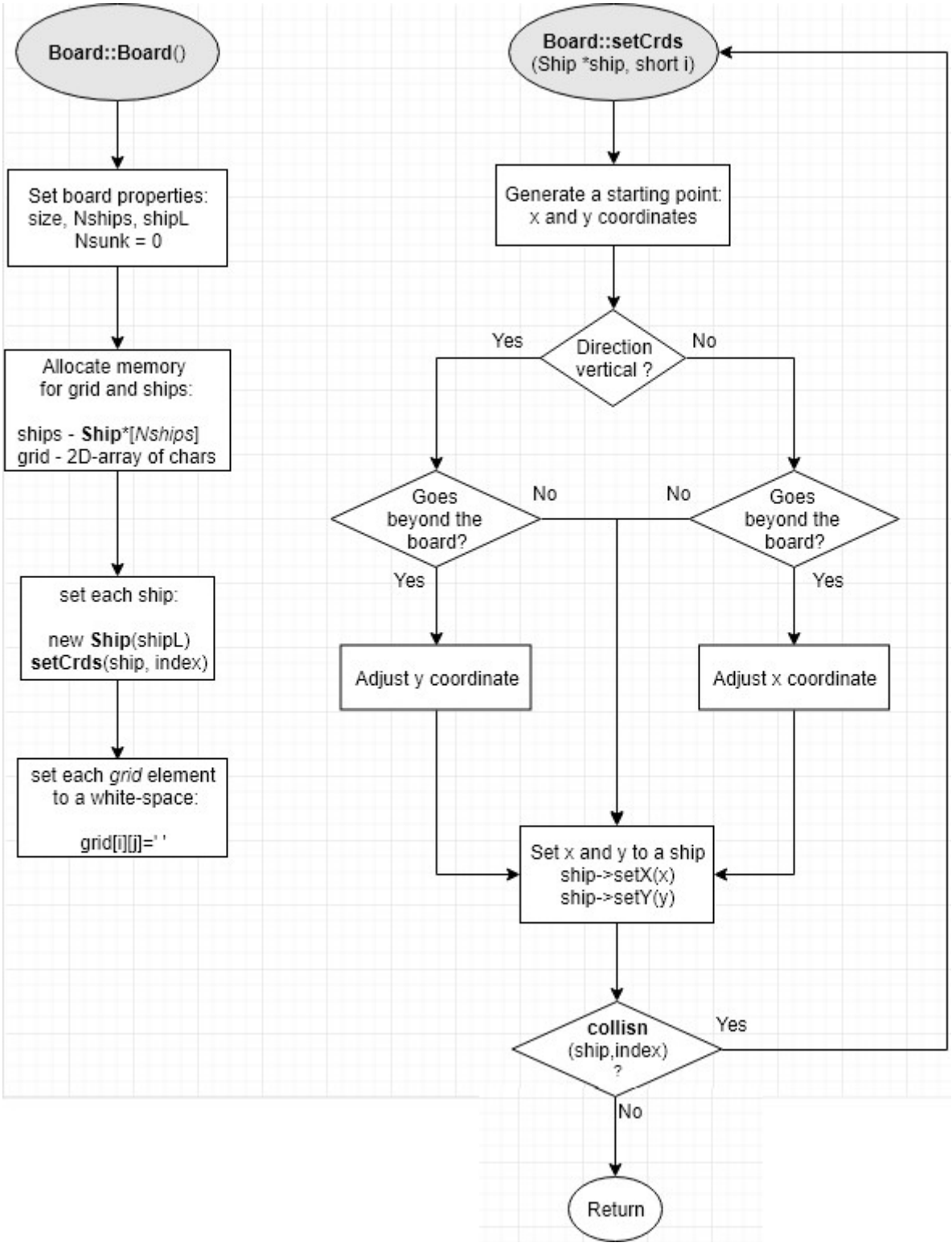




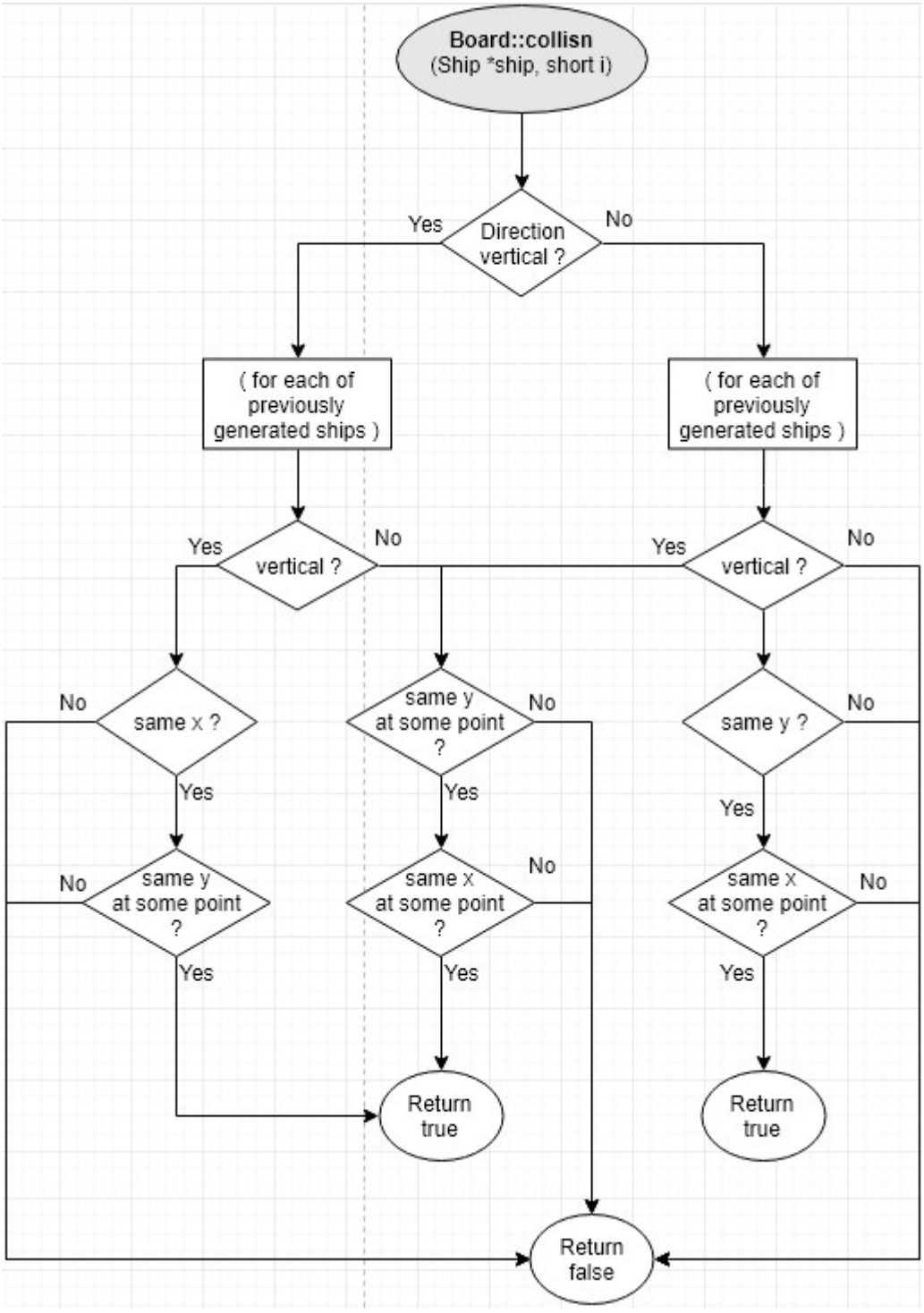
(continued on the next page)

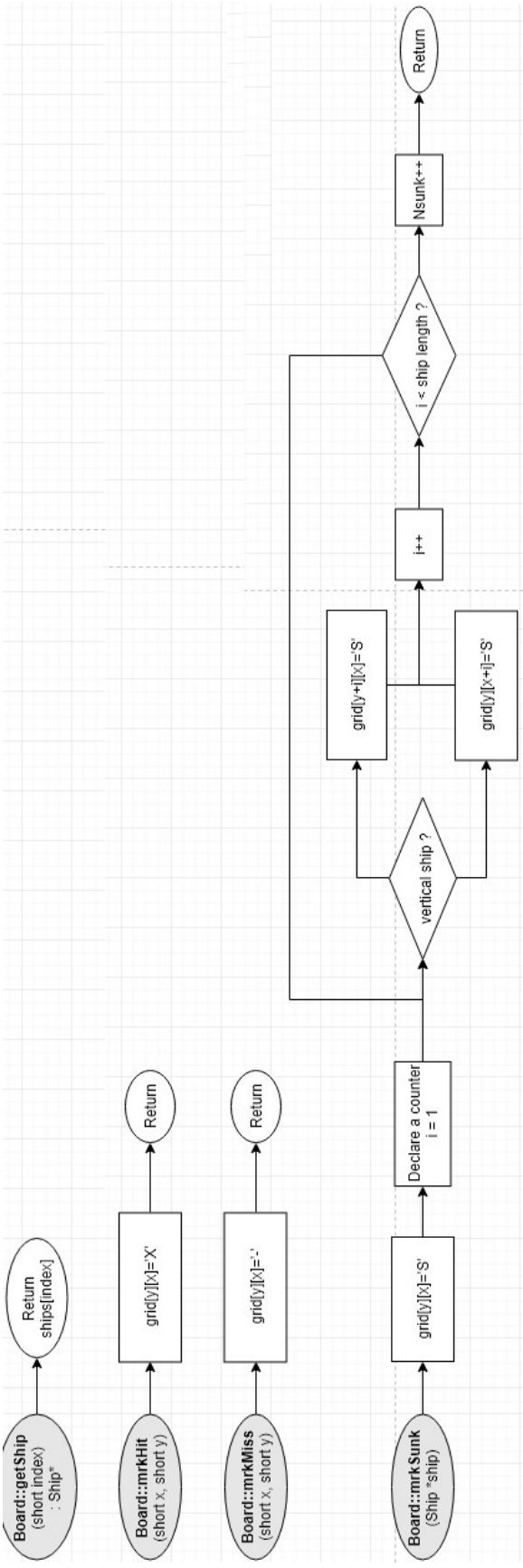

Player::fire
(Player &rival, guess)
: bool
(continued)
Grab the 1st ship
isHit
(ship, guess)
Yes
mrkHit(x,y)
Grab the next ship
No
No
Was it
the last ship
?
Yes
isSunk
(ship, guess)
Yes
No
mrkSunk
(Ship *ship)
"Hit!"
mrkMiss(x,y)
"Sunk!"
"Missed."
All ships sunk
No
Return false
Yes
"Game over!"
Return true

2.4.3 Board functions



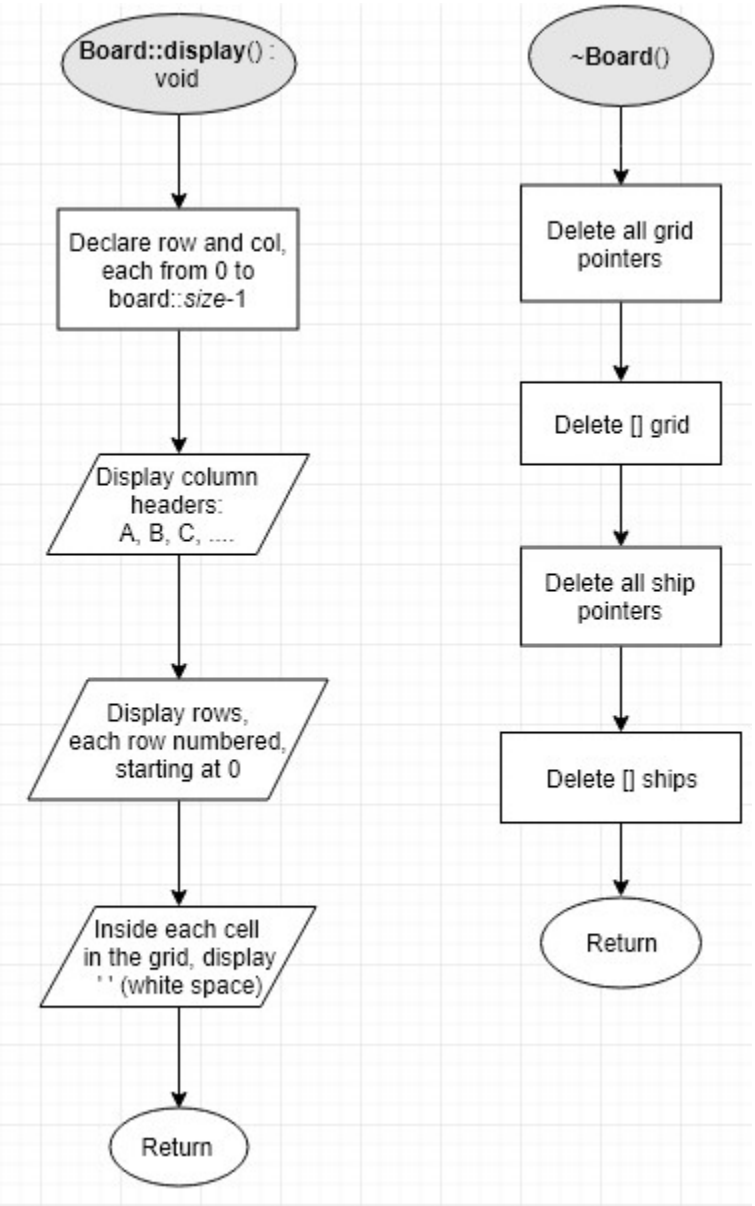
Board functions (cont)



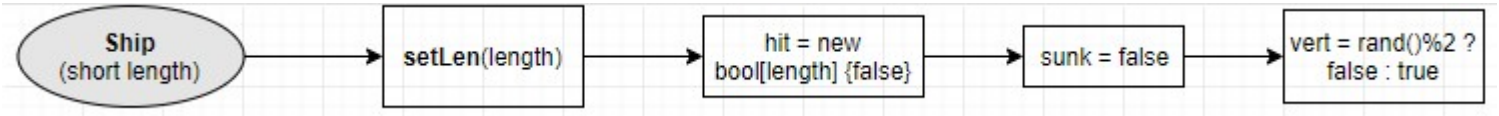


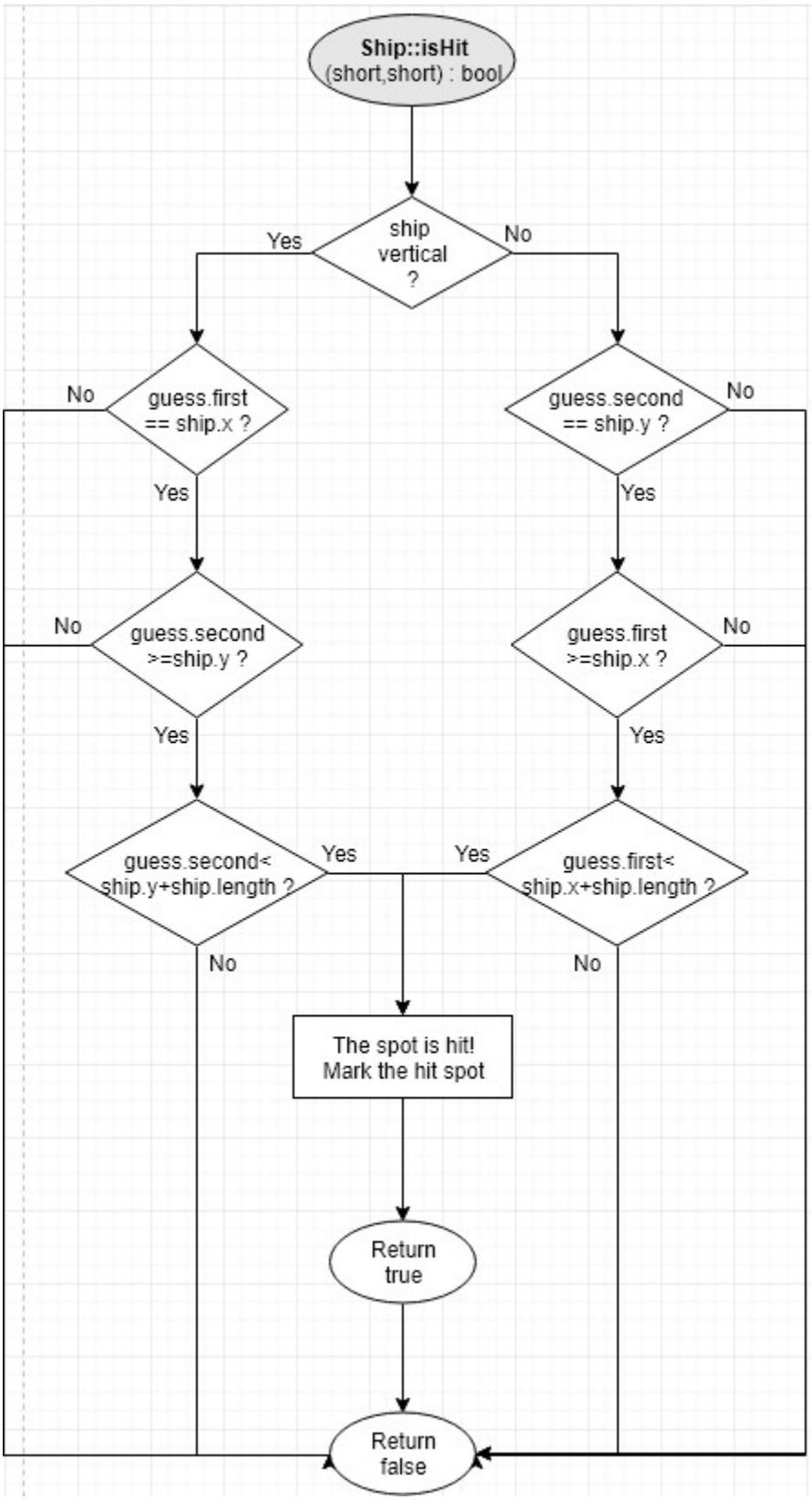
(see the better version in Board_FC .jpg)

Board functions (cont)

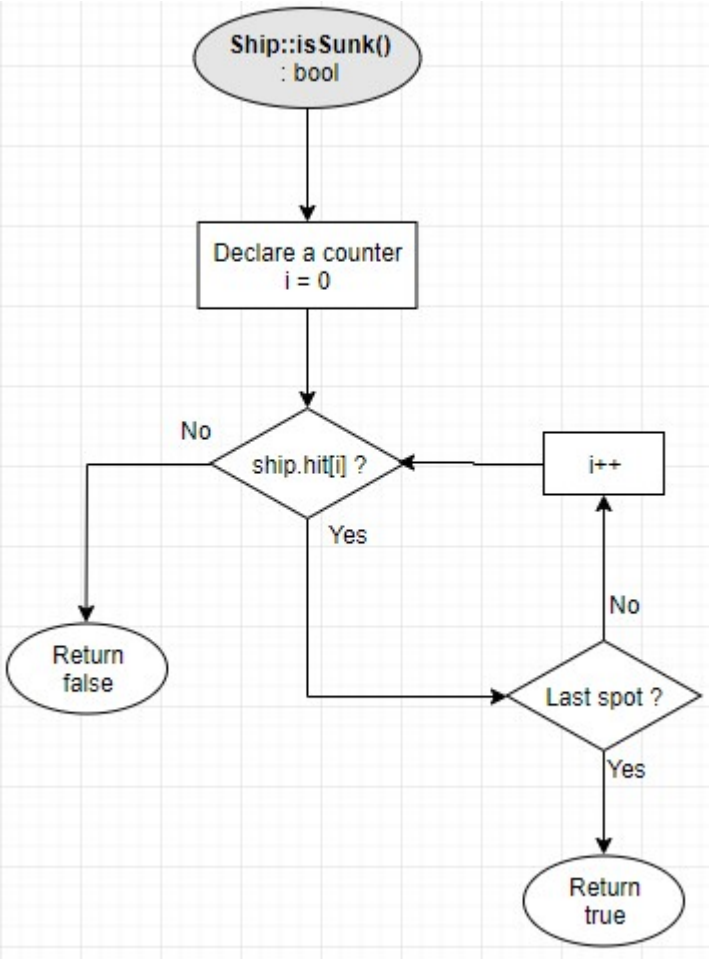


2.4.5 Ship functions

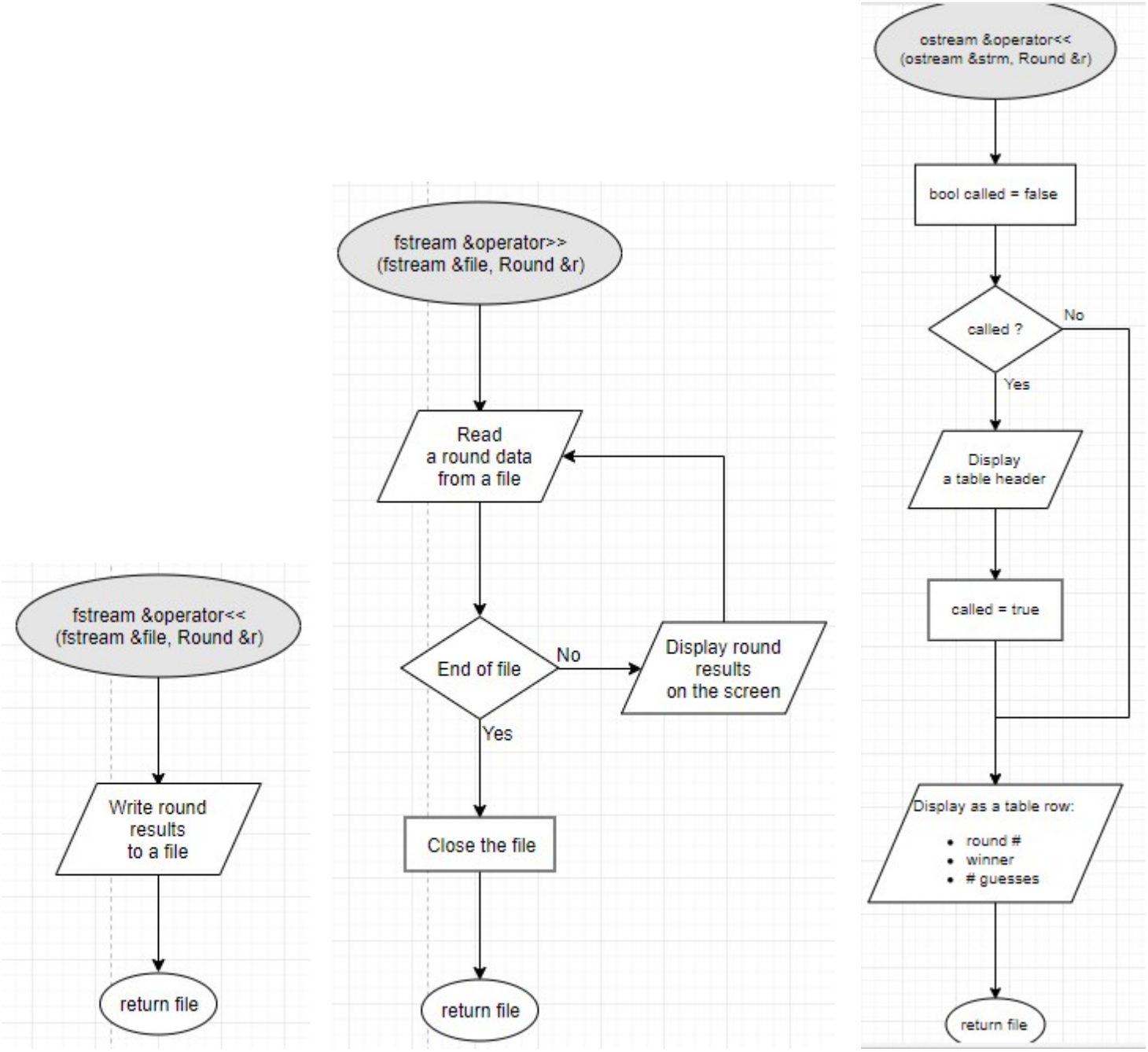



Ship::isHit
(short,short) : bool
ship vertical ?
Yes
No
guess.first == ship.x ?
guess.second == ship.y ?
guess.second >=ship.y ?
guess.first >=ship.x ?
guess.second< ship.y+ship.length ?
guess.first< ship.x+ship.length ?
The spot is hit! Mark the hit spot
Return true
Return false

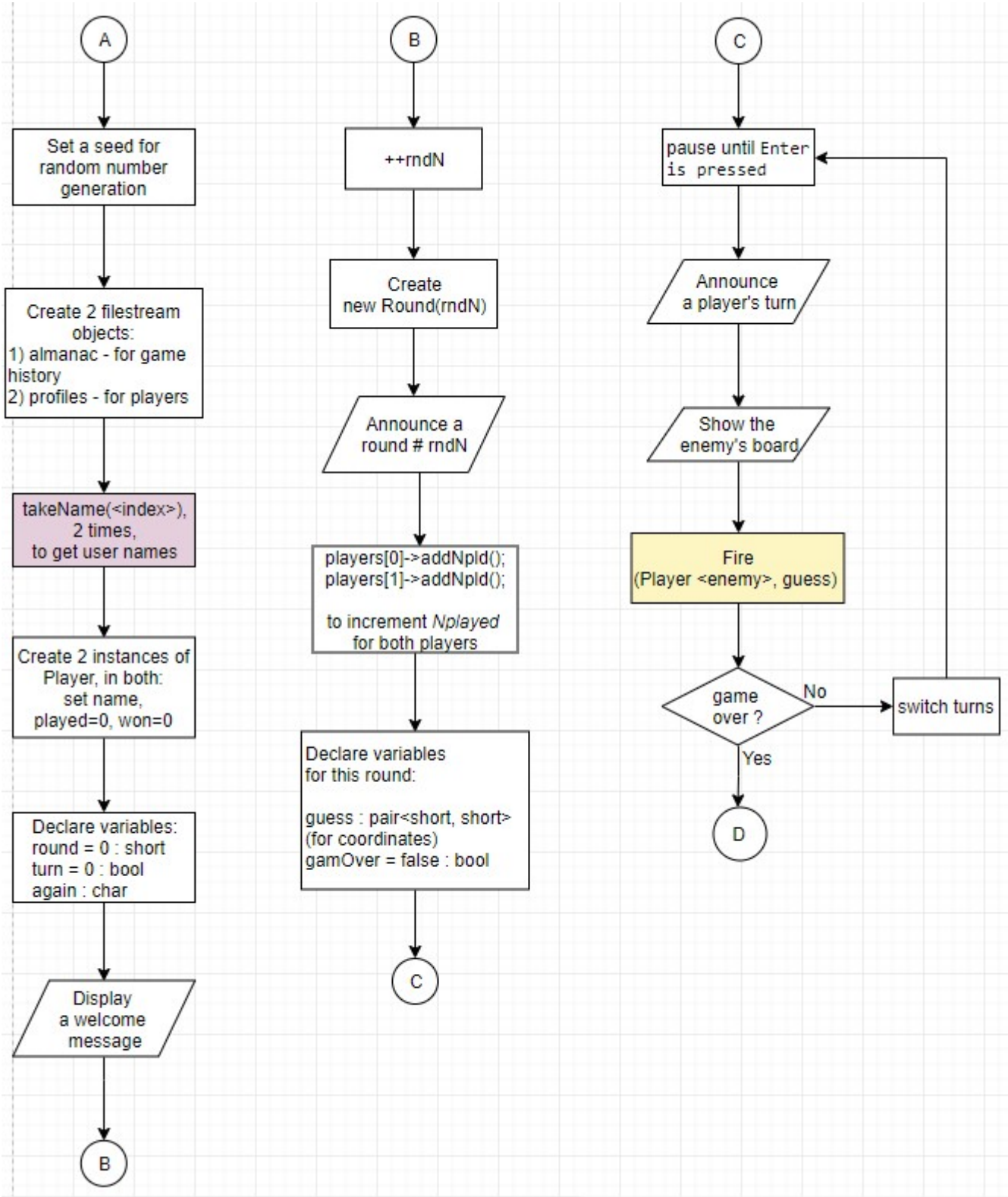
Ship functions (cont)

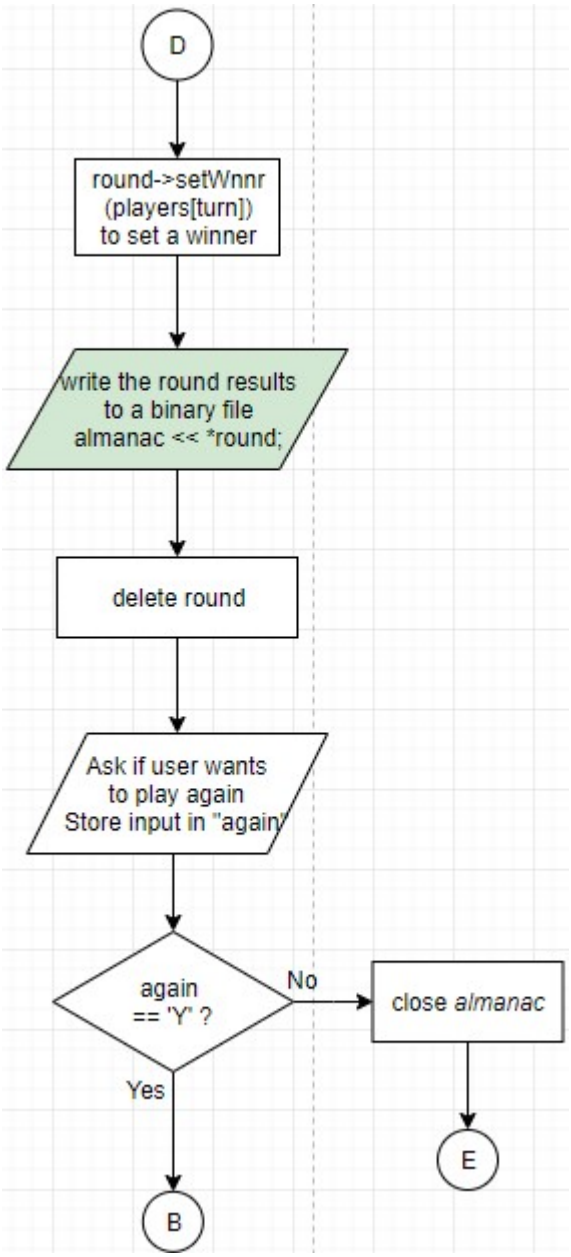


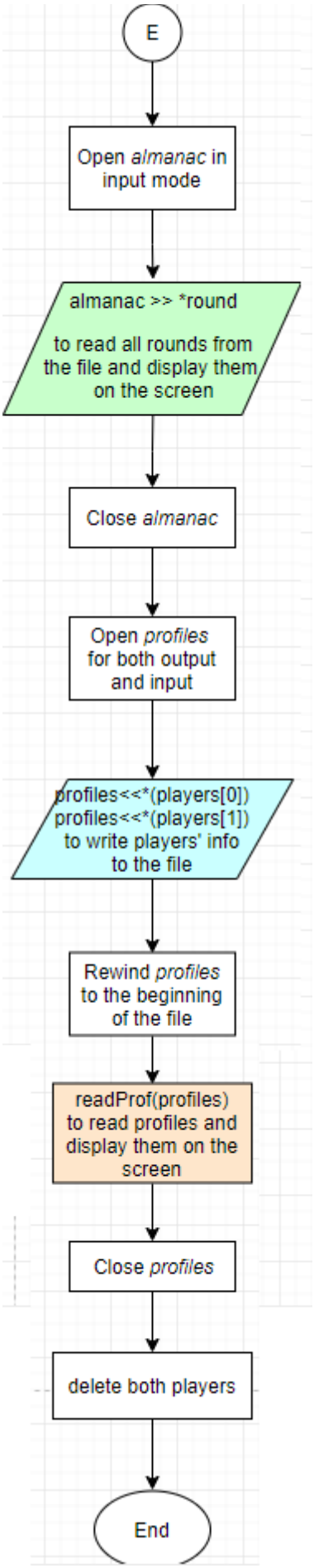
2.4.6 Round Friend functions



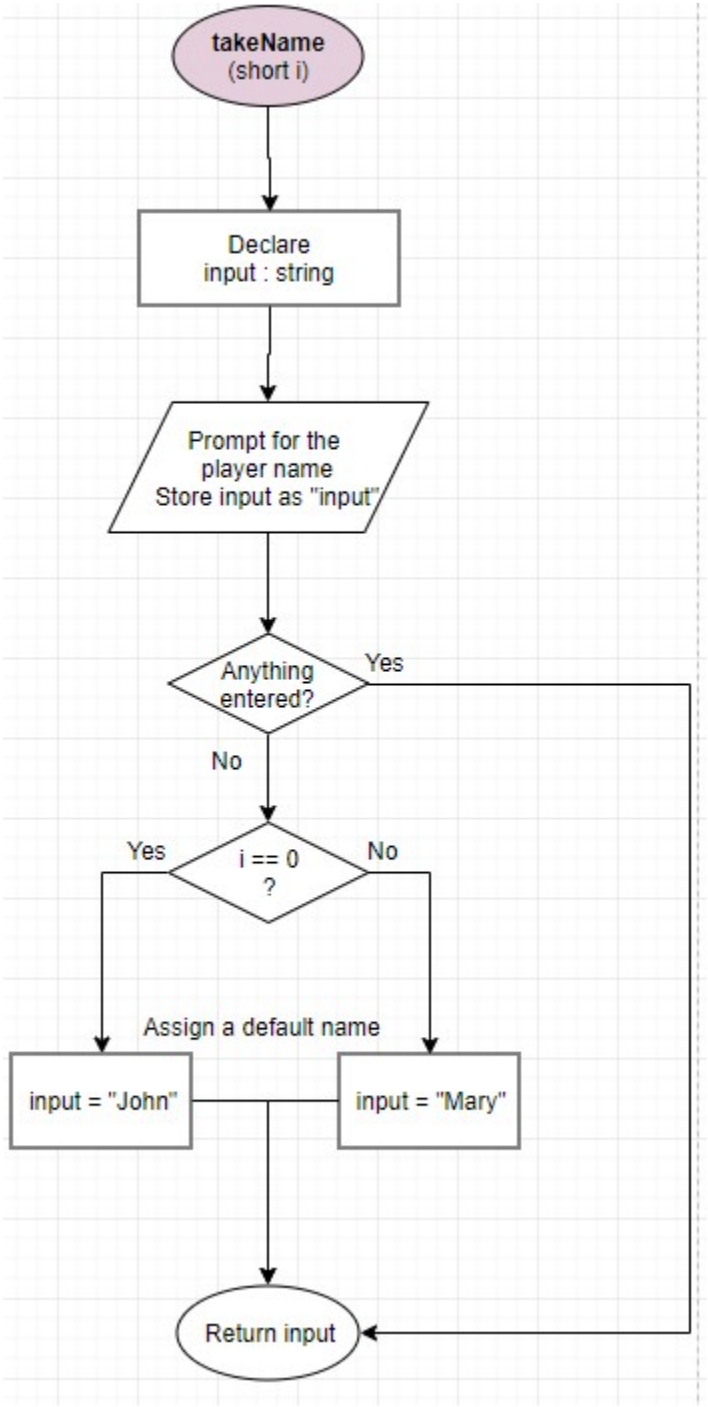
2.4.7 Functions in main.cpp

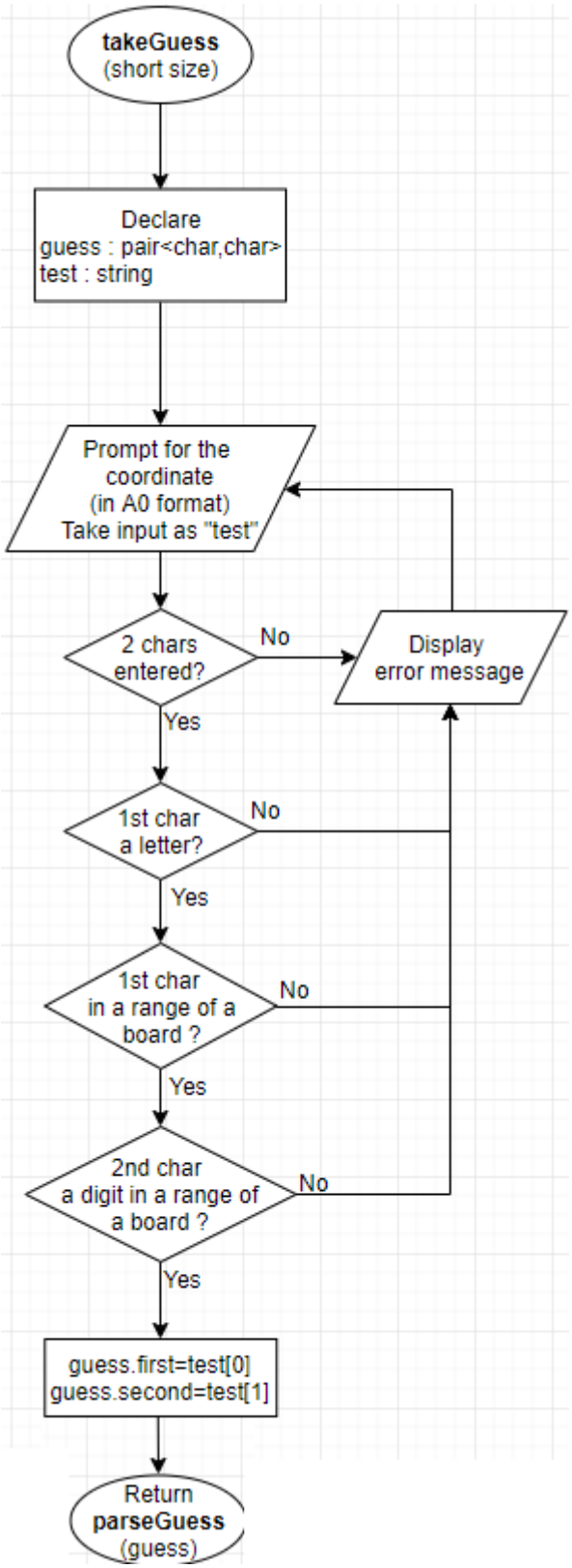


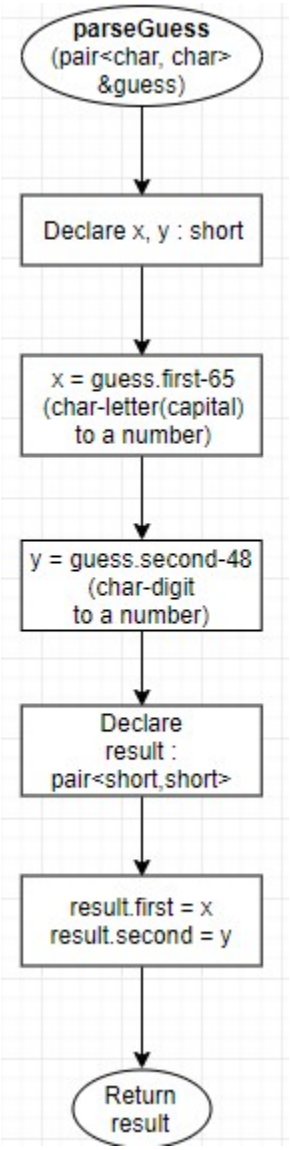

D
round->setWnnr
(players[turn])
to set a winner
write the round results
to a binary file
almanac << *round;
delete round
Ask if user wants
to play again
Store input in "again"
again
== 'Y' ?
No
close almanac
Yes
E
B

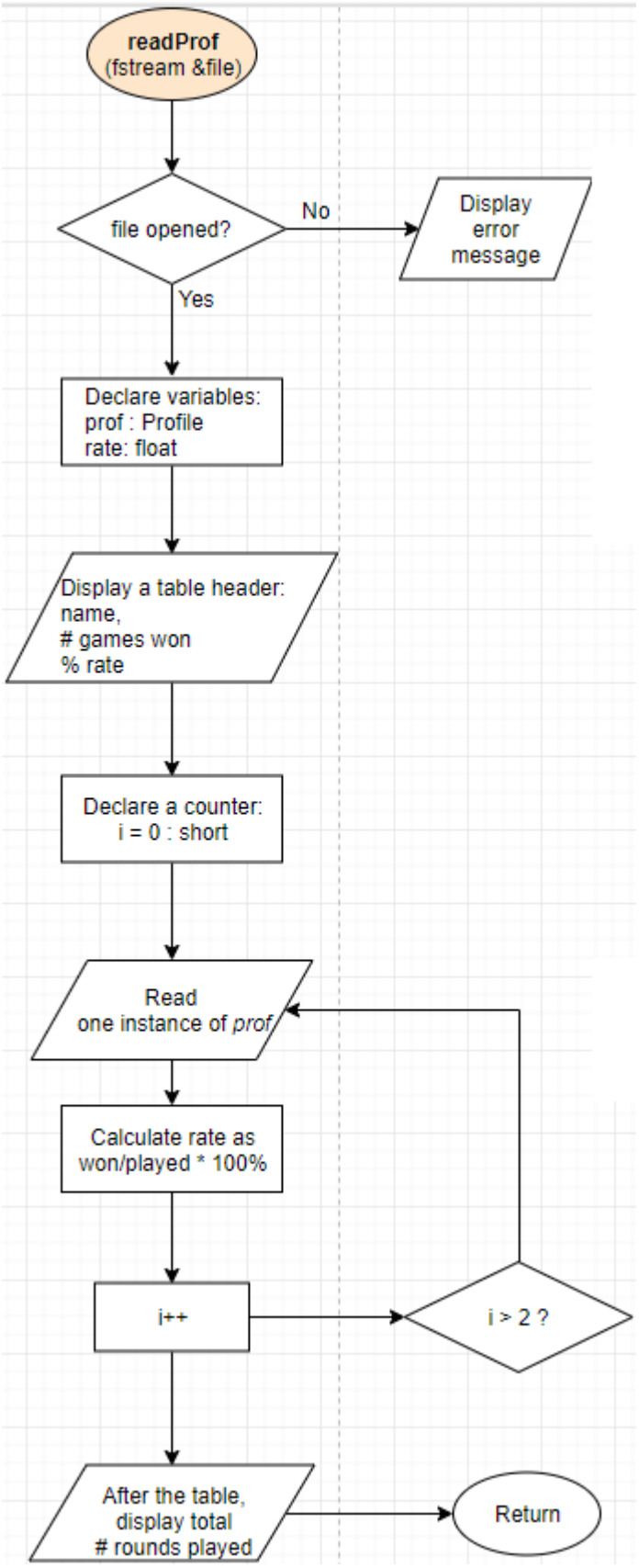

E
Open almanac in
input mode
almanac >> *round
to read all rounds from
the file and display them
on the screen
Close almanac
Open profiles
for both output
and input
profiles<<*(players[0])
profiles<<*(players[1])
to write players' info
to the file
Rewind profiles
to the beginning
of the file
readProf(profiles)
to read profiles and
display them on the
screen
Close profiles
delete both players
End

Main.cpp functions (cont)




takeGuess
(short size)
Declare
guess : pair<char,char>
test : string
Prompt for the
coordinate
(in A0 format)
Take input as "test"
2 chars
entered?
No
Display
error message
Yes
1st char
a letter?
No
Yes
1st char
in a range of a
board ?
No
Yes
2nd char
a digit in a range of
a board ?
No
Yes
guess.first=test[0]
guess.second=test[1]
Return
parseGuess
(guess)


parseGuess
(pair<char, char>
&guess)
Declare x, y : short
x = guess.first-65
(char-letter(capital)
to a number)
y = guess.second-48
(char-digit
to a number)
Declare
result :
pair<short,short>
result.first = x
result.second = y
Return
result


readProf
(fstream &file)
file opened?
No
Display
error
message
Yes
Declare variables:
prof : Profile
rate: float
Display a table header:
name,
games won
% rate
Declare a counter:
i = 0 : short
Read
one instance of prof
Calculate rate as
won/played * 100%
i++
i > 2 ?
After the table,
display total
rounds played
Return

4. References

- The idea is taken from the textbook:
  Head First JavaScript Programming, by Robson and Freeman, 2014: 1st Ed., chapter 8.
- Instead of setting firm ship locations, like `locs[3] = { “A1”, “A2”, “A3” },`
  I generated only the first XY coordinate, and then, depending on a ship direction, extended a ship down (if vertical) or right (if horizontal). This idea was first suggested by Jason Lyllard.

Conclusion

This game is a text-based, utilizing classes and objects, pointers, a template, throw points and exception handling.

However, some of the issues remain unsolved:

1. In main.cpp, function readProf(), when only one round is played, the calculated rate could not be displayed.
   Temporary solution: when only 1 round was played, instead of calling Profile::getRate(), I hard-coded the formula by embedding it into readProf function body.
   (*see* else if(prof.played()==1) { … })
2. In main.cpp, function main(), in rare cases, the program gets stuck after the user types:
   1) player's name (in main() or takeName() functions)
   2) "Y" in response to "Do you want to play again?" after a round finished
3. In extremely rare cases: before starting a new round (second, etc.), the board with ships is not allocated properly. The output just says “Run failed”, and the program aborts. May be Board::setCrds() creates an infinite loop due to Board::collisn() returning “true” too many times, or the previous ships fail to be deleted completely. Could not find solution up to now.

Attachments

Flowcharts:

- Board_FC
- Main_FC
- Player_FC
- Profile_FC
- Round_FC
- Ship_FC

UML Diagrams

- Classes_UML

HTML folder:
Doxygen-documented web-page