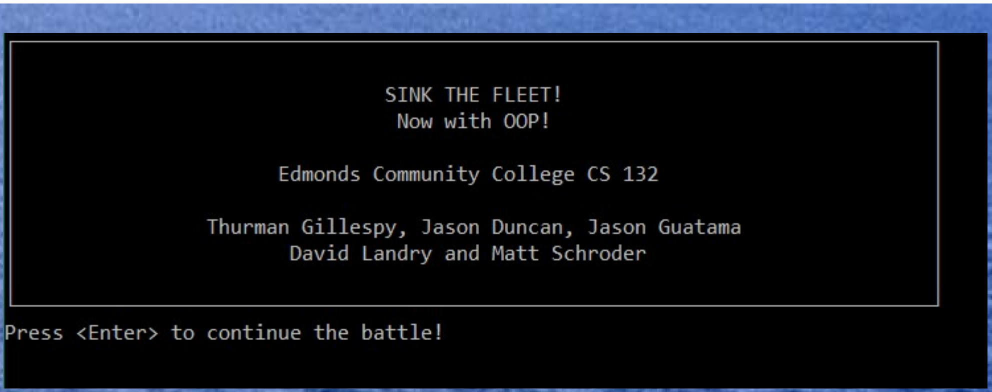
Sink the Fleet

tested by David Landry

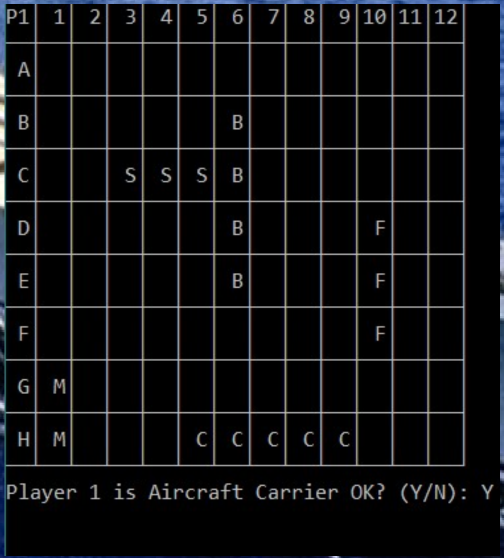
designed by Thurman Gillespey, Jason Gautama, Matt Schroder, Jason Duncan, and David Landry

Overview

- Sink the Fleet is a Battleship program developed for CS 132, taught by Paul Bladek. It was developed in C++ and is almost entirely composed of objects instantiated from one of 6 different classes.



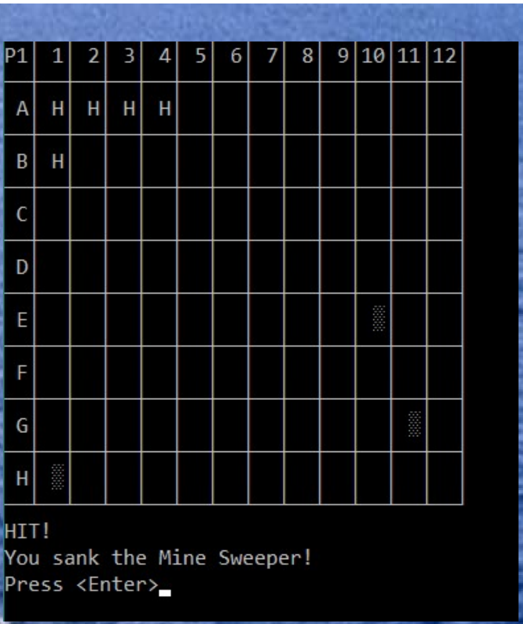
```
                    SINK THE FLEET!
                    Now with OOP!

             Edmonds Community College CS 132

     Thurman Gillespy, Jason Duncan, Jason Guatama
            David Landry and Matt Schroder

Press <Enter> to continue the battle!
```



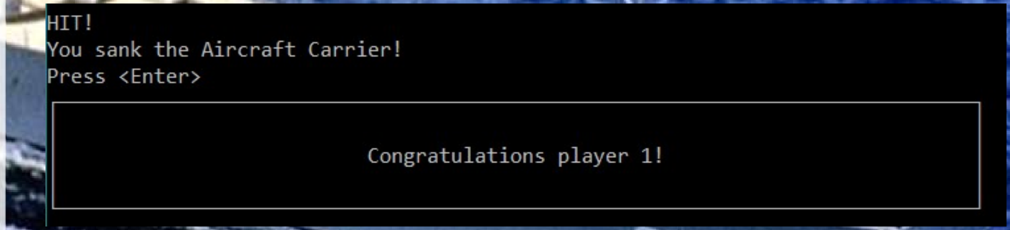
```
P1|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12|
 A |   |   |   |   |   |   |   |   |   |   |   |   |
 B |   |   |   |   |   | B |   |   |   |   |   |   |
 C |   |   | S | S | S | B |   |   |   |   |   |   |
 D |   |   |   |   |   | B |   |   |   | F |   |   |
 E |   |   |   |   |   | B |   |   |   | F |   |   |
 F |   |   |   |   |   |   |   |   |   | F |   |   |
 G | M |   |   |   |   |   |   |   |   |   |   |   |
 H | M |   |   |   | C | C | C | C | C |   |   |   |
Player 1 is Aircraft Carrier OK? (Y/N): Y
```

Overview

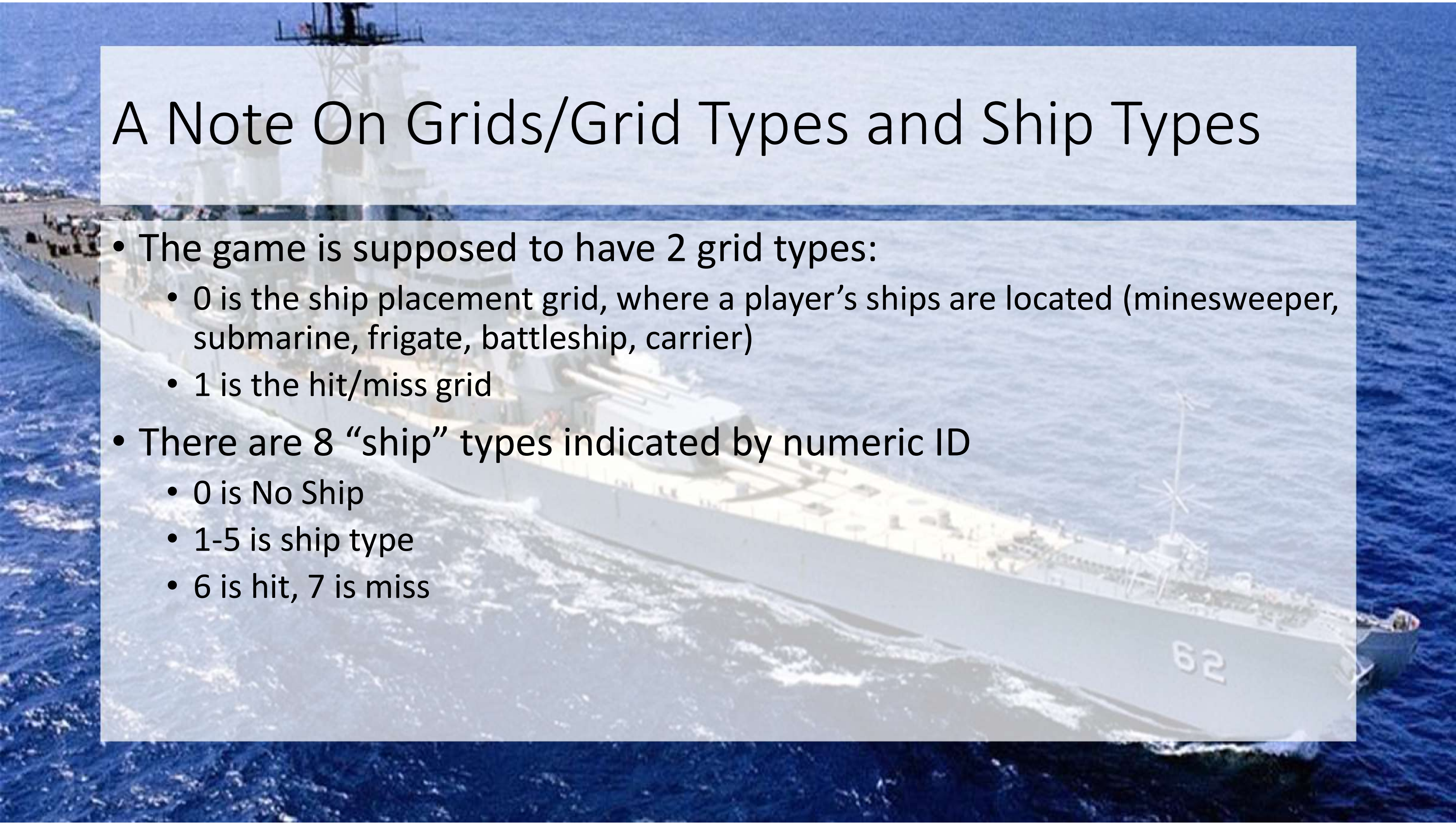
- Main classes focused on:
  - CPlayer- Workhorse of the project. All about the players and their grids.
  - CShipInfo- Instantiates a full ship with all info including location & pieces left.
  - CSinkTheFleet- Class containing the game flow. Buddies with CPlayer.



```
P1| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|
A | H| H| H| H|  |  |  |  |  |  |  |  |
B | H|  |  |  |  |  |  |  |  |  |  |  |
C |  |  |  |  |  |  |  |  |  |  |  |  |
D |  |  |  |  |  |  |  |  |  |  |  |  |
E |  |  |  |  |  |  |  |  |  | ▒|  |  |
F |  |  |  |  |  |  |  |  |  |  |  |  |
G |  |  |  |  |  |  |  |  |  |  | ▒|  |
H | ▒|  |  |  |  |  |  |  |  |  |  |  |
HIT!
You sank the Mine Sweeper!
Press <Enter>_
```

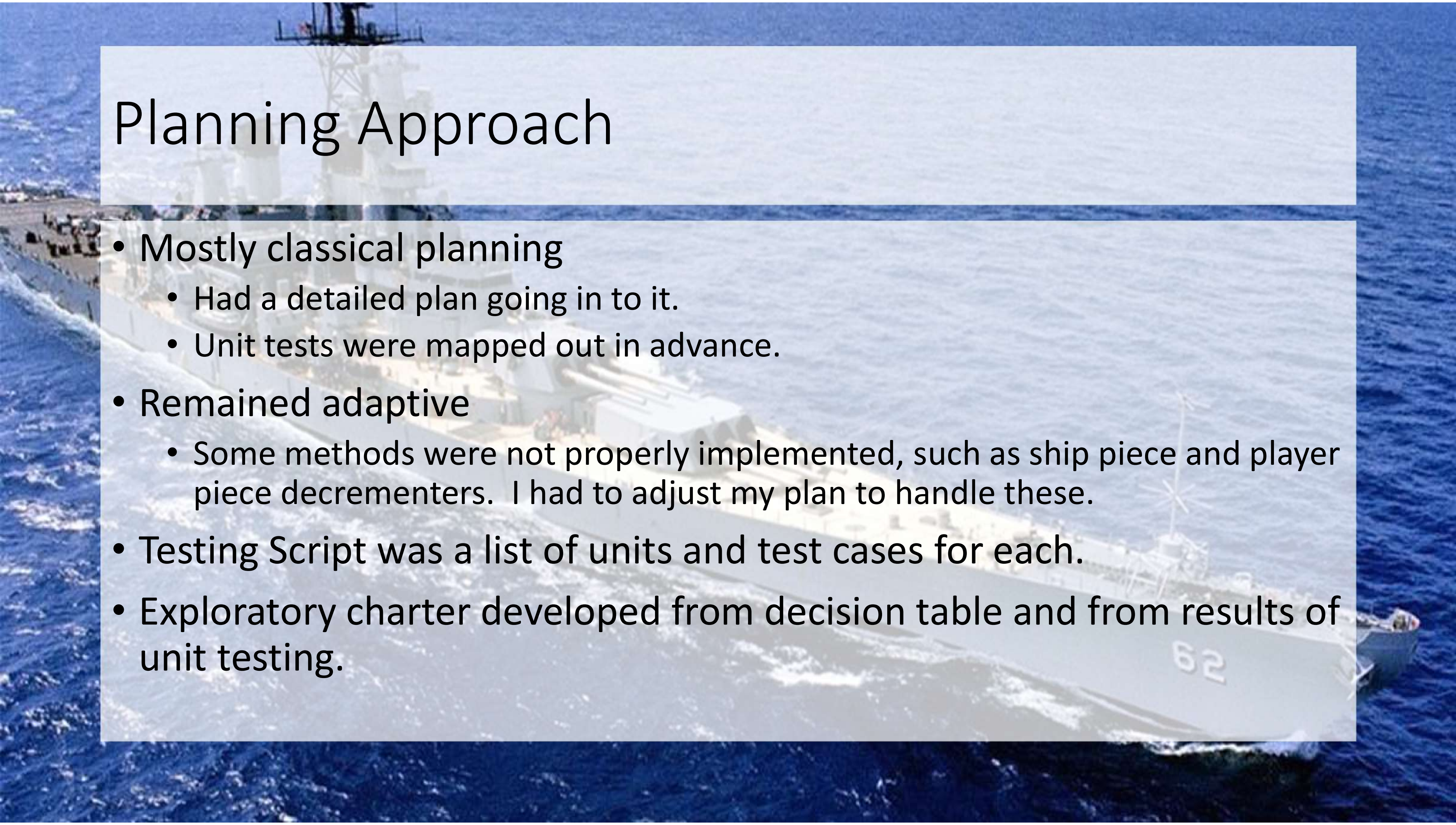


```
HIT!
You sank the Aircraft Carrier!
Press <Enter>

              Congratulations player 1!
```



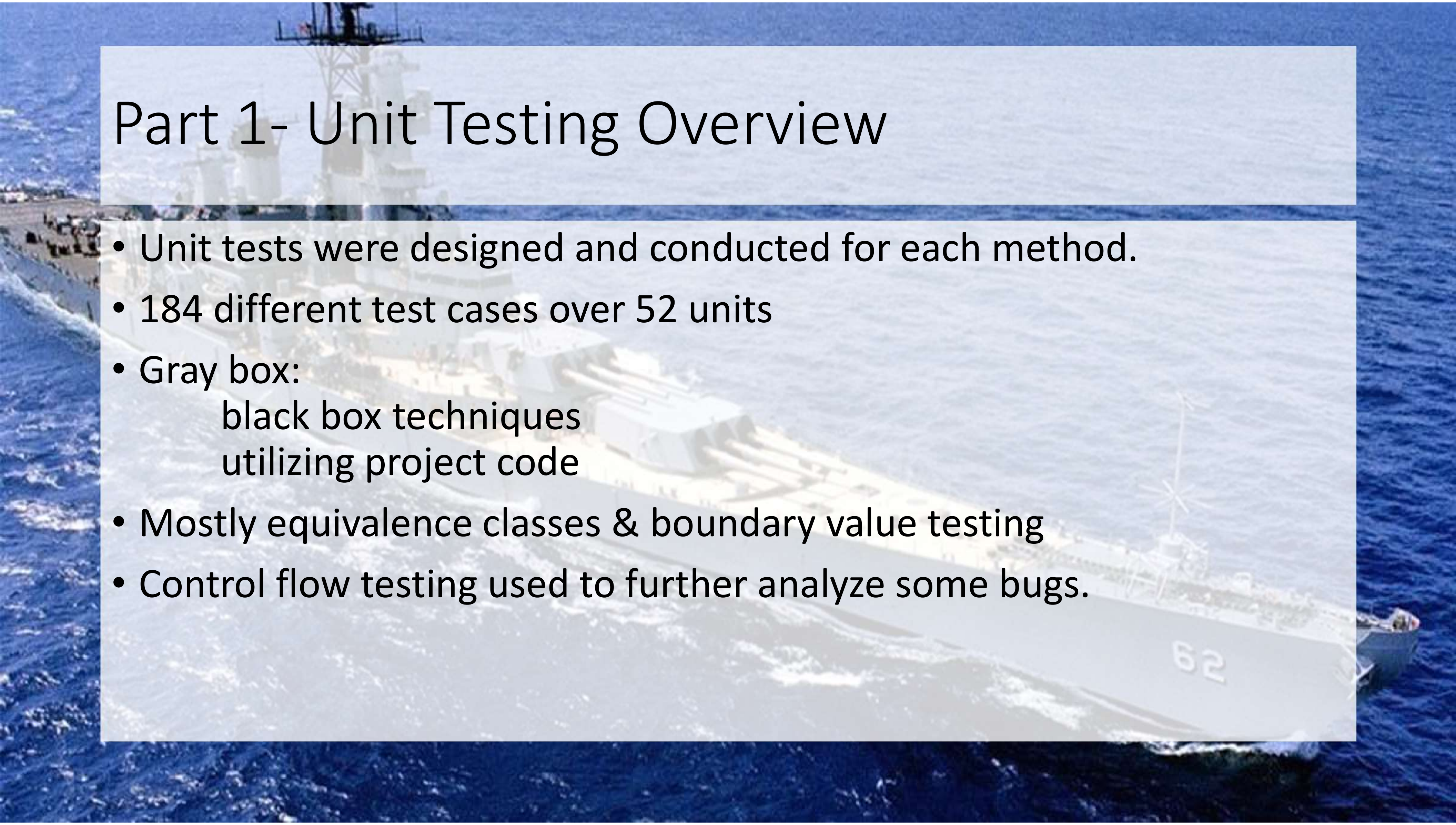
A Note On Grids/Grid Types and Ship Types

- The game is supposed to have 2 grid types:
  - 0 is the ship placement grid, where a player’s ships are located (minesweeper, submarine, frigate, battleship, carrier)
  - 1 is the hit/miss grid
- There are 8 “ship” types indicated by numeric ID
  - 0 is No Ship
  - 1-5 is ship type
  - 6 is hit, 7 is miss



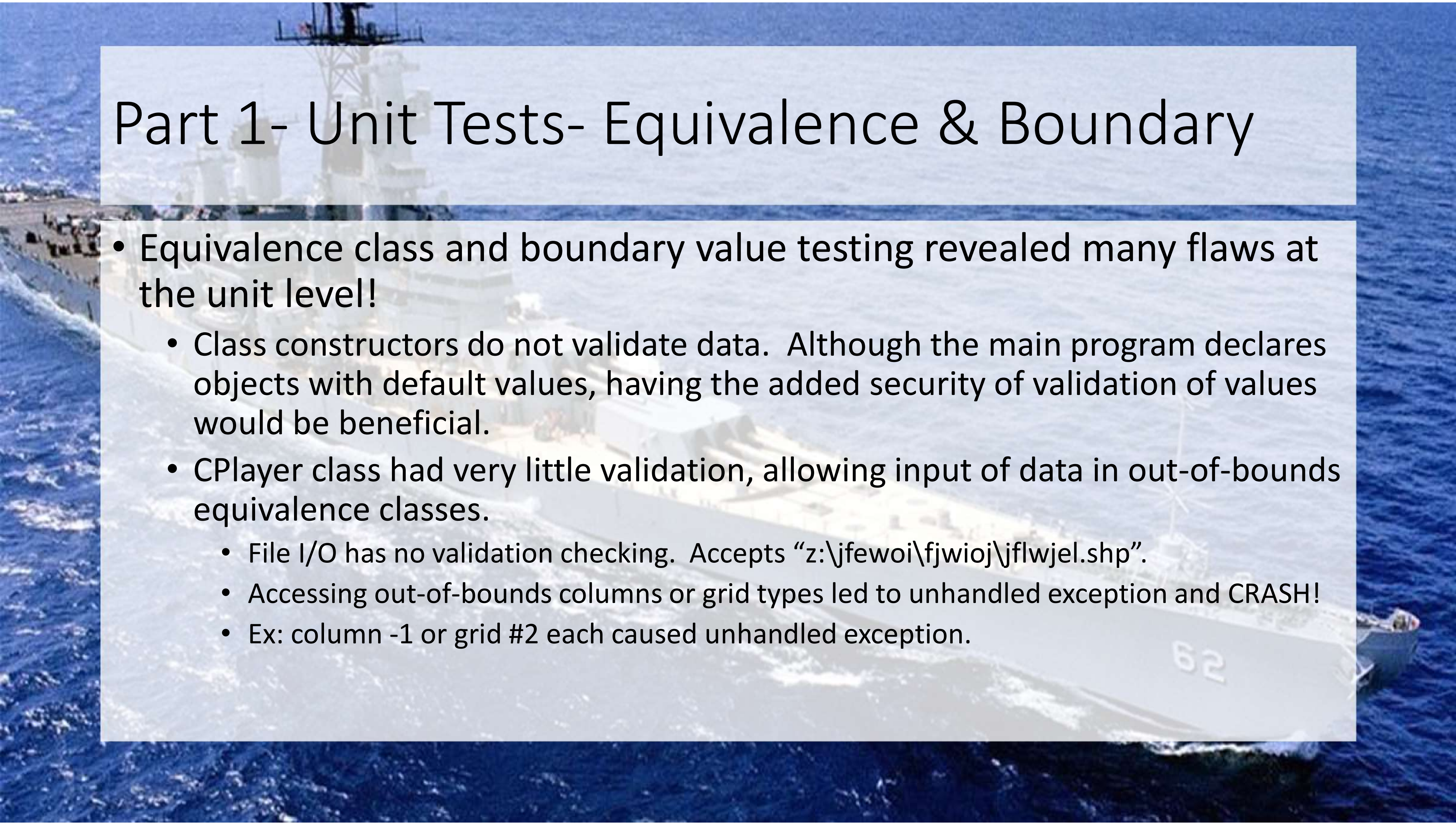
Planning Approach

- Mostly classical planning
  - Had a detailed plan going in to it.
  - Unit tests were mapped out in advance.
- Remained adaptive
  - Some methods were not properly implemented, such as ship piece and player piece decrementers. I had to adjust my plan to handle these.
- Testing Script was a list of units and test cases for each.
- Exploratory charter developed from decision table and from results of unit testing.



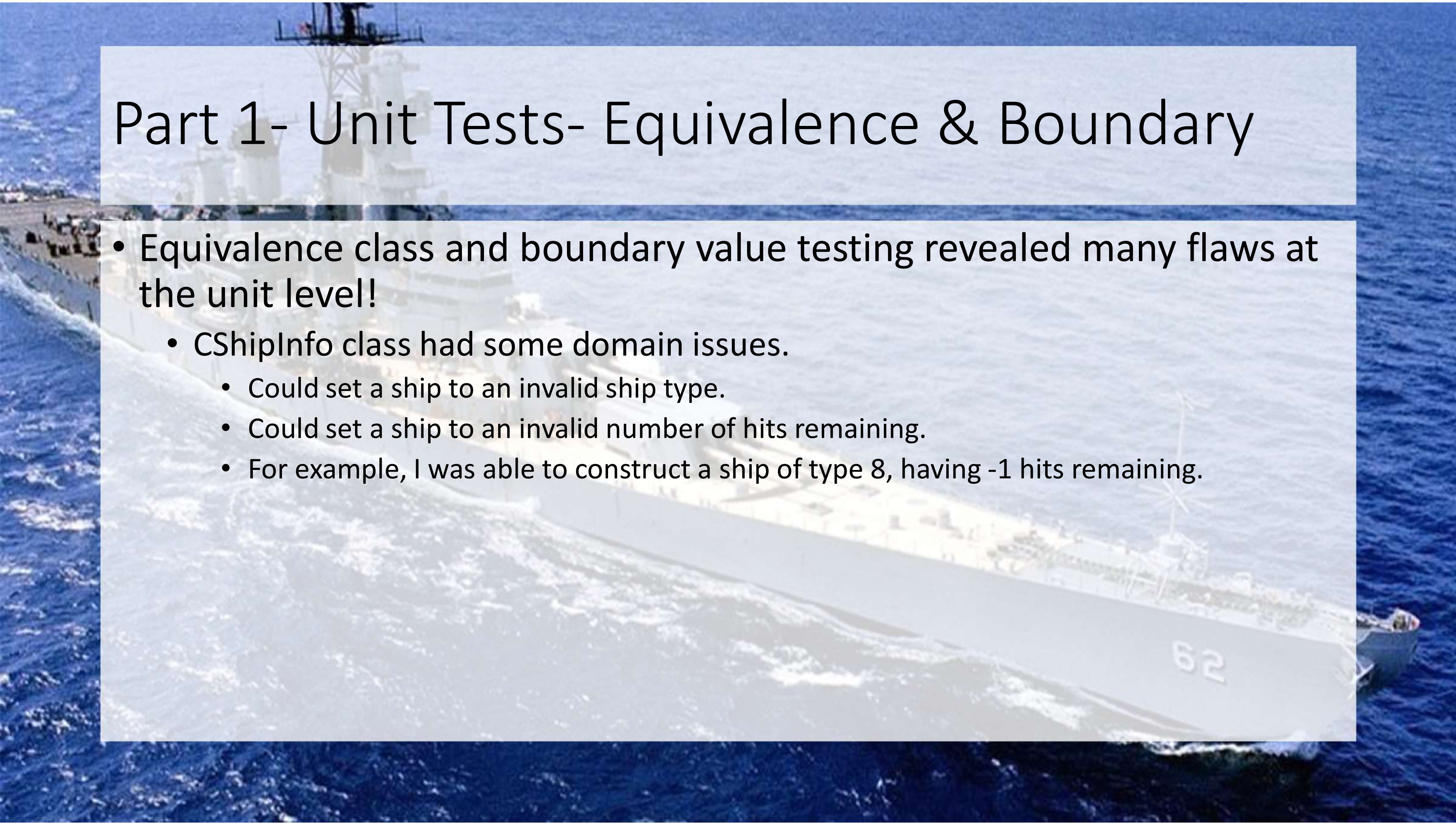
Part 1- Unit Testing Overview

- Unit tests were designed and conducted for each method.
- 184 different test cases over 52 units
- Gray box:
  - black box techniques
  - utilizing project code
- Mostly equivalence classes & boundary value testing
- Control flow testing used to further analyze some bugs.



Part 1- Unit Tests- Equivalence & Boundary

- Equivalence class and boundary value testing revealed many flaws at the unit level!
  - Class constructors do not validate data. Although the main program declares objects with default values, having the added security of validation of values would be beneficial.
  - CPlayer class had very little validation, allowing input of data in out-of-bounds equivalence classes.
    - File I/O has no validation checking. Accepts “z:\jfewoi\fjwioj\jflwjel.shp”.
    - Accessing out-of-bounds columns or grid types led to unhandled exception and CRASH!
    - Ex: column -1 or grid #2 each caused unhandled exception.



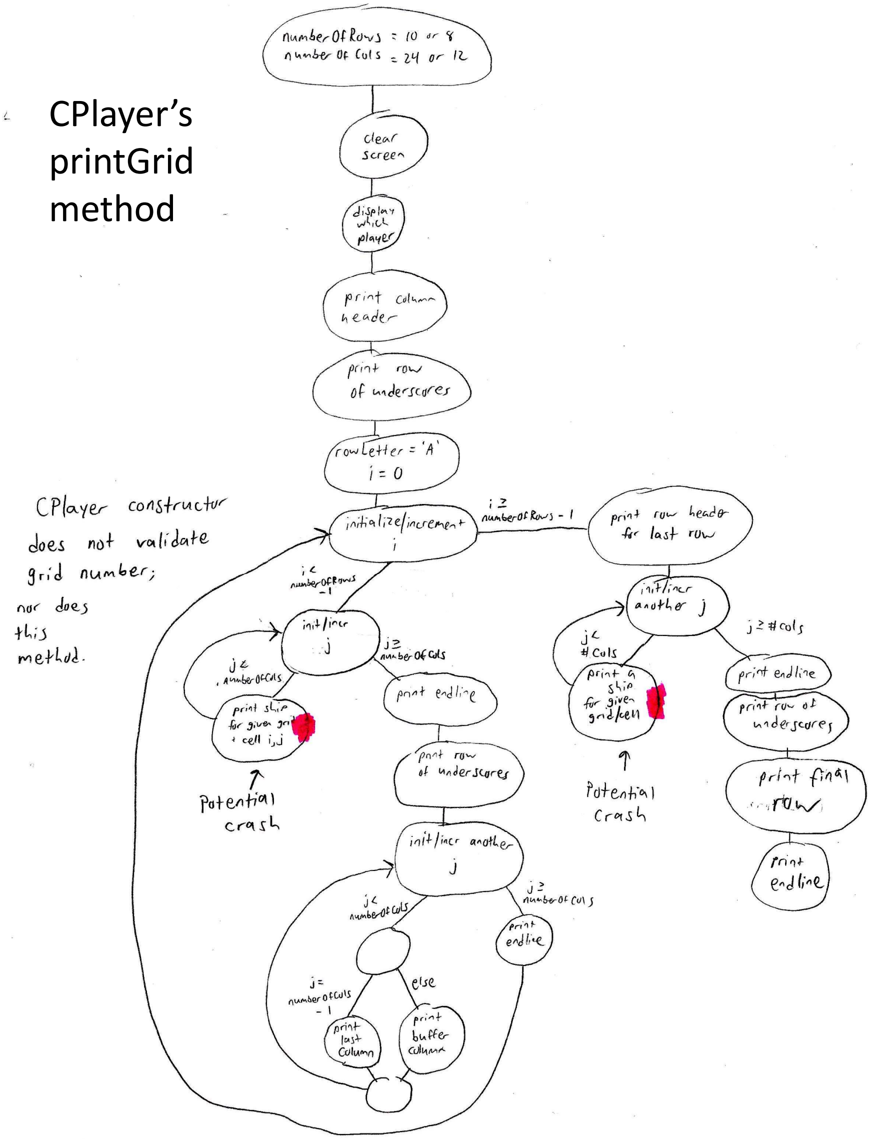
Part 1- Unit Tests- Equivalence & Boundary

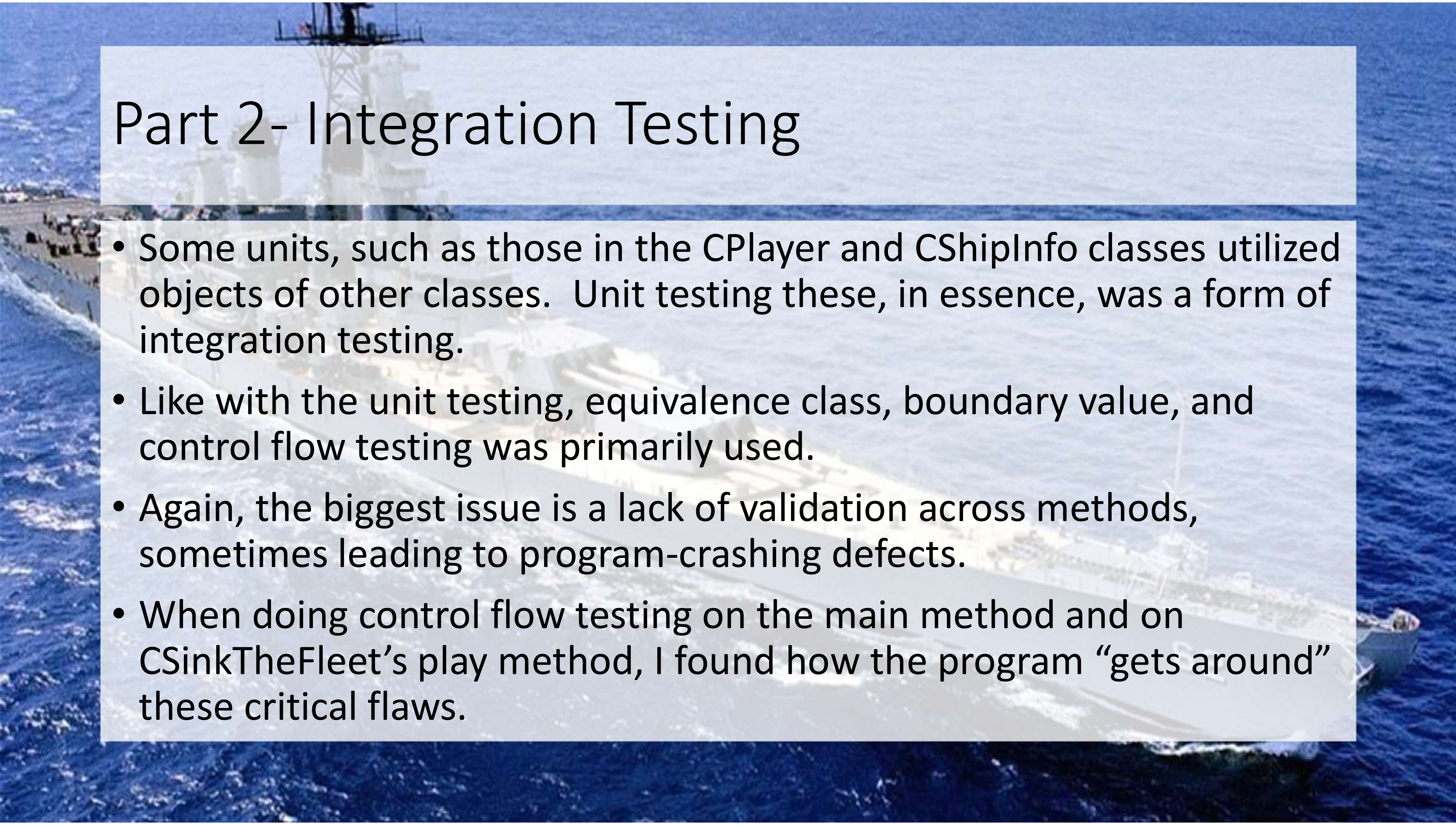
- Equivalence class and boundary value testing revealed many flaws at the unit level!
  - CShipInfo class had some domain issues.
    - Could set a ship to an invalid ship type.
    - Could set a ship to an invalid number of hits remaining.
    - For example, I was able to construct a ship of type 8, having -1 hits remaining.

Part 1- Unit Tests- Control Flow

- Control Flow testing was able to pinpoint a potential program-killing defect in CPlayer’s printGrid method
  - The CPlayer constructor does not validate grid type.
  - CPlayer’s printGrid method also does not validate grid type.
  - Accessing bad grid type (e.g., grid 2 or grid -1) = no memory allocated = OOPS!!!

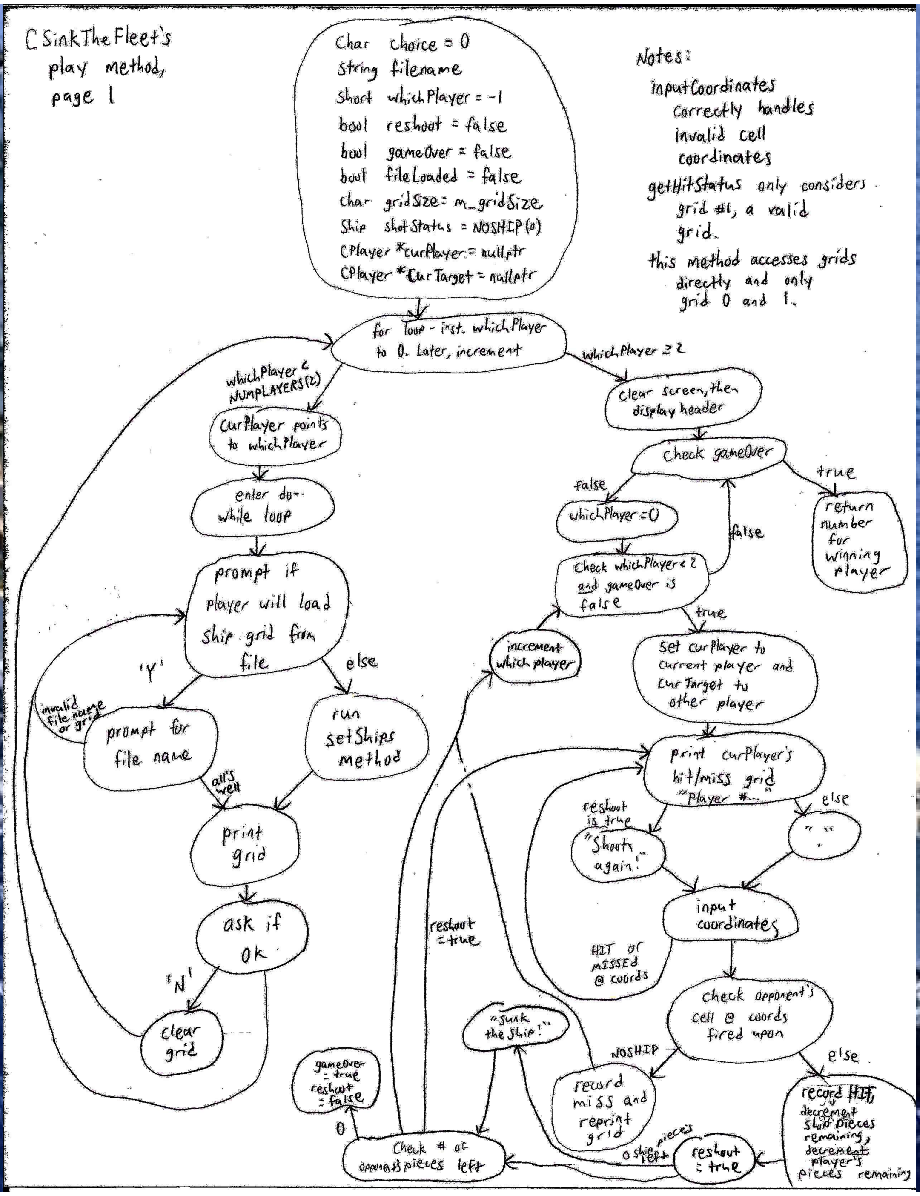
CPlayer’s printGrid method





Part 2- Integration Testing

- Some units, such as those in the CPlayer and CShipInfo classes utilized objects of other classes. Unit testing these, in essence, was a form of integration testing.
- Like with the unit testing, equivalence class, boundary value, and control flow testing was primarily used.
- Again, the biggest issue is a lack of validation across methods, sometimes leading to program-crashing defects.
- When doing control flow testing on the main method and on CSinkTheFleet’s play method, I found how the program “gets around” these critical flaws.



CSinkTheFleet's play method, page 1

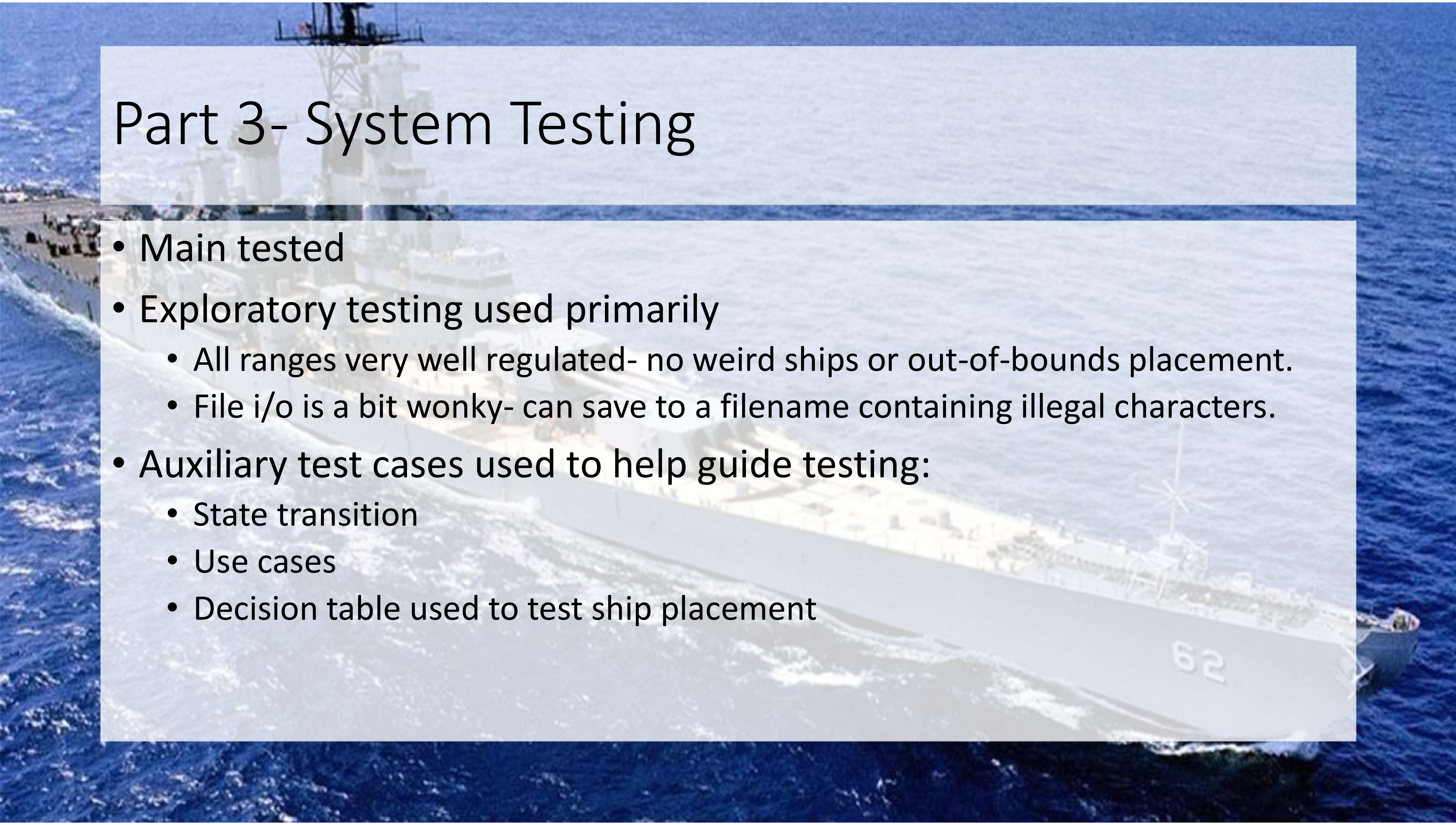
Notes:

inputCoordinates correctly handles invalid cell coordinates

getHitStatus only considers grid #1, a valid grid.

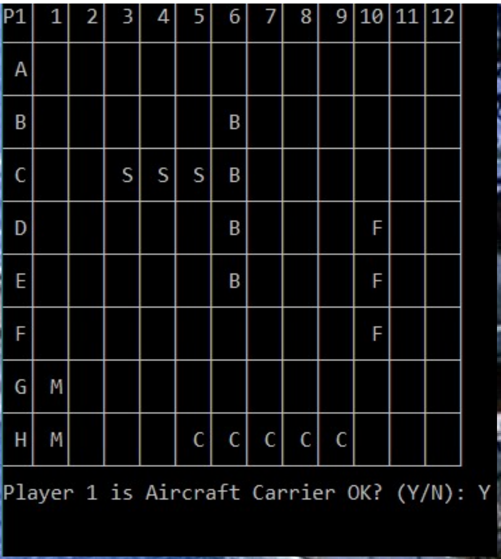
this method accesses grids directly and only grid 0 and 1.

Char choice = 0
String filename
Short whichPlayer = -1
bool reshoot = false
bool gameOver = false
bool fileLoaded = false
Char gridSize = m_gridSize
Ship shotStatus = NOSHIP(0)
CPlayer *curPlayer = nullptr
CPlayer *CurTarget = nullptr
for loop - inst. whichPlayer to 0. Later, increment
whichPlayer < NUMPLAYERS(2)
curPlayer points to whichPlayer
enter do-while loop
prompt if player will load ship grid from file
'Y'
else
invalid file name or grid
prompt for file name
all's well
run setShips method
print grid
ask if ok
'N'
clear grid
whichPlayer ≥ 2
Clear screen, then display header
Check gameOver
true
return number for winning player
false
whichPlayer = 0
false
Check whichPlayer < 2 and gameOver is false
true
increment which player
Set curPlayer to current player and curTarget to other player
print curPlayer's hit/miss grid "Player #..."
reshoot is true
"Shoots again!"
else
" "
input coordinates
HIT or MISSED @ coords
check opponent's cell @ coords fired upon
NOSHIP
record miss and reprint grid
else
record HIT, decrement ship pieces remaining, decrement player's pieces remaining
reshoot = true
0 ship pieces left
"Sunk the Ship!"
Check # of opponent's pieces left
0
gameOver = true reshoot = false
reshoot = true

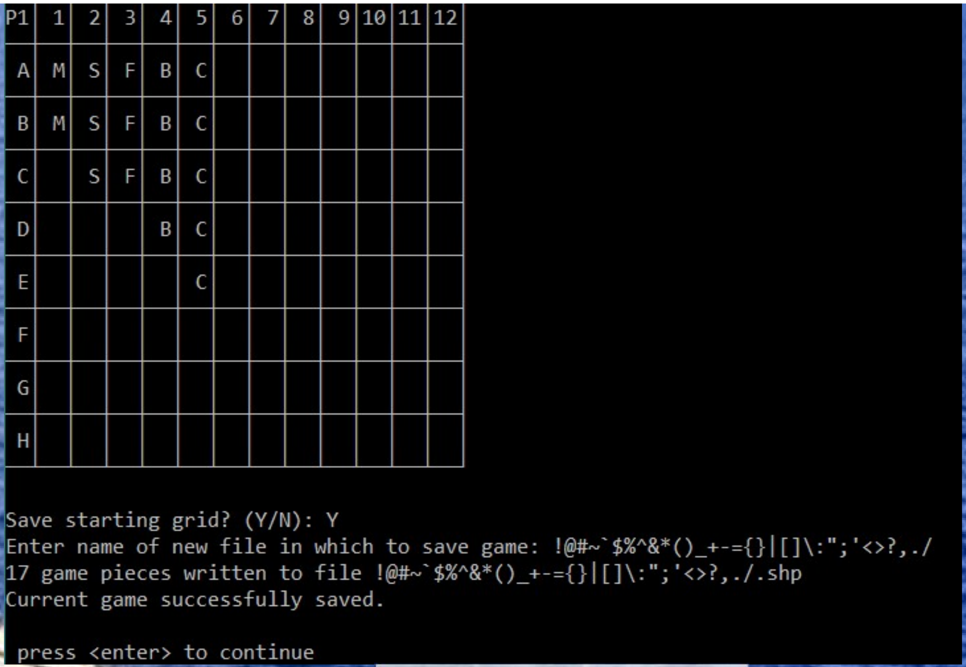


Part 3- System Testing

- Main tested
- Exploratory testing used primarily
  - All ranges very well regulated- no weird ships or out-of-bounds placement.
  - File i/o is a bit wonky- can save to a filename containing illegal characters.
- Auxiliary test cases used to help guide testing:
  - State transition
  - Use cases
  - Decision table used to test ship placement



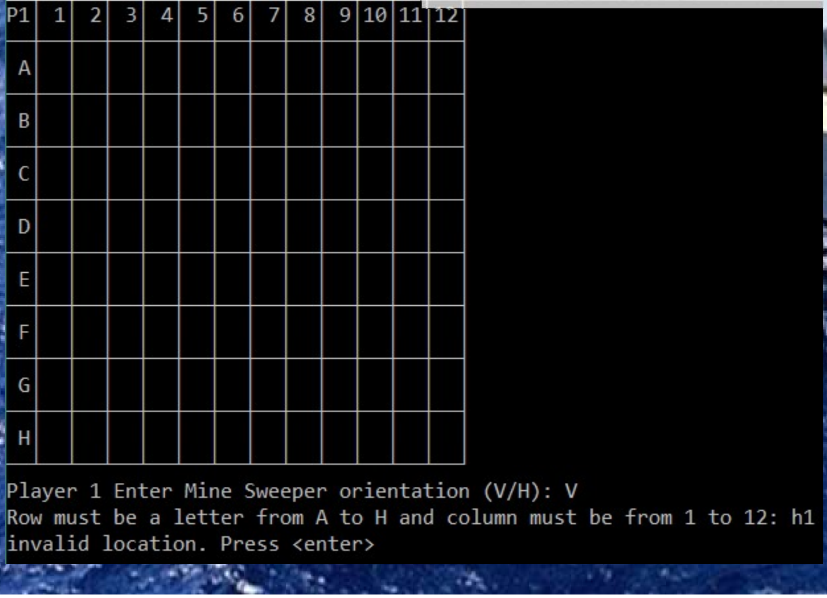
```
P1  1  2  3  4  5  6  7  8  9 10 11 12
 A
 B                 B
 C        S  S  S  B
 D                 B           F
 E                 B           F
 F                             F
 G  M
 H  M           C  C  C  C  C
Player 1 is Aircraft Carrier OK? (Y/N): Y
```



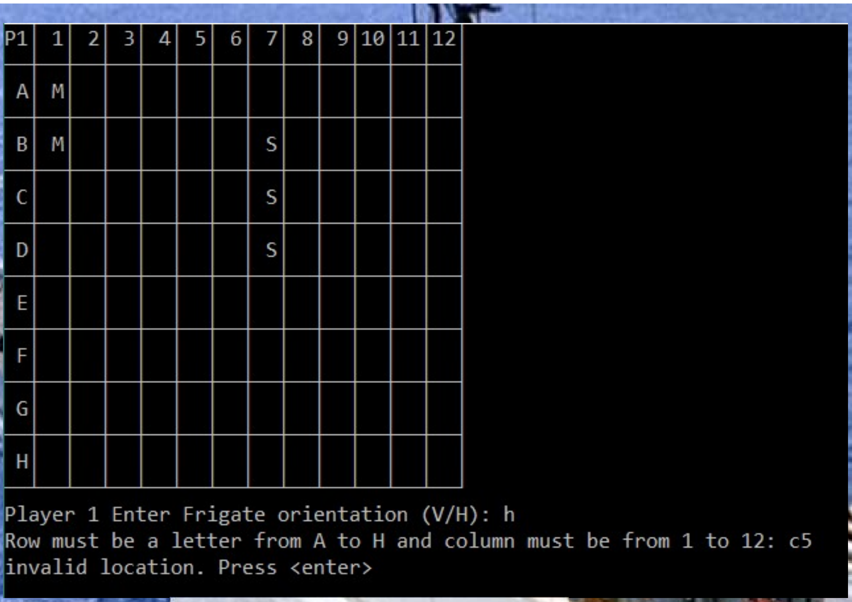
```
P1  1  2  3  4  5  6  7  8  9 10 11 12
 A  M  S  F  B  C
 B  M  S  F  B  C
 C     S  F  B  C
 D           B  C
 E              C
 F
 G
 H

Save starting grid? (Y/N): Y
Enter name of new file in which to save game: !@#~`$%^&*()_+-={}|[]\:";'<>?,./
17 game pieces written to file !@#~`$%^&*()_+-={}|[]\:";'<>?,./.shp
Current game successfully saved.

 press <enter> to continue
```

Out-Of-Bounds Caught

I/O Error not caught!

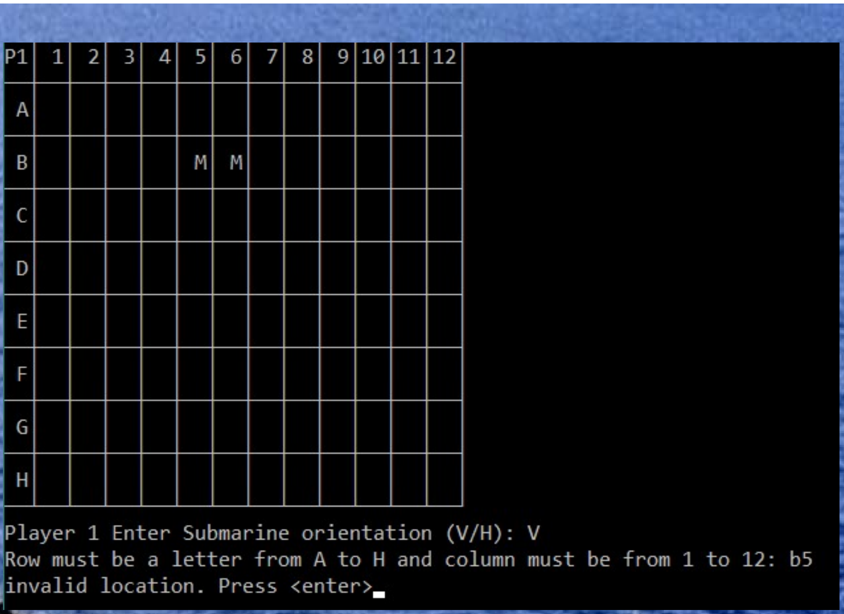


```
P1  1  2  3  4  5  6  7  8  9 10 11 12
 A
 B
 C
 D
 E
 F
 G
 H
Player 1 Enter Mine Sweeper orientation (V/H): V
Row must be a letter from A to H and column must be from 1 to 12: h1
invalid location. Press <enter>
```



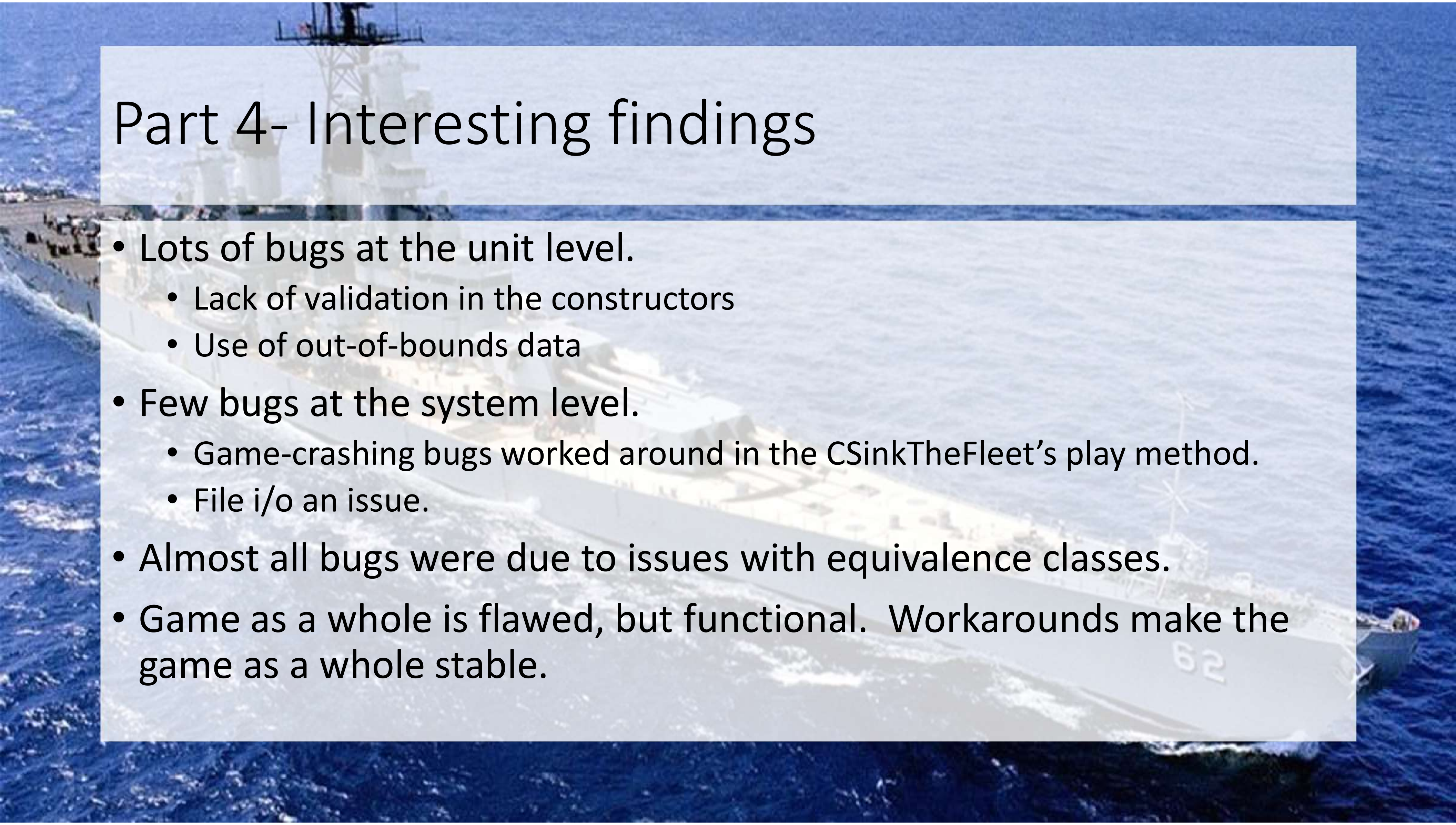
```
P1| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|
 A| M|  |  |  |  |  |  |  |  |  |  |  |
 B| M|  |  |  |  |  | S|  |  |  |  |  |
 C|  |  |  |  |  |  | S|  |  |  |  |  |
 D|  |  |  |  |  |  | S|  |  |  |  |  |
 E|  |  |  |  |  |  |  |  |  |  |  |  |
 F|  |  |  |  |  |  |  |  |  |  |  |  |
 G|  |  |  |  |  |  |  |  |  |  |  |  |
 H|  |  |  |  |  |  |  |  |  |  |  |  |

Player 1 Enter Frigate orientation (V/H): h
Row must be a letter from A to H and column must be from 1 to 12: c5
invalid location. Press <enter>
```

Ship overlap detected.



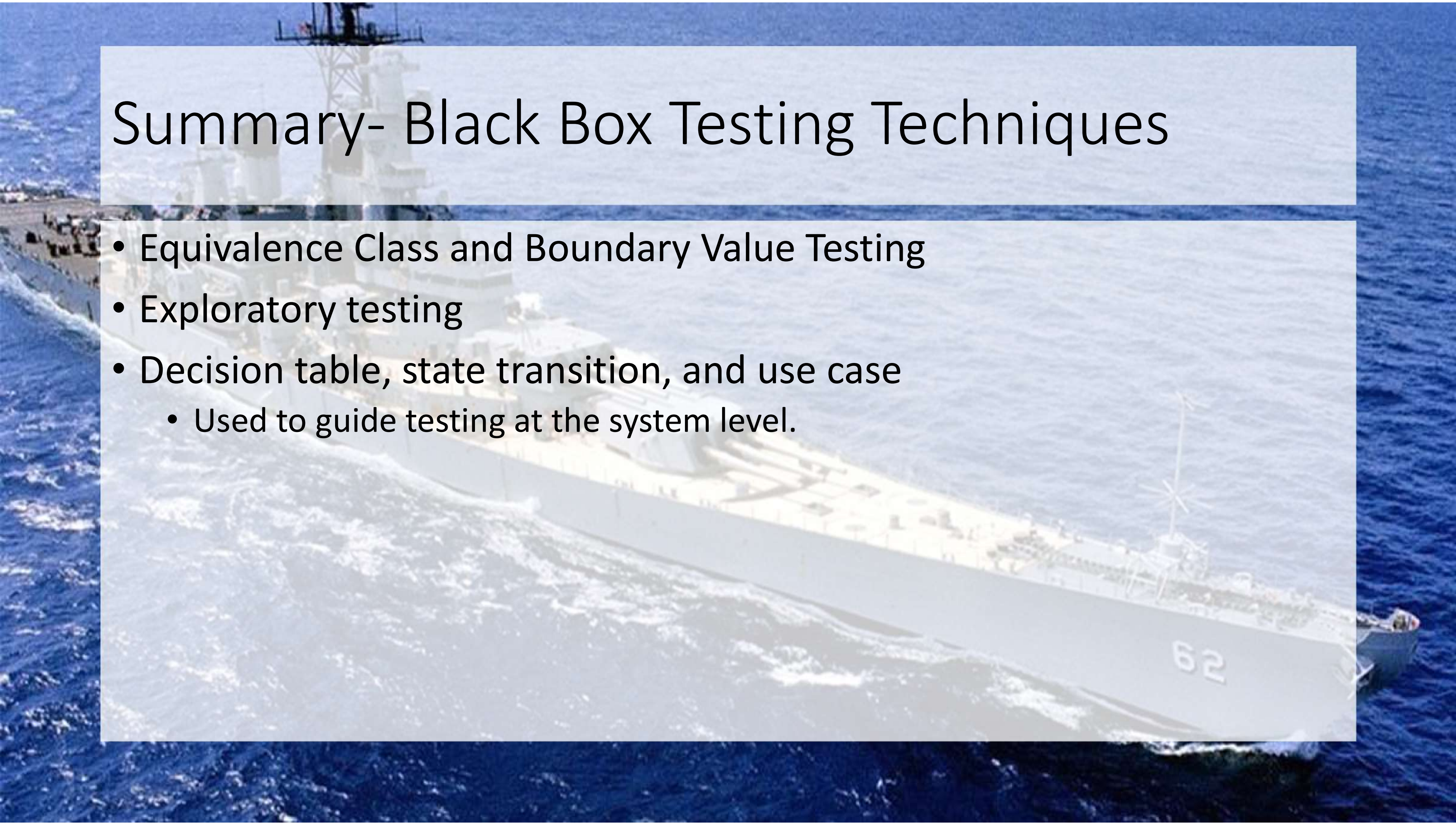
```
P1| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|
 A|  |  |  |  |  |  |  |  |  |  |  |  |
 B|  |  |  |  | M| M|  |  |  |  |  |  |
 C|  |  |  |  |  |  |  |  |  |  |  |  |
 D|  |  |  |  |  |  |  |  |  |  |  |  |
 E|  |  |  |  |  |  |  |  |  |  |  |  |
 F|  |  |  |  |  |  |  |  |  |  |  |  |
 G|  |  |  |  |  |  |  |  |  |  |  |  |
 H|  |  |  |  |  |  |  |  |  |  |  |  |

Player 1 Enter Submarine orientation (V/H): V
Row must be a letter from A to H and column must be from 1 to 12: b5
invalid location. Press <enter>
```

Bow collision detected.



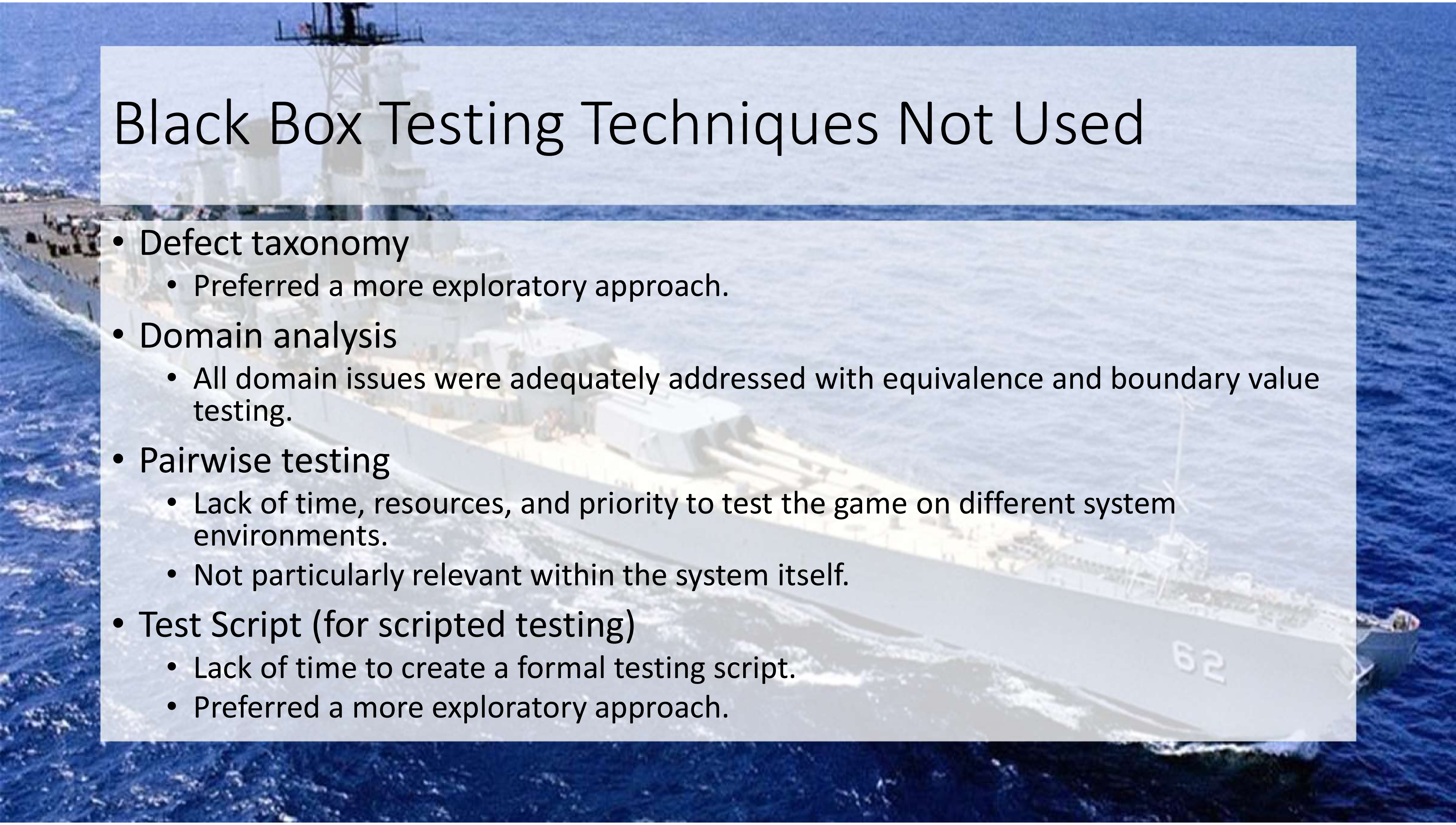
Part 4- Interesting findings

- Lots of bugs at the unit level.
  - Lack of validation in the constructors
  - Use of out-of-bounds data
- Few bugs at the system level.
  - Game-crashing bugs worked around in the CSinkTheFleet’s play method.
  - File i/o an issue.
- Almost all bugs were due to issues with equivalence classes.
- Game as a whole is flawed, but functional. Workarounds make the game as a whole stable.



Summary- Black Box Testing Techniques

- Equivalence Class and Boundary Value Testing
- Exploratory testing
- Decision table, state transition, and use case
  - Used to guide testing at the system level.

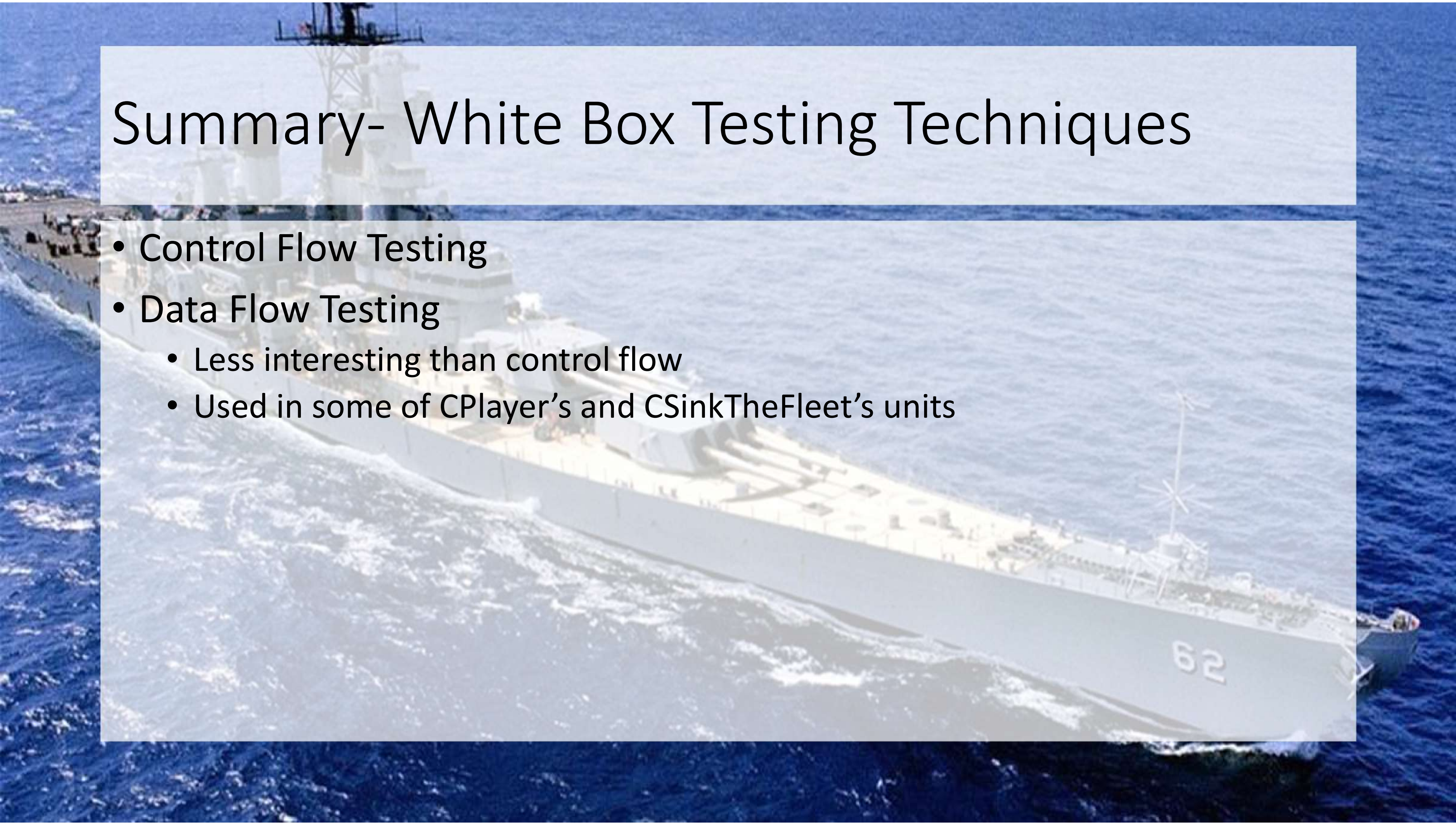


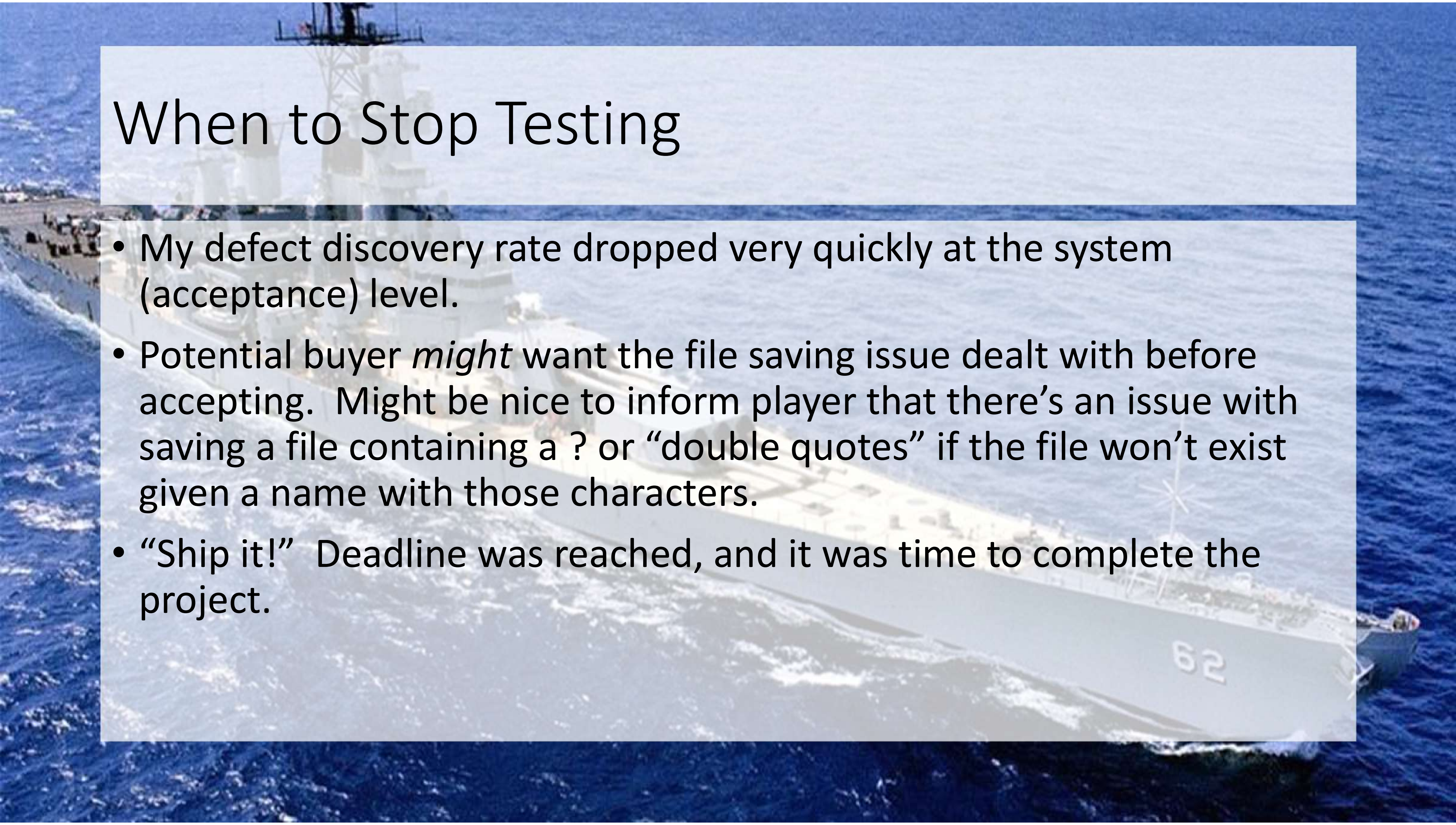
Black Box Testing Techniques Not Used

- Defect taxonomy
  - Preferred a more exploratory approach.
- Domain analysis
  - All domain issues were adequately addressed with equivalence and boundary value testing.
- Pairwise testing
  - Lack of time, resources, and priority to test the game on different system environments.
  - Not particularly relevant within the system itself.
- Test Script (for scripted testing)
  - Lack of time to create a formal testing script.
  - Preferred a more exploratory approach.

Summary- White Box Testing Techniques

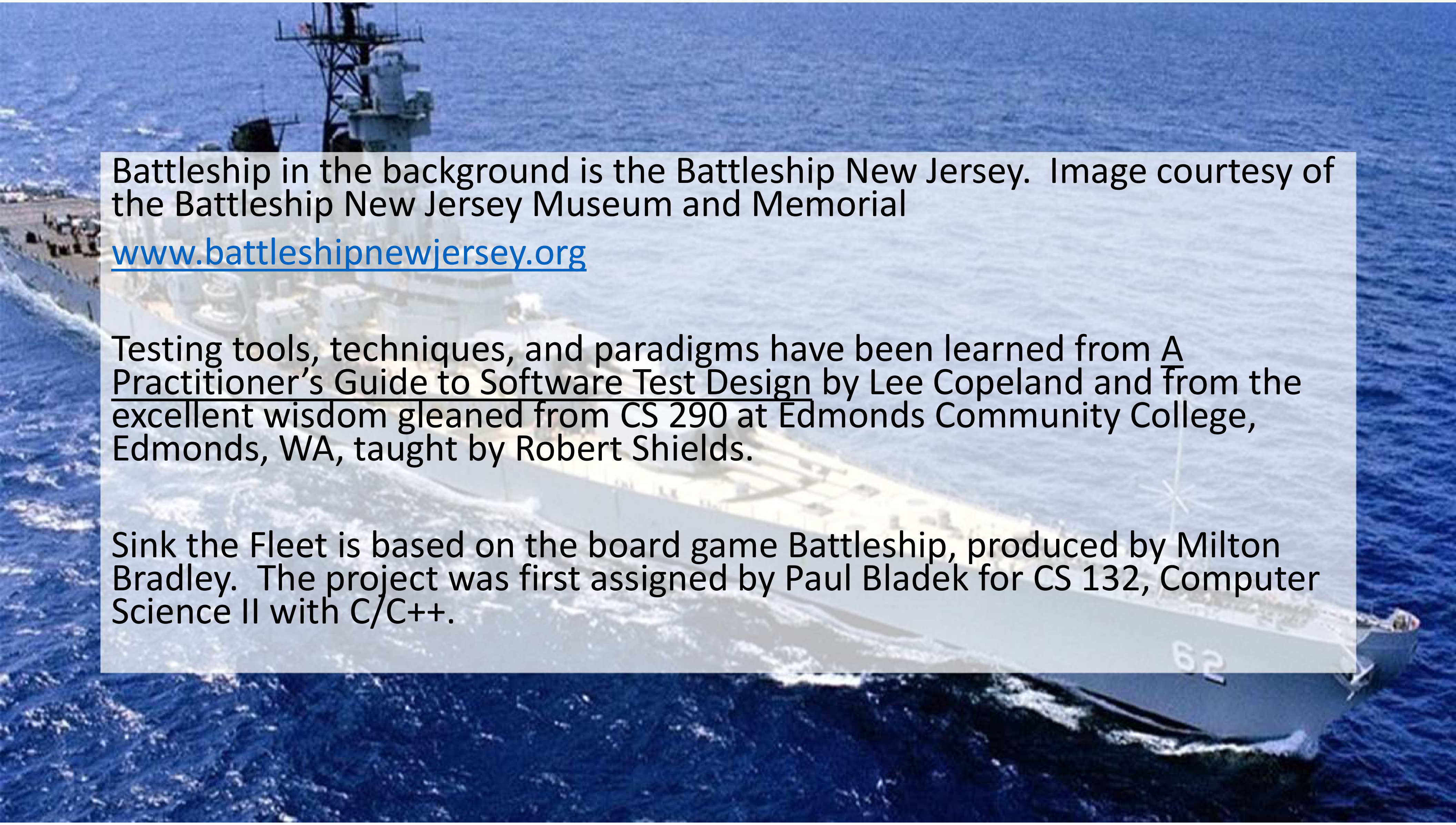
- Control Flow Testing
- Data Flow Testing
  - Less interesting than control flow
  - Used in some of CPlayer’s and CSinkTheFleet’s units





When to Stop Testing

- My defect discovery rate dropped very quickly at the system (acceptance) level.
- Potential buyer *might* want the file saving issue dealt with before accepting. Might be nice to inform player that there’s an issue with saving a file containing a ? or “double quotes” if the file won’t exist given a name with those characters.
- “Ship it!” Deadline was reached, and it was time to complete the project.



Battleship in the background is the Battleship New Jersey. Image courtesy of the Battleship New Jersey Museum and Memorial

www.battleshipnewjersey.org

Testing tools, techniques, and paradigms have been learned from A Practitioner’s Guide to Software Test Design by Lee Copeland and from the excellent wisdom gleaned from CS 290 at Edmonds Community College, Edmonds, WA, taught by Robert Shields.

Sink the Fleet is based on the board game Battleship, produced by Milton Bradley. The project was first assigned by Paul Bladek for CS 132, Computer Science II with C/C++.