COP 3503 Term Project: Report

Team members: Eion Blanchard, John-Michael Dieckmann, Zachary Goins, Callum Jago, Annie Luc, Armin Lukic

For the COP 3503 Term Project, our group created a Battleship simulation, which imitated the classic Battleship game. This project asks a user to set ships based on length and desired left-most position, and then proceeds to play the game turn-by-turn until game-over, when either the computer or the user wins. This idea was developed by the team because we wanted to create a game, but also practice fundamental programming concepts thoroughly. Battleship was the perfect way to implement everything that was taught and learned in the class throughout the entire semester: pointers, memory allocation, object-oriented programming, abstraction, and others that were not covered.

The UML of the major classes are shown below:

|  |
| --- |
| Board |
| - typedef struct tile  - typedef struct full\_board  - int possibleStatus[]  + struct tile values[101] |
| + Board()  + void populate()  + void print\_user\_board()  + void print\_computer\_board()  + string getId(int index)  + int setStatus(int index, int status) |

|  |
| --- |
| Ship |
| - int length  - int startX  - int startY  - bool isSunk  - bool isHorizontal  + int index |
| + Ship(int length, string name, bool isHorizontal, char startX, int startY)  + bool sinkShip()  + bool getIsSunk(Board \*board)  + int getLength()  + int getStart()  + char getStartX()  + int getStartY()  + bool getIsHorizontal()  + void Initialize(Board \*\*b, int index) |

|  |
| --- |
| Computer |
| + Board \*board  + Ship \*carrier  + Ship \*battleship  + Ship \*submarine  + Ship \*cruiser  + Ship \*destroyer  - string desPos  - int choice |
| + Computer()  + void setCarrier(bool \*addSuccess)  + void setBattleship(bool \*addSuccess)  + void setSubmarine(bool \*addSuccess)  + void setCruiser(bool \*addSuccess)  + void setDestroyer(bool \*addSuccess)  + int guess()  + int formulate(int length)  + int isValidPos(string desPos, int length, int \*index\_return)  + Board getBoard() |

|  |
| --- |
| User |
| + Board \*board  + Ship \*carrier  + Ship \*battleship  + Ship \*submarine  + Ship \*cruiser  + Ship \*destroyer  - string desPos  - int choice |
| + User()  + void setCarrier(bool \*addSuccess)  + void setBattleship(bool \*addSuccess)  + void setSubmarine(bool \*addSuccess)  + void setCruiser(bool \*addSuccess)  + void setDestroyer(bool \*addSuccess)  + int guess()  + int isValidPos(string desPos, int length, int \*index\_return)  + Board getBoard() |

|  |
| --- |
| Turn |
| friend class Player  friend class Ship  - bool computer\_carrier  - bool computer\_battleship  - bool computer\_submarine  - bool computer\_cruiser  - bool computer\_destroyer;  - bool user\_carrier  - bool user\_battleship  - bool user\_submarine  - bool user\_cruiser  - bool user\_destroyer |
| + Turn()  + bool gameover(Computer \*computer, User \*user)  + bool lose(User \*user)  + bool win(Computer \*computer)  + void user\_hit\_ship(User \*user, Computer \*computer, Board \*board\_user, Board \*board\_computer)  + void computer\_hit\_ship(User \*user, Computer \*computer, Board \*board\_user, Board \*board\_computer)  + void turn(User \*user, Computer \*computer, Board \*board\_user, Board \*board\_computer)  + void frame(User \*user, Computer \*computer, Board \*&board\_user, Board \*&board\_computer); |

The User and Computer classes both implement the abstract class Player, and the main method is written in a separate file named ‘battleship\_sim.cpp’. Each User and Computer has five Ships, each representing a Battleship ship, and also they have a board that represents their respective ships, hits, and misses. A makefile has been created to compile all of the source code. Moreover, to increase the graphical quality of the game, when the game is played, several screens pop up from the terminal to show and update the board of the user and the computer.

In working on the project, several lessons were learned:

- Plan the design of the project well in advance, with everything mapped out with fine detail, so that there is time to fix any problems with implementation as the deadline approaches.

- Make sure it is clear the goals and duties of each person in the group, along with making sure it is balanced and fair to all.

- Hold people accountable for their work, and also be sure to have many backup plans in case of imminent failure.

- Know who is good at doing what, and delegate responsibilities accordingly.

- Try to have an earlier push for generating the first bits of code--our momentum came late, but strong.

Additionally, though we were not able to implement these in the current Battleship game simulator, the group had several goals and future plans for the program that would increase its complexity and also success:

- It would be great to have a graphical interface that is separate from the terminal, which could make the software design a little more personable and user-friendly.

- It would be awesome to implement a more advanced AI component on the part of the computer that not only guesses randomly about the placement of ships, but remembers previous games’ positions and tries its best to win using human-playing strategy.

- Create different levels of difficulty on the AI component, allowing a more user-friendly game for a more variety of ages.

- Consider adding options for User v. User or Computer v. Computer (+ difficulty levels).

- Implement more sophisticated AI that seeks the most efficient guess (i.e. bombing spaces near previous hits to sink ships more quickly) rather than random locations.

- Add in a variant pause for AI’s “thinking” to make more realistic experience for user.