Project 2 Honors Section
Ethan Osmundson and Jonathan Leibovich
osmun046 and leibo023

To create board:

```
public BattleBoatsBoard(String gamemode){
      this.gamemode = gamemode;
      if (gamemode.equals("standard")){
            board = new Boats[8][8];
      }
```

Simply creating the board is a O(1) time complexity because it is a constant amount of actions (always one in this case), regardless of the gamemode.

At the same time, placing the boats is also a O(1) time complexity because it relies on the length of boat sizes which is either 5 or 10 depending on the game mode. In the nested loops, it also relies on the constant size of the boat array or the boat sizes themselves (which is also constant). The user cannot choose the number of boats they want on the board, they can only choose between 5 and 10. If you were to consider the number of boats and their respective sizes to be of size n, it would clearly be O(n^2) because of the loop running through the number of boats needed and the nested loop running through each boat's length. Similarly, if the boat size is considered constant then the time complexity would be O(n). We were unsure of what you wanted here, and the clarification confused us more. See code excerpt below:

Only the first possibility (Standard gamemode and horizontal left placement shown)

```
if (this.gamemode.equals("standard")){
      Boats boat5a = new Boats(5, "a");
      Boats boat4a = new Boats(4, "a");
      Boats boat3a = new Boats(3, "a");
      Boats boat3b = new Boats(3, "b");
      Boats boat2a = new Boats(2, "a");

      Boats[] boatSizes = {boat5a, boat4a, boat3a, boat3b, boat2a}; //Constant
size for both standard and expert, either 5 or 10

      for (int i =0; i < boatSizes.length; i++){  //Relies on the constant
.length attribute of the boatSizes array
            boolean fits = false;

            while (fits == false){
                  int x = (int)Math.floor(Math.random() * 8.0);
                  int y = (int)Math.floor(Math.random() * 8.0);

                  boolean horizontal = true;
                  if (Math.random() < 0.5){
                        horizontal = false;
```

```
                  }

                  boolean left = true;
                  if (Math.random() < 0.5){
                        left = false;
                  }

                  boolean inTheWay = false;

                  if (horizontal && left){ //Horizontal and left
                  if (((x+1) - boatSizes[i].getLength()) >= 0){//Checks if
boat can fit on board

                        for (int k=0; k<boatSizes[i].getLength(); k++){
                  //relies on the constant sizes of the boat object length
                        if (board[y][x-k] instanceof Boats){ //check if
different boat is in the way                              inTheWay = true;
                              break;
                        }
                        }

                        if (!inTheWay){
                              fits = true;
                              for (int j=0; j<boatSizes[i].getLength(); j++){
//again relies on the constant sizes of the boat object length
                              board[y][x-j]=boatSizes[i];
                        }
                  }
            }
}
```

Taking a turn (firing a shot):

The action of firing a shot is contained in a while loop. The while loop continues while the input the user gives is invalid, this is then O(n). If the user beats the game in this turn, the fire code calls the helper method `printBoard()` which prints the original board using two nested for loops. If the board is of size n by n then `printBoard()` is of O(n^2) complexity because it loops through the rows and columns with nested loops. If we assume the size of the board to be constant (as it actually is) the method is of constant complexity. It either loops through 64 or 144 times depending on the board size chosen. The method always calls `updateCoordinate()` which is constant as it has no loops. Finally the method calls `fire()`.The fire method again has no loops so it is a constant complexity. The complexity overall if the board is considered to be a constant size is O(n) because the while loop at the beginning is the O(n) and the rest of the fire code is constant (O(1)). If we were to consider the length of the board array to be nxn, then

the complexity would be O(n^2) since it is not nested in the O(n) while loop but printBoard() contains a nested for loop referring to the size of the board.

Below are some code excerpts with comments indicating important parts:

```
if (action.equals("fire")){

	boolean inBounds = false;
	int row = -1;
	int col = -1;

	while (inBounds ==  false){//This takes n times based on the amount of
times it takes the user to input valid coordinates
		System.out.println("Enter a row to fire on... (between 0 and " +
(userBoard.getLength()-1)+")");
		row = s.nextInt();
		System.out.println("Enter a column to fire on... (between 0 and "
+ (userBoard.getLength()-1)+")");
		col = s.nextInt();
		if ((0 <= row) && (row < userBoard.getLength()) && (0 <= col) &&
(col < userBoard.getLength())){
			inBounds  = true;
		}
		else{
			System.out.println("Point out of bounds, try again.\n");
		}
	}

	int result = masterBoard.fire(row, col);
	if (result == 0){
		System.out.println("\nMISS!");
		userBoard.updateCoordinate(row, col, result); //Calls
updateCoordinate
	}
	else if (result == 1){
		System.out.println("\nHIT!");
		userBoard.updateCoordinate(row, col, 1);//Calls updateCoordinate

		totalHealth--;
		if (totalHealth < 1){
			System.out.println("Wow you are SO cool! You sunk all the
ships! You managed this amazing feat in "+ turns+ " turns.");
		System.out.println("Here is the revealed board:");
			printBoard(originalBoard);
			cont = false;
		}
	}
```

```
      else if ((result == -1) || (result ==  -2)){
            turns++;
            System.out.println("\nPenalty! This spot has already been hit, one
extra turn added.");
      }
}
```

printBoard():

```
public static void printBoard(String[][] originalBoard){
      String boardResult = "";
      for (int i = 0; i < originalBoard.length; i++){//Original Board is a
constant length (either 8 for standard or 12 for expert)
            boardResult += "\n";
            for (int j = 0; j < originalBoard.length; j++){
                  boardResult += originalBoard[i][j];
            }
      }
      System.out.println(boardResult);
}
```

fire():

```
public int fire(int row, int col){//No loops therefore O(1) time complexity

      if (board[row][col] == null){
            board[row][col] = Boats.missedSpot;
            return 0;
      }

      else if (board[row][col].getHealth() == -1 ||board[row][col].getHealth()
== -2){
            return -1;
      }

      else if (board[row][col].getHealth() > 0){
            board[row][col].loseHealth();
            if (board[row][col].getHealth() == 0){
                  System.out.println("You have sunk a boat!");
            }

            board[row][col] = Boats.hitBoat;
            return 1;
      }

      else{
            return -8;
      }
}
```

```
updateCoordinate()://No loops therefore O(1) time complexity
public void updateCoordinate(int row, int col, int entry){
	if (entry == 0){
		userBoard[row][col] = "[OO] ";
	}
	else if (entry == 1){
		userBoard[row][col] = "[XX] ";
	}
	else if ((entry == -1) || (entry == -2)){
		userBoard[row][col] = userBoard[row][col];
	}

	else{
		userBoard[row][col] = "ERROR";
	}
}
```

Sending the drone:

The droneContinue while loop just ensures the drone runs only one time and has no bearing on the complexity of the code overall. As we mentioned with the fire method, the while loop `validDirection` continues while the user input is invalid, and is thus O(n). The code also has another while loop that continues while the index the user provides is invalid, this is also O(n). This is all of the loops in the drone section in BattleBoats.java. The code in BattleBoats also calls `.getLength()`, a helper in the Boats class. It simply returns a static attribute of an object and is O(1). Finally, the actual `drone` function from the BattleBoatsBoard class. The drone function has two non-nested for loops that loop for a constant number of times (the length of the row or column). This is O(1). Or it could be considered O(n) if the board was an arbitrary n x n size (in our case it is not). It also calls `getHealth()` which is another basic getter method and is constant, O(1). Overall the complexity is O(n) because the while loop overrides the other constant methods called, regardless of whether or not we consider the Board size to be length n.

Drone from BattleBoats:

```
else if (action.equals("drone")){
	String direction = "";
	int index = -1;
	boolean droneCont = true;

	while (droneCont = true){
		if (dronesRemaining > 0){

			boolean validDirection = false;
```

```
                while (validDirection == false){
                System.out.println("Would you like to scan a 'row' or
'column'?");
                direction = s.next().trim().toLowerCase();
                      if (direction.equals("row") ||
direction.equals("column")){
                            validDirection = true;
                      }
                      else{
                            System.out.println("Enter 'row' or 'column'.");
                      }
          }

          boolean validIndex = false;
          while (validIndex == false){
                System.out.println("What " + direction + " would you like to
scan? (between 0 and " + (userBoard.getLength() - 1) + ")");
                                                //Constant getter
                index = s.nextInt();
                if (index > 0 && index <= userBoard.getLength()){
                      validIndex = true;
                }
                else {
                      System.out.println("Must be between 1 and " +
(userBoard.getLength() - 1) +".");
                }
          }

          if (direction.equals("row")){
                System.out.println("\n" + masterBoard.drone("row", index) +
" ships were found in this " + direction + " (hit or unhit).");
                droneCont = false;
                dronesRemaining --;
                System.out.println("\nYou now have " + dronesRemaining + "
drones remaining.");
                break;
          }

          else if (direction.equals("column")){
                System.out.println("\n" + masterBoard.drone("column",
                                        //Drone is constant overall
index) + " ships were found in this " + direction + " (hit or unhit).");
                droneCont = false;
                dronesRemaining --;
                System.out.println("\nYou now have " + dronesRemaining + "
drones remaining.");
                break;
                }
```

```
            }

            else{
                  System.out.println("You have no more drones remaining, you
imbecile!");
                  droneCont = false;
                  turns --;
                  break;
            }
      }
}
```

Drone from BattleBoatsBoard:

```
public int drone(String direction, int index) {
      int count = 0;

      if (direction.equals("row")){
            for (int i = 0; i < this.board[index].length; i++){  //Constant
Boat Object Size
                  if ((this.board[index][i] instanceof Boats) &&
(this.board[index][i].getHealth() != -2)){  //Constant getter
                        count ++;
                  }
            }
      }

      else if (direction.equals("column")){
            for (int i = 0; i < this.board[0].length; i++){  //Constant Boat
Object Size
                  if (this.board[i][index] instanceof Boats &&
this.board[i][index].getHealth() != -2){  //Constant getter
                        count ++;
                  }
            }
      }
      return count;
}
```