```
from itertools import cycle
from collections import namedtuple
from string import ascii_uppercase

SIZE = 10
LINE_LENGTH = (SIZE * 3) + 3
EMPTY = "-"
SHIP = "S"
HIT = "*"
MISS = "M"
SUNK = "Sunk"
SEPERATOR = "  "
BOARD_SEPERATOR = "  |  "
PROMPT = ">"
START_SHIPS = [5,4,3,2,1]
Point = namedtuple("Point", "x y")
Ship = namedtuple("Ship", "position length orientation")


class Board:
    def __init__(self, size):
        """Set up key variables."""

        self.size = size
        self.ships = {}
        self.guesses = []
        self.hits = 0

    def is_on_board(self, position):
        """Return True if the position is on the board."""
        for coordinate in position:
            if not (0 <= coordinate < self.size):
                return False
        return True

    def is_ship_on_board(self, ship):
        """Return True if ship fits on the board."""

        row, column = ship.position

        if ship.orientation == "H":
            long, short = column, row
        elif ship.orientation == "V":
            long, short = row, column

        if short > self.size:
            return False

        if long + ship.length > self.size:
            return False

        for direction in ship.position:
            if direction < 0:
                return False
```

```
        return True

    @staticmethod
    def generate_ship(ship):
        """Generate a ship as a list of coordinates."""

        row, column = ship.position

        ship_positions = []
        for i in range(ship.length):
            if ship.orientation == "H":
                ship_positions.append(Point(row, column + i))
            elif ship.orientation == "V":
                ship_positions.append(Point(row + i, column))

        return ship_positions

    def check_ship_collisions(self, ship):
        """Check if the current ship overlaps any other ships."""

        for other_ship in self.ships:
            other_ship_set = set(self.generate_ship(other_ship))
            current_ship_set = set(self.generate_ship(ship))
            if not other_ship_set.isdisjoint(current_ship_set):
                return True
        return False

    def place_ship(self, ship):
        """Place the current ship on the board."""
        self.ships[ship] = ship.length

    def delete_ship(self, ship):
        """Delete the current ship from the board."""
        del self.ships[ship]

    def is_guessed(self, position):
        """Return True if the position has already been guessed."""

        return position in self.guesses

    def guess_position(self, position):
        """Accepts game position and updates board. Also return
hit/miss."""

        self.guesses.append(position)

        for ship in self.ships:
            if position in self.generate_ship(ship):
                self.ships[ship] -= 1
                if self.ships[ship] == 0:
                    result = SUNK
                else:
                    result = HIT
```

```
                self.hits += 1
                break
        else:
            result = MISS

        return result

    def print_rows(self, show_ships=True):
        """Return a generator that produces the rows in the board."""

        seperator_length = len(SEPERATOR) + 1
        column_headers = "".join(
            [str(i).ljust(seperator_length) for i in range(1, self.size +
1)]
        )

        yield "   " + column_headers

        for row in range(self.size):
            row_letter = ascii_uppercase[row]

            row_text = ""
            for column in range(self.size):
                current_position = Point(row, column)

                for ship in self.ships:
                    if current_position in self.generate_ship(ship):
                        if current_position in self.guesses:
                            value = HIT
                        else:
                            if show_ships:
                                value = SHIP
                            else:
                                value = EMPTY
                        break
                else:
                    if current_position in self.guesses:
                        value = MISS
                    else:
                        value = EMPTY
                row_text += value + SEPERATOR

            yield row_letter + "  " + row_text

    def display(self, show_ships=True):
        """Display the board."""

        for row in self.print_rows(show_ships):
            print(row)

    def all_ships_sunk(self):
        """Return True if all ships have been sunk."""

        for ship_health in self.ships.values():
```

```
            if ship_health > 0:
                return False
        return True

    @staticmethod
    def convert_position(position):
        """Convert position from 'C6' notation to (2, 5) format."""
        row = position[0]
        column = position[1:]

        row_number = ascii_uppercase.index(row)
        column_number = int(column) - 1

        return Point(row_number, column_number)


def clear_screen():
    """Clear the screen."""

    print("\n" * 100)


def center(text, width=LINE_LENGTH):
    """Centre colorful strings."""

    length = len(text)
    start = (width - length) // 2
    end = width - length - start
    output = (" " * start) + text + (" " * end)
    return output


class Game:
    def __init__(self):
        """Set up key variables and play the game."""

        # Set up the variables
        self.players = "AB"
        self.player_names = {}
        self.boards = {player: Board(SIZE) for player in self.players}
        self.ships = START_SHIPS
        self.welcome()

        for player in self.players:
            self.set_up(player)
        # Play the game
        for player in cycle(self.players):
            if self.round(player):
                break

        winning_player = self.player_names[player]
        print(winning_player + " Won ")
        other_player = self.players.replace(player, "")
        with open("score.txt", 'a') as f:
```

```
            f.write(winning_player+"-
"+str(self.boards[other_player].hits)+"\n")
        self.play_again()

    def play_again(self):
        val = input("Do you want to play again ? (y/n)")
        if val.lower() == "y":
            Game()
        elif val.lower() == "n":
            exit()
        else:
            print("Please select between y or n")
            self.play_again()

    def welcome(self):
        """Print the initial input screen and wait for initial input."""

        print(" LET's PLAY BATTLESHIPS ")
        try:
            with open("score.txt", 'r') as f:
                print("Previous Record:")
                data = f.read()
                n_list = data.split('\n')
                n_list.reverse()
            for i in n_list[0:6]:
                if i:
                    val = i.split("-")
                    print(val[0].ljust(30) + "\t\t" + val[1])
        except:
            print("No Previous Record")
        # print("Press ENTER to Start Game !")
        # input()

    def set_up(self, player):
        """Allow the players to set up their screens."""

        # clear_screen()
        board = self.boards[player]

        input(f"Pass the computer to player {player}. \nPress enter when
ready.")

        print("Enter Name: ")
        self.player_names[player] = input(PROMPT)

        clear_screen()
        ship_names = ["Aircraft carrier", "Cruiser", "Ship", "Frigate",
"Submarine"]
        for number, ship in enumerate(self.ships, start=1):
            self.set_ship(player, ship, number, ship_names[0])
            ship_names.pop(0)

    def set_ship(self, player, ship, number, ship_name):
        """Set up one ship, including validating input."""
```

```
        board = self.boards[player]

        while True:
            board.display()

            print(
                "\nHere is your board!"
                + f"\nEquipment number {number} is "
                + ship_name
                + f" (length: {ship})"
                + "\nEnter the start point of the equipment (e.g A1)"
            )
            start_point = input(PROMPT).upper()
            print("Enter the orientation of the equipment" "\n Horizontal
or Vertical (h/v)")
            orientation = input(PROMPT).upper()

            # Checks
            try:
                start_position = board.convert_position(start_point)
            except BaseException:
                self.print_error("The start position is invalid.")
                continue

            if orientation != "H" and orientation != "V":
                self.print_error("The orientation entered is not either H
or V.")
                continue

            current_ship = Ship(start_position, ship, orientation)

            if not board.is_ship_on_board(current_ship):
                self.print_error("Some of the equipment would fall
outside of the board.")
                continue

            if board.check_ship_collisions(current_ship):
                self.print_error("The equipment entered collides with
another equipment.")
                continue

            board.place_ship(current_ship)
            board.display()

            print("Would you like to keep the equipment? (C) for
confirm.")
            if input(PROMPT).upper() != "C":
                board.delete_ship(current_ship)
                continue
            else:
                break

    def print_error(self, error):
```

```
        """Print the error message if invalid input."""
        print("Error \n {0} Please try again.".format(error))
        input()

    def round(self, player):
        """Go through one round of battleships."""

        player_name = self.player_names[player]
        other_player = self.players.replace(player, "")

        board = self.boards[player]
        other_board = self.boards[other_player]

        clear_screen()
        input(f"Pass the computer to {player_name}.Press Enetr when
ready.")

        clear_screen()
        print(f"Hello {player_name}.")

        while True:
            self.display_both_boards(player)

            print("Please enter your guess.")
            guess = input(PROMPT).upper()

            try:
                guess_position = other_board.convert_position(guess)
            except BaseException:
                self.print_error("The position entered is invalid.")
                continue

            if not other_board.is_on_board(guess_position):
                self.print_error("The guess is outside of the board.")
                continue

            if other_board.is_guessed(guess_position):
                self.print_error("This position has already been
guessed.")
                continue

            result = other_board.guess_position(guess_position)

            if result == HIT or result == SUNK:
                if result == HIT:
                    print("You HIT Enemy Equipment! ")
                elif result == SUNK:
                    print("You SUNK Enemy Equipment!")

                input()
                if other_board.all_ships_sunk():
                    return True
            elif result == MISS:
                print("You Hit in Water")
```

```
                input()
                break

        else:
            return False

    def display_both_boards(self, player):
        """Display the current board on the left and the other board on
the right."""

        other_player = self.players.replace(player, "")

        this_board_display =
self.boards[player].print_rows(show_ships=True)
        other_board_display =
self.boards[other_player].print_rows(show_ships=False)

        text_template = "{}  -- Score: {}"

        player_text = text_template.format(
            self.player_names[player], self.boards[other_player].hits
        )

        other_player_text = text_template.format(
            self.player_names[other_player], self.boards[player].hits
        )

        player_header = center(player_text)
        other_player_header = center(other_player_text)

        print(player_header + (" " * len(BOARD_SEPERATOR)) +
other_player_header)

        for this_board, other_board in zip(this_board_display,
other_board_display):
            print(this_board + BOARD_SEPERATOR + other_board)


if __name__ == "__main__":
    Game()
```