board.py

Code:

```
from ship import Ship

class Board:
    def __init__(self):
        self.size = 10
        self.grid = [['~' for _ in range(self.size)] for _ in range(self.size)]
        self.ships = []

    def print_board(self):
        """Prints the board grid for the player to view."""
        print("  " + " ".join(chr(65 + i) for i in range(self.size)))  # A-J
        for i in range(self.size):
            print(f"{i + 1} " + " ".join(self.grid[i]))

    def place_ship(self, ship):
        """Places the ship on the board if it passes bounds and overlap checks."""
        if not ship.is_within_bounds(self.size):
            print("Ship cannot be placed. It exceeds the board limits.")
            return False

        for existing_ship in self.ships:
            if ship.overlaps_with(existing_ship):
                print("Ship overlaps with another ship. Choose a different location.")
                return False

        # Place the ship if all checks pass
        for x, y in ship.coordinates:
            self.grid[x][y] = 'S'

        self.ships.append(ship)
        return True

    def receive_fire(self, x, y):
        """Handles the result of a guess (hit or miss)."""
        for ship in self.ships:
            if (x, y) in ship.coordinates and not ship.destroyed:
                self.grid[x][y] = 'X'  # Hit
                ship.destroyed = True  # Mark the ship as destroyed
                print(f"Ship at {x+1},{chr(y+65)} has been destroyed!")
                return True
        if self.grid[x][y] == '~':
            self.grid[x][y] = 'O'  # Miss
        return False

    def all_ships_sunk(self):
        """Checks if all ships on the board have been sunk."""
        return all(ship.destroyed for ship in self.ships)
```

Notes:

Board Class Documentation

The `Board` class represents the game board in the Battleship game program. It is responsible for managing the game grid, placing ships, handling player guesses, and tracking the status of ships.

Initialization

The `__init__` method initializes a new `Board` instance with the following attributes:

- `size`: The size of the game grid, set to 10 by default.
- `grid`: A 2D list representing the game grid, initialized with all cells set to '~' (water).
- `ships`: An empty list to store the ships placed on the board.

Methods

`print_board`

Prints the current state of the game grid to the console. The grid is displayed with row and column labels (A-J and 1-10, respectively).

`place_ship`

Attempts to place a ship on the board. The method performs the following checks:

- Bounds check: Ensures the ship does not exceed the board limits.
- Overlap check: Verifies that the ship does not overlap with any existing ships on the board.

If both checks pass, the ship is placed on the board by updating the grid cells to 'S' (ship). The ship is then added to the `ships` list.

`receive_fire`

Handles the result of a player's guess (hit or miss). The method checks if the guessed coordinates match a ship's location. If a ship is hit, the grid cell is updated to 'X' (hit), and the ship is marked as destroyed. If the guess is a miss, the grid cell is updated to 'O' (miss).

`all_ships_sunk`

Checks if all ships on the board have been sunk by verifying that all ships in the `ships` list have their `destroyed` attribute set to `True`.

- The `Ship` class is assumed to have the following methods and attributes:
  - `is_within_bounds(size)`: Checks if the ship is within the board limits.
  - `overlaps_with(other_ship)`: Verifies if the ship overlaps with another ship.
  - `coordinates`: A list of coordinates representing the ship's location.
  - `destroyed`: A boolean attribute indicating if the ship has been destroyed.

—---

<u>main.py</u>

Code:

```
from player import Player

def main():
    player1 = Player(input("Enter name for Player 1: "))
    player2 = Player(input("Enter name for Player 2: "))

    player1.place_ships()
    player2.place_ships()

    while True:
        print(f"\n{player1.name}'s turn to guess:")
        player1.make_guess(player2)

        if player2.board.all_ships_sunk():
            print(f"{player1.name} wins! All ships of {player2.name} are sunk.")
            break

        # Player 2's turn
        print(f"\n{player2.name}'s turn to guess:")
        player2.make_guess(player1)

        if player1.board.all_ships_sunk():
            print(f"{player2.name} wins! All ships of {player1.name} are sunk.")
            break

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

<u>Notes:</u>

Main Game Loop Documentation

The `main` function serves as the entry point for the Battleship game program. It is responsible for initializing the game, managing player turns, and determining the winner.

Initialization

The `main` function initializes two `Player` instances, `player1` and `player2`, with names input by the users.

Game Loop

The game loop consists of the following steps:

1. Ship Placement: Each player is prompted to place their ships on their respective boards using the `place_ships` method.
2. Player Turns: The game loop alternates between `player1` and `player2`, with each player taking turns to make a guess using the `make_guess` method.
3. Win Condition Check: After each guess, the game checks if all ships on the opponent's board have been sunk using the `all_ships_sunk` method. If a player's ships are all sunk, the game ends, and the other player is declared the winner.

Notes

- The `Player` class is assumed to have the following methods and attributes:
    - `place_ships`: Prompts the player to place their ships on their board.
    - `make_guess(opponent)`: Allows the player to make a guess on the opponent's board.
    - `name`: The player's name.
    - `board`: The player's game board, an instance of the `Board` class.

Entry Point

The `if __name__ == "__main__":` block ensures that the `main` function is only executed when the script is run directly (i.e., not when it is imported as a module by another script).

player.py

Code:
```
from board import Board
from ship import Ship

class Player:
    def __init__(self, name):
        self.name = name
        self.board = Board()
        self.guesses = Board()

    def place_ships(self):
        num_ships = int(input(f"{self.name}, enter the number of ships (1-5): "))

        while num_ships < 1 or num_ships > 5:
            print("Please enter a valid number of ships (1-5).")
            num_ships = int(input(f"{self.name}, enter the number of ships (1-5): "))

        for size in range(1, num_ships + 1):
            valid_position = False

            while not valid_position:
                position = input(f"Place your {size}x1 ship (e.g., B3): ").upper()
                if len(position) < 2 or not position[0].isalpha() or not position[1:].isdigit():
                    print("Invalid input format. Please use the format 'LetterNumber' (e.g.,
B3).")
                    continue

                x = int(position[1:]) - 1
                y = ord(position[0]) - 65

                if x < 0 or x >= self.board.size or y < 0 or y >= self.board.size:
                    print("Position out of bounds. Please choose a valid position on the board.")
                    continue
                orientation = input("Choose orientation (H for horizontal, V for vertical):
").upper()
                if orientation not in ['H', 'V']:
                    print("Invalid orientation. Please enter 'H' for horizontal or 'V' for vertical.")
                    continue

                ship = Ship(size, (x, y), orientation)
                if self.board.place_ship(ship):
                    valid_position = True

    def make_guess(self, opponent):
        valid_guess = False

        while not valid_guess:
            guess = input(f"{self.name}, enter your guess (e.g., B3): ").upper()

            if len(guess) < 2 or not guess[0].isalpha() or not guess[1:].isdigit():
                print("Invalid input format. Please use the format 'LetterNumber' (e.g., B3).")
                continue

            x = int(guess[1:]) - 1
            y = ord(guess[0]) - 65

            if x < 0 or x >= self.guesses.size or y < 0 or y >= self.guesses.size:
                print("Guess out of bounds. Please choose a valid position on the board.")
                continue

            hit = opponent.board.receive_fire(x, y)
            self.guesses.grid[x][y] = 'X' if hit else 'O'
            if hit:
                print("It's a hit!")
            else:
                print("It's a miss!")
            valid_guess = True
```

Notes:

Player Class Documentation

The `Player` class represents a player in the Battleship game program. It is responsible for managing the player's game board, placing ships, and making guesses.

Initialization

The `__init__` method initializes a new `Player` instance with the following attributes:

- `name`: The player's name.
- `board`: The player's game board, an instance of the `Board` class.
- `guesses`: A separate board to track the player's guesses, also an instance of the `Board` class.

Methods

`place_ships`

Prompts the player to place their ships on their board. The method:

1. Asks the player to enter the number of ships (1-5).
2. Loops until the player enters a valid number of ships.
3. For each ship, prompts the player to enter the ship's position (e.g., B3) and orientation (H or V).
4. Validates the input and creates a `Ship` instance.
5. Attempts to place the ship on the board using the `place_ship` method. If successful, the ship is placed, and the loop continues.

`make_guess`

Allows the player to make a guess on the opponent's board. The method:

1. Prompts the player to enter their guess (e.g., B3).
2. Validates the input and converts it to coordinates (x, y).
3. Checks if the guess is within the board bounds.

4. Calls the `receive_fire` method on the opponent's board to determine if the guess is a hit or miss.
5. Updates the player's `guesses` board with the result (X for hit, O for miss).
6. Prints a message indicating whether the guess was a hit or miss.

Notes

- The `Board` class is assumed to have the following methods and attributes:
  - `place_ship(ship)`: Attempts to place a ship on the board.
  - `receive_fire(x, y)`: Handles the result of a guess (hit or miss).
  - `size`: The size of the game grid.
- The `Ship` class is assumed to have the following attributes:
  - `size`: The size of the ship.
  - `coordinates`: A tuple representing the ship's position (x, y).
  - `orientation`: A string indicating the ship's orientation (H or V).

<u>ship.py</u>

Code:

```
class Ship:
    def __init__(self, size, position, orientation):
        self.size = size  #tracks size
        self.position = position #tracks the position on the grid
        self.orientation = orientation #tracks the orientation such as horizontal or vertical.
        self.coordinates = self.get_coordinates() #this will ensure cordinates the a ship is
allowed to be placed on.
        self.destroyed = False  # Flag to track if the ship is destroyed

    def get_coordinates(self):
        """Generate all coordinates that this ship will cover."""
        x, y = self.position
        coordinates = []
        #Detemines the valid positions for the horizontal co-ordinates.
        if self.orientation == 'H':
            coordinates = [(x, y + i) for i in range(self.size)]
        #Detemines the valid positions for the vertical co-ordinates.
        else:
            coordinates = [(x + i, y) for i in range(self.size)]
        return coordinates

    def is_within_bounds(self, board_size):
        """Check if the ship is within board boundaries."""
        #Takes the column name "A-J" and row number "1-10" and makes sure that they
are within the bound.
        for x, y in self.coordinates:
            if x < 0 or x >= board_size or y < 0 or y >= board_size:
                return False
        return True

    def overlaps_with(self, other_ship):
        """Check if this ship overlaps with another ship."""
        #Boolean function, that returns true if it overlaps, else returns False.
        for coord in self.coordinates:
            if coord in other_ship.coordinates:
                return True
        return False
```

Notes:

Ship Class Documentation

The `Ship` class represents a ship in the Battleship game program. It is responsible for managing the ship's properties, such as its size, position, orientation, and coordinates.

Initialization

The `__init__` method initializes a new `Ship` instance with the following attributes:

- `size`: The size of the ship.
- `position`: A tuple representing the ship's position (x, y) on the grid.
- `orientation`: A string indicating the ship's orientation (H for horizontal or V for vertical).
- `coordinates`: A list of tuples representing the ship's coordinates, generated by the `get_coordinates` method.

- `destroyed`: A flag indicating whether the ship is destroyed (initially set to `False`).

Methods

`get_coordinates`

Generates all coordinates that the ship will cover based on its size and orientation.

- If the ship is horizontal (H), generates coordinates by incrementing the y-coordinate.
- If the ship is vertical (V), generates coordinates by incrementing the x-coordinate.

`is_within_bounds`

Checks if the ship is within the board boundaries.

- Take the board size as an argument.
- Iterates through the ship's coordinates and checks if any of them are outside the board boundaries.
- Returns `False` if any coordinate is out of bounds, otherwise returns `True`.

`overlaps_with`

Checks if the ship overlaps with another ship.

- Takes another `Ship` instance as an argument.
- Iterates through the ship's coordinates and checks if any of them match the coordinates of the other ship.
- Returns `True` if an overlap is found, otherwise returns `False`.

Notes

- The `Ship` class is designed to be used in conjunction with the `Board` class, which is assumed to have a `size` attribute representing the size of the game grid.
- The `Ship` class is also designed to be used in conjunction with the `Player` class, which is assumed to have a `board` attribute representing the player's game board.

Disclaimer: These notes were made using the help of Chatgpt to create the notes in an easy-to-understand fashion so the next team has a good understanding of what is going on. There were no code documentation specifications yet as we will learn more about documentation artifacts later.