Code Documentation

Filename: `game.py`

The `game.py` module defines the Game class, which manages the overall game flow of a Battleship game. This class handles player turns, the various game phases (menu, ship placement, attack), and player actions such as placing ships and attacking. The game progresses through different stages, including the menu, ship placement phase, and attack phase, all driven by a central game loop.

Constructor:

> The `__init__()` method initializes the game by setting up player turns and creating two player objects. It also defines the states for the different game phases: the menu phase, ship placement phase, attack phase, and game end phase. Utility lookup tables are created to map turn numbers to the respective players and their enemies. Additionally, various messages are initialized for user interface display, including prompts and notifications about the game state.

Phases:

> **Menu Phase:** The game starts in the menu_phase, where players choose the number of ships to play with. The show_menu method handles the ship count input (from 1 to 5), prepares the game with the corresponding ships, and then transitions to the ship placement phase.
>
> **Ship Placement Phase:** The show_place_ship_phase method manages the process of placing ships. Each player places their ships by clicking on the grid, and the orientation can be toggled between horizontal and vertical with a right-click. Once all ships are placed for both players, the game transitions to the attack phase.
>
> > get_placement is responsible for handling player ship placement on the grid. It checks for mouse clicks to place or rotate ships and ensures that placements are valid.
>
> **Attack Phase:** The attack phase is managed by the show_attack_phase method. Players take turns attacking each other's boards, and hits, misses, or sunken ships are reflected in real-time through messages. The attacking player can toggle between viewing their own board and the enemy's board by pressing 'B'.

get_attack handles the mechanics of attacking the enemy's board. It checks for valid attacks, updates the board, and notifies players of hits, misses, or sunken ships. If all ships are destroyed, the game enters the end phase.

Game End Phase: Once all of one player's ships are destroyed, the game moves to the game_end_phase. The losing player's board is displayed, and a message declares the winner through the show_game_end_phase method.

Methods:

The **`get_placement(player)`** method is responsible for handling ship placement for the specified player. It processes mouse clicks to allow players to place their ships on their boards. Players can rotate their ships between horizontal and vertical orientations using the right mouse button. If a ship is successfully placed, it is removed from the player's available ship list. If an attempted placement is invalid, an error message is displayed to inform the player. When a player has placed all their ships, they are marked as finished, allowing the game to proceed.

The **`get_attack(player, enemy)`** method manages the attack phase, enabling the current player to attack the enemy player's board. The game checks if the clicked cell is valid for an attack and provides feedback based on the result, which can include notifications for hits, misses, or sunk ships. If an attack is successfully made, the method returns `True`; otherwise, it returns `False`.

In the **`show_menu()`** method, the initial menu is displayed, allowing players to select the number of ships they wish to use. Players can choose between 1 and 5 ships by pressing the corresponding key on their keyboard. Once the selection is made, the game transitions to the ship placement phase, indicating the next steps for the players.

The **`show_place_ship_phase()`** method manages the ship placement phase for both players. It draws the current player's board and waits for ship placements. As players place their ships, the method handles turn-switching and ensures that both players complete their placements before transitioning to the attack phase.

The **`toggle_show_board()`** method allows players to switch between viewing their own board and the enemy's board during the attack phase by pressing the 'B' key. This enhances the gameplay experience by giving players the option to strategize based on what they can see.

In the **`show_attack_phase()`** method, the attack phase is managed, allowing players to take turns attacking each other's ships. The current player's turn is indicated, and attacks can be made on the enemy's board unless they are currently viewing their own board. The method handles the logic of determining whether an attack is valid and provides

feedback on the attack's outcome. If one player sinks all the ships of their opponent, they are declared the winner, and the game transitions to the game end phase.

The **`show_game_end_phase()`** method displays the final phase of the game, showing the losing player's board along with all of their sunk ships. This provides closure to the game and highlights the outcome.

The **`draw_info_messages()`** method is responsible for drawing various informational messages on the screen during gameplay. This includes the current turn information, results of attacks (hit or miss), remaining ships to place, and the winner's message at the end of the game. The messages are color-coded to enhance readability and player experience.

Finally, the **`game_loop()`** method serves as the main loop that continuously updates the game state. Depending on the current phase of the game, it either displays the menu, manages ship placements, handles attacks, or concludes the game. This loop ensures smooth transitions between phases and allows for real-time updates during gameplay, keeping the players engaged and informed.

Filename: `player.py`

The `player.py` module defines the `Player` class for a Battleship game, managing the player's interactions with their board, ships, attacks, and special power ups. It provides functionality for placing ships, executing attacks, and checking whether a player has lost the game. The `Player` class also includes several power ups that can significantly alter gameplay strategy, adding complexity and excitement to the traditional Battleship format.

The `Powerup` enumeration defines the available power ups that can be used during gameplay. Each powerup offers a unique tactical advantage. The powerups are: `MULTISHOT`, which allows 7 individual 1x1 shots; `BIG_SHOT`, which targets a 3x3 area; `LINE_SHOT`, which fires along an entire row or column; `RANDOM_SHOT`, which selects 10 random cells to attack; and `REVEAL_SHOT`, which uncovers a 2x6 area on the enemy’s board, assisting the player in locating ships. These powerups introduce an element of unpredictability and dynamic decision-making.

The `Player` class represents an individual player in the Battleship game. It manages the player's fleet of ships, tracks the state of the board, handles attacks, and implements the powerup system. The class is responsible for key game operations such as placing ships on the board, registering hits and misses during attacks, and determining whether a player has lost the game by checking if all their ships have been destroyed.

Constructor:

The `__init__` method initializes a new `Player` object. It takes the player's number as an argument, which is used to differentiate between players. During initialization, several important attributes are set up. These include the player's list of ships, which starts empty, and dictionaries to track remaining ship parts (`ship_count`) and hit locations (`ship_hits`). The player's board is also created as an instance of the `Board` class, and variables related to powerups, such as `powerup_options`, `powerup_choice`, and `powerup_locked`, are initialized to manage the player's powerup interactions. The constructor also sets the default ship orientation and initializes other attributes needed for advanced gameplay.

Methods:

The **`get_ships`** method generates the player's fleet based on the specified number of ships. It creates ships of varying sizes, ranging from 1 to the given number, and updates the `ship_count` and `ship_hits` dictionaries to track the state of each ship. The total number of cells occupied by ships is also calculated.

The **`place_ship`** method handles the logic for placing a ship on the board. It checks whether the specified ship can be placed at the given coordinates in the chosen orientation (horizontal or vertical). If valid, the method updates the board by marking the cells where the ship is located. This method ensures that ships are placed correctly without overlapping or exceeding board boundaries.

The **`change_cells_to_sunk`** method is called when a ship is completely destroyed. It changes the state of all cells occupied by the sunken ship to a special `SUNK_CELL` status, visually marking the ship’s destruction on the player's board. This helps players track their progress in sinking enemy ships.

The **`place_attack`** method processes an attack on the player’s board. It checks whether an attack at the specified coordinates hits or misses a ship. If a ship is hit, the method updates the relevant ship’s status in the `ship_count` and `ship_hits` dictionaries, and if the attack sinks a ship, the affected cells are marked as sunk. If the attack misses, the corresponding cell is marked as a miss. The method also prevents players from targeting the same cell multiple times by returning a signal if the cell has already been attacked.

The **`get_shots_from_attack`** method calculates the coordinates affected by an attack, factoring in any power ups the player has selected. Depending on the active powerup, the method determines whether the attack should target a single cell, multiple cells, or a special area like a full row or a random selection of cells. This method ensures that the player’s powerups are correctly applied during their turn, enhancing the tactical options available.

The **`get_preview_cells`** method provides a visual preview of which cells will be affected by an attack, taking into account any power ups that might modify the area of effect. This preview is displayed when the player hovers over potential attack locations, allowing them to make more informed decisions about where to target. The method supports all power ups, showing how each would influence the board before the player finalizes their attack.

The **`get_new_powerup_options`** method randomly assigns two powerup options to the player at the start of their turn. These powerups offer different strategic possibilities, and the player can choose one to use during their turn. The method resets the powerup state at the beginning of each turn to ensure that power ups are not carried over from previous turns.

The **`is_loss`** method determines whether a player has lost the game. A player loses when all their ships have been destroyed, meaning that the total number of ship cells remaining (`num_ship_cells`) reaches zero. This method checks the player's ship status and returns a boolean value indicating whether the player has lost.