Класс Player
Этот класс реализует логику игрока в морском бое, включая управление кораблями, взаимодействие с игровым полем и обработку выстрелов.

PlayerInfo
Эта структура хранит основную информацию об игроке.
Поля структуры:
1. **name** (string)
   Имя игрока.
2. **person** (bool)
   Тип игрока:
   - true — игрок человек.
   - false — игрок управляется ИИ.
3. **titul_player** (string)
   Заголовок или обозначение игрока (например, "Player 1").
4. **my_ships** (Board)
   Игровое поле игрока, где расположены его корабли.
5. **enemy_ships** (Board)
   Поле противника (для отображения состояния игры).

Методы структуры:
1. **PlayerInfo(string n, bool person, string titul_player, Board my, Board enemy)**
   Конструктор с параметрами.
2. **PlayerInfo()**
   Конструктор по умолчанию.

PlayerStatistic
Эта структура хранит статистику игрока за текущую игру.
Поля структуры:
1. **drawned_ships** (int)
   Количество потопленных кораблей противника.
2. **processed_shots** (int)
   Общее количество сделанных выстрелов.
3. **sucessful_shots** (int)
   Количество успешных попаданий.
4. **percent_suc_shots** (int)
   Процент успешных попаданий (рассчитывается как отношение успешных попаданий к общему числу выстрелов).
5. **stat_alive_enemy_ships** (vector<ShipType>)
   Список живых кораблей противника (с указанием их размером и количества).
6. **stat_dead_enemy_ships** (vector<ShipType>)
   Список потопленных кораблей противника (с указанием их размеров и количества).

7. **initial_ships** (vector<ShipType>)
   Список кораблей в начале игры (с указанием их размеров и количества).

Методы структуры:

1. **PlayerStatistic(int x, int y, int z, int b, vector<ShipType> ships)**
   Конструктор с параметрами. Инициализирует объект заданными значениями и заполняет списки stat_alive_enemy_ships и stat_dead_enemy_ships.
2. **PlayerStatistic()**
   Конструктор по умолчанию.

PlayerResultOfStep

Эта структура хранит результаты хода игрока.

Поля структуры:

1. **win_player** (bool)
   Статус победы:
   - true — текущий игрок победил.
   - false — игра продолжается.
2. **in_a_row** (bool)
   Указывает, является ли текущий ход частью успешной серии попаданий.
3. **target_mode** (bool)
   Режим "уничтожения" для ИИ:
   - true — включен (ИИ продолжает стрелять по кораблю).
   - false — выключен (ИИ ищет новый корабль).
4. **target_queue** (vector<COORD>)
   Очередь целей для атак (используется ИИ для добивания повреждённых кораблей).
5. **result_shot** (PlayerResultOfShot)
   Структура. Результат последнего выстрела:

Методы структуры:

1. **PlayerResultOfStep(bool win_player, bool in_a_row, bool target_mode, PlayerResultOfShot result_shot)**
   Конструктор с параметрами.
2. **PlayerResultOfStep()**
   Конструктор по умолчанию.

Поля класса:

1. **info** (PlayerInfo)
   Структура, содержащая информацию об игроке:
2. **statistic** (PlayerStatistic)
   Структура, содержащую статистику игрока:

3. **result_of_step** (PlayerResultOfStep)
   Структура, содержащая результаты последнего хода игрока (в том числе выстрела).
4. **coords** (COORD)
   Текущие координаты курсора игрока.
5. **ship_sells** (int)
   Количество «живых» клеток всех кораблей игрока (для проверки победы).

Методы-конструкторы:

1. **Player(string name, Board my_ships, Board enemy_ships, vector<ShipType> ships, int ship_sells, bool person, string titul_player)**

Описание

Создаёт игрока с заданным именем, полями, типом (человек/ИИ), списком кораблей и начальными параметрами.

Параметры:

- name (string) — имя игрока.
- my_ships (Board) — игровое поле игрока (с кораблями).
- enemy_ships (Board) — игровое поле противника.
- ships (vector<ShipType>) — список кораблей игрока.
- ship_sells (int) — количество клеток всех кораблей игрока.
- person (bool) — тип игрока (true для человека, false для ИИ).
- titul_player (string) — титул игрока (например, "Player 1").

Возвращаемое значение:

Не возвращает значения. Он инициализирует объект Player.

2. **Player()**

Описание

Конструктор по умолчанию.

Возвращаемое значение:

Не возвращает значения. Он инициализирует объект Player с пустыми значениями.

Методы размещения кораблями на поле боя:

1. **AutoBoardShipPlacement(string name, Player& another_player)**

Описание

Автоматически размещает корабли игрока на поле.

Параметры:

- name (string) — имя игрока (для отображения в консоли).
- another_player (Player&) — объект противника, для размещения его кораблей.

Возвращаемое значение:

Метод не возвращает значения. Он изменяет состояние поля игрока и противника, размещая корабли.

2. **BoardShipPlacement(string name, Player& another_player)**

Описание

Реализует ручное размещение кораблей игроком. Пользователь выбирает позиции и ориентацию кораблей.

Параметры:

- name (string) — имя игрока (для отображения в консоли).
- another_player (Player&) — объект противника, на чьем поле размещаются корабли.

Возвращаемое значение:

Метод не возвращает значений. Он изменяет игровое поле игрока, размещая корабли вручную.

3. **paintFutureShip(int x, int y, ShipType ship_player, bool vertical)**

Описание

Предварительное отображение корабля на поле перед размещением.

Параметры:

- x (int) — координата по оси X для начала размещения корабля.
- y (int) — координата по оси Y для начала размещения корабля.
- ship_player (ShipType) — тип корабля, который нужно отобразить.
- vertical (bool) — ориентация корабля (true — вертикальная, false — горизонтальная).

Возвращаемое значение:

Метод не возвращает значений. Он обновляет визуальное отображение корабля на поле.

Методы атаки:

1. Attack_manual(Player* another_player)

Описание

Осуществляет атаку вручную. Игрок выбирает координаты для выстрела.

Параметры:

- another_player (Player) — объект противника, по которому будет произведён выстрел.

Возвращаемое значение:

Метод не возвращает значений. Он изменяет состояние поля противника в зависимости от результата выстрела.

2. Attack_computer(Player* another_player)

Описание

Осуществляет автоматическую атаку. Используется ИИ, с учётом стратегии (поиск и уничтожение повреждённых кораблей).

Параметры:

- another_player (Player) — объект противника, на поле которого ИИ выполняет выстрел.

Возвращаемое значение:
Метод не возвращает значений. Он изменяет состояние поля противника и выполняет выстрел на основе стратегии ИИ.

3. **check_zone(int x, int y, bool& horizontal, int& move, bool& move_minus)**
   Проверяет зону вокруг координаты для расчёта следующего выстрела.

Параметры:
- x (int) — координата X текущего выстрела.
- y (int) — координата Y текущего выстрела.
- horizontal (bool&) — ориентация для следующего выстрела (будет изменена в случае необходимости).
- move (int&) — количество шагов для выстрела, если требуется корректировка.
- move_minus (bool&) — флаг, показывающий направление движения для корректировки (если true, выстрел сдвигается в одну сторону, если false, то в другую).

Возвращаемое значение:
Возвращает false, если зона проверки пустая и выстрел возможен, иначе true.

Методы для ведения статистики:

1. **CalcStatShips(vector<ShipType> ships, vector<ShipType> ships_player, vector<ShipType> initial_ships, Board board)**
   Вычисляет текущее состояние кораблей игрока (живые и уничтоженные).

Параметры:
- ships (vector<ShipType>) — список кораблей противника.
- ships_player (vector<ShipType>) — список кораблей игрока.
- initial_ships (vector<ShipType>) — начальное количество кораблей игрока.
- board (Board) — поле игрока, для которого проверяется состояние кораблей.

Возвращаемое значение:
Возвращает обновлённый список кораблей игрока, с учётом потопленных.

2. **my_stat(PlayerResultOfShot rezult)**
   Обновляет статистику игрока после выстрела, включая успешные попадания, процент попаданий и потопленные корабли.

Параметры:
- rezult (PlayerResultOfShot) — результат выстрела (попадание или промах, потопленный корабль и так далее).

Возвращаемое значение:
Метод не возвращает значений. Он обновляет статистику игрока.

3. **output_stat(int fieldSize, int y)**
   Выводит текущую статистику игрока в консоль, включая живые и потопленные корабли.

Параметры:
- fieldSize (int) — размер поля.
- y (int) — вертикальная позиция для вывода статистики в консоли.

Возвращаемое значение:
Метод не возвращает значений. Он выводит информацию в консоль.

Системные методы:

1. **SetCursor(int x, int y)**
   Устанавливает курсор в заданные координаты консоли (используется для вывода графики).

Параметры:
- x (int) — горизонтальная координата.
- y (int) — вертикальная координата.

Возвращаемое значение:
Метод не возвращает значений. Он изменяет положение курсора в консоли.

2. **~Player()**
   Деструктор (по умолчанию).

Параметры:
- нет параметров.

Возвращаемое значение:
Деструктор не возвращает значений. Он выполняет очистку ресурсов объекта при его уничтожении.

Алгоритм метода Attack_computer:

1. **Инициализация переменных:**
   - fieldSize — размер игрового поля противника (в ячейках).
   - x и y — координаты для выстрела, которые будут вычислены в процессе работы метода.
2. **Проверка режима уничтожения (target_mode):**
   - Если **target_mode** включен (предполагается, что ИИ уже попадал в корабль и хочет продолжить атаку на соседние клетки), то:
     - ИИ вытягивает координаты следующей цели из **target_queue**.
     - Если очередь не пуста, то использует координаты из конца очереди и удаляет их из списка.
     - Если очередь пуста, продолжается выполнение случайных выстрелов.
   - Если **target_mode** выключен (то есть ИИ не попадал в корабль и не нужно продолжать атаку), то:
     - ИИ будет случайным образом выбирать координаты на поле для выстрела.
3. **Генерация случайных координат для выстрела:**
   - В случае обычной атаки (когда **target_mode** выключен):
     - Генерируются случайные координаты x и y на поле.
     - Процесс повторяется, если выстрел попадает в клетку, которая уже была обстреляна или является мёртвой зоной (отмечена как 'O' или 'X').
4. **Проверка зоны вокруг выстрела (метод check_zone):**
   - После того как случайные координаты были выбраны, вызывается метод **check_zone**, который проверяет, можно ли стрелять в этих координатах, не нарушив правила (например, попадание в занятые или подбитые клетки). Также метод может скорректировать координаты атаки для оптимизации урона врагу.
   - Если зона вокруг координат не пустая, то ИИ корректирует координаты и пытается выстрелить снова.
5. **Выполнение выстрела:**
   - Если координаты прошли проверку зоны, выполняется выстрел:
     - Вызов метода **processShot** на поле противника для обработки выстрела.
     - Метод **processShot** на игровом поле (my_ships) обрабатывает выстрел и возвращает информацию о результате выстрела.
6. **Обработка результатов выстрела:**
   - Если результат выстрела — **попадание** (возвращаемое значение rezult_of_shot == 1), то:
     - Уменьшается счётчик оставшихся клеток кораблей.

        - Если все корабли противника уничтожены (поля кораблей игрока исчерпаны), ИИ завершает игру с победой.
        - Если корабль потоплен, ИИ сбрасывает **target_mode** (выйти из режима уничтожения) и очищает очередь целей.
        - Если корабль не потоплен и проверка в 4-х направления показала повреждение секции корабля (помимо текущей), ИИ сохраняет **target_mode** включённым и включает в очереди атаки клетки, находящиеся спереди носа и позади кормы корабля.
        - Если поврежденные секции корабля (помимо текущей) не обнаружены, тогда метод добавляет соседние клетки в **target_queue** для дальнейших атак.
    - Если выстрел не попал, процесс продолжается с новым выстрелом, и **in_a_row** устанавливается в **false** для прекращения серии попаданий.
7. **Завершение работы режима уничтожения:**
    - Когда режим уничтожения завершён (например, когда корабль уничтожен), очередь целей **target_queue** очищается, и ИИ возвращается к случайным атакам на поле.

Алгоритм Check_zone:

1. **Определение минимального размера корабля:** Метод начинает с определения минимального размера корабля, проверяя все типы кораблей и увеличивая минимальный размер для потопленных кораблей.
2. **Подсчёт доступных клеток в каждом направлении:** Для каждого из четырёх направлений (вверх, вниз, влево, вправо) рассчитывается, сколько шагов можно пройти, не сталкиваясь с занятой клеткой. Это делается с помощью функции **checkDirection**, которая возвращает количество доступных клеток в направлении.
3. **Проверка, есть ли место для выстрела или размещения:** После того, как получены значения для шагов в каждом направлении, метод проверяет, есть ли в строке или колонке достаточно места для корабля или выстрела. Если да, то соответствующие координаты обновляются, и метод возвращает **false**, чтобы сигнализировать о том, что размещение или выстрел возможны.
4. **Корректировка позиции для выстрел**а, если в строке или колонке достаточно места: Проверка на достаточность места. Если в строке (**in_a_row**) или колонке (**in_a_column**) есть достаточно свободного места для корабля, то:
    1. Для **горизонтального** размещения проверяются клетки слева и справа. Если слева или справа корабль не помещается, вычисляется сдвиг вправо или влево соотвествтенно.

2. Для **вертикального** размещения проверяются клетки выше и ниже. Если сверху или снизу корабль не помещается, вычисляется сдвиг вниз или вверх соответственно.