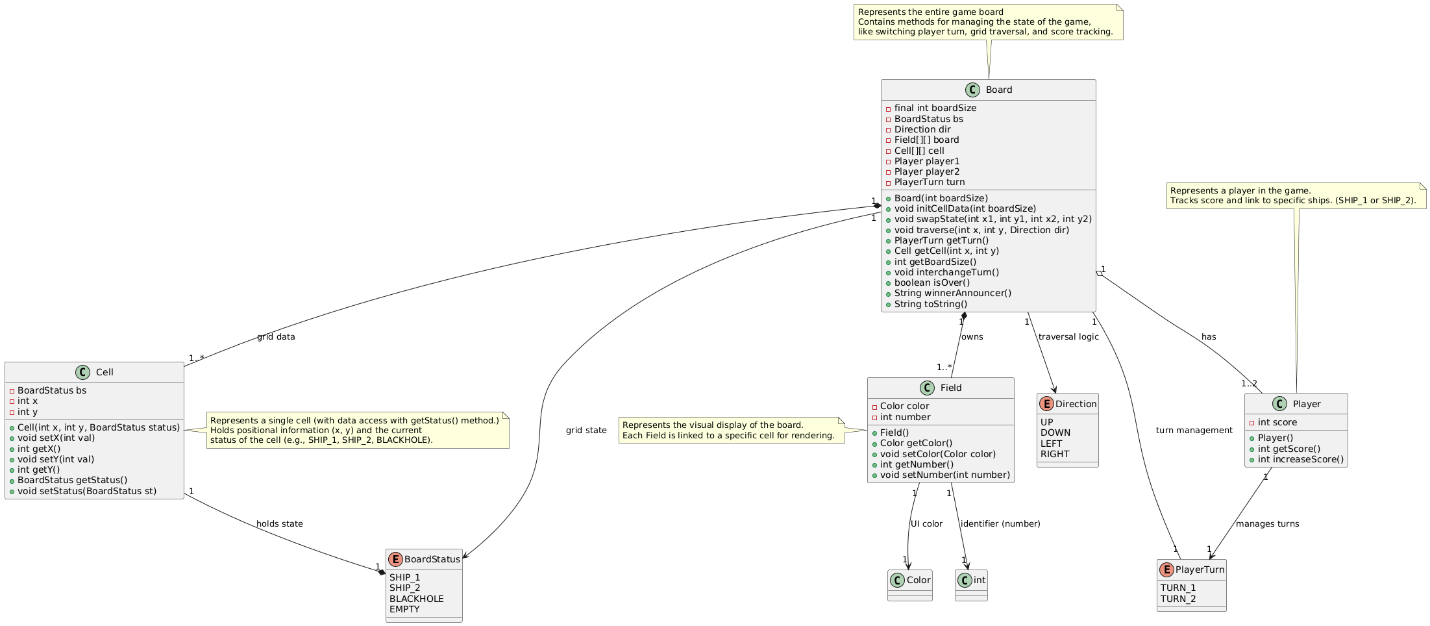
**Tran Quoc Tuan – UG2F20**

**Task 9:**  
Black hole is a two-player game, played on a board consists of n x n fields, which has a black hole at its center. Each player has n-1 spaceships, which are placed initially in the lower (upper) diagonal of the board (so the same colored spaceships placed on the same side). The players take turns moving one of their own spaceships. The black hole interferes with the navigation system of the spaceships, so they cannot move only one place, but they move until they reach the edge of the board, the black hole, or an other spaceship. A spaceship cannot jump over an other one. A player wins, if he manages to move half of his spaceships into the black hole. Implement this game, and let the board size be selectable (5x5, 7x7, 9x9). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

**SOLUTION:**

UML Diagram



***METHOD:***

**Class: Board**

1. **Board(int boardSize)**:  
   Constructor that initializes the board with a specified size, populates it with Cell objects, and sets up initial positions for ships and a black hole.
2. **initCellData(int boardSize)**:  
   Populates the Cell array with initial data, including ships, black hole, and empty cells.
3. **swapState(int x1, int y1, int x2, int y2)**:  
   Swaps the state (BoardStatus) of two cells, ensuring coordinates are within bounds.
4. **traverse(int x, int y, Direction dir)**:  
   Moves a ship in the specified direction, checks for valid moves, and handles collisions with other ships or black holes.
5. **getTurn()**:  
   Returns the current player's turn.
6. **getCell(int x, int y)**:  
   Retrieves the Cell object at the specified coordinates.
7. **getBoardSize()**:  
   Returns the size of the board.
8. **interchangeTurn()**:  
   Switches the turn to the other player.
9. **isOver()**:  
   Checks if the game is over based on player scores.
10. **winnerAnnouncer()**:  
    Returns a string announcing the winner (Player 1 or Player 2).
11. **toString()**:  
    Provides a textual representation of the board, showing the current turn, scores, and cell statuses.

**Class: Cell**

1. **Cell(int x, int y, BoardStatus status)**:  
   Constructor that initializes a cell with its coordinates and status.
2. **setX(int val)**:  
   Sets the x coordinate of the cell.
3. **getX()**:  
   Returns the x coordinate of the cell.
4. **setY(int val)**:  
   Sets the y coordinate of the cell.
5. **getY()**:  
   Returns the y coordinate of the cell.
6. **getStatus()**:  
   Returns the status (BoardStatus) of the cell.
7. **setStatus(BoardStatus st)**:  
   Sets the status (BoardStatus) of the cell.

**Class: Field**

1. **Field()**:  
   Constructor that initializes the field with a gray color and a default number of -1.
2. **getColor()**:  
   Returns the color of the field.
3. **setColor(Color color)**:  
   Sets the color of the field.
4. **getNumber()**:  
   Returns the number associated with the field.
5. **setNumber(int number)**:  
   Sets the number associated with the field.

**Class: Player**

1. **Player()**:  
   Constructor that initializes the player with a score of 0.
2. **getScore()**:  
   Returns the player's current score.
3. **increaseScore()**:  
   Increases the player's score by 1 and returns the updated score.

**Enums**

1. **BoardStatus**:  
   Represents the state of a cell:
   * SHIP\_1: Ship belonging to Player 1.
   * SHIP\_2: Ship belonging to Player 2.
   * BLACKHOLE: A black hole cell.
   * EMPTY: An empty cell.
2. **Direction**:  
   Represents the directions a ship can move:
   * UP, DOWN, LEFT, RIGHT.
3. **PlayerTurn**:  
   Represents the current player's turn:
   * TURN\_1 for Player 1.
   * TURN\_2 for Player 2.

**Connections between the events and event handlers:**

**1. Button Click Events**

**Event Source**: JButton instances in the grid (gameboard).  
**Event Trigger**: Clicking a button (e.g., selecting a cell on the board).  
**Handler**: ActionListener attached to each button. When a button is clicked, the handleShipSelection(int x, int y) method is called.

* **Connection**:

button.addActionListener(e -> handleShipSelection(x, y));

**2. Keyboard Input Events (WASD for Movement)**

**Event Source**: KeyStroke mappings for 'w', 'a', 's', and 'd'.  
**Event Trigger**: Pressing a movement key while the game window is focused.  
**Handler**: ActionMap actions for moveUp, moveDown, moveLeft, moveRight.

The input map associates a key press (e.g., 'w') with an action identifier (e.g., "moveUp"). The action map then maps this identifier to an AbstractAction handler.

* **Connection**:

boardPanel.getInputMap(JPanel.WHEN\_IN\_FOCUSED\_WINDOW).put(KeyStroke.getKeyStroke('w'), "moveUp");

boardPanel.getActionMap().put("moveUp", new AbstractAction() {

@Override

public void actionPerformed(ActionEvent e) {

if (currentX > 0) {

move(Direction.UP);

}

}

});  
  
  
**3. Move Events**

**Event Source**: Logic inside the move(Direction direction) method.  
**Event Trigger**: Triggered by keyboard actions mapped to WASD keys.  
**Handler**: board.traverse(currentX, currentY, direction).

* + **Connection**: Performs the actual ship movement.

private void move(Direction direction) {

board.traverse(currentX, currentY, direction);

refreshAll();

checkGameEnd();

}

1. **Game Over Event**

**Event Source**: checkGameEnd method in BoardGUI.  
**Event Trigger**: When a player wins the game (based on score conditions).  
**Handler**: Shows a dialog and restarts the game.

* **Connection**: Detects if the game has ended.

private void checkGameEnd() {

if (board.isOver()) {

JOptionPane.showMessageDialog(boardPanel, board.winnerAnnouncer(), "Game Finished!", JOptionPane.PLAIN\_MESSAGE);

restartGame();

}

}

1. **Menu Item Events**

**Event Source**: JMenuItem instances in the game menu (e.g., "New Game" options).  
**Event Trigger**: Clicking on a menu item for a new game size or exiting the game.  
**Handler**: Attached ActionListener for each menu item.

* + **Connection**: Reinitializes the game board with the selected size.

sizeMenuItem.addActionListener(e -> {

frame.getContentPane().remove(boardGUI.boardPanel);

boardGUI.initializeGame(boardSize);

frame.getContentPane().add(boardGUI.boardPanel, BorderLayout.CENTER);

frame.pack();

});

1. **Restart Game Event**

**Event Source**: restartGame() method in BoardGUI.  
**Event Trigger**: Called when the game ends and needs to restart.  
**Handler**: Closes the current game window and reopens a new one.

* **Connection**: Restarts the game from scratch with the same board size.

private void restartGame() {

if (parentFrame != null) {

parentFrame.dispose();

}

new GameFrame();

}