File game.cs:

Classes:

1. **Pos Class**
   - **Purpose**: Represents a position on the game board with x and y coordinates.
   - **Members**:
     - int x, y: Coordinates of the position.
   - **Constructor**:
     - Pos(int x, int y): Initializes the position with given coordinates.
2. **Move Class**
   - **Purpose**: Represents a move in the game, consisting of a starting position and a target position.
   - **Members**:
     - Pos from, to: Starting and target positions of the move.
   - **Constructor**:
     - Move(Pos from, Pos to): Initializes the move with given positions.
3. **IsolatedIslandsFinder Class**
   - **Purpose**: Contains methods to find and process isolated islands in a given matrix. An isolated island is a set of cells that either contains empty cells surrounded by inactive cells or contains cells with some disks but there is no move to do using these disks.
   - **Members**:
     - static int Size: Size of the matrix (default is 5).
   - **Methods**:
     - public static List<List<(int, int)>> findIsolatedIslands(int[,] matrix, int[,] active): Finds isolated islands in the matrix where all cells in an island are of the same type.
     - static bool isValid(int[,] matrix, int row, int col, bool[,] visited): Checks if a cell can be included in the Depth-First Search (DFS).
     - static void DFS(int[,] matrix, int row, int col, bool[,] visited, List<(int, int)> island): Performs DFS to explore an island.
     - public static List<List<(int, int)>> FindAllIslands(int[,] M): Finds all islands in the matrix.
4. **Game Class**
   - **Purpose**: Implements the main game logic, including the game board, player turns, move validation, and capturing pieces.
   - **Constants**:
     - public const int Size = 5: Size of the game board.
   - **Members**:

        - public int[,] squares: Holds the pieces on the game board.
        - public int[,] squaresState: Holds the state of each square (active/inactive).
        - public Dictionary<(int, int), int> diction: Stores isolated island cells.
        - public int[] pieces: Tracks the number of pieces owned by each player.
        - public int turn: Indicates the current player's turn.
        - public int moves: Tracks the number of moves made.
        - public int winner: Indicates the winning player.
    - **Constructor**:
        - Game(): Initializes the game board with default pieces and states.
    - **Methods**:
        - public Game clone(): Creates an independent copy of the game state.
        - bool valid(Pos pos): Checks if a position is valid on the board.
        - public bool validMove(Move move): Checks if a move is valid.
        - public List<Move> possibleMoves(): Returns a list of all possible moves for the current player.
        - public bool hasValidMoves(): Checks if the current player has any valid moves.
        - public (int, int) winLine(): Determines the win line based on inactive rows.
        - public bool move(Move m): Updates the game state based on a valid move and returns if a capture was made.
        - public void unmove(Move m, bool wasCapture): Reverses a previous move.
5. **Player Interface**
    - **Purpose**: Defines a strategy for playing the game.
    - **Methods**:
        - Move chooseMove(Game game): Decides what move to make given the game state.
6. **Program Class**
    - **Purpose**: Entry point of the program.
    - **Main Method**:
        - static void Main(): Initializes players and runs the game with a graphical view.

File view_gtk.cs:
Classes:

1. **View Class**
    - **Purpose**: Manages the graphical interface of the game, rendering the game board, handling user interactions, and updating the display based on the game state.
    - **Inheritance**: Inherits from Gtk.Window.
    - **Members**:
        - Game game: Holds the current game state.
        - Player?[] players: Array of players, where each element represents a player or is null for a human player.
        - Move? lastMove: The last move made in the game.
        - Pos? moveFrom: The starting position of a move.
        - bool wasCapture: Indicates if the last move resulted in a capture.
        - Stack<(Move, bool)> undoStack: Stack to keep track of moves for undo functionality.
        - bool undone: Indicates if the last action was an undo.
        - const int Square = 100: The size of each square on the game board in pixels.
        - Pixbuf blackDisk, redDisk: Images for the black and red game pieces.
    - **Constructor**:
        - View(Game game, Player?[] players): Initializes the view with the given game state and players, sets up the window size, event handlers, and title.
    - **Methods**:
        - void setTitle(): Sets the window title based on the player types (human or agent).
        - void move(): Handles the logic for making a move, including choosing the move, updating the game state, and checking for valid moves and game end conditions.
        - void unmove(): Reverses the last move using the undo stack.
        - static RGBA color(string name): Converts a color name to an RGBA object.
        - static void drawLine(Context c, RGBA color, int lineWidth, int x1, int y1, int x2, int y2): Draws a line on the context c with the given parameters.

- static void drawRectangle(Context c, RGBA color, int lineWidth, int x, int y, int width, int height): Draws a rectangle on the context c with the given parameters.
- static void fillRectangle(Context c, RGBA color, int x, int y, int width, int height): Fills a rectangle on the context c with the given parameters.
- static void drawImage(Context c, Pixbuf pixbuf, int x, int y): Draws an image at the specified location on the context c.
- void highlight(Context c, RGBA color, int x, int y): Highlights a square on the board.
- protected override bool OnDrawn(Context c): Handles the drawing of the game board and pieces. It is called whenever the window needs to be redrawn.
- bool gameOver(): Checks if the game is over and quits the application if it is.
- protected override bool OnButtonPressEvent(EventButton e): Handles mouse button press events to allow user interaction with the game board.
- protected override bool OnDeleteEvent(Event ev): Handles the window close event to quit the application.
- public static void run(Game game, Player?[] players): Initializes the GTK application and runs the game view.

File my_agent.cs:

Classes:

1. MyAgent Class
   - Purpose: Represents an automated player that uses a Minimax algorithm with alpha-beta pruning to decide on the best move in the game.
   - Inheritance: Inherits from Player.
   - Methods:
     - bool underAttack(Game game, int x, int y): Checks if a piece at the specified position (x, y) is under attack.
     - bool nextMoveWins(Game game): Determines if the current player can win on their next move.
     - int eval(Game game): Evaluates the game position and returns a score indicating the favorability of the position for the current player.
     - int minimax(Game game, int depth, int alpha, int beta, out Move bestMove): Implements the Minimax algorithm with alpha-beta pruning to determine the best move.
     - public Move chooseMove(Game game): Selects the best move for the current player.