## **Klotski (华容道) 项目结构和方法详解**

### **项目根目录结构**

SUSTech-JavaProject-KlotskiApp-master/  
├── .idea/ # IntelliJ IDEA 项目配置文件  
│ ├── misc.xml  
│ ├── modules.xml  
│ └── vcs.xml  
├── src/ # 源代码目录  
│ ├── Main.java # 程序入口类  
│ ├── css/ # CSS 样式文件目录  
│ │ └── wave.css  
│ ├── game/ # 游戏逻辑相关类  
│ │ ├── AboutGame.java  
│ │ ├── AutoSolve.java  
│ │ ├── Direction.java  
│ │ ├── GameLogic.java  
│ │ ├── LevelManager.java  
│ │ └── Solver.java  
│ ├── model/ # 数据模型相关类  
│ │ ├── Block.java  
│ │ ├── Board.java  
│ │ ├── GameState.java  
│ │ └── Level.java  
│ ├── ui/ # 用户界面相关类  
│ │ ├── KlotskiApp.java  
│ │ └── controls/ # 自定义 UI 控件  
│ │ ├── WavePasswordConfirm.java  
│ │ ├── WavePasswordField.java  
│ │ └── WaveTextField.java（这几个是我随便加的）  
│ ├── user/ # 用户管理相关类  
│ │ ├── User.java  
│ │ └── UserManager.java  
│ └── util/ # 工具类  
│ └── GameFileManager.java  
└── Klotski.iml # IntelliJ IDEA 模块文件 (从 modules.xml 推断)

### **各 Java 文件详解**

#### **1. src/Main.java**

* **包名:** (默认包)
* **类名:** Main
* **主要功能:** 程序的入口点。
* **主要方法:**
  + public static void main(String[] args): 启动 JavaFX 应用程序，调用 KlotskiApp.launch(KlotskiApp.class, args)。

#### **2. src/game/AboutGame.java**

* **包名:** game
* **类名:** AboutGame
* **主要功能:** 提供一个显示“关于游戏”信息的窗口。
* **主要方法:**
  + public AboutGame(): 构造函数，初始化并显示“关于游戏”的 Stage (窗口)。
  + public void show(): 显示“关于游戏”窗口。

#### **3. src/game/AutoSolve.java**

* **包名:** game
* **类名:** AutoSolve
* **主要功能:** 实现游戏的自动求解功能。它可能使用某种搜索算法（如 BFS）来找到解决方案。
* **主要方法:**
  + public AutoSolve(Board initialBoard): 构造函数，接收初始棋盘状态。
  + public List<Board> solve(): 执行求解算法，返回从初始状态到目标状态的棋盘状态序列。
  + private List<Board> reconstructPath(Map<Board, Board> predecessors, Board targetState):根据前驱节点映射重构解决方案路径。
  + public boolean isSolvable(): 判断当前棋盘是否有解。

#### **4. src/game/Direction.java**

* **包名:** game
* **类名:** Direction (枚举类型)
* **主要功能:** 定义棋子可以移动的四个方向：上、下、左、右。
* **枚举常量:**
  + UP
  + DOWN
  + LEFT
  + RIGHT
* **主要方法:**
  + public int getDeltaX(): 获取该方向在 X 轴上的变化量。
  + public int getDeltaY(): 获取该方向在 Y 轴上的变化量。

#### **5. src/game/GameLogic.java**

* **包名:** game
* **类名:** GameLogic
* **主要功能:** 处理游戏的核心逻辑，如移动棋子、检查游戏是否胜利、加载关卡等。
* **主要方法:**
  + public GameLogic(Level level): 构造函数，根据指定的关卡初始化游戏。
  + public Board getBoard(): 获取当前棋盘状态。
  + public boolean moveBlock(Block block, Direction direction): 尝试向指定方向移动一个棋子。如果移动成功，返回 true，否则返回 false。
  + public boolean moveBlock(int blockId, Direction direction): (推测) 根据棋子 ID 移动棋子。
  + public boolean isWin(): 检查当前是否达到胜利条件。
  + public void loadLevel(Level level): 加载新的游戏关卡。
  + public void resetGame(): 重置当前关卡到初始状态。
  + public int getSteps(): 获取当前移动步数。
  + private boolean isValidMove(Block block, int newX, int newY): 检查棋子移动到新位置是否有效（是否越界、是否与其他棋子重叠）。

#### **6. src/game/LevelManager.java**

* **包名:** game
* **类名:** LevelManager
* **主要功能:** 管理游戏关卡，包括加载预设关卡、保存/加载自定义关卡。
* **主要方法:**
  + public LevelManager(): 构造函数。
  + public Level getLevel(int levelNumber): 获取指定编号的预设关卡。
  + public List<Level> getAllLevels(): 获取所有预设关卡。
  + public void saveCustomLevel(Level level, String filePath): (推测) 将自定义关卡保存到文件。
  + public Level loadCustomLevel(String filePath): 从文件加载自定义关卡。
  + private void loadDefaultLevels(): 加载内置的默认关卡数据。

#### **7. src/game/Solver.java**

* **包名:** game
* **类名:** Solver
* **主要功能:** (与 AutoSolve 类似或为其一部分) 提供解决华容道谜题的算法。这是一个更通用的解谜器接口或实现。
* **主要方法:**
  + public List<GameState> findPath(GameState initialState): 寻找从初始状态到解决状态的路径。
  + private List<GameState> bfs(GameState initialState): 使用广度优先搜索 (BFS) 算法。
  + private List<GameState> getNeighbors(GameState state): 获取给定状态所有可能的下一步状态。

#### **8. src/model/Block.java**

* **包名:** model
* **类名:** Block
* **主要功能:** 表示游戏中的一个棋子。包含棋子的位置、大小（宽度和高度）、类型（例如，曹操、兵、将等）和唯一标识。
* **主要属性 (推测):**
  + id (int): 棋子唯一标识。
  + x, y (int): 棋子左上角在棋盘上的坐标。
  + width, height (int): 棋子的宽度和高度（以格子为单位）。
  + name (String) or type (enum): 棋子的名称或类型。
* **主要方法:**
  + 构造函数: public Block(int id, int x, int y, int width, int height, String name)
  + Getter 和 Setter 方法: 用于获取和设置棋子的各个属性。
  + public void move(Direction direction): 根据方向更新棋子坐标，但不检查移动有效性。
  + public boolean occupies(int r, int c):判断棋子是否占据棋盘上的 (r, c) 位置。
  + public Block copy(): 创建并返回当前棋子的一个副本。
  + @Override public boolean equals(Object o): 比较两个棋子是否相同（通常基于 ID）。
  + @Override public int hashCode(): 为棋子生成哈希码。

#### **9. src/model/Board.java**

* **包名:** model
* **类名:** Board
* **主要功能:** 表示游戏棋盘的状态。包含棋盘的尺寸（行数和列数）以及棋盘上所有棋子的集合。
* **主要属性 (推测):**
  + rows, cols (int): 棋盘的行数和列数 (通常是 5x4)。
  + blocks (List): 棋盘上所有棋子的列表。
  + grid (int[][] or Block[][]): (推测) 一个二维数组，表示棋盘上每个格子的状态（例如，被哪个棋子占据，或者为空）。
* **主要方法:**
  + 构造函数: public Board(int rows, int cols, List<Block> initialBlocks)
  + public List<Block> getBlocks(): 获取棋盘上所有棋子。
  + public Block getBlockAt(int r, int c): 获取指定位置的棋子，如果该位置为空则返回 null。
  + public boolean isOccupied(int r, int c): 判断指定位置是否被占据。
  + public boolean canMove(Block block, Direction direction): 判断指定的棋子是否可以向指定方向移动（检查边界和碰撞）。
  + public Board moveBlock(Block block, Direction direction): 移动棋子并返回一个新的棋盘状态 (immutable)；或者直接修改当前棋盘状态 (mutable)。
  + public boolean isGoalState(): 判断当前棋盘是否为胜利状态（例如，曹操棋子在目标位置）。
  + public Board copy(): 创建并返回当前棋盘状态的一个深拷贝。
  + @Override public boolean equals(Object o): 比较两个棋盘状态是否相同（所有棋子位置和类型都相同）。
  + @Override public int hashCode(): 为棋盘状态生成哈希码，常用于在集合中存储和查找状态（例如在 BFS 算法中）。
  + public void addBlock(Block block): 向棋盘添加棋子。
  + public void removeBlock(Block block):从棋盘移除棋子。
  + public String toString(): 返回棋盘状态的字符串表示，方便调试。

#### **10. src/model/GameState.java**

* **包名:** model
* **类名:** GameState
* **主要功能:** 表示游戏的一个完整状态，通常包含当前的棋盘布局 (Board) 以及可能的其他信息，如到达此状态的步数或前一个状态。这在搜索算法中非常有用。
* **主要属性 (推测):**
  + board (Board): 当前的棋盘状态。
  + previousState (GameState): 到达当前状态的前一个状态 (用于路径重构)。
  + move (String or Object): 到达此状态所执行的移动。
  + depth or steps (int): 从初始状态到当前状态的步数。
* **主要方法:**
  + 构造函数: public GameState(Board board, GameState previousState, ...)
  + Getter 方法: 获取状态的各个属性。
  + public List<GameState> getPossibleNextStates():生成所有可能的下一个游戏状态。
  + @Override public boolean equals(Object o): 比较两个游戏状态是否相同 (通常基于棋盘状态)。
  + @Override public int hashCode(): 为游戏状态生成哈希码 (通常基于棋盘状态的哈希码)。

#### **11. src/model/Level.java**

* **包名:** model
* **类名:** Level
* **主要功能:** 定义一个游戏关卡，包括关卡的名称或编号，以及初始的棋子布局。
* **主要属性 (推测):**
  + levelName (String) or levelNumber (int): 关卡名称或编号。
  + initialBlocks (List): 该关卡的初始棋子配置。
  + targetBlockId (int): 目标棋子 (曹操) 的 ID。
  + targetX, targetY (int): 目标棋子需要到达的目标位置。
  + rows, cols (int):该关卡棋盘的尺寸。
* **主要方法:**
  + 构造函数: public Level(String levelName, List<Block> initialBlocks, ...)
  + Getter 方法: 获取关卡的各个属性。
  + public Board createInitialBoard(): (推测) 根据关卡配置创建一个初始的 Board 对象。

#### **12. src/ui/KlotskiApp.java**

* **包名:** ui
* **类名:** KlotskiApp
* **父类:** javafx.application.Application
* **主要功能:** JavaFX 应用程序的主类，负责创建和管理用户界面 (UI)，包括游戏棋盘的显示、按钮、菜单等。
* **主要方法:**
  + public static void main(String[] args): (通常不直接使用，而是通过 Main.java 启动)
  + @Override public void start(Stage primaryStage): JavaFX 应用程序的入口点，在此设置主场景 (Scene) 和舞台 (Stage)。
  + private Scene createLoginScene(): 创建登录界面。
  + private Scene createGameScene(Level level): 创建游戏主界面，包括棋盘、控制按钮等。
  + private void drawBoard(Board board): 在 UI 上绘制当前棋盘状态。
  + private void handleBlockClick(Block block) or private void handleKeyPress(KeyEvent event): 处理用户输入（点击棋子、键盘操作）。
  + private void updateGameView(): 当游戏状态改变时更新 UI 显示。
  + private void showWinMessage(): 当游戏胜利时显示提示信息。
  + private void showLoseMessage(): (如果适用) 游戏失败时的提示。
  + private void setupMenu(Stage primaryStage):设置应用程序的菜单栏 (文件、编辑、帮助等)。
  + 事件处理方法 (例如 onNewGameClicked, onSolveClicked, onUndoClicked, onSaveGameClicked, onLoadGameClicked 等)。

//后面三个是我自己随便加的，先不管，到时候你界面ui设计的时候可以自己更改

#### **13. src/ui/controls/WavePasswordConfirm.java**

* **包名:** ui.controls
* **类名:** WavePasswordConfirm
* **父类 :** javafx.scene.layout.VBox 或 javafx.scene.layout.GridPane
* **主要功能:** 一个自定义 UI 控件，用于密码输入和确认。通常包含两个密码输入框，并验证两次输入的密码是否一致。
* **主要方法:**
  + 构造函数: public WavePasswordConfirm()
  + public String getPassword(): 获取输入的密码 (如果两次输入一致)。
  + public boolean isValid(): 检查两次输入的密码是否一致且符合密码策略。
  + public PasswordField getPasswordField(): (推测) 获取第一个密码输入框。
  + public PasswordField getConfirmPasswordField(): (推测) 获取确认密码输入框。

#### **14. src/ui/controls/WavePasswordField.java**

* **包名:** ui.controls
* **类名:** WavePasswordField
* **父类:** javafx.scene.control.PasswordField 或对其进行封装的自定义控件。
* **主要功能:** 一个自定义的密码输入框，可能带有一些特殊的样式或验证逻辑 (例如 "wave" 效果)。
* **主要方法:**
  + 构造函数: public WavePasswordField()
  + 可能包含自定义的验证逻辑或样式设置方法。

#### **15. src/ui/controls/WaveTextField.java**

* **包名:** ui.controls
* **类名:** WaveTextField
* **父类 (推测):** javafx.scene.control.TextField 或对其进行封装的自定义控件。
* **主要功能:** 一个自定义的文本输入框，可能带有一些特殊的样式或验证逻辑 (例如 "wave" 效果，可能与 wave.css 文件相关)。
* **主要方法:**
  + 构造函数: public WaveTextField()
  + 可能包含自定义的验证逻辑或样式设置方法。

#### **16. src/user/User.java**

* **包名:** user
* **类名:** User
* **主要功能:** 表示一个用户，包含用户名、密码（可能是哈希过的）以及用户的游戏进度等信息。
* **主要属性 (推测):**
  + username (String)
  + hashedPassword (String)
  + highestLevelCompleted (int)
  + gameStates (Map<String, GameState>): (推测) 保存的用户游戏状态。
* **主要方法:**
  + 构造函数: public User(String username, String password)
  + Getter 和 Setter 方法。
  + public boolean checkPassword(String password): (推测) 验证输入的密码是否正确。
  + public void saveProgress(Level level, int steps): (推测) 保存用户在某个关卡的进度。

#### **17. src/user/UserManager.java**

* **包名:** user
* **类名:** UserManager
* **主要功能:** 管理用户信息，包括用户注册、登录、加载和保存用户信息到文件。
* **主要方法:**
  + public UserManager(): 构造函数，可能从文件加载现有用户数据。
  + public boolean registerUser(String username, String password): 注册新用户。
  + public User loginUser(String username, String password): 用户登录，成功返回 User 对象，失败返回 null。
  + public void saveUsers(): 将所有用户信息保存到文件 (例如 JSON 或序列化对象)。
  + private void loadUsers(): 从文件加载用户信息。
  + public User getCurrentUser(): 获取当前登录的用户。
  + public void logoutUser(): 用户登出。

#### **18. src/util/GameFileManager.java**

* **包名:** util
* **类名:** GameFileManager
* **主要功能:** 提供文件操作的工具方法，例如保存和加载游戏状态、用户数据、关卡数据等。
* **主要方法:**
  + public static void saveGame(GameState gameState, String filePath): 将游戏状态保存到文件。
  + public static GameState loadGame(String filePath): 从文件加载游戏状态。
  + public static void saveUsers(List<User> users, String filePath): 保存用户列表到文件。
  + public static List<User> loadUsers(String filePath):从文件加载用户列表。
  + public static void saveLevel(Level level, String filePath): 保存关卡数据到文件。
  + public static Level loadLevel(String filePath): 从文件加载关卡数据。
  + 可能包含通用的读写 JSON、XML 或二进制文件的方法。

### **CSS 文件**

* **src/css/wave.css**: 这个 CSS 文件很可能用于定义 WavePasswordField 和 WaveTextField 等自定义控件的波浪动画效果或其他视觉样式。