# ChessBot README

# ChessBot  
  
ChessBot is a Python-based chess engine and interactive game environment designed with a strong emphasis on \*\*clean architecture\*\*, \*\*modularity\*\*, and \*\*extensibility\*\*.   
It separates \*\*movement logic\*\*, \*\*piece/rank definitions\*\*, and \*\*board management\*\*, enabling both human-vs-human and AI-vs-human gameplay.  
  
---  
  
## 🎯 Goals  
- Maintain a clear separation between \*\*game rules\*\* and \*\*board state\*\*.  
- Support \*\*AI-driven moves\*\* through destination evaluation logic.  
- Enforce \*\*move legality\*\* and \*\*capture rules\*\* through centralized validation.  
- Enable \*\*transaction-safe game updates\*\* for undo/redo and rollback.  
- Provide a \*\*Pygame-powered interface\*\* for drag-and-drop piece movement.  
  
---  
  
## 🏛 Architecture Overview  
  
The design is built around \*\*layered responsibilities\*\*:  
  
### \*\*Core Entities\*\*  
- \*\*`ChessPiece`\*\*  
 - Holds unique piece ID, immutable `rank`, and `position\_history`.  
 - Maintains `captured` state.  
 - Enforces that the `rank` matches its piece type at construction.  
 - Implements capture handling through a `captor` reference.  
  
- \*\*`Rank`\*\*  
 - Encapsulates movement rules via `MovementStrategy`.  
 - Includes methods like `path\_to\_coordinate\_exists()` and `line\_fits\_definition()`.  
 - Special ranks (e.g., `Pawn`, `King`) implement `RankPromotable` for promotions.  
  
- \*\*`Board`\*\*  
 - Stores and manages `Square` objects, each holding coordinates and optional `ChessPiece`.  
 - Performs \*\*bounds checking\*\* and manages piece movement/capture.  
 - Does not contain game logic — purely spatial awareness.  
  
- \*\*`Player`\*\*  
 - Identified by ID, name, and `Team`.  
 - Team handles prisoners (captured enemy pieces).  
  
- \*\*`CaptureRecord`\*\*  
 - Records the prisoner, captor, and capture square for game history.  
  
---  
  
### \*\*Movement & Strategy\*\*  
Movement logic is encapsulated in \*\*movement strategy classes\*\*:  
- `BishopMovement` – diagonal lines.  
- `RookMovement` – vertical/horizontal lines.  
- `KnightMovement` – L-shaped jumps.  
- `PawnMovement` – forward movement with capture diagonals and promotion rules.  
- `QueenMovement` – union of bishop and rook logic.  
- `KingMovement` – one-square radius, castling rules.  
  
All movements use \*\*motion definitions\*\* (`DiagonalDefinition`, `VerticalDefinition`, etc.) to check if a line is valid.  
  
---  
  
### \*\*Destination Selection\*\*  
The `DestinationSelector` rates legal moves:  
- Prioritizes highest-value enemy captures.  
- Falls back to random selection when no valuable targets exist.  
- Risk assessment may be added in the future.  
  
---  
  
## ✅ Validation & Transactions  
- \*\*Move Validation\*\*  
 - Prevents friendly fire (cannot capture your own pieces).  
 - Ensures destination is within rank’s movement definition.  
 - Verifies path clearance for sliding pieces (rook, bishop, queen).  
 - Confirms legality before updating `position\_history`.  
  
- \*\*Transaction Management\*\*  
 - Every move is wrapped in a transaction-like structure for safe rollback.  
 - `TransactionResult` stores method name, outcome, and error details.  
 - Supports a one-move-per-player undo system.  
  
---  
  
## 🖥 Pygame UI  
- Drag-and-drop piece interaction.  
- Visual cues for piece type (color-coded shapes).  
- Highlighted legal moves (planned feature).  
- Adjustable board dimensions via config constants (`CELL\_PX`, `BORDER\_PX`, `SCREEN\_WIDTH`, `SCREEN\_HEIGHT`).  
  
---  
  
## 📂 Project Structure  
```  
chessbot/  
├── chess/  
│ ├── board/  
│ │ ├── chess\_board.py  
│ │ ├── square.py  
│ │ └── coordinate.py  
│ ├── pieces/  
│ │ ├── chess\_piece.py  
│ │ ├── ranks/  
│ │ │ ├── bishop.py  
│ │ │ ├── rook.py  
│ │ │ ├── pawn.py  
│ │ │ ├── king.py  
│ │ │ └── ...  
│ │ ├── movement/  
│ │ │ ├── bishop\_movement.py  
│ │ │ ├── rook\_movement.py  
│ │ │ └── ...  
│ ├── players/  
│ │ ├── player.py  
│ │ └── team.py  
│ ├── common/  
│ │ ├── config.py  
│ │ └── logging\_setup.py  
│ └── game/  
│ ├── capture\_record.py  
│ ├── destination\_selector.py  
│ └── transaction\_result.py  
├── main.py  
└── README.md  
```  
  
---  
  
## 🛠 Installation Instructions  
  
To get started, first clone this repository:  
  
```bash  
git clone https://github.com/yourusername/chessbot.git  
cd chessbot  
```  
  
Next, run the bootstrap script:  
  
```bash  
python bootstrap.py  
```  
  
This script will:   
- Create a `.venv` virtual environment.   
- Install required dependencies (e.g., `pygame`).   
- Set up necessary folders (`assets`, `fonts`, `levels`, `src`).   
- Create `main.py` if it’s missing.   
- Display instructions for activating the virtual environment and running the game.   
  
If everything goes well, you’ll see a message at the end showing exactly how to launch ChessBot.  
  
---  
  
### 📦 Manual Setup (Alternative)  
If you prefer to set things up manually or run into issues with the bootstrap script:  
  
```bash  
python -m venv .venv  
```  
  
Activate the virtual environment:  
  
- \*\*macOS/Linux\*\*:  
 ```bash  
 source .venv/bin/activate  
 ```  
  
- \*\*Windows (cmd)\*\*:  
 ```cmd  
 .venv\Scripts\activate.bat  
 ```  
  
- \*\*Windows (PowerShell)\*\*:  
 ```powershell  
 .venv\Scripts\Activate.ps1  
 ```  
  
Install dependencies:  
  
```bash  
pip install -r requirements.txt  
```  
  
---  
  
### ▶ Running the Game  
Once dependencies are installed, launch ChessBot with:  
  
```bash  
python main.py  
```  
  
If using \*\*PyCharm\*\*:   
- Open the project.   
- Create or select a virtual environment from the interpreter settings.   
- Install dependencies from `requirements.txt`.   
- Run `main.py` from the project tree.  
  
---  
  
### ⚙ Python Version  
ChessBot requires \*\*Python 3.11 or later\*\*.  
  
---  
  
### 📌 Adding New Dependencies  
If you install new packages during development:  
  
```bash  
pip install <package\_name>  
pip freeze > requirements.txt  
```  
  
---  
  
## 🔮 Planned Features  
- AI opponent with risk-aware move selection.  
- Network multiplayer.  
- Timed matches.  
- PGN (Portable Game Notation) export.  
- Move hints and visual overlays.  
  
---  
  
## 📜 License  
MIT License. See [LICENSE](LICENSE) for details.