Khreibaga Zero: Technical Specification

A Reference Manual for AlphaZero-Style Reinforcement Learning

System Architecture & Implementation Guide

February 6, 2026

Abstract

This document serves as the authoritative reference for developing an AlphaZero-based Reinforcement Learning agent for the game of **Khreibaga** (Mauritanian Alquerque). It provides a rigorous definition of the game environment, including edge cases for "Flying Kings" and "Majority Capture." Furthermore, it details the state-space representation, the source-target action space ($N = 625$), and the specific modifications required for the Monte Carlo Tree Search (MCTS) algorithm to handle multi-step compulsory captures.

Contents

1 Part I: The Environment Specification (Game Logic)

The accuracy of the RL agent depends entirely on the fidelity of the environment. The "Rule Engine" must strictly enforce every constraint defined below.

1.1 1.1 Board Geometry & Initialization

- **Topology:** A 5×5 grid of intersecting lines.
- **Topology:** A 5×5 grid of points forming 16 internal squares.
- **Connectivity Constraints:**
  - Orthogonal connections exist between all adjacent horizontal and vertical points.
  - **Diagonal connections are alternating.**
  - A point (r, c) has diagonal connections if and only if $(r + c)$ is even (assuming a checkerboard parity of the squares themselves, or hard-coded based on the visual pattern).
  - *Visual Rule:* Only squares that correspond to "dark squares" on a chessboard (or vice versa depending on indexing) contain 'X' crosses. Squares between them are empty of diagonals.
- **Coordinate System:** We utilize a 0-indexed integer system for efficiency.
  - Rows: 0 (Bottom) to 4 (Top).
  - Columns: 0 (Left) to 4 (Right).
  - **Flat Indexing:** $SquareID = Row \times 5 + Col$. (Range $0 \dots 24$).
- **Initial Setup:**
  - **Total Points:** 25.
  - **Piece Count:** 12 Black (Player 1), 12 White (Player 2).
  - **Configuration:** All points filled *except* the center point (Index 12 / C3).
  - **Player 1 (Black):** Occupies Rows 0 and 1 fully, plus indices 10 and 11 (left side of Row 2).
  - **Player 2 (White):** Occupies Rows 3 and 4 fully, plus indices 13 and 14 (right side of Row 2).

1.2 1.2 Piece Mechanics

1.2.1 Men (Unpromoted)

- **Simple Move:** One step to an adjacent empty point.
- **Directionality:** Strictly **Forward** or **Forward-Diagonal** relative to the player's side. (Black moves $Row+$, White moves $Row-$).
- **Backward Movement:** Illegal for simple moves. Permitted *only* during capture sequences.

- **Promotion:** A Man is immediately promoted to a **King** if it ends its turn on the opponent's back rank (Row 4 for Black, Row 0 for White).
  - *Note:* If a Man reaches the promotion rank *during* a multi-capture sequence but must continue jumping out of the rank, it does **not** promote. Promotion only occurs when the piece *stops* on the final rank.

1.2.2 Kings (Promoted)

- **Type:** "Flying Kings" (similar to International Draughts).
- **Movement:** Can move any distance along a clear line (orthogonal or diagonal) to an empty square.
- **Obstructions:** Cannot jump over friendly pieces. Cannot move through enemy pieces without capturing.

1.3 1.3 Capturing Logic (The Critical Path)

In Khreibaga, capturing is **compulsory** and governed by strict priority rules.

1. **Compulsory Capture:** If any legal capture exists on the board, non-capturing moves are illegal.
2. **Man Capture:** Leaps over an adjacent enemy to the empty square immediately behind. Can capture in **any direction** (360 degrees).
3. **King Capture (Long Leap):** Leaps over a single enemy piece along a line. The King may land on **any** empty square beyond the captured piece, provided the path is clear.
4. **Immediate Removal:** The captured piece is removed from the board **instantly**.
   - *Implication:* A piece may cross the same square twice in one sequence if the removal clears the path.
5. **The Majority Capture Rule:**
   - If multiple capture sequences are available, the player **must** choose a sequence that captures the maximum number of pieces.
   - *Example:* If Option A captures 2 pieces and Option B captures 3 pieces, Option A is strictly illegal.
   - *Tie-Breaking:* If two sequences both capture the maximum number (e.g., both 3), the player may choose either.

1.4 1.4 Terminal Conditions

The game ends when:

1. **Elimination:** One player has 0 pieces remaining.
2. **Stalemate:** The current player has pieces but no legal moves (immobilized). This counts as a loss for the immobilized player.
3. **Draws:**
    - **50-Move Rule:** 50 half-moves (25 turns each) pass without a capture or a Man moving.
    - **3-Fold Repetition:** The exact same board state occurs 3 times.

2 Part II: Reinforcement Learning Specification

This section defines the input/output interfaces for the Neural Network.

2.1 2.1 State Representation (Input Tensor)

The board state is converted into a tensor of shape $5 \times 5 \times C$. We recommend $C = 7$ planes to capture full context.

Canonical Form: Before inputting to the network, the board is always oriented such that the **current player** is moving from "bottom to top." This allows the network to learn a single set of features regardless of color.

Plane	Feature	Description
0	P1 Pieces (Men)	1 if current player's Man is present, else 0.
1	P1 Pieces (Kings)	1 if current player's King is present, else 0.
2	P2 Pieces (Men)	1 if opponent's Man is present, else 0.
3	P2 Pieces (Kings)	1 if opponent's King is present, else 0.
4	Repetitions	1 if this board configuration has appeared before (to avoid cycles).
5	Colour Plane	All 0s for White, All 1s for Black (Optional, helps if dynamics differ).
6	Move Count	Normalized scalar (t/MAX_STEPS) to sensing draw proximity.

Table 1: Input Feature Planes

2.2 2.2 Action Space (Output Vector)

We utilize a **Source-Target (From-To)** discrete action space. This is the optimal representation for "Flying Kings" where the landing square is variable.

- **Total Actions:** $25 \times 25 = 625$ logits.
- **Mapping Logic:**

$$\text{Action Index} = (\text{SourceSquare} \times 25) + \text{TargetSquare}$$

- **SourceSquare:** Integer $0 \dots 24$.
- **TargetSquare:** Integer $0 \dots 24$.

2.2.1 Interpretation of Indices

The neural network outputs a probability distribution over these 625 indices.

- **Index 0 ($0 \to 0$):** Illegal (piece cannot move to itself).
- **Index 6 ($0 \to 6$):** Move from $(0, 0)$ to $(1, 1)$.
- **Index 24 ($0 \to 24$):** Move from $(0, 0)$ to $(4, 4)$ (Long diagonal jump).

Ambiguity Resolution: The Action Space does not distinguish between a "Move" and a "Capture." The Environment resolves this based on the board state. If a piece at A moves to B and jumps an enemy, it is a capture. If the path is clear, it is a move.

2.3 2.3 Handling Multi-Step Actions (Chains)

This is the most critical implementation detail for Khreibaga.

The Atomic Action Principle

The Neural Network does not output a full sequence (e.g., A1-C3-E5). It outputs one atomic hop at a time. The environment maintains the "Turn State."

The Sequence Flow:

1. **State S_0:** Player 1 has a multi-jump available starting at square 0.
2. **Masking:** The environment masks ALL moves except the valid jumps from square 0.
3. **NN Action:** Agent picks $0 \to 12$.
4. **Environment:**
    - Executes jump $0 \to 12$.
    - Removes captured piece.
    - Checks: "Can piece at 12 capture again?" $\to$ YES.
    - **Does NOT switch player.**
    - Returns State $S_{0.5}$ (Intermediate state).
5. **State $S_{0.5}$:**
6. **Masking:** Mask is zeros everywhere except valid jumps starting from square 12.
7. **NN Action:** Agent picks $12 \to 24$.
8. **Environment:**
    - Executes jump.
    - Checks: "Can piece at 24 capture again?" $\to$ NO.
    - Switches player.
    - Returns State S_1 (New Turn).

3 Part III: Implementation Algorithms

3.1 3.1 The Legal Move Generator (With Majority Rule)

This function must be implemented in the Environment class. It is the gatekeeper for the Neural Network.

Algorithm 1 Legal Move Mask Generation

```
function GetActionMask(BoardState)
    Mask ← Zeros(625)
    AllMoves ← []
    CaptureChains ← []
                                        ▷ Phase 1: Detect all possible moves for current player
    for piece in CurrentPlayerPieces do
        chains ← FindMaxCaptureChains(piece, BoardState)
        if chains is not empty then
            CaptureChains.extend(chains)
        else
            AllMoves.extend(FindSimpleMoves(piece))
        end if
    end for
                                                    ▷ Phase 2: Apply Majority Rule Priority
    if CaptureChains is not empty then
        MaxLen ← max(len(c) for c in CaptureChains)
        BestChains ← [c for c in CaptureChains if len(c) == MaxLen]
        for chain in BestChains do
            FirstHop ← chain[0]                  ▷ Extract only the immediate next atomic move
            Index ← (FirstHop.src × 25) + FirstHop.dst
            Mask[Index] ← 1
        end for
    else                                         ▷ No captures available, allow simple moves
        for move in AllMoves do
            Index ← (move.src × 25) + move.dst
            Mask[Index] ← 1
        end for
    end if
    return Mask
end function
```

3.2 3.2 Neural Network Architecture

Given the 5×5 board, a large ResNet (like in AlphaGo) is overkill. We define a compact architecture.

- **Input:** $(B, 7, 5, 5)$
- **Convolutional Block:** Conv2d(64 filters, 3×3), BN, ReLU.

- **Residual Tower:** 6 Blocks.
  - Each Block: Conv(64) → BN → ReLU → Conv(64) → BN → Skip Connection → ReLU.
- **Policy Head:**
  - Conv(2 filters, 1×1) → BN → ReLU.
  - Flatten → Linear($50 \to 625$).
  - Output: Logits (Do not apply Softmax here; Softmax is part of the loss function).
- **Value Head:**
  - Conv(1 filter, 1×1) → BN → ReLU.
  - Flatten → Linear($25 \to 64$) → ReLU → Linear($64 \to 1$).
  - Output: Tanh (Range -1 to 1).

3.3 3.3 The Training Loop

The self-play cycle follows the standard AlphaZero definition:

1. **Self-Play:** The best current model plays against itself.
   - Execute 200 MCTS simulations per turn.
   - **Exploration:** For the first 10 moves, sample actions from the MCTS visit distribution (Temperature $\tau = 1$).
   - **Exploitation:** After 10 moves, select the action with the highest visit count ($\tau \to 0$).
   - Store tuples $(State, Policy_\pi, Reward_z)$.
2. **Optimization:**
   - Sample a batch of games.
   - Loss: $l = (z - v)^2 - \pi^T \log(p) + c\|\theta\|^2$.
   - Update weights using SGD or Adam.
3. **Evaluation:**
   - Every 50 training steps, pit New Model vs Old Model (50 games).
   - If New Model wins $> 55\%$ of games, it becomes the new data generator.

4 Part IV: Specific Implementation Difficulties

4.1 4.1 Flying King Branching Factor

Problem: A King capturing a piece might have 3 different squares it can land on behind the enemy. **Solution:** The "Source-Target" action space naturally handles this. $A1 \to C3$ and $A1 \to D4$ are distinct indices. The Legal Move Generator must verify which landing spots are valid (i.e., path is clear up to the jump, and destination is empty).

4.2 4.2 Cyclic States

Problem: Players may shuffle pieces back and forth to avoid losing. **Solution:** 1. Include the "Repetition Plane" in the input tensor. 2. In the Environment, store a hash of every board state in the current game history. 3. If a move results in a hash that appeared 2 times previously, declare the game a Draw immediately.

4.3 4.3 First-Move Advantage

Problem: On a 5×5 board, the first player (Black) likely has a strong theoretical advantage. **Solution:** During training, strictly enforce the temperature parameter $\tau = 1$ for the opening phase. This forces the agent to explore suboptimal openings, preventing it from overfitting to a single "winning" line that might actually be brittle.