Proyecto Mancala - IA jeje

Adrian Eduardo Ruiz Cerquera[a], Leonardo Velázquez Colin[a], Gustavo Ortiz Loaiza[a], Tomas Henrique Martinez Gomez[a]

[a] *Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

Este documento detalla el desarrollo de un jugador de inteligencia artificial para el juego de mesa africano Mancala. Se realiza un análisis exhaustivo del juego, incluyendo su origen, componentes, reglas de movimiento, condiciones de captura, obtención de turnos extra y finalización del juego. Se exploran estrategias para un jugador sintético, como la maximización de turnos y capturas, y la adaptación al estado actual de la partida.

Keywords: algoritmo, mancala, inteligencia artificial, juego por turnos

1. Análisis del problema

El Mancala es un juego de mesa de origen africano que cuenta con un tablero con varios receptáculos u hoyos, donde se debe mover fichas durante el juego, también se le conoce como Kalaha o Mangala. El objetivo principal del juego es mover las piezas alrededor del tablero e intentar capturar el mayor número de fichas del oponente. [1]

Se dice que su origen proviene de África pero también de la época arábica, por lo tanto, podemos decir que es una mezcla de culturas las que se encuentran en este juego. La palabra "Mancala" significa transferir o movimiento lo cual se atribuye a la forma de jugarlo. [1]

Se compone de:

- Sea $P \in \mathbb{N}_0^{2\times 6}$ la matriz de hoyos (*pits*), donde:
  - $P_{i,j}$ representa el número de semillas en el hoyo j de la fila i.
  - $\mathbb{N}_0$ denota los enteros no negativos: $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.
- Sea $S = (S_1, S_2) \in \mathbb{N}_0^2$ el vector de almacenes, donde:

Email addresses: ruiz-adrian@javeriana.edu.co (Adrian Eduardo Ruiz Cerquera), levelazquez@javeriana.edu.co (Leonardo Velázquez Colin), g.ortizl@javeriana.edu.co (Gustavo Ortiz Loaiza), tomashmartinez@javeriana.edu.co (Tomas Henrique Martinez Gomez)

  - S_1 es el almacén del jugador 1.
  - S_2 es el almacén del jugador 2.

- Sea $J = \{1, 2\}$ el conjunto de jugadores. El juego se desarrolla en turnos alternados, aunque puede haber repetición de turnos:
  - Si la última semilla cae en $S_j \Rightarrow$ el jugador j juega de nuevo.

1.1. Turno del Jugador

- Para un jugador $j \in J$:

  Sea $H_j \subseteq \{0, 1, \ldots, 5\}$ el conjunto de índices de hoyos propios:
  - Si $j = 1 \Rightarrow H_1 = \{0, 1, 2, 3, 4, 5\}$ en la fila inferior (P_1).
  - Si $j = 2 \Rightarrow H_2 = \{0, 1, 2, 3, 4, 5\}$ en la fila superior (P_0).

- Se recogen todas las semillas y se reparten en sentido antihorario (una a una).
  1. Selecciona $h \in H_j$ tal que $P_{f(j),h} > 0$, donde $f(j)$ indica la fila del jugador j.
  2. Se toman $k = P_{f(j),h}$ semillas y se distribuyen en sentido antihorario, una por hoyo, excluyendo el hoyo de origen.
  3. Se actualizan la matriz P y los almacenes S según las reglas de distribución y captura.

1.2. Reglas

- Si la última semilla cae en $S_j \Rightarrow$ el jugador j repite turno.
- Si la última semilla cae en $P_{f(j),h} = 1$ (vacío) y $P_{1-f(j),h} > 0 \Rightarrow$

$$S_j \leftarrow S_j + 1 + P_{1-f(j),h}; \quad P_{f(j),h} \leftarrow 0; \quad P_{1-f(j),h} \leftarrow 0$$

- Solo se puede seleccionar h tal que $P_{f(j),h} > 0$.

1.3. Fin del Juego

- El juego termina cuando uno de los jugadores no tiene semillas en ninguno de sus hoyos.

$$\sum_{h \in H_j} P_{f(j),h} = 0 \quad \text{para algún } j \in J$$

- Cuando se da el fin, el otro jugador recoge todas las semillas restantes de su lado y las pone en su almacén.

$$S_{\bar{j}} \leftarrow S_{\bar{j}} + \sum_{h \in H_{\bar{j}}} P_{f(\bar{j}),h}; \quad P_{f(\bar{j}),h} \leftarrow 0 \quad \forall h$$

  Donde $\bar{j}$ es el oponente de j.

- Gana el jugador j tal que:

$$S_j > S_{\bar{j}}$$

1.4. Estrategias del jugador sintético

- Maximizar $\mathbb{E}$[turnos extra]
- Maximizar $\mathbb{E}$[capturas]
- Minimizar $\mathbb{E}$[oportunidades de captura del oponente]
- Condiciones estratégicas según el estado del juego
  - Si $S_j > S_{\bar{j}}$:
    Buscar acabar el juego vaciando $P_{f(j),\cdot}$.
  - Si $S_j < S_{\bar{j}}$:
    Evitar terminar el juego, manteniendo al menos un hoyo activo:
    $\exists h \in H_j : P_{f(j),h} > 0$

1.5. Extra

- Hay algunas posiciones de salida (como jugar desde el hoyo 3 o 4) que generan secuencias con turnos extra o capturas rápidas.

2. Diseño del problema

Entradas: $(P, S, t, j_{\text{actual}})$
Donde:

- $P \in \mathbb{N}_0^{2\times 6}$: matriz de pits.
- $S \in \mathbb{N}_0^2$: vector de almacenes.
- $j_{\text{actual}} \in \{1, 2\}$: jugador actual.

Salidas: $j \in \{0, 1, 2, 3, 4, 5\}$

- Donde j es el índice de la columna del hoyo elegido por el algoritmo en la fila correspondiente al jugador actual.

3. Algoritmo de solución

3.1. Algoritmo Voraz (Greedy) para Mancala

Este algoritmo intenta tomar la mejor decisión local en cada turno, priorizando movimientos que otorgan un turno extra, luego aquellos que resultan en una captura, y finalmente, otros movimientos basados en heurísticas simples.

Algoritmo 1 VorazMancala

Require: P (Matriz de hoyos), S (Vector de almacenes), j_{actual} (Jugador actual)
Ensure: $h_{elegido}$ (Índice del hoyo seleccionado)

Variables:
    posibles_movimientos: LISTA de ENTERO
    mejor_movimiento_actual: ENTERO ← −1
    movimientos_con_turno_extra: LISTA de HOYO_Y_PUNTAJE
    movimientos_con_captura: LISTA de HOYO_Y_PUNTAJE
posibles_movimientos ← ObtenerHoyosValidos(P, j_{actual})
if ESTA_VACIA(*posibles_movimientos*) **then**
    return −1
    // No hay movimientos posibles
end if
for all h **en** *posibles_movimientos* **do**
    $(P_{sim}, S_{sim}, es\_turno\_extra, semillas\_capturadas\_val)$ ← SimularMovimientoCompleto(P, S, j_{actual}, h)
    if *es_turno_extra* **then**
        Añadir (h, P[Fila(j_{actual}), h]) a *movimientos_con_turno_extra*
    else if *semillas_capturadas_val* > 0 **then**
        Añadir (h, *semillas_capturadas_val*) a *movimientos_con_captura*
    end if
end for
if NO ESTA_VACIA(*movimientos_con_turno_extra*) **then**
    mejor_movimiento_actual ← ElegirMejorDeLista(*movimientos_con_turno_extra*, "MAX_PUNTAJ
    return EXTRACTO_HOYO(*mejor_movimiento_actual*)
end if
if NO ESTA_VACIA(*movimientos_con_captura*) **then**
    mejor_movimiento_actual ← ElegirMejorDeLista(*movimientos_con_captura*, "MAX_PUNTAJE)
    return EXTRACTO_HOYO(*mejor_movimiento_actual*)
end if
mejor_puntaje_general ← −1
for all h **en** *posibles_movimientos* **do**
    if P[Fila(j_{actual}), h] > *mejor_puntaje_general* **then**
        mejor_puntaje_general ← P[Fila(j_{actual}), h]
        mejor_movimiento_actual ← h
    end if
end for
return *mejor_movimiento_actual*

3.2. Algoritmo Minimax

procedure DecisionMinimax($P, S, j_{actual}, profundidad_{max}$)
Require: P (Matriz de hoyos), S (Vector de almacenes), j_{actual} (Jugador actual), $profundidad_{max}$

Ensure: $h_{elegido}$ (Índice del hoyo seleccionado)
Variables:
$posibles\_movimientos$: LISTA de ENTERO
$mejor\_hoyo\_encontrado$: ENTERO $\leftarrow -1$
$max\_valor\_evaluado$: REAL $\leftarrow -\infty$
$alfa$: REAL $\leftarrow -\infty$
$beta$: REAL $\leftarrow +\infty$
$posibles\_movimientos \leftarrow$ ObtenerHoyosValidos(P, j_{actual})
if ESTA_VACIA($posibles\_movimientos$) **then**
return -1
end if
for all h **en** $posibles\_movimientos$ **do**
$(P_{siguiente}, S_{siguiente}, es\_turno\_extra, \_) \leftarrow$ SimularMovimientoCompleto(P, S, j_{actual}, h)
$valor\_movimiento\_actual$: REAL
if es_turno_extra **then**
$valor\_movimiento\_actual \leftarrow$ ValorMax$(P_{siguiente}, S_{siguiente}, j_{actual}, profundidad_{max} - 1, alfa, beta, j_{actual})$
else
$j_{oponente} \leftarrow$ Oponente(j_{actual})
$valor\_movimiento\_actual \leftarrow$ ValorMin$(P_{siguiente}, S_{siguiente}, j_{oponente}, profundidad_{max} - 1, alfa, beta, j_{actual})$
end if
if $valor\_movimiento\_actual > max\_valor\_evaluado$ **then**
$max\_valor\_evaluado \leftarrow valor\_movimiento\_actual$
$mejor\_hoyo\_encontrado \leftarrow h$
end if
$alfa \leftarrow$ MAX$(alfa, max_valor_evaluado)$
end for
return $mejor\_hoyo\_encontrado$
end procedure

function ValorMax$(P_{estado}, S_{estado}, j_{turno_max}, profundidad, alfa, beta, j_{perspectiva_eval})$
if $profundidad = 0$ **o** EsFinDeJuego$(P_{estado}, j_{turno_max})$ **then**
return FuncionEvaluacionEstado$(P_{estado}, S_{estado}, j_{perspectiva_eval})$
end if
$posibles\_movimientos \leftarrow$ ObtenerHoyosValidos$(P_{estado}, j_{turno_max})$
if ESTA_VACIA($posibles\_movimientos$) **then**
return FuncionEvaluacionEstado$(P_{estado}, S_{estado}, j_{perspectiva_eval})$
end if
$valor\_max \leftarrow -\infty$
for all h **en** $posibles\_movimientos$ **do**
$(P_{siguiente}, S_{siguiente}, es\_turno\_extra, \_) \leftarrow$ SimularMovimientoCompleto$(P_{estado}, S_{estado}, j_{turno_m}$
$eval\_actual$: REAL
if es_turno_extra **then**
$eval\_actual \leftarrow$ ValorMax$(P_{siguiente}, S_{siguiente}, j_{turno_max}, profundidad - 1, alfa, beta, j_{perspectiva_eval})$

```
        else
            j_oponente ← OPONENTE(j_turno_max)
            eval_actual ← VALORMIN(P_siguiente, S_siguiente, j_oponente, profundidad−
    1, alfa, beta, j_perspectiva_eval)
        end if
        valor_max ← MAX(valor_max, eval_actual)
        alfa ← MAX(alfa, valor_max)
        if beta ≤ alfa then
            break                                              ▷ Poda Beta
        end if
    end for
    return valor_max
end function

function VALORMIN(P_estado, S_estado, j_turno_min, profundidad, alfa, beta, j_perspectiva_eval)
    if profundidad = 0 o ESFINDEJUEGO(P_estado, j_turno_min) then
        return FUNCIONEVALUACIONESTADO(P_estado, S_estado, j_perspectiva_eval)
    end if
    posibles_movimientos ← OBTENERHOYOSVALIDOS(P_estado, j_turno_min)
    if ESTA_VACIA(posibles_movimientos) then
        return FUNCIONEVALUACIONESTADO(P_estado, S_estado, j_perspectiva_eval)
    end if
    valor_min ← +∞
    for all h en posibles_movimientos do
        (P_siguiente, S_siguiente, es_turno_extra, _) ← SIMULARMOVIMIENTOCOMPLETO(P_estado, S_estado, j_turno_m
        eval_actual: REAL
        if es_turno_extra then
            eval_actual ← VALORMIN(P_siguiente, S_siguiente, j_turno_min, profundidad−
    1, alfa, beta, j_perspectiva_eval)
        else
            j_oponente ← OPONENTE(j_turno_min)
            eval_actual ← VALORMAX(P_siguiente, S_siguiente, j_oponente, profundidad−
    1, alfa, beta, j_perspectiva_eval)
        end if
        valor_min ← MIN(valor_min, eval_actual)
        beta ← MIN(beta, valor_min)
        if beta ≤ alfa then
            break                                              ▷ Poda Alfa
        end if
    end for
    return valor_min
end function
```