Documentación Técnica del Juego Mancala

Gabriel Alejandro Camacho Rivera

Índice

1. Requerimientos

1.1. Requerimientos Funcionales

- **RF01:** El sistema debe permitir iniciar una partida entre dos jugadores humanos (modo Jugador vs. Jugador).
- **RF02:** El sistema debe mostrar el estado actual del tablero antes de cada turno.
- **RF03:** El jugador debe poder seleccionar un pozo para realizar su jugada durante su turno.
- **RF04:** El sistema debe validar si el movimiento del jugador es legal según las reglas del juego.
- **RF05:** El sistema debe distribuir las piedras o gemas correctamente en sentido antihorario al ejecutar un movimiento.
- **RF06:** El sistema debe aplicar correctamente las reglas especiales del juego (captura, turno adicional, final de partida).
- **RF07:** Al finalizar la partida, el sistema debe calcular y mostrar el resultado indicando el jugador ganador.
- **RF08:** El sistema debe permitir el uso de un jugador sintético (bot) que tome decisiones automáticamente.
- **RF09:** El sistema debe ser capaz de utilizar distintas estrategias mediante el patrón Strategy.

1.2. Requerimientos No Funcionales

- **RNF01:** Se debe aplicar el patrón de diseño Modelo-Vista-Controlador (MVC) para mantener una buena separación de responsabilidades.
- **RNF02:** El sistema debe ser modular, permitiendo la incorporación futura de mejoras o variantes del juego.
- **RNF03:** El sistema debe permitir la incorporación de nuevas estrategias sin modificar el código base (uso del patrón Strategy).

2. Diagrama de Clases

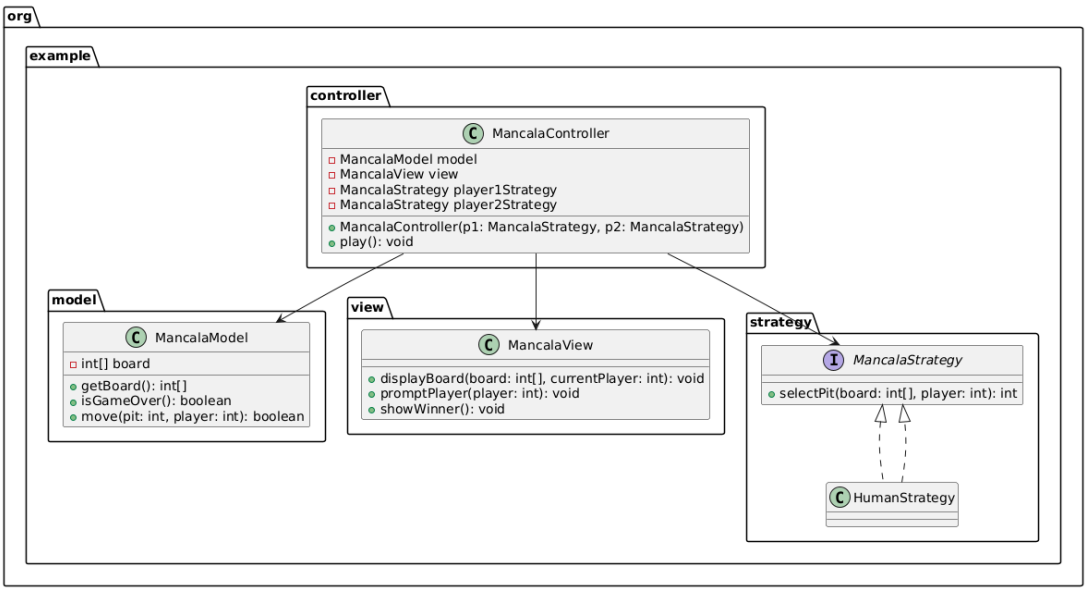

Figura 1: Diagrama de clases del sistema Mancala

3. Diagrama de Flujo del Método `play()`

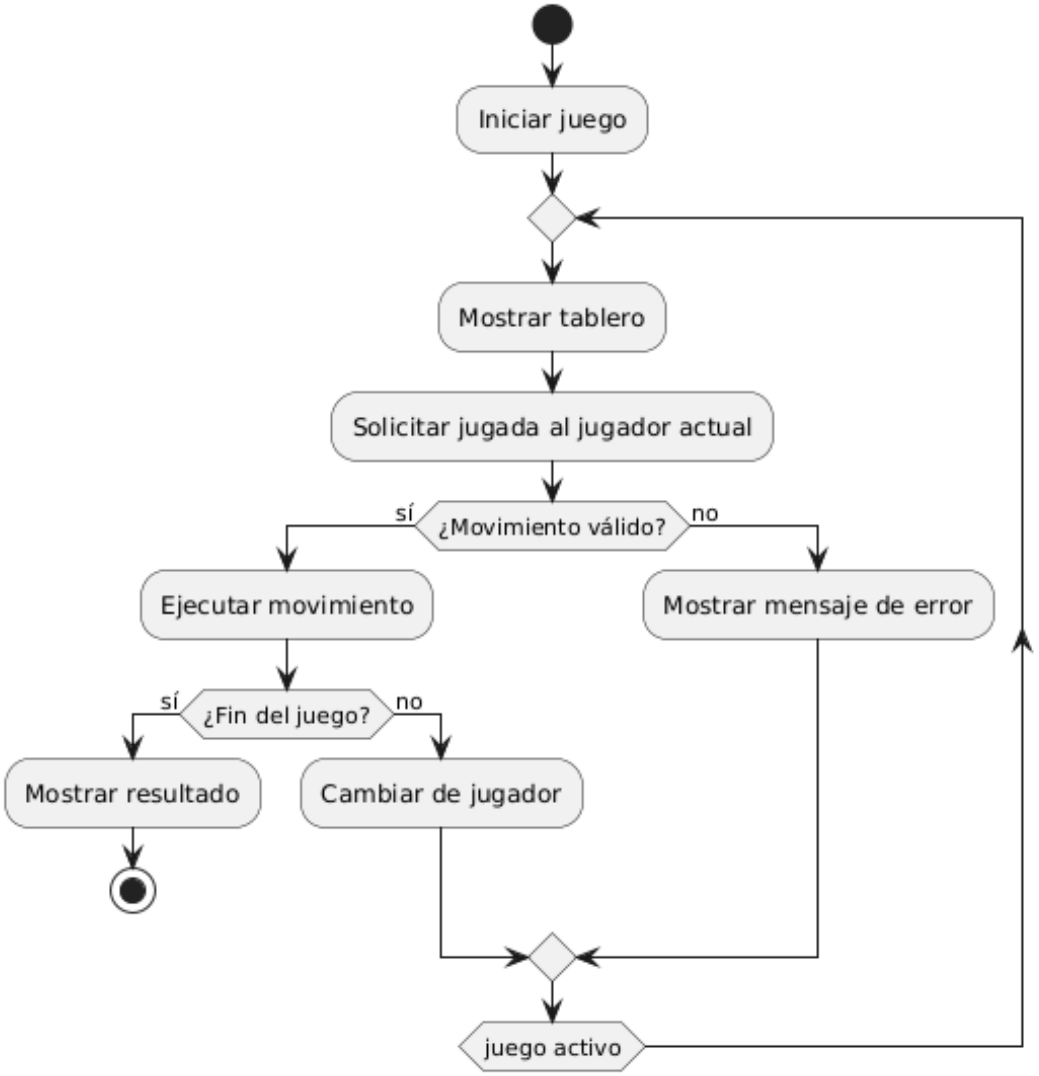

Figura 2: Diagrama de flujo del método `play()`

4. Estrategia Sintética Codiciosa (`SyntheticGreedyStrategy`)

La clase `SyntheticGreedyStrategy` implementa una lógica para bots basada en seleccionar el pozo que otorgue el mayor beneficio inmediato (turno adicional). Si hay varias opciones con el mismo beneficio, elige aleatoriamente entre ellas.

```
public class SyntheticGreedyStrategy implements MancalaStrategy {
    private final int player;
    private final Random random = new Random();

    public SyntheticGreedyStrategy(int player) {
        this.player = player;
    }

    @Override
```

```
    public int selectPit(int[] board, int player) {
        int start = (player == 1) ? 0 : 7;
        int end = (player == 1) ? 5 : 12;

        int bestGain = Integer.MIN_VALUE;
        List<Integer> bestPits = new ArrayList<>();

        for (int pit = start; pit <= end; pit++) {
            if (board[pit] == 0) continue;

            int gain = simulateGain(board.clone(), pit, player);

            if (gain > bestGain) {
                bestGain = gain;
                bestPits.clear();
                bestPits.add(pit);
            } else if (gain == bestGain) {
                bestPits.add(pit);
            }
        }

        if (bestPits.isEmpty()) {
            for (int pit = start; pit <= end; pit++) {
                if (board[pit] > 0) bestPits.add(pit);
            }
        }

        if (bestPits.isEmpty()) {
            System.out.println("Bot␣Voraz␣no␣encuentra␣pozos␣
               v lidos.");
            return -1;
        }

        int chosen = bestPits.get(random.nextInt(bestPits.size()));
        System.out.println("Bot␣Voraz␣juega␣en␣el␣pozo:␣" + chosen)
           ;
        return chosen;
    }

    private int simulateGain(int[] boardCopy, int pitIndex, int
       player) {
        int stones = boardCopy[pitIndex];
        boardCopy[pitIndex] = 0;

        int index = pitIndex;
        while (stones > 0) {
            index = (index + 1) % boardCopy.length;
            if ((player == 1 && index == 13) || (player == 2 &&
               index == 6)) continue;
            boardCopy[index]++;
            stones--;
```

```
        }

        int houseIndex = (player == 1) ? 6 : 13;
        return (index == houseIndex) ? 1 : 0;
    }
}
```

Listing 1: Implementación de SyntheticGreedyStrategy.java

5. Diagrama de Flujo del Método `selectPit()`

El siguiente diagrama de flujo representa la lógica de selección de pozo implementada por el bot codicioso (`SyntheticGreedyStrategy`). Este algoritmo recorre todos los pozos válidos del jugador, simula las posibles ganancias, y elige el mejor movimiento según heurística codiciosa.

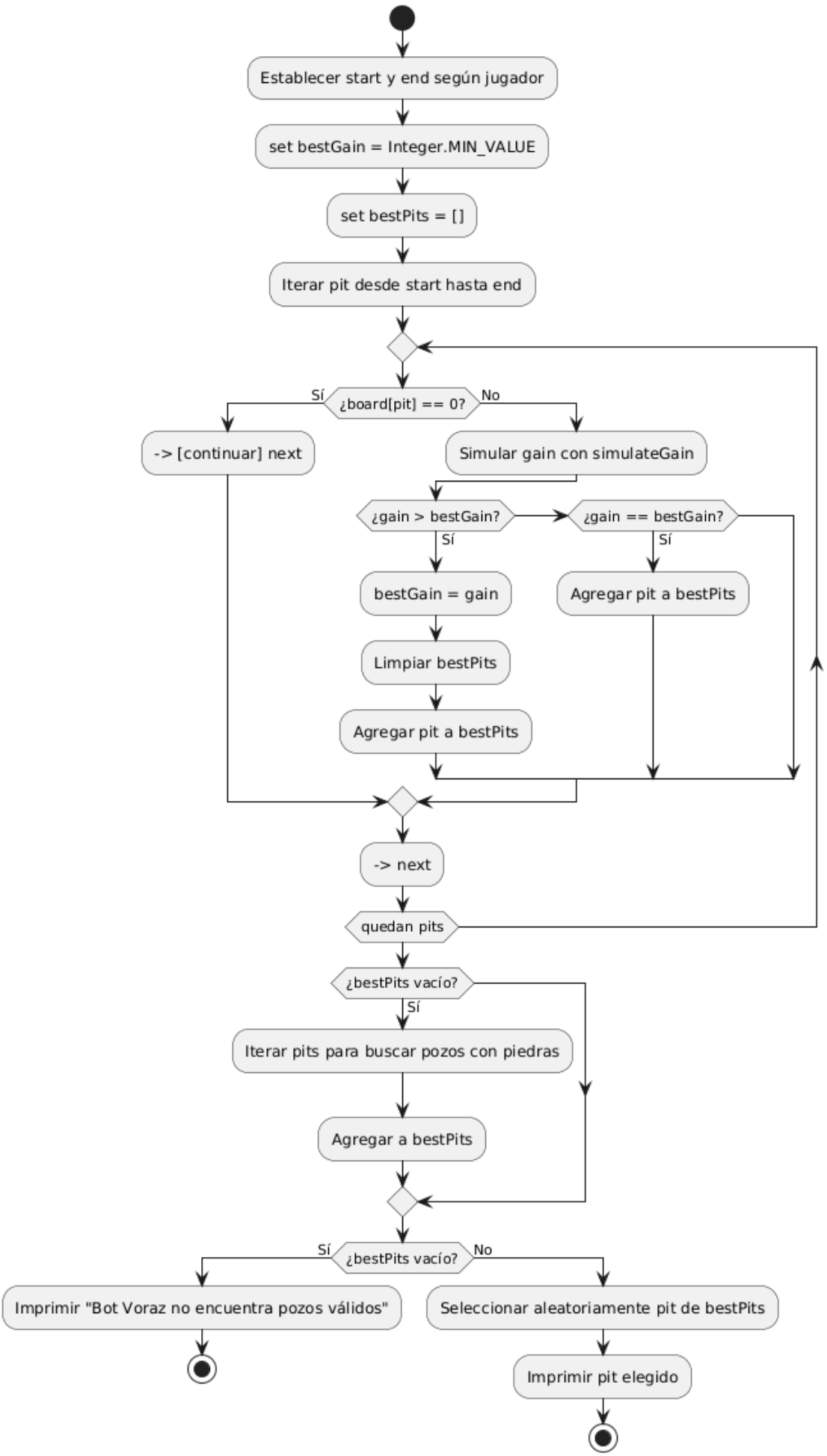

Figura 3: Diagrama de flujo del método `selectPit()`