Programación Lógica

Laboratorio 2

Implementación de juego - Owarelog

Integrantes

Nombre	CI	Correo
Nicolás Santos	5.244.700-0	nicolas.santos@fing.edu.uy
Pedro Zunino	4.739.354-3	pedro.zunino.severgnini@fing.edu.uy
Germán Ouviña	4.823.566-1	german.ouvina@fing.edu.uy

Índice

1. Introducción

En el presente informe se detallan los puntos mencionados en la consigna del **Laboratorio 2**. A continuación, se destacan ciertas consideraciones a tener en cuenta:

1. Se agregó el directorio *resources*, donde se almacenaron todas las imágenes de la solución, para mayor comodidad al visualizar los archivos. A su vez, se modificó el archivo `graficos.pl` para reflejar las nuevas rutas.

2. Se agregó el directorio *saves*, donde se almacenan todos los archivos de guardado. Se puede elegir el nombre del archivo a guardar, pero este siempre se almacenará en dicho directorio y con extensión `j`.

2. Desarrollo

2.1. Diseño

La solución se separó en varios módulos, con el fin de agrupar predicados de similar utilidad y facilitar la lectura de cada módulos en cuestión. Los mismos son:

- `owarelog.pl` - Módulo principal, el cual ejecuta el loop del juego y llama a los predicados que sean necesarios para hacerlo avanzar.
- `core.pl` - Lógica general del juego, cuenta con los predicados que controlan el tablero, las jugadas y
- `maquina.pl` - Lógica general de un jugador máquina, implementado predicados para la toma de decisiones automática.
- `utils.pl` - Predicados auxiliares utilizados por el resto de módulos o para funciones no específicas de la lógica del juego (como guardar o cargar partidas).

Cabe destacar que todos los módulos creados utilizan el módulo `utils.pl`, y tanto `owarelog.pl` como `maquina.pl` implementan predicados de `core.pl`.

2.2. Predicados

A continuación se detallan los cabezales así como una breve descripción de los principales predicados implementados, separándolos según el módulo al que pertenecen y ordenándolos según importancia.

2.2.1. Manejo del juego - owarelog.pl

Este módulo contaba de antemano con los predicados de `iniciar_juego`, `loop` y `process_command`, por lo que sólo se trataran los predicados agregados:

- `comprobar_fin_partida(Visual,MsgCmd,Jugador1,Jugador2,SiguienteTurno,NuevoTablero2, NuevoScore1,NuevoScore2)`
  Predicado utilizado para evaluar el estado actual del juego y determinar si se alcanzó el final de la partida, ya sea por medios "normales" (empate o alguno de los jugadores superando 25 puntos) o por falta de jugadas válidas.
- `loopFinal(Comando,Visual,MsgCmd,Jugador1,Jugador2,Turno,Tablero,Score1,Score2)`
  Predicado llamado por `comprobar_fin_partida` y utilizado en caso de que efectivamente sea el fin de la partida. Para evitar sobreescribir `loop`, se agregó este predicado que trata el mensaje final según la causa del fin de la partida.

Se destaca que, si bien no se agregaron otros predicados, se modificó ligeramente el comportamiento del predicado `loop` para comprobar si los jugadores son humanos o máquina.

También es importante mencionar que el predicado `process_command` con evento *click(Casa)* maneja el flujo general de la jugada producto de hacer click en la casa *Casa*.

A su vez, a dicho predicado se le agregaron 3 eventos para interpretar el fin del juego de forma gráfica: *empatar*, *ganar1* y *ganar2* (los cuales modifican el entorno gráfico acorde a un empate, victoria de jugador 1 y victoria de jugador 2 respectivamente).

2.2.2. Lógica del juego - core.pl

Este módulo cuenta con los siguientes predicados principales:

- `movimiento(+Casa, +Jugador, +Tablero, -Nuevo_Tablero, -Final)`
  Ejecuta un movimiento producto de hacer click en la casa *Casa*, actualizando el tablero en *NuevoTablero* y devolviendo el número de la casilla final en *Final*.
- `recoger_semillas(+CasilleroFinal, +Jugador, +Tablero, -NuevoTablero +Score1, -NuevoScore1, +Score2, -NuevoScore2)`
  Sirviendose de la casilla final resultante del predicado anterior, comienza a comprobar recursivamente hacia atrás si es posible recoger las semillas de la casilla en cuestión.
- `comprobar validez(+Tablero)`
  Comprueba si el tablero actual, resultante de una potencial jugada anterior, es válido. Dicha validez proviene de controlar que ningún jugador haya dejado al otro sin semillas y se haya quedado sin semillas a la vez. También controla el caso borde donde esto si es posible (cuando un jugador consume todas las semillas restantes al ganar).
- `quedan_movimientos_validos(+Turno, +Tablero)`
  Comprueba si el tablero actual, resultante de una potencial jugada anterior por parte del jugador *Turno* es válido. Dicha validez provieen de controlar que el jugador no deje a su contricante sin semillas. Este predicado se utiliza en el flujo del juego luego del predicado anterior, por lo cual se elimina la posibilidad

de quitar las semillas restantes del tablero. De esta manera, si el resultado es `false`, significa que ninguno de los jugadores puede seguir jugando y se procederá a hacer un recuento de las semillas restantes y ver que jugador gana.

- `terminar_partida_invalida(+Score1, +Score2, +Tablero, -NuevoScore1, -NuevoScore2)`
  En el caso donde lo anteriormente mencionado aplica, se realiza el recuento de las semillas de cada jugador y se suman las semillas de sus casas.

2.2.3. Lógica del jugador automático - maquina.pl

Este módulo cuenta con un único predicado principal, `iniciar_maquina`, el cual determina la siguiente jugada de un jugador máquina utilizando un algoritmo *minimax*. No obstante, para implementarlo se utilizan varios predicados auxiliares, siendo algunos de ellos:

- `minimax(+Profundidad, +Turno, +Tablero, +Score1, +Score2,`
  -Alpha, -Beta, -CasaElegida, -HeuristicValue)—
  Predicado base para la ejecución del algoritmo *minimax*, generando un valor *HeuristicValue* en los pasos base (ya sea profundidad 0 o un nodo terminal) y llamando al predicado `mejor_casa` en el paso inductivo.
- `mejor_casa(+Casas, +Depth, +Turno, +Tablero, +Score1, +Score2, -Alpha, -Beta, -Casa, -HeuristicVal`
  En base a la situación actual del tablero, ejecuta la idea del algoritmo *minimax*, simulando movimientos para cada casa de la lista *Casas* y comprobando el valor intrínseco de cada uno en base a *HeuristicValue*. Utilizando las nociones de *Alpha* y *Beta*, va intercalando entre jugadores siguiendo la simulación hasta alcanzar la profundidad *Profundidad* o un nodo terminal. Luego reconstruye el valor heurístico recursivamente, devolviendo la mejor casa en el atributo *Casa*.

Cabe destacar que se implementaron múltiples predicados más, comentándose en el código su utilidad. No se explaya sobre ellos en el informe, ya que su cometido es muy puntual y se autoexplica.

2.2.4. Predicados auxiliares - utils.pl

Debido a que se trata de predicados auxiliares, no se entrará en detalle de las especificidades de cada predicado. No obstante, cabe destacar que se dividen en 3 grupos:

- **Predicados para el juego** - Todos aquellos predicados cortos que no valía la pena introducir en `core.pl` y hacen referencia a la lógica del juego, se encuentran en este grupo. Son de naturaleza sencilla y se utilizan en todos los módulos.
- **Predicados para archivos** - Los predicados `cargar` y `guardar` se encuentran en este grupo, manejando los estados de partidas guardadas.
- **Predicados genéricos** - Otros predicados para manejo de listas que resultaron útiles en múltiples escenarios se encuentran en este grupo.