

UCM

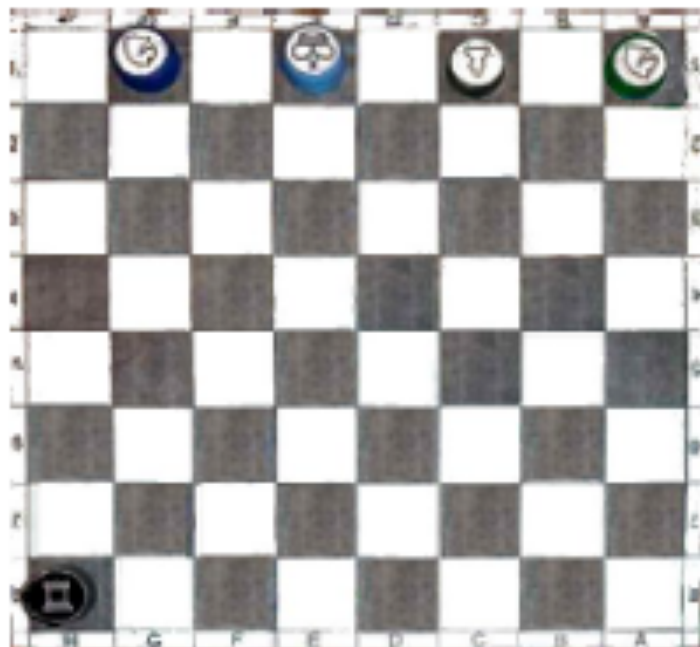
JUEGO DEL ZORRO Y LOS SABUESOS

Anghel Laurentiu Dulceanu | Jaime Rascón García

Juego del zorro y los sabuesos

¿Qué es?

El Zorro y los sabuesos es un juego muy simple pero divertido. Se juega sobre un tablero de ajedrez. Los sabuesos son cuatro fichas de un color que ocupan las cuatro casillas negras en un extremo del tablero, en el extremo opuesto una ficha de otro color, el zorro, ocupa cualquier casilla negra. No se salta ni se come. Por turno puede moverse un sabueso una casilla en diagonal hacia adelante. El zorro puede avanzar o retroceder en diagonal una casilla por turno. El zorro debe lograr llegar al extremo opuesto del tablero, los sabuesos deben lograr encerrar e inmovilizar al zorro antes de que éste llegue a su destino.



Motivación

Hemos elegido este juego como trabajo para la asignatura PDA porque es un buen ejemplo para poner en práctica las nuevas funcionalidades de Prolog aprendidas durante el curso, y explotar la potencia de este lenguaje.

Teníamos la convicción de que no nos iba a dar problemas la representación del tablero y del estado de juego y esto nos iba a permitir meternos de lleno en el estudio del problema y generar diferentes heurísticas, que es la parte interesante del juego.

Estado del arte

Partimos de la base de que existen implementaciones de este juego que exploran en profundidad un árbol de jugadas, de tal modo que si la máquina controla a los sabuesos es imposible ganar, se puede realizar un juego perfecto con los sabuesos tal como se explica en el libro *Winning Ways for your Mathematical Plays*.

Una cuestión interesante sería preguntarse qué heurísticas utilizar de tal modo que el juego siga siendo desafiante para un usuario pero al mismo tiempo éste sea capaz de ganar la partida, esta cuestión la iremos desarrollando conforme estudiemos más el problema.

Existen implementaciones en la tienda Google Play y Windows Store pero son de muy poca popularidad y tienen varias quejas de los usuarios. También hemos encontrado alguna implementación en lenguajes imperativos como Pascal, pero ninguna en lenguajes declarativos, lo que nos da la oportunidad de hacer algo nuevo.

Tecnologías Utilizadas

Hemos hecho uso del lenguaje declarativo Prolog para la codificación del programa.

Como implementación de Prolog utilizamos SWI-Prolog y hacemos uso de la librería de dominios finitos “clpfd” y otras funciones definidas en la implementación.

Implementación

El código ha sido dividido en secciones para tener una mejor legibilidad. En este apartado explicamos grosso modo cada sección y explicaremos con más detalle las diferentes heurísticas codificadas que simulan la inteligencia artificial de los sabuesos, utilizando en la explicación algunos bocetos que trazamos durante la codificación.

Las definiciones pormenorizadas de las funciones pueden encontrarse en el código fuente del proyecto.

SECCIONES

- **Representación del Juego:** Una muestra de cómo sería el tablero y las posiciones iniciales del zorro y de los sabuesos
- **Constantes:** Constantes globales que delimitan el tablero de juego o nos sirven para realizar algún cómputo
- **Inicialización:** Llama a la librería clpfd e inicia una partida
- **Gráficos:** Funciones para pintar el estado actual de juego

- **Lógica del Juego:** Funciones básicas para mover al zorro y los sabuesos por el tablero y son utilizadas en las partidas de 1 vs 1.
- **Condiciones de Victoria:** Comprobaciones para detectar el fin de la partida
- **Iniciar Partida:** Inicia una nueva partida y muestra los diferentes menús.
- **Bucles de Juego:** Gestiona los turnos de cada jugador y de la CPU.
- **Heurísticas:** Diferentes estrategias que puede seguir la CPU durante la partida.
- **Funciones Auxiliares Genéricas:** Funciones que no son específicas de este problema.
- **Funciones Auxiliares del Juego:** Funciones implicadas en el juego.
- **Funciones Auxiliares de Heurísticas:** Funciones implicadas en las heurísticas de los sabuesos.

HEURÍSTICAS

En el caso de que el usuario elija jugar contra la CPU, el jugador podrá mover el zorro y el ordenador calculará y realizará los movimientos de los sabuesos.

1. **mueveSiVaAGanar:** Prueba todos los movimientos posibles de cada sabueso y comprueba si gana
2. **mueveSiHaceBarrera:** Prueba todos los movimientos posibles de cada sabueso y comprueba si los sabuesos forman una barrera de izquierda a derecha infranqueable para el zorro.

Las ilustraciones 1 y 2 muestran situaciones en las que existen barreras.

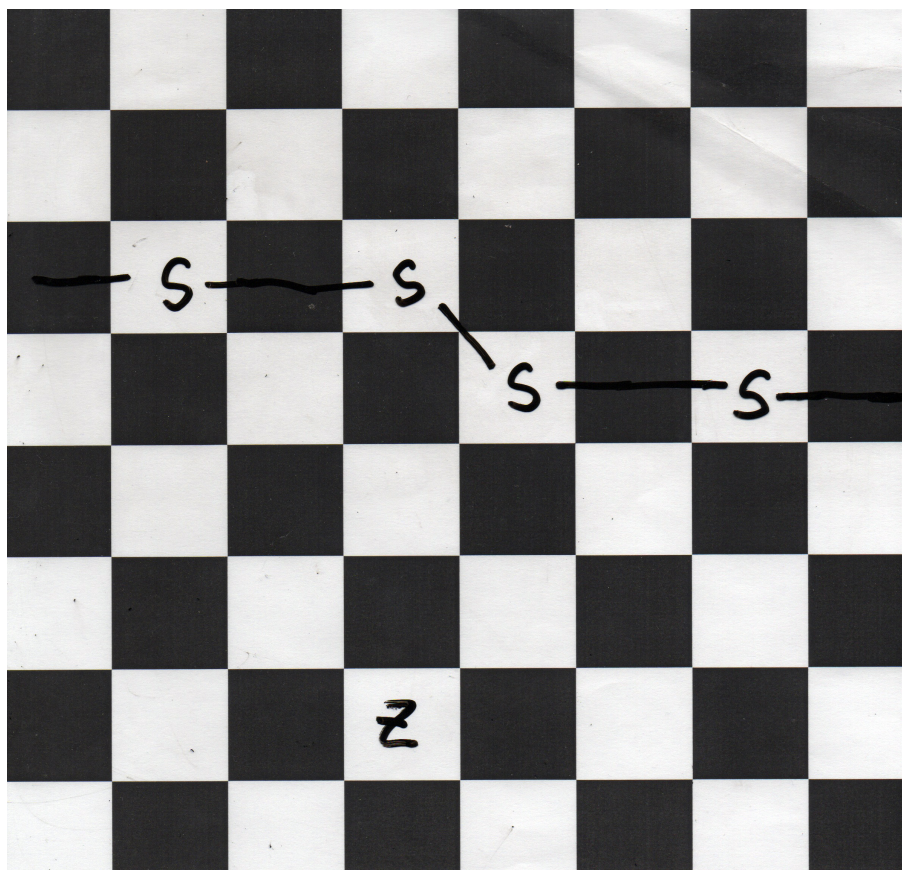


Ilustración 1: Barrera Irregular

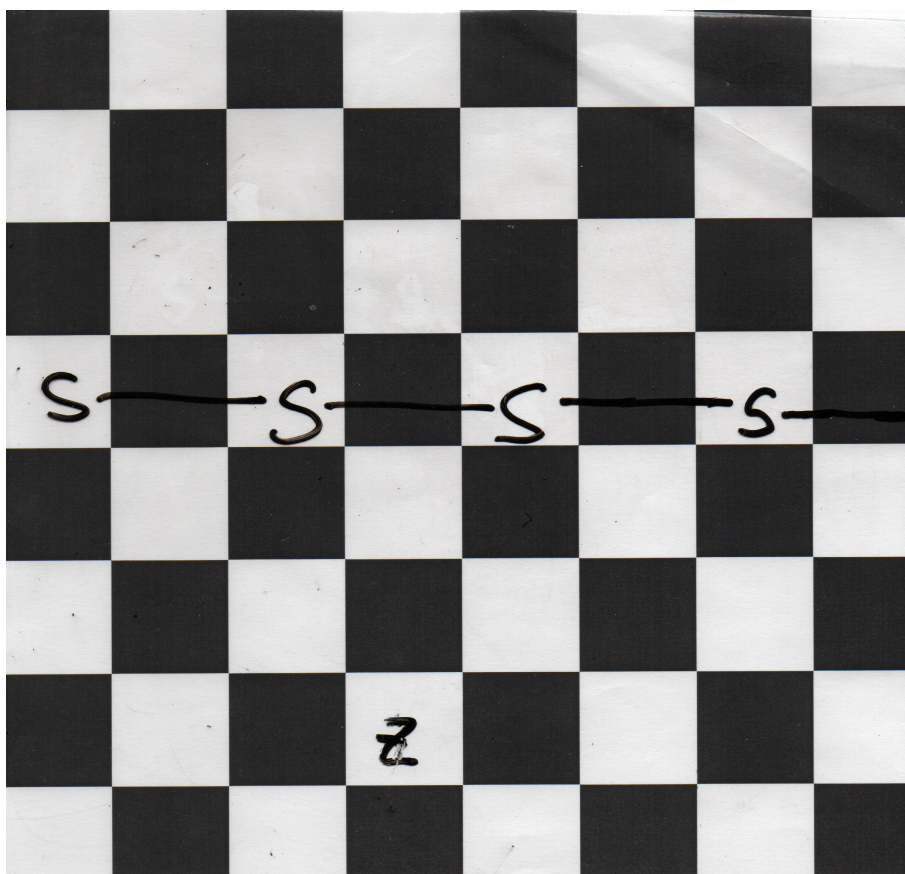


Ilustración 2: Barrera de una fila

3. **mueveElMasLejano**: Mover en dirección al zorro al sabueso más lejano.

En la ilustración 3 se puede ver que el más lejano no puede mover hacia el zorro, así que se prueba con los de la siguiente fila.

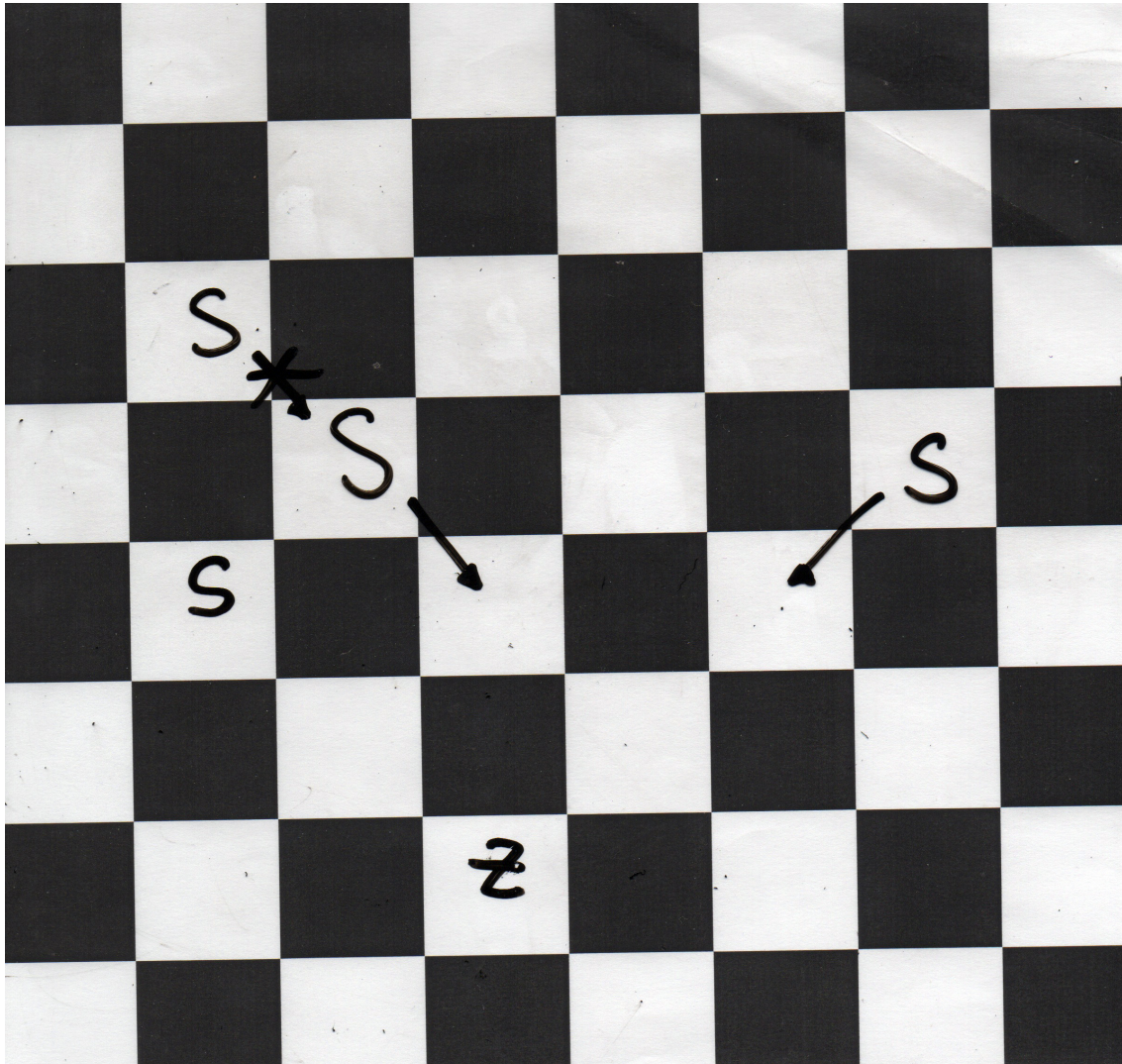


Ilustración 3: Mueve el más lejano

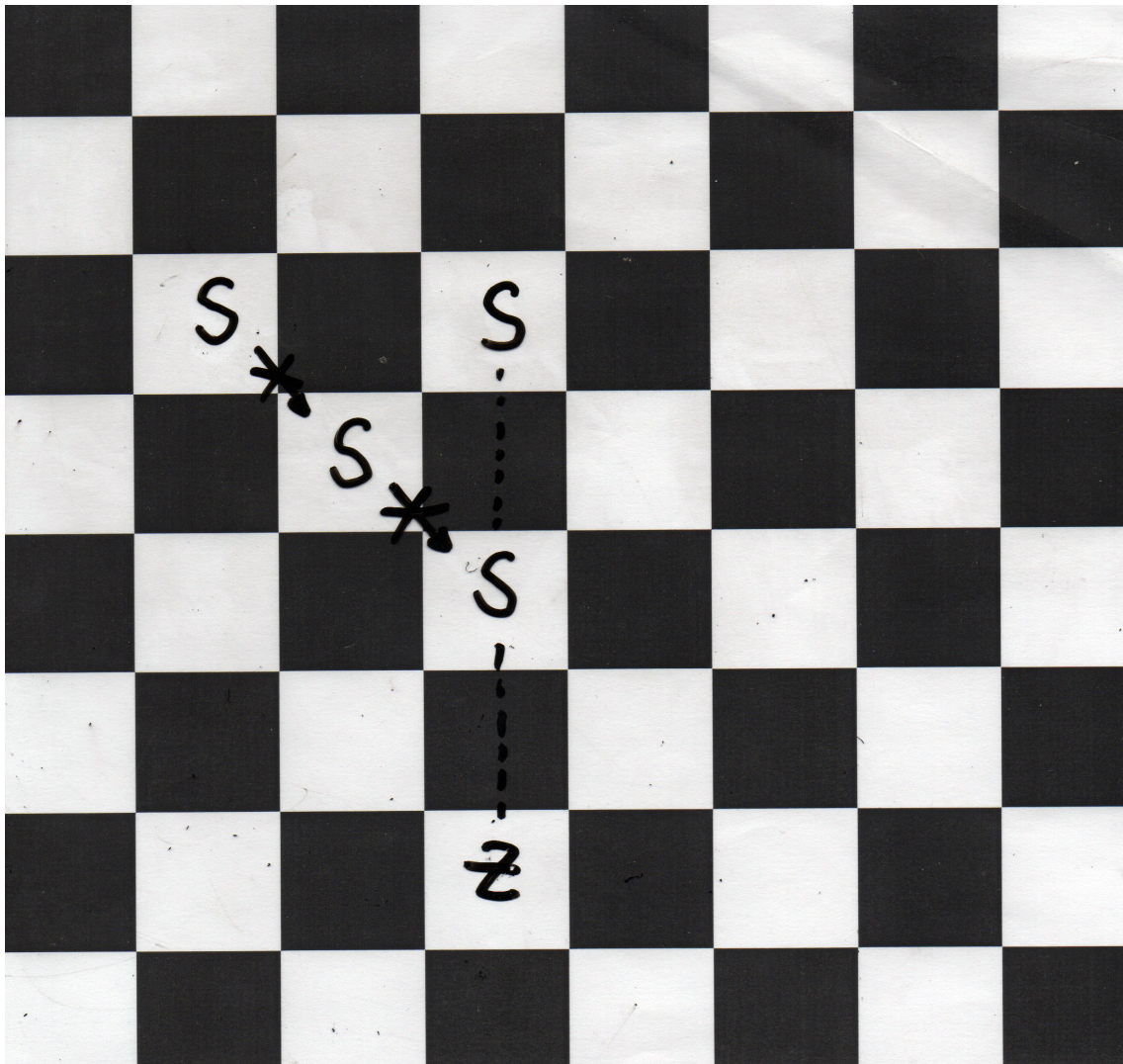


Ilustración 4: Imposible mover hacia el zorro

En la ilustración 4 se ve que ningún sabueso puede acercarse más a la vertical del zorro, en cuyo caso falla y se probaría otra heurística

4. **mueveAleatorio**: Intenta mover al primer sabueso, y si no puede al siguiente, etc.
5. **mueveMasCercano**: Mueve al sabueso que más cerca está del zorro
6. **mueveMenosInaccesibles**: Calcula para cada movimiento posible cuántas casillas quedan inaccesibles para los sabuesos y escoge un movimiento que minimice esta cantidad.

En la imagen podemos ver numeradas las casillas por las que es imposible que pase un sabueso en la partida actual, ya que éstos no pueden volver hacia atrás. El valor de la heurística en esta situación sería 12.

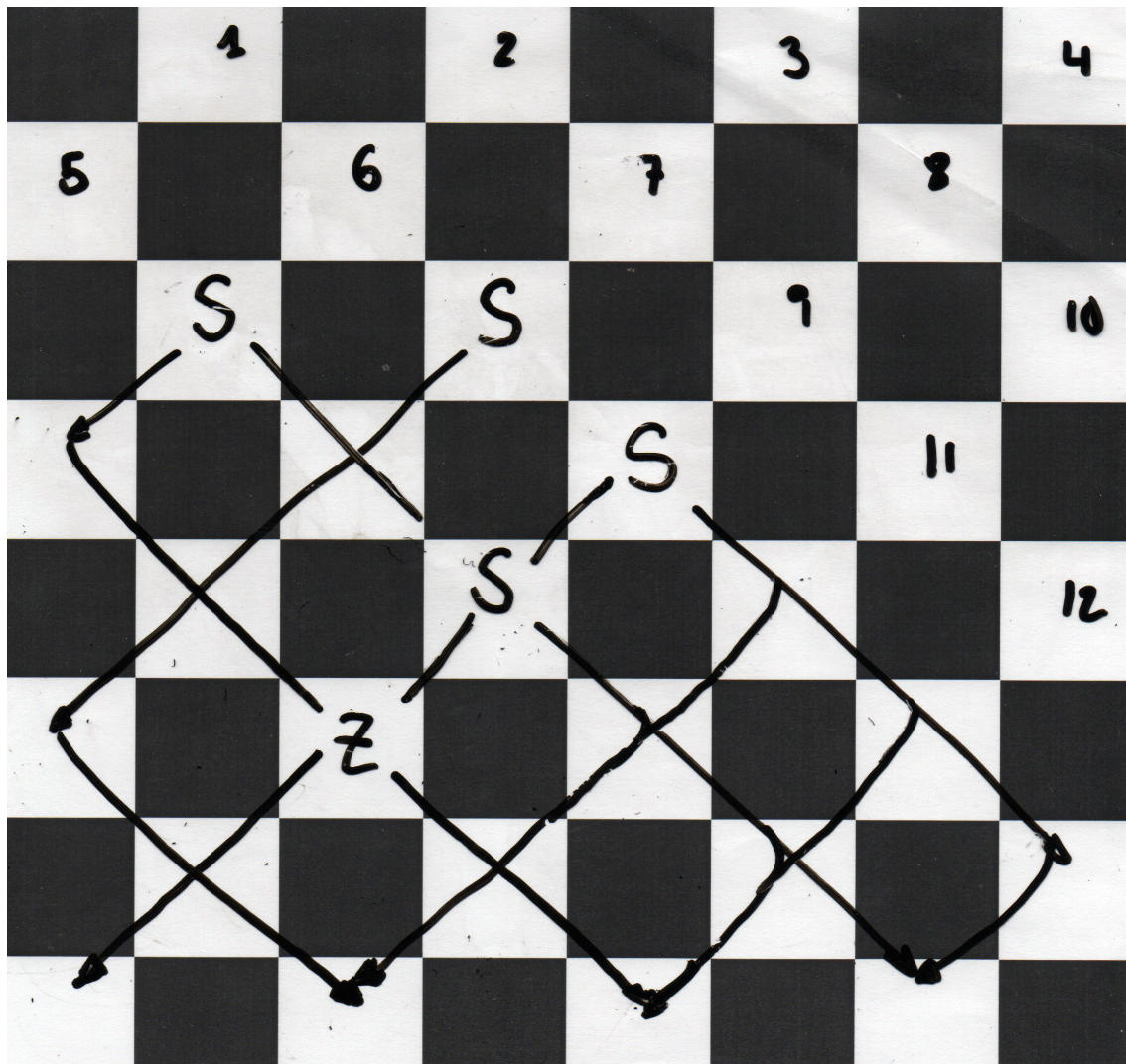


Ilustración 5: Cálculo de casillas inaccesibles

DETALLES DE IMPLEMENTACIÓN

Comprobación de condición de barrera:

- Recursivamente comprobamos que exista una barrera de izquierda a derecha, columna a columna.
- Se escoge una casilla y se comprueba si la casilla es blanca y hay un sabueso, o si es una casilla negra, en ambos casos, esa casilla forma barrera.
- A continuación comprobamos la casilla inmediatamente a la derecha si es blanca y tiene sabueso, o si es negra.
 - En caso de ser negra se comprueba la casilla de la fila anterior y posterior, que debe ser blanca y tener un sabueso.

- En caso de incumplirse estas propiedades, falla y prueba otra rama si hay, y al llegar al lado derecho, si hay barrera realizamos un corte para no explorar más.

Cálculo de casillas no accesibles por sabuesos

Son aquellas casillas que:

- Están en una fila anterior a cualquier sabueso.
- Están en la misma fila que el sabueso más alejado y no están ocupadas.
- Están en una fila posterior (por delante) a un sabueso pero no es accesible debido a que está muy a la izquierda o muy a la derecha y debido al movimiento diagonal no es accesible (ver ilustración 5).

Manual de Usuario

- Abrir SWI-Prolog
- Introducir [foxAndHounds]. Y pulsar ENTER
- Introducir la opción de juego y pulsar ENTER

```
=====
ZORRO Y SABUESOS
=====
```

```
Esta es una implementacion en prolog del juego de los sabuesos
Para mover al zorro debes insertar la direccion del movimiento de 1 a 4
El zorro solo puede moverse a una casilla libre en diagonal, se representa con una z
Los sabuesos se representan con una S
```

```
1 2
  z
3 4
```

```
= [ OPCIONES DE JUEGO ] =
```

- ```
1. Jugador vs Jugador
2. Jugador vs PC (Dificultad: Facil)
3. Jugador vs PC (Dificultad: Dificil)
```

```
Introduzca "1.", "2." o "3." para seleccionar la opcion y pulse ENTER
|: 2.
```

- Si se introduce una entrada errónea o se acaba la partida, se puede empezar una nueva introduciendo "nuevaPartida." Y pulsando ENTER.
- El programa es compatible con SWI-Prolog para Windows y compatible parcialmente con otras implementaciones de SWI-Prolog (debido a algunas funciones para dar formato al tablero en pantalla).

# Conclusiones

- Hemos profundizado en nuestros conocimientos de Prolog con este proyecto, en concreto al implementar las heurísticas, para las que hemos utilizado predicados de corte, “fail”, “findall”, y otras funciones de las librerías de SWI-Prolog, para tratar de maximizar el rendimiento y mejorar la legibilidad.
- Los consejos del profesor de la asignatura nos han sido de gran utilidad para implementar algunas funciones, y para evitar seguir pensando de modo “iterativo” y no forzar los hábitos que tenemos en otros lenguajes de programación.
- Aunque no hemos exprimido todo el potencial de la librería de dominios finitos, nos ha sido de utilidad para imponer algunas restricciones sobre los datos.
- Podemos decir que las decisiones que hemos tomado en el diseño y la implementación del juego elegido han sido acertadas ya que no hemos tenido problemas para dividir el trabajo.
- Nos ha costado bastante encontrar documentación técnica sobre este juego y no ha sido fácil conseguir que las heurísticas implementadas se comporten como queríamos pero al final creemos que hemos hecho un buen trabajo.

# Trabajo Futuro

- Incorporación de la aleatoriedad del estado inicial del tablero, adaptar para tableros de diferente tamaño y distinto número de sabuesos.
- Implementar una heurística basada en el algoritmo minimax con poda alfa-beta
- Añadir más niveles de dificultad al juego mezclando de distintas maneras las heurísticas implementadas.

# Bibliografía

---

- Introduction of Game AI – Neil Kirby
- Wikipedia – Fox Games
- Website – Reglas de Juegos Simples

# Autorización

---

Los abajo firmantes autorizan a que se publique el proyecto que han realizado para la asignatura de PROGRAMACIÓN DECLARATIVA AVANZADA de forma que podrá estar online y accesible para terceros, bajo las siguientes licencias: el software liberado bajo la licencia BSD (\*) y la memoria, documentación, gráficos u otro contenido liberado bajo la licencia Creative Commons Atribución 4.0 Internacional (\*\*). La autoría será respetada en todos los casos.

(\*): [http://directory.fsf.org/wiki/License:BSD\\_3Clause](http://directory.fsf.org/wiki/License:BSD_3Clause)

(\*\*): <https://creativecommons.org/licenses/by/4.0/deed.es>

Nombre, DNI y firma:

Nombre, DNI y firma:

Madrid, a 18 de Junio de 2014