PECL1

CONOCIMIENTO Y RAZONAMIENTO AUTOMATIZADO.

AKINATOR.

LENGUAJAES DE PROGRAMACIÓN.

ARQUITECTURA DEL JUEGO.

Hemos adoptado una arquitectura basada en una fase inicial de preparación de los datos y otra fase de juego que se repite hasta que o bien hemos encontrado la solución o determinamos que no podemos encontrarla pues nos hemos quedado sin preguntas (no se puede llegar con los datos iniciales pero se incluye esta comprobación por los que pueda meter el jugador) o porque no tenemos un lenguaje que cumpla las respuestas proporcionadas por el usuario en cuyo caso se le dará la opción de añadir un lenguaje nuevo tras responder de nuevo a las preguntas que anteriormente no contestó o contestó con ambigüedad (no sé, puede que si, puede que no).
Adicionalmente daremos la posibilidad de volver a jugar.
Mantendremos en memoria una lista con las respuestas del usuario, otra con los lenguajes y sus características y otra con las preguntas. Las dos últimas listas decrecerán en tamaño según explicaremos en el apartado de optimización.
Por cada iteración del bucle de juego realizamos las siguientes acciones:

- Elegir la pregunta óptima para realizar.
- Realizar la pregunta.
- Leer y almacenar la respuesta.
- Reducir la lista de lenguajes y de características lo máximo posible según la respuesta obtenida.
- Analizar si el juego ha terminado. En caso de que lo haya hecho actuaremos como ya hemos explicado y en caso contrario repetiremos el bucle de juego con la nueva lista de lenguajes y de preguntas.

REPRESENTACIÓN DE DATOS EN LA KNOWLEDGE BASE.

La elección de la representación de los datos en la base de conocimiento no ha sido trivial, si no se ha elegido teniendo en mente la optimización de las consultas. No obstante, a efectos explicativos, se expondrá antes este apartado para estar ya familiarizados con la estructura de representación de los datos cuando se explique la optimización.

Los datos se almacenan en la base de conocimiento con la siguiente estructura:

lenguaje (nombre_del_lenguaje**, [** Lista_de_características **]).**

caracteristicas([Lista_de_preguntas **]).**

Esta estructura nos permite cargar de forma sencilla tanto las preguntas como los lenguajes en lista en memoria al inicio del juego, así como guardar lenguajes nuevos de una forma cómoda de nuevo al archivo cuando un nuevo lenguaje sea descubierto al jugar.

- Lista_de_características: contiene las características de un lenguaje en el siguiente formato:
  - 0 → NO tiene la característica.
  - 1 → PUEDE que se piense que tiene esa característica.
  - 2 → SI tiene la característica.
- Lista_de_preguntas: es un listado de las preguntas que se deben realizar al usuario sobre los lenguajes de programación

La restricción que relaciona preguntas con características es la posición de estos en la lista. La pregunta en la posición n en Lista_de_preguntas esta en la posición n en Lista_de_características.

MANEJO DE LA INCERTIDUMBRE.

Para tratar las respuestas de tipo no sé, puede que si y puede que no se realizan las siguientes acciones.

- **No sé**: no se descarta ningún lenguaje, simplemente se descarta la pregunta.
- **Puede que si**: nos quedaremos con aquellos lenguajes cuya característica a evaluar cumpla que sí lo cumple o que puede que alguien piense que lo haga. La característica es > 0 por tanto.
- **Puede que no**: nos quedaremos con aquellos lenguajes cuya característica a evaluar cumpla que no lo cumple o que puede que alguien piense que lo haga. La característica es < 2 por tanto.

OPTIMIZACIÓN.

La optimización realizada se base en ver la base de conocimiento de los lenguajes como una matriz que relaciona lenguajes con características en un espacio bidimensional.

Con la realización de una pregunta reducimos la longitud de la dimensión preguntas en una unidad y la longitud de la dimensión características en tantas unidades como lenguajes no cumplan con la respuesta proporcionada.

Adicionalmente podemos analizar el estado de características resultante de modo que aquellas que no aporten capacidad para discriminar lenguajes serán eliminadas pues en el caso de ser preguntadas no aportarían información nueva. Este es el caso de que todos los datos de una característica sean el mismo. (contando los 1s tanto como 0s como 2s)

Por último, realizamos otra optimización adicional mediante la selección de la pregunta a realizar. Dicha selección se basará en tomar con un 90% de probabilidad aquella pregunta que pueda dividir a los lenguajes restantes en dos grupos de igual tamaño y con un 10% aquella pregunta que divide a los lenguajes en grupos de la mayor diferencia posible en tamaño. De este modo con un 90% de probabilidad realizaremos la estrategia más segura, búsqueda binaria pero tendremos la opción de realizar un salto de rendimiento con un 10% de probabilidad.

INTERFAZ JAVA CONECTADA POR UDP.

Deseábamos crear una interfaz para mostrar todos los datos que nuestro programa prolog producía, pero deseábamos que esta fuera lo menos intrusiva posible lo cual nos llevó a descartar otras posibles implementaciones como embeber prolog.

Para realizar la interfaz hemos creado un programa en JAVA que lanza tres hilos: uno que maneja la interfaz, otro que se encarga de crear un proceso que ejecute el código prolog y un último que se encargue de escuchar en un socket udp al proceso prolog.