Práctica de Búsqueda Local de Inteligencia Artificial

Josep Maria Olivé Fernández, Pol Monroig, Yaiza Cano

20 de marzo del 2020

Introducción

Este proyecto está enfocado principalmente a aprender a resolver un problema mediante algoritmos de búsqueda local explicados en teoría.
En primer lugar, debemos aprender a razonar sobre la naturaleza del problema y determinar el planteamiento a seguir. Para conseguir esto, se nos ha proporcionado una librería de algoritmos *AIMA* y una que recoge un conjunto de clases que generan los escenarios simulados *DistribFileSystem* que utilizaremos para construir parte de la inicialización nuestro programa.

Toda la práctica se ha desarrollado en Java, cada experimento se ha realizado 1000 veces y se han calculado valores medios para conseguir resultados más fiables. Los parámetros principales que se han medido a lo largo de la experimentación han sido:

- El tiempo total que se tarda en hallar la solución.
- El tiempo que tarda el servidor más lento en gestionar sus peticiones.
- El tiempo total de transmisión.
- El balance de carga de los servidores.

Para ilustrar los resultados, se ha hecho uso de gráficas y tablas para complementar las conclusiones y razonamientos extraídos durante la realización de este trabajo.

Por último, comentar que este proyecto ha sido realizado gracias a la colaboración y el trabajo duro de tres personas.

Descripción del Problema

Muchas aplicaciones en internet necesitan tolerancia a fallos y alta disponibilidad. Una solución frecuente para este problema es el tener un sistema de ficheros distribuido a lo largo de varios servidores y tener otro grupo de servidores dedicados a redireccionar las peticiones de ficheros a los servidores que tienen la copia del fichero pedido.

El punto clave de esta solución es cómo el servidor que atiende a las peticiones decide qué servidores van a enviar su copia del fichero a su destinatario. Así pues, el objetivo de esta práctica es experimentar con algoritmos de búsqueda local para resolver la tarea de, dado un conjunto de peticiones de ficheros, decidir qué servidores van a responder a dicha petición.

Una vez encontrada una solución, ¿cómo sabemos si ésta es suficientemente buena? Para evaluar la solución usaremos dos criterios a lo largo de la experimentación que se deberán de optimizar en caso de querer lograr una solución mejor; dichos criterios son:

1. **Minimizar el tiempo de transmisión** de los ficheros para el servidor que necesita más tiempo para trasmitir sus peticiones.
2. **Minimizar el tiempo total de trasmisión** de los ficheros pero con la **restricción** de que los tiempos de transmisión de los servidores han de ser lo más similares posibles entre ellos.

Definición de tipos

Implementación del estado

El estado que representa las soluciones tiene que almacenar de alguna manera las peticiones que hacen los usuarios, es decir tiene que guardar la relación entre petición, servidor e usuario. Al principio pensamos muchas maneras distintas de hacerlo pero muchas tenían inconvenientes, no por que guardaran menos información ni porque no representaran el estado de manera correcta si no porque no eran ineficientes. Lo primero que almacenamos de manera compartida es la información sobre los servidores, esta no representa el estado pero es necesaria para poder cambiar de estado. Después almacenamos una estructura de datos representada por un vector en el cual cada posición del vector representa un fichero y por posición hemos guardado un conjunto de peticiones. Este conjunto de peticiones lo hemos representado con un *MaxHeap* ya que preveíamos su utilidad para poder aplicar los operadores lo mas rápido posible. Por ultimo guardamos otras cualidades sobre el estado para poder calcular los heurísticos mas rápidamente, como por ejemplo un vector donde cada posición de este tenga el sumatorio de tiempos de transición del servidor de esa posición.

Tipos de estados iniciales

- **initalState1:** Este estado inicial mueve cada petición al servidor que tarda menos en enviarla, escoge el servidor de entre los posibles que pueden enviar ese fichero a dicho usuario. Hemos implementado esta solución inicial ya que hemos pensado que si por cada fichero lo ponemos en un servidor que ya va muy rápido, entonces estamos ayudando al algoritmo a encontrar un estado muy bueno inicialmente. Aunque equilibremos la carga entre servidores.

- **initialState2:** Este estado mueve cada petición a un servidor aleatorio de entre los posibles que pueden enviar ese fichero. Hemos elegido esta implementación de solución inicial ya que pensamos basarnos un poco en Simulated Annealing que es estocástico. Al escoger el estado inicial de manera aleatorio estamos provocando una exploración inicial que no esta presente en ningún momento en el Hill Climbing, ya que es un algoritmo que explota y escoge los mejores estados siempre, en otras palabras es puramente greedy. Con esta exploración inicial no estamos seguros de que si obtendremos mejores resultados al o empeoraremos, pero dado que la exploración es mínima lo mas probable es que acabemos obteniendo peores resultados que con el*initialState1*.

Tipos de operadores

- **moveMaxFile:** Este operador selecciona el fichero que más tarda del servidor que tiene su suma de tiempos de transmisión más alta y lo mueve a cualquiera de los otros servidores posibles, de manera que tiene un *branch factor* de $k-1$ donde k es el numero de servidores que contienen dicho fichero. Hemos escogido este heurístico porque creemos que al mover un fichero del servidor mas lento, estamos equilibrando la carga entre servidores.

- **moveRandomFile:** Este operador selecciona el fichero que más tarda de un servidor aleatorio y lo mueve a otro servidor aleatorio de los servidores posibles. Tiene el mismo *branch factor* que el operador anterior y lo hemos escogido para ver las diferencias entre un operador fijo y uno aleatorio.

- **op3:** Este operador combina los primeros dos operadores de la siguiente forma: moveMaxFile $\cup$ moveRandomFile. Tiene un *branch factor* de $\mathrm{O}(2(k-1))$ donde k es el número de servidores que contienen el fichero que se mueve. Lo hemos escogido para estudiar el comportamiento de la combinación de dos operadores distintos.

- **op4:** Este operador combina los primeros dos de la siguiente forma: moveMaxFile $\cap$ moveRandomFile. Tiene un **brach factor** de $\mathrm{O}(k-1)$ donde k es el número de servidores que contiene el fichero que se mueve. Lo hemos escogido en contrapunto al operador anterior.

Tipos de heurístico

- **FirstHeuristicFunction:** Este heurístico tiene en cuenta el sumatorio de tres criterios

$$h(state) = h_1(state) + h_2(state) + h_3(state) \quad (1)$$

  donde h_1 es el el tiempo del servidor que tarda mas, h_2 es el total de todos los tiempos de transmisión y h_3 es el equilibrio de tiempos de transmisión entre servidores y esta representada por la varianza.

- **SlowestServerHeuristicFunction** Este heurístico representa el criterio h_1.

- **TwoHeuristicFunction** Este heurístico representa el criterio h_1 + el criterio h_2.

Tipos de generadores de sucesores

- **FirstSuccessorFunction:** Genera todos los sucesores de un estado aplicando el operador *moveMaxFile*.

- **SecondSuccessorFunction:** Genera todos los sucesores de un estado aplicando el operador *moveRandomFile*.

- **ThirdSuccessorFunction:** Genera todos los sucesores de un estado aplicando el operador op3, que es una combinación entre moveMaxFile y moveRandomFile.

Experimento 1

En este experimento, nuestro objetivo es descubrir qué combinación de nuestros operadores es el más efectivo, es decir, con el que obtenemos mejores resultados a partir de una misma inicialización. El criterio que determinará la resolución de la prueba es el de minimizar el tiempo de transmisión del servidor que más tarda en gestionar sus peticiones.

Hipótesis

"El operador *moveMaxFile*, comparado con los otros operadores bajo las mismas condiciones, será el que obtenga el menor tiempo de transmisión del servidor más lento".

Condiciones iniciales del experimento

- Nº de usuarios: 200.
- Nº máximo de peticiones por usuarios: 5.
- Nº servidores: 50.
- Nº mínimo de replicaciones: 5.
- Algoritmo: Hill Climbing.
- Estrategia de inicialización: initialState2.
- Heurístico usado: SlowestServerHeuristicFunction.
- Operadores usados: Todos, ya que son el sujeto de la prueba.

Resultados del experimento

Fichero que más tarda: *MaxTransmissionTime*

Operadores	Situación inicial	Situación final
moveMaxFile	56.883	39.864
moveRandomFile	56.883	56.876
moveMaxFile ∪ moveRandomFile	56.883	39.863
moveMaxFile ∩ moveRandomFile	56.883	56.833

Conclusiones

Analizando los resultados obtenidos vemos que nuestra hipótesis era errónea. En un principio pensamos que con el primer operador obtendríamos los mismos resultados que con el tercero pero en menor tiempo y, por lo tanto, sería mejor. Lo que nos llevo a creer esto es que el tercer operador no es más que lo que obtenemos del primero más la unión de un seguido de estados sucesores generados de manera aleatoria por el segundo operador; así que pensamos que estos

últimos estados no tendrían relevancia ya que el primer operador está programado con el objetivo de obtener lo que nosotros considerábamos los mejores estados sucesores posibles.
De todas formas, decidimos llevar este experimento un paso más allá aun que no se pedía: al ver que la diferencia entre resultados del op1 con el op3, decidimos valorar también el tiempo que tardaban cada uno en encontrar solución y los resultados obtenidos son los siguientes: 22ms vs 43ms. El op3 tarda el doble que el op1, así que hemos decidido quedarnos con este último.
De este experimento hemos aprendido a no subestimar la aleatoriedad y que añadir un poco de ellas a nuestras fórmulas no está de más. Además, para hacer un buen trabajo, a veces hay que seguir el criterio de aquellos que lo realizan y no solo de los establecidos matemáticamente.

Experimento 2

El objetivo de este segundo experimento es averiguar qué estrategia de estado inicial da mejores resultados y fijar, de esta manera, el mejor estado sobre el que se partirá en la realización de los siguientes experimentos. El criterio que determinará la resolución de la prueba es el mismo que el del experimento anterior.

Hipótesis

"El estado *initialState1*, comparado con el otro estado bajo las mismas condiciones, será el que obtenga el menor tiempo de transmisión del servidor más lento".

Condiciones iniciales del experimento

- Nº de usuarios: 200.
- Nº máximo de peticiones por usuarios: 5.
- Nº servidores: 50.
- Nº mínimo de replicaciones: 5.
- Algoritmo: Hill Climbing
- Estrategia de inicialización: Todas, son el sujeto de la prueba.
- Heurístico usado: SlowestServerHeuristicFunction.
- Operador usado: op1.

Resultados del experimento

Fichero que más tarda: MaxTransmissionTime

Estados	Situación inicial	Situación Final
initialState1	19.022	14.709
initialState2	56.883	39.864

Conclusiones

Analizando los resultados obtenidos podemos comprobar la certeza de nuestra hipótesis.
En este experimento teníamos bastante claro que el primer estado daría mejores resultados que el segundo pero tampoco nos imaginábamos que la diferencia sería tan grande.
Como complemento a la reflexión de los resultados del experimento anterior vemos que aunque un poco de aleatoriedad esté bien, mucha puede ser contraproducente. Nuestro segundo estado tiene un factor de aleatoriedad muy grande y hace que sea difícil encauzarlo a una solución que se pueda considerar suficientemente buena.

También nos gustaría comentar que nos ha sorprendido que el operador y el heurístico hayan reducido tanto el tiempo, pensábamos que al partir de un estado inicial tan malo, no serían tan eficaces pero han conseguido reducirlo en un 29.9% vs el 22.7% del primer estado.
De este experimento hemos interiorizado el concepto de que la eficacia del operador y del heurístico no dependen del estado al que se le aplique pero que tampoco pueden hacer magia.

Experimento 3

Una vez tenemos la mejor estrategia de inicialización y la mejor combinación de operadores, aplicamos el mismo heurístico y definimos las mismas condiciones de prueba pero cambiando el algoritmo inteligente. El algoritmo a usar es *Simulated Annealing* y el objetivo de este experimento es averiguar con qué parámetros da mejores resultados. El criterio que determinará la resolución de la prueba será el mismo que hemos estado utilizando hasta ahora.

El planteamiento de este experimento es un poco caótico, hemos decidido que la mejor manera de obtener resultados es incrementando el valor de uno de los parámetros en cada ejecución mientras mantenemos fijado el de los otros tres.

Los parámetros son los siguientes:

Parámetros	Significado
steps	número de temperaturas.
stiter	número de iteraciones dentro de cada temperatura.
k	valor temperatura inicial.
lamb	decremento de temperatura.

Sus valores son estos:

- Para *steps* variable partiendo de un valor de 250 e incrementándolo 250 en cada ejecución hasta un límite de 2000:
  - stiter: 50.
  - k: 50.
  - lamb: 0.005.
- Para *stiter* variable partiendo de un valor de 25 e incrementándolo 25 en cada ejecución hasta un límite de 300:
  - steps: 500.
  - k: 50.
  - lamb: 0.005.
- Para *k* variable partiendo de un valor de 25 e incrementándolo 25 en cada ejecución hasta un límite de 300:
  - steps: 500.
  - stiter: 50.
  - lamb: 0.005.
- Para *lamb* variable partiendo de un valor
  - steps: 500.
  - stiter: 50.
  - k: 225.

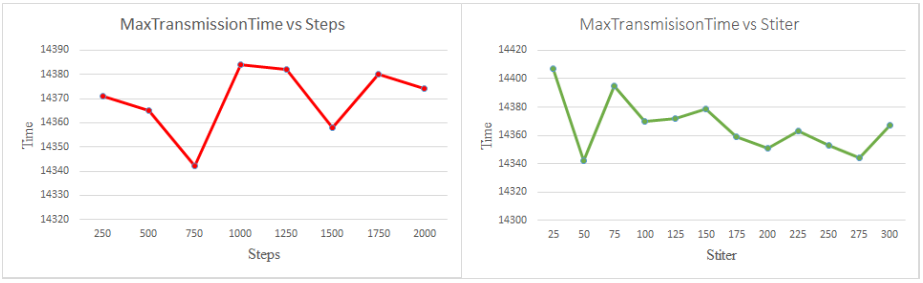
Hipótesis

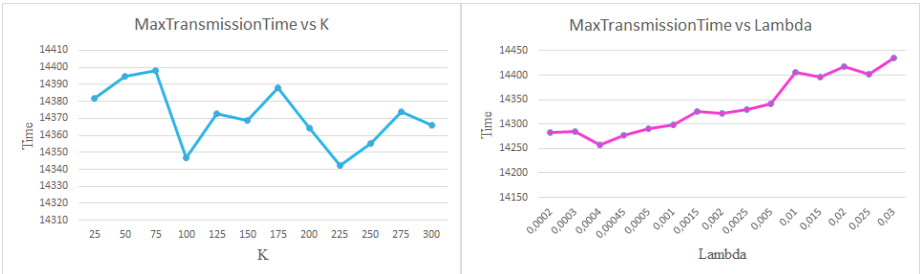
"Los valores que dan mejor resultado en el algoritmo inteligente *Simulated Anneling* son: *steps* = 1000", *stiter* = 150, *k* = 50 i *lambda* = 0.005."

Condiciones iniciales del experimento

- Nº de usuarios: 200.
- Nº máximo de peticiones por usuarios: 5.
- Nº servidores: 50.
- Nº mínimo de replicaciones: 5.
- Algoritmo: Simulated Annealing.
- Parámetros del algoritmo: sujetos de la prueba.
- Estrategia de inicialización: initialState1.
- Heurístico usado: SlowestServerHeuristicFunction.
- Operador usado: op1.

Resultados del experimento





Conclusiones

La primera conclusión clara que podemos extraer de este experimento es que nuestra hipótesis inicial era totalmente errónea, y que el valor óptimo para *steps* es de 750, y no de 1000. Mientras que el valor óptimo para el *Stiter* es 50, para la *K* es 225 y para *Lambda* 0,004.

Experimento 4

Esta vez queremos estudiar la evolución del tiempo de ejecución del algoritmo *Hill Climbing* según el número de usuarios y el número de servidores. Al tener dos parámetros distintos a evaluar, lo que haremos es fijar uno de ellos e ir incrementando el otro hasta que se pueda datar su tendencia y luego viceversa. Primero fijaremos el número de servidores en 50 y los usuarios los iremos incrementando de 100 en 100 partiendo de un valor de 100. Una vez encontrada la tendencia, fijaremos el número de usuarios en 200 e incrementaremos los servidores de 50 en 50, partiendo de un valor de 50.
El criterio que determinará la resolución de la prueba será el mismo que hemos estado utilizando hasta ahora.

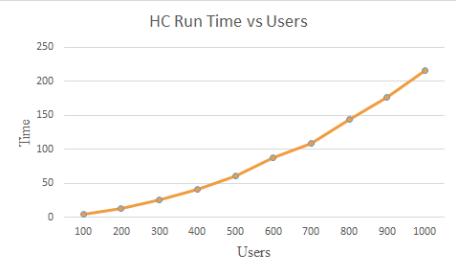
Hipótesis

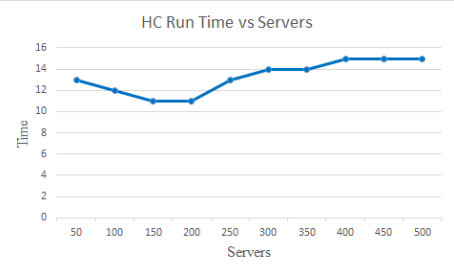
"A menor numero de servidores, mayor sera el rendimiento."

Condiciones iniciales del experimento

- Nº de usuarios: variable.
- Nº máximo de peticiones por usuarios: 5, constante.
- Nº servidores: variable.
- Nº mínimo de replicaciones: 5, contante.
- Algoritmo: Hill climbing.
- Estrategia de inicialización: initialState1.
- Heurístico usado: SlowestServerHeuristicFunction.
- Operador usado: op1.

Resultados del experimento





Conclusiones

Vemos que lo que afecta realmente al rendimiento no es tanto el numero de servidores, sino el numero de replicaciones mínimo que hay que mantener, ya que es este numero el que nos va a marcar el coste de nuestros operadores. Juntamente con el numero de usuarios que debemos gestionar.

Experimento 5

En este caso se ha implementado un nuevo heurístico que tenga en cuenta dos criterios distintos, y a partir de esto se calcula la diferencia entre el tiempo total de transmisión y el tiempo para hallar la solución, las condiciones son las mismas que las del primer apartado.

Hipótesis

"Creemos que este nuevo heurístico dará un resultado final mejor que un heurístico que solo tenga en cuenta un solo criterio."

Condiciones iniciales del experimento

- Nº de usuarios: 200.
- Nº máximo de peticiones por usuarios: 5.
- Nº servidores: 50.
- Nº mínimo de replicaciones: 5.
- Algoritmo: Hill Climbing.
- Estrategia de inicialización: initialState1.
- Heuristico usado: FirstHeuristicFunction y SlowServerHeuristicFunction.
- Operador usado: op1.

Resultados del experimento

Time	First	Slow
Elapsed	6	13
Total Transmisión	515.636	562.434

Conclusión

Vemos finalmente que el heurístico FirstHeuristicFunction que tiene en cuenta dos criterios más que el SlowServerHeuristicFunction da resultados mucho mejores, pues esta mucho más quilificado para ver si un estado es de mayor o menor calidad. A demás vemos también que el tiempo empleado por este heurístico es menor, y por lo tanto también es más eficiente. Aun así, SlowServerHeuristicFunction es mejor si solo deseamos reducir el tiempo mínimo, ya que al solo fijarse en este factor, realiza mejor esta tarea en concreto.

Experimento 6

Consiste en repetir el experimento anterior cambiando el algoritmo inteligente a *Simulated Annealing*.

Hipótesis

"Visto el resultado anterior, suponemos que FirstHeuristicFunction nos dará mejores resultados."

Condiciones iniciales del experimento

- Nº de usuarios: 200
- Nº máximo de peticiones por usuarios: 5
- Nº servidores: 50
- Nº mínimo de replicaciones: 5
- Algoritmo: SimulatedAnnealing
- Parámetros del algoritmo:
  - steps = 750.
  - stiter = 50.
  - k = 225.
  - lambda = 0.004.
- Estrategia de inicialización: initialState1
- Heuristico usado: FirstHeuristicFunction y SlowServerHeuristicFunction.
- Operador usado: op1.

Resultados del experimento

Time	First	Slow
Elapsed	193	195
Total Transmisión	507.456	609.689

Conclusión

Vemos otra vez que FirstHeuristicFunction nos da un tiempo de transmisión final mejor que SlowServerHeuristicFunction, y que a demás, el tiempo que necesita para darnos la solución es muy ligeramente menor, por lo tanto seria arriesgado decir que FirstHeuristicFunction es más eficiente en este caso, pero aun así es innegable que el resultado de este es mejor.

Experimento 7

Este experimento consiste en medir el tiempo total de transmisión y el tiempo necesario para hallar la solución para diferentes valores de número de replicaciones de un fichero.
Usando el escenario inicial decidido en el *Experimento 2*, el algoritmo *Hill Climbing* y ambas heurísticas, iremos incrementando el valor de número de replicaciones de 5 en 5, partiendo desde 5 y llegando a un límite de 25.

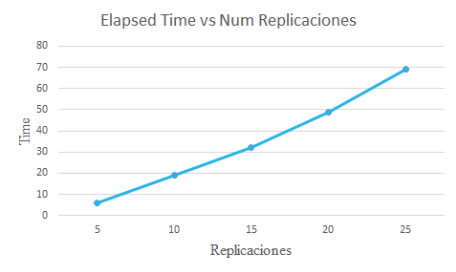
Hipótesis

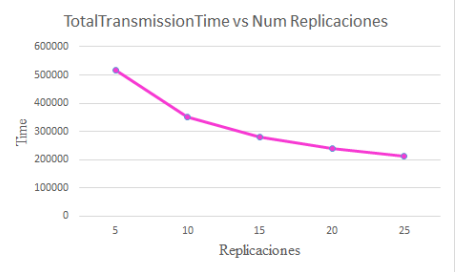
"Basándonos en resultados anteriores, creemos que a menor numero de replicaciones mayor será nuestro tiempo de ejecución."

Condiciones iniciales del experimento

- Nº de usuarios: 200.
- Nº máximo de peticiones por usuarios: 5.
- Nº servidores: 50.
- Nº mínimo de replicaciones: variable, son el sujeto de la prueba.
- Algoritmo: Hill Climbing.
- Estrategia de inicialización: initialState1
- Heuristico usado: FirstHeuristicFunction.
- Operador usado: op1.

Resultados del experimento





Conclusión

Vemos que nuestra hipótesis era correcta, y que a menor numero de replicas, menor es el coste de nuestros operadores y por lo tanto menor tiempo de ejecución tenemos. Por otro lado, a mayor numero de replicas, menor puede llegar a ser nuestro tiempo de transmisión total, ya que tendremos la posibilidad de usar un mayor numero de servidores para el mismo fichero, facilitándonos reducir la carga individual de cada uno, y encontrando un mayor numero de rutas óptimas o sub-óptimas.

Conclusión del proyecto

A lo largo de este trabajo hemos tenido nuestra primera toma de contacto con el mundo de inteligencia artificial y hemos podido ver fácilmente la cantidad de posibilidades que ofrece la librería *AIMA* y lo útil que nos podrá ser cuando hagamos algún proyecto que se pueda resolver utilizando algoritmos de *búsqueda local*, técnica de resolución de problemas explicada en clase de teoría y que hemos logrado comprender después de invertir horas a esta práctica.
Además, en este proyecto hemos aprendido cómo diseñar experimentos, recopilar datos, extraer resultados, analizarlos y sacar conclusiones. También hemos aprendido a trabajar en equipo y la importancia del uso de herramientas colaborativas como *git* y *Overleaf*.

Como bonus, nos gustaría comentar que nos hemos dado cuenta de la importancia de pasar todos los juegos de pruebas posibles para cerciorarse de que el output proporcionado es aceptable. Si no hubiera sido porque decidimos invertir tiempo necesario para realizar el experimento especial y obtener feedback antes de la fecha de entrega de la práctica, no nos hubiéramos dado cuenta de un error en la implementación y no hubiéramos sido capaces de entender los resultados obtenidos durante la parte de experimentación.

INFORMACIÓN DEL TRABAJO DE INNOVACIÓN

Para nuestro trabajo de innovación hemos decidido realizar un estudio sobre *AlphaStar*, la primera IA que ha sido capaz de derrotar a uno de los mejores jugadores profesionales de *StarCraft II*, un videojuego de estratégia creado por *Blizzard Entertainment*.

Dicho trabajo aún está en proceso y lo hemos divido en las siguientes secciones:

1. Introducción: terminada y hecha por **Yaiza Cano**.
2. Descripción del producto: terminada y hecha por **Yaiza Cano**.
3. Métodos y técnicas utilizadas: en proceso y hecho por **Pol Monroig**.
4. Factor innovador del producto: terminado y hecho por **Josep Maria Olivé**.
5. Impacto del producto en la empresa: en proceso.
6. Impacto del producto en el usuario o en la sociedad: en proceso.

Para documentar, hicimos uso de la siguiente lista de referencias:

- Reinforcement learning: libro utilizado para los apartados **Métodos y técnicas utilizadas i Factor innovador del producto**.
- AlphaStar Grandmaster level in StarCraft II using multi-agent reinforcement learning: artículo en el blog *Deepmind* utilizado para los apartados **Descripción del producto, Métodos y técnicas utilizadas i Factor innovador del producto**.
- AlphaStar Grandmaster level in StarCraft II using multi-agent reinforcement learning: paper en la revista *Nature* utilizado para los apartados **Descripción del producto, Métodos y técnicas utilizadas i Factor innovador del producto**.
- Artificial intelligence in video games: artículo en *Wikipedia* utilizado para el apartado **Introducción**.
- AlphaStar: Mastering the Real-Time Strategy Game StarCraft II: artículo en el blog *Deepmind* utilizado para los apartados **Métodos y técnicas utilizadas, Factor innovador del producto**.
- Play nim game: juego en *Archimedes-lab* utilizado para el apartado **Factor innovador del producto**.

El hecho de que *StarCraft II* sea conocido por su dificultad, además de ser uno de los juegos más famosos (por no decir el más) y con más visualizaciones en los *E-Sports*, ha hecho que sea realmente complicado encontrar información útil o suficientemente técnica que podamos aprovechar ya que, aunque la creación revolucionaria de la IA *AlphaStar* haya dado lugar a que exista mucha cantidad de información sobre ella y que se hable de ella en muchas páginas web, dicha información está enfocada a su público, un público genérico y no especializado.