Minesweeper solver

David Santiago Flórez Alsina* and Laura Sofía Ortiz Arcos†
Universidad del Rosario, Escuela de Ingeniería, Ciencia y Tecnología, Bogotá, Colombia
(Dated: October 31, 2022)

Resumen

En este artículo se muestra el análisis aplicado a nuestra solución para el problema del buscaminas. Lo anterior, por medio de lógica proposicional, es decir definiendo las reglas necesarias para que nuestro agente pudiese jugar al buscaminas implementado. Finalmente, se realizó una comprobación del funcionamiento del agente en diversos tableros de buscaminas, teniendo en cuenta el tiempo y la proporción de victorias.

Keywords: *Lógica proposicional, reglas, buscaminas.*

I. INTRODUCCIÓN

El Buscaminas es un videojuego creado en 1989 por Curt Johnson, y portado a Windows por Robert Donner. Como se conoce, el objetivo de este juego es despejar un campo, el cuál tiene minas, evitando elegir alguna de estas.

Fue lanzado por primera vez como parte del Microsoft Entertainment Pack. Luego se integró de manera definitiva a partir de la versión de Windows 3.X. Posteriormente, el juego continuó con el Sistema operativo hasta la plataforma de Windows 8.

El Buscaminas, junto con el Solitatio y otros juegos, fueron de los primeros implementados en Windows, pero un dato curioso es que la función original de estos, no era brindar entretenimiento a los usuarios. El sistema de Windows los implementó con el objetivo de familiarizar a los usuarios con el ratón y la interfaz gráfica.

Por ejemplo el Solitario, tenía como objetivo acostumbrar al usuario el arrastre del ratón. Y lo mismo ocurre con el Buscaminas, esta vez con el fin de mostrar cómo se utiliza el botón derecho e izquierdo del ratón, así como su precisión.

* Correspondence email address: davidsa.florez@urosario.edu.co
† Correspondence email address: lauraso.ortiz@urosario.edu.co

A. Reglas del juego

Como se mencionó anteriormente, el juego se trata de despejar todas y cada una de las casillas de la pantalla, sin elegir una que tenga mina, si no, esta detonará y perderás la partida.

Por lo tanto, las reglas son:

1. Algunas casillas contienen un número, el cuál indica la cantidas de minas adyacentes a esa casilla. Por ejemplo, si una casilla tiene el número 2, significa que de las ocho casillas que hay alrededor, hay 2 con minas y el resto sin minas.
2. En caso de tener una casilla sin número, es porque no existe ninguna mina adyacente a la casilla, y éstas se descubren automáticamente *(es decir se expanden en el tablero)*.
3. Si se descubre una casilla con una mina se pierde la partida.
4. Se puede poner una bandera en las casillas donde hay posibilidad de que se encuentre una mina, esto para ayudar a descubrir las que están cerca.

B. Tipos de Buscaminas

El Buscaminas es un juego que lo puedes encontrar de diferentes versiones, aparte de la clásica:

- Buscaminas de forma circular.
- Buscaminas hexagonal.
- Triangular.

- En código ASCII.

Para este proyecto se decidió trabajar con la versión clásica, la cuál consta de 4 niveles:

1. Principiante: 8x8 casillas y 10 minas.
2. Intermedio: 16x16 casillas y 40 minas.
3. Avanzado: 16x30 casillas y 99 minas.
4. Personalizado: Donde el usuario puede elegir el número de casillas *(teniendo en cuenta que la altura debe estar entre 8-24 y el ancho 8-32)* y el número de minas *(teniendo en cuenta la dimensión elegida del tablero)*.

Pero también tendremos algunos cambios y es que trabajaremos únicamente con tableros cuadrados, es decir el avanzado no se tendra en cuenta.

C. Ejemplos del problema

Ejemplo 1

El siguiente tablero muestra como sería el nivel principiante, y su solución, así como en que ocasión perdería:

Figure 1. Solución a un tablero 8x8.

Figure 2. Partida perdida al descubrir una casilla con mina.

Ejemplo 2

El siguiente tablero muestra como sería el nivel intermedio, y su solución, así como en que ocasión perdería:

Figure 3. Solución a un tablero 16x16.

Figure 4. Partida perdida al marcar una bandera en una casilla que no tenía mina.

Ejemplo 3

El siguiente tablero muestra como sería el nivel avanzado, y su solución, así como en que ocasión perdería:

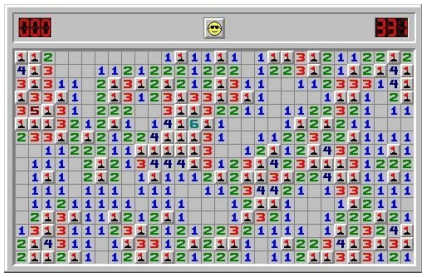

Figure 5. Solución a un tablero 16x30.

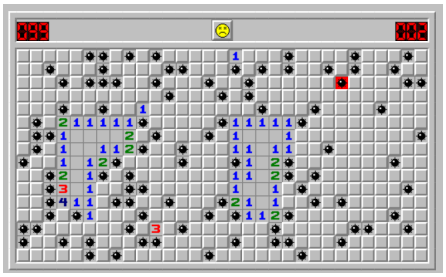

Figure 6. Partida perdida al descubrir una casilla con mina.

Por otro lado, nuestro problema se tienen las siguientes propiedades del entorno:

Opción 1	Opción 2
Completamente observable	✓ Parcialmente observable
✓ Agente único	Multiagente
✓ Determinista	Estocástico
Episódico	✓ Secuencial
✓ Estático	Dinámico
✓ Discreto	Continuo
✓ Conocido	Desconocido

II. MÉTODOS

La definicion formal del problema del juego Buscaminas, puede ser descrita como:

- Estado inicial: Tablero sin ninguna casilla descubierta.
- Posibles acciones: Las reglas definidas para poder solucionar el problema con lógica proposicional.
- Función de transiciones: Cada una de las reglas que llevan de un estado a otro.
- Prueba de satisfacción del objetivo: Que el agente logre completar el juego.
- Función de costo: En el caso general, se asume que la función de costo es constante para todas las transiciones.

Los métodos empleados para resolver este problema, fueron por medio de lógica proposicional y algunas estrategias de deducción *(Forward chaining y Backward chaining)*.

Lo anterior, a través de bases de conocimiento, las cuáles se construyen de hechos *(Atómos)* y reglas *(Implicaciones)*. Donde los hechos nos ayudan a representar una situación, es decir cuantos más hechos se conozcan, más precisa es la situación conocida, y las implicaciones que nos ayudan a representar el conocimiento lógico.

Para que nuestro agente logre resolver el juego se definieron las siguiente fórmulas:

A. Fórmulas

1. *Códigos*

Para manejar los estados definimos unos códigos como sigue:

- 9 = Casilla desconocida.
- 10 = Mina marcada.
- 11 = Hay mina.

2. *Unicidad por casilla*

Nuestra primera regla consiste en que cada casilla del tablero debe de tener un número y soló un número. Para esto definimos una regla que es de la forma:

$$\begin{aligned} (casilla\ actual, i) \rightarrow \neg(casilla\ actual, j) \\ \forall j \in OtrosNumeros, \\ OtrosNumeros \coloneqq \{j \neq i, j \in Numeros\}, \\ Numeros \coloneqq \{0,1,2,3,4,5,6,7,8,10,11\} \end{aligned} \tag{1}$$

3. *No hay minas*

Esta fórmula me permite entender el juego de la siguiente manera: *"Sé que en esta casilla hay un número y en estas otras no hay bomba, entonces la bomba debe estar en aquella casilla."*

Esto se logra con un proceso como este: Tomar la casilla que se supone tiene X bombas adyacentes *(caso a consideración)*, después hace una Itoria de casos en que una casilla adyacente no tiene bomba y junta la Itoria con el caso a consideración. Para agregar como consecuente el hecho de que en una casilla de las adyacentes hay bomba. Componentes del **antecedente**:

$$(casilla\ actual, i) \tag{2}$$

$$Y\neg(casilla\ adyacente\ 1, 11) \ldots Y\neg(casilla\ adyacente\ n, 11) \tag{3}$$

El **consecuente**:

$$(casilla\ adyacente\ w, 11) \tag{4}$$

usando las ecuaciones anteriores $(2), (3)$ y el consecuente(4), podemos expresar la regla así:

$$(2)(3) \rightarrow (4)$$

4. Hay minas

Esta regla la podemos entender como el caso complementario al de la fórmula anterior, así: *"Sé que en esta casilla hay un número y en estas otras hay bomba, entonces no debe haber bomba en aquella casilla."*

Dicha regla la generamos en un proceso como este: Toma la casilla que se supone tiene X bombas adyacentes *(caso a consideración)*, después hace una Itoria de casos en que una casilla adyacente tiene bomba y junta la Itoria con el caso a consideración.

Para finalmente agregar como consecuente el hecho de que en una casilla de las adyacentes no hay bomba. Componentes del **antecedente**:

$$(casilla\ actual, i) \tag{5}$$
$$Y(casilla\ adyacente\ 1, 11) \dots Y(casilla\ adyacente\ n, 11) \tag{6}$$

El **consecuente**:

$$\neg(casilla\ adyacente\ w, 11) \tag{7}$$

usando las ecuaciones anteriores $(5), (6)$ y el consecuente (7), podemos expresar la regla así:

$$(5)(6) \rightarrow (7)$$

B. Nuestro Agente

El agente que implementamos hace lo siguiente para solucionar el juego:

1. Expande una ***casilla aleatoria*** como primer paso.
2. Con lo descubierto en ***(1)*** actualizamos el conocimiento del agente a través de la función actualizar conocimiento.
3. ***Obtengo la lista de casillas que conozco***, en particular las que son distintas de 0 y 11 *(es decir las que no tienen mina adyacente y las que tienen mina)* y empiezo a preguntarme sobre cada una de las casillas adyacentes a las casillas conocidas, en concreto itero sobre las adyacentes que no son conocidas *(para no desperdiciar iteraciones o entrar en bucles innecesarios)*.
4. Para la casilla adyacente desconocida se crea una nueva base de conocimiento completamente desde cero, generando las fórmulas que competen solo a la casilla sobre la cual se va a decidir, y después se actualiza la base de conocimiento con toda la información que se sabe. *Esto para evitar que con tanta información el algoritmo de búsqueda se vuelva muy lento con información que no le compete al caso que se va a explorar, y también para evitar contradicciones entre reglas o casos innecesarios para el actual contemplado.*
5. La -***primera pregunta***- es si puedo marcar con bandera la casilla adyacente desconocida seleccionada, si no puedo marcarla con certeza entonces me hago la -***segunda pregunta***- ¿puedo expandirla? Si tampoco puedo con certeza entonces paso a analizar la siguiente casilla.
6. repito el paso ***(5)*** hasta que me quede sin casillas adyacentes desconocidas, y sin casillas conocidas que tengan vecinas por revisar, ***si en ningún momento pude tomar una acción*** entonces hago una expansión aleatoria sobre las casillas que no conozco, y con esta información actualizo el conocimiento. ***Si en algún momento pude marcar o expandir con certeza*** entonces dejo de buscar en las vecinas y actualizo el conocimiento adquirido, para nuevamente empezar desde el paso ***(3)*** hasta que pierda o gane el juego.

III. RESULTADOS Y DISCUSIÓN

Una vez comprendido el entorno y el problema del juego, además de implementadas las diferentes fórmulas para la solución del problema, se procedió a analizar los tiempos de ejecución para distintos tamaños de problema, y su proporción de vistorias.

Los resultados fueron los siguientes:

A. 15 porciento de dificultad

La siguiente gráfica muestra los tiempos de ejecución que tuvo para problemas de 3x3 hasta 13x13, cada uno con 3 muestras del juego *(es decir se jugó 3 juegos para cada tamaño y se tomó su promedio)*, en donde podemos ver que el más lento fue el problema de 11×11, la explicación que tenemos para esto es que en los tres intentos que se hicieron para medir el tiempo de ejecución en 13×13 el agente tuvo que tomar una elección aleatoria, y pisó una bomba, es por ello que vemos que los tiempos de ejecución son muy cortos y la cantidad de victorias es cero.

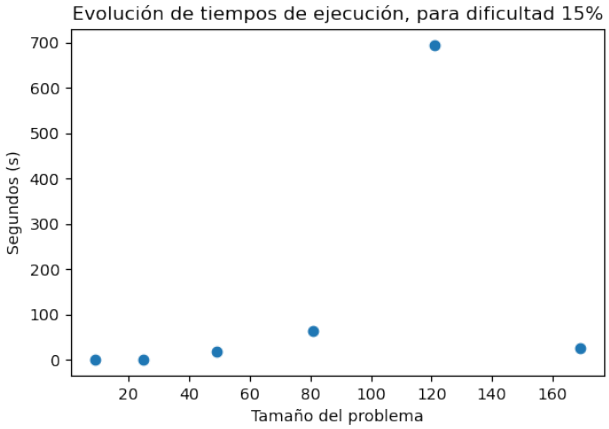

Figure 7. Evolución de los tiempos de ejecución, para una dificultad del 15%.

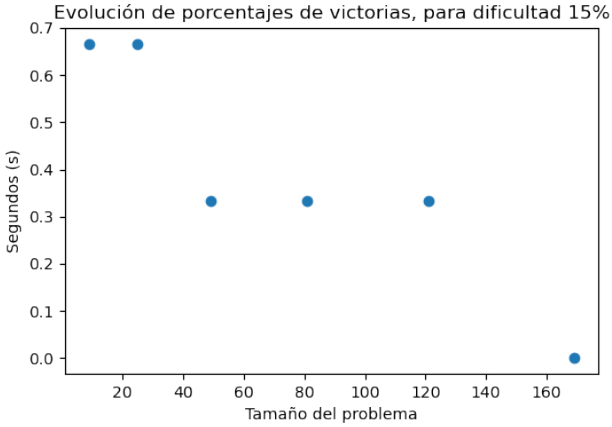

Figure 8. Porcentajes de victoria, para una dificultad del 15%

Sin embargo podemos entrever siguiendo la tendencia de la curva de tiempos de ejecución, que el aumento de tiempo que toman tiene tendencia exponencial.

B. 20 porciento de dificultad

La siguiente gráfica muestra los tiempos de ejecución que tuvo para problemas de 3×3 hasta 13×13, cada uno con 10 muestras del juego *(es decir se jugó 10 juegos para cada tamaño y se tomó su promedio)*, en donde podemos ver que el más demorado fue el problema más grande, es decir, 13×13 casillas.

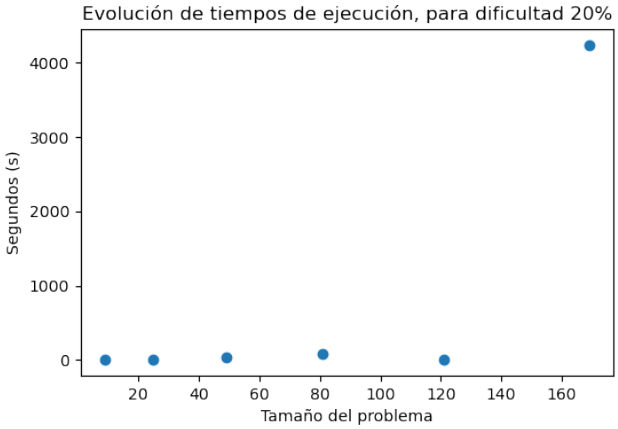

Figure 9. Evolución de los tiempos de ejecución, para una dificultad del 20%.

Se le hizo zoom a la gráfica anterior, para mostrar un mejor análisis de la evolución de los tiempos de ejecución para los problemas en donde tuvo un mejor rendimiento. En donde podemos observar que le llevo más tiempo solucionar problemas de 9×9 que 11×11.

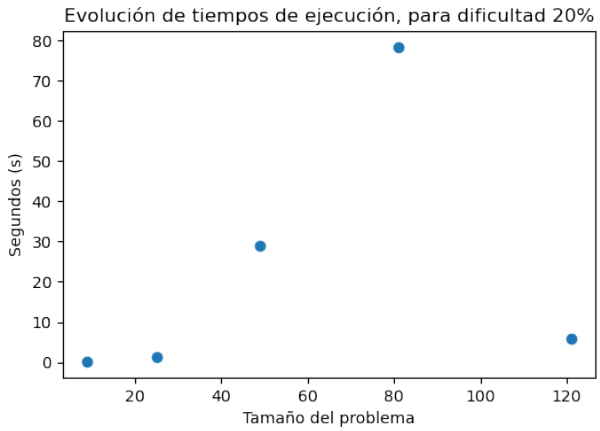

Figure 10. Evolución de los tiempos de ejecución más rápidos, para una dificultad del 20%

Por otro lado, las siguientes gráficas muestran el porcentaje de victorias que tuvo, en donde podemos ver que para casos más pequeños el porcentaje será mucho mejor.

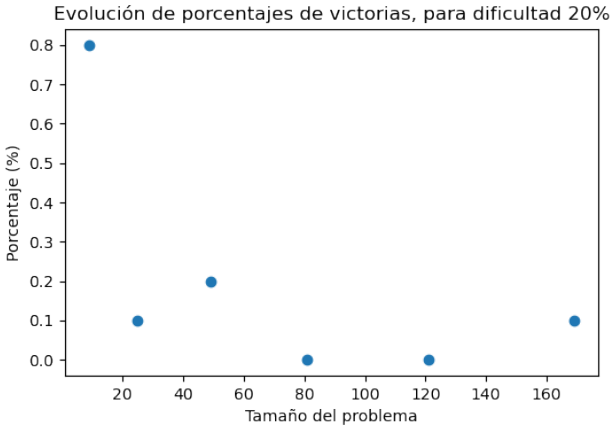

Figure 11. Porcentajes de victoria, para una dificultad del 20%

Haciendo zoom podemos ver que el porcentaje de victorias es más grande para problemas 13×13, que para los de 9×9 y 11×11 casillas.

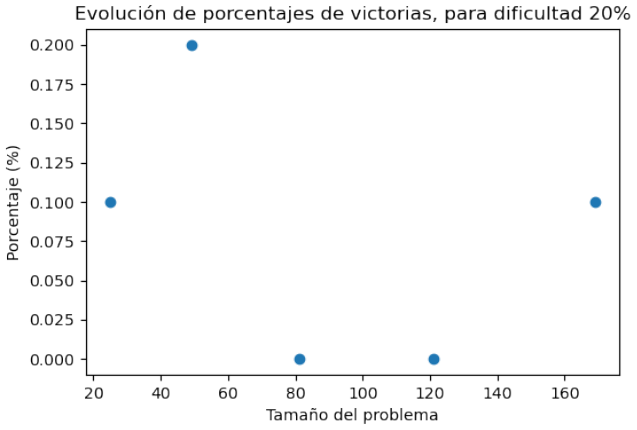

Figure 12. Porcentajes de victoria más bajos, para una dificultad del 20%

C. 25 porciento de dificultad

La siguiente gráfica muestra los tiempos de ejecución que tuvo para problemas de 3×3 hasta 13×13, pero esta vez uno con 6 muestras del juego, ya que la dificultad aumento y se tardaba más*(es decir se jugó 6 juegos para cada tamaño y se tomó su promedio)*, en donde podemos ver que con una dificultad mayor es mucho más lento resolviendo el problema.

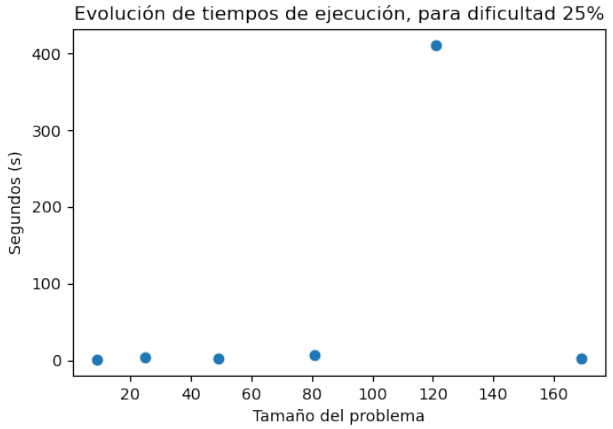

Figure 13. Evolución de los tiempos de ejecución, para una dificultad del 25%.

Por otro lado, las siguientes gráficas muestran el porcentaje de victorias que tuvo, en donde podemos ver que para casos más pequeños el porcentaje será mucho mejor.

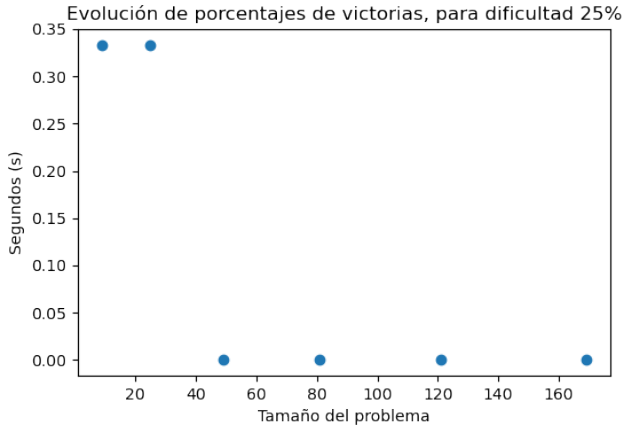

Figure 14. Porcentajes de victoria, para una dificultad del 25%

D. Análisis

Dentro de nuestros hallazgos encontramos que la implementación de ASK utilizada *(un método recursivo para hallar solución)* es mucho más lenta que la implementación del agente usando el método $pl\_fc\_entails$ *(método secuencial para hallar solución)*.

También vimos que para evitar contradicciones o bucles indeseados lo ideal es recrear una base de conocimientos con sus reglas para el caso que se va a explorar, esto no solo mejora los tiempos de búsqueda sino que también los resultados.

E. Siguientes pasos

Como próximos pasos para mejorar queremos hacer una revisión completa de las reglas planteadas y su suficiencia para interpretar el juego mejor.

Otra próxima mejora es implementar una regla que nos permita saber la cantidad de minas que tiene el tablero. Lo anterior nos podría ayudar a mejorar el rendimiento de solución por medio de inferencias sabiendo cuantas minas faltan por descubrir.

También, buscamos mejorar las acciones que toma el agente, podrían haber casos en los que llegue a marcar mal una bomba y podríamos programarlo para que considere si está bien o mal marcada, una vez nueva información se da a conocer.

Finalmente queremos hacer mejoras en la velocidad de ejecución, sabemos que ya el algoritmo en sí mismo es bastante ineficiente, sin embargo aún puede haber margen para mejorar su tiempo de ejecución.

IV. CONCLUSIONES

La forma de solucionar buscaminas a través de lógica propocicional es bastante interesante y ha demostrado ser un camino que funciona para solucionar el problema, sin embargo debido a su ineficiencia computacional y a la complejidad del algoritmo recomendaríamos otro tipo de solución para estos problemas.

Respecto a lo que desarrollamos pudimos observar las siguientes cosas:

- El algoritmo se comporta bien en muchos casos para solucionar el problema, selecciona bien donde se puede expandir y también donde hay minas marcadas *(la mayoría de las veces)*.

V. BIBLIOGRAFÍA

Wikipedia contributors. (2022, October 25). Minesweeper (video game). In Wikipedia, The Free Encyclopedia, from `https://en.wikipedia.org/w/index.php?title=Minesweeper_(video_game)&oldid=1118169711`