Universidad del Rosario

Automatizando el juego de Buscaminas
Reporte del segundo problema del curso
"IA: Representación y solución de problemas"

Juan Miguel Gutiérrez Vidal
Juan Esteban Murcia

5 de diciembre de 2020

1. Introducción

El juego de buscamunas, o mas conocida como "Minesweeper",fue inventado por Robert Donner en 1989, el objetivo del juego es despejar un campo de minas sin detonar ninguna. Se ha demostrado que el problema del buscaminas es equivalente a un rango de complejidad computacional que sucede en otros juegos importantes en la literatura los cuales se llaman NP-Completos. En ideas generales estos problemas toman mucho tiempo en ser solucionados y a medida que aumenta su complejidad tanto en habilidad como espacio se vuelven mas difíciles de solucionar. Nadie ha demostrado que existe una manera eficiente de solucionarlos, o que por el contrario que no existe una solución a este tipo de problemas.

Buscamininas es un juego que a medida que aumenta el tablero de minas, crece de manera abrupta el espacio de posibles estados. Consideremos w : el ancho del tablero, h :la altura del tablero, k el numero de minas y sea

$$n = w \cdot h$$

El numero de posibles estados, s, se encuentran dados por

$$s = \binom{n}{k} \tag{1.1}$$

Por ejemplo, para un tablero $8 \cdot 8$ con 10 bombas, el numero posible de estados son 151 billones.

1.1. Reglas

- **Objetivo:** Descubrir todo el mapa sin descubrir una mina .
- **Numero de jugadores:** 1.
- **Configuración:** El jugador juega primeramente una casilla aleatoria y a partir de lo que se descubra comienza a jugar "iteligentemente" con las reglas que se le pasan (Figura **??**).
- **Movimientos**:
  - **Bandera:** Pone una bandera en una zona cuando sabe que existe una mina en esa posición.

- **Destapar:** Destapa una casilla.
- **Quitar Bandera:**Quita una bandera cuando se retracta de que ya sabe que ahí no se encuentra una bomba.

- **Bombas:** Si el jugador destapa una casilla donde se encuentra una mina, pierde.

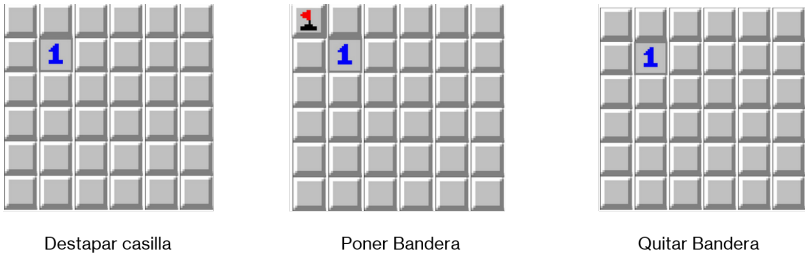

Figura 1: Reglas del jugador para el buscaminas

2. Métodos

La solución al problema requiere una definición formal previa a la implementación en python, que es la siguiente:

Estado Inicial: Situación del entorno desde el cual comienza el juego. En el caso del buscaminas, el estado inicial es el tablero con todas las casillas tapadas.

Funciones de transiciones $((x, y), accion)$: Descripción del entorno que resulta de la ejecución de destapar la casilla (x, y) con alguna acción (destapar,quitar bandera, poner bandera).

Prueba de objetivo (): Permite determinar si el juego se termina cuando se obtiene el estado s. Esto sucede si el jugador reconoció todas las mina que se encontraban en el tablero.

Una vez implementado el problema así definido, se procedió a definir el entorno del jugador:

- **Entorno**: Un mapa representado por una rejilla $n \times n$ bordeada por muros. El agente siempre comienza con todas las casillas tapadas mirando a la derecha. Comienza abriendo una casilla al azar. El numero de bombas en las diferentes casillas equivale al techo del 20 % de las casillas totales.
- **Actuadores**: El jugador puede elegir destapar una casilla o marcar bomba (Poner bandera).
- **Sensores**: El jugador puede percibir todas las casillas que ha decidido destapar y el numero de bombas de banderas que le quedan y cuantas bombas existen en el mapa.
- **Medida de desempeño**: No existe medida desempeño para el juego.

Después procedimos a codificar las casillas de la siguiente manera:

- si la casilla posee valor entre $0 \leq n \leq 8$ representa el numero de bombas n adyacentes a esa casilla.
- **9**: si hay una bomba
- **10** Casilla Tapada
- **11** : Bandera sobre la casilla

3. Agente de Conocimiento

Ahora para crear un agente que se base en el conocimiento, para esto definimos la codificación de las casillas de la siguiente manera para la columna (x') y la fila (y):

- $P(x, y, 10)$ es verdadero sii en la casilla (x, y) se sabe que esta tapada.
- $P(x, y, 0)$ es verdadero sii el agente cree que en la casilla (x, y) no existen casillas adyacentes con bombas.
- $P(x, y, 1)$ es verdadero sii en la casilla (x, y) se percibe 1 bomba adyacente a la casilla.
- $P(x, y, 2)$ es verdadero sii en la casilla (x, y) se percibe 2 bombas adyacentes a la casilla.
- $P(x, y, 3)$ es verdadero sii en la casilla (x, y) se percibe 3 bombas adyacentes a la casilla.
- $P(x, y, 4)$ es verdadero sii en la casilla (x, y) se percibe 4 bombas adyacentes a la casilla.
- $P(x, y, 5)$ es verdadero sii en la casilla (x, y) se percibe 5 bombas adyacentes a la casilla.
- $P(x, y, 6)$ es verdadero sii en la casilla (x, y) se percibe 6 bombas adyacentes a la casilla.
- $P(x, y, 7)$ es verdadero sii en la casilla (x, y) se percibe 7 bombas adyacentes a la casilla.
- $P(x, y, 8)$ es verdadero sii en la casilla (x, y) se percibe 8 bombas adyacentes a la casilla.
- $P(x, y, 9)$ es verdadero sii en la casilla (x, y) se percibe que hay una bomba.

3.1. Reglas de unicidad

Ahora usamos fórmulas de la lógica proposicional para representar aspectos del mundo. Comenzamos con una parte de la información sobre la relación entre el numero de bombas alrededor en una casilla y los caso que no deben suceder (unicidad) . Necesitamos incluir en la base de conocimiento las siguientes fórmulas:

- $$P(0, 0, 1) \rightarrow \big(\neg P(0, 0, 2) \wedge ...\neg P(0, 0, 8)\big)$$

  Esto es , si hay un número 1 en (0,0) no hay los números 2,3,4,5,6,7,8 o bomba en (0,0)

Con el fin de implementar los algoritmos de búsqueda, usaremos fórmulas en forma 'pseudo' cláusulas de Horn por lo que reescribimos las anteriores fórmulas así el conjunto de fórmulas que expresan la fórmula de arriba:

- $P(0, 0, 1) \rightarrow \neg P(0, 0, 2)$
- $P(0, 0, 1) \rightarrow \neg P(0, 0, 3)$

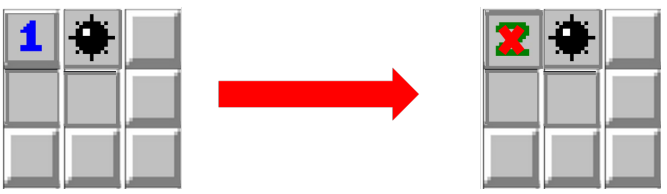

Figura 2: Si existe una bomba adyacente en la casilla (0,0), no es posible que en esa misma casilla hayan dos bombas adyacentes

3.2. Regla de reconocimiento no bomba

Ahora definimos las formulas para que el jugador sepa donde no hay una bomba utilizando la base de conocimiento.

- $$P(0,0,1) \land P(0,1,9) \rightarrow \neg P(1,1,9) \land \neg P(1,1,9)$$

  Esto es , si hay una bomba adyacente en $(0,0)$ y hay una bomba en $(0,1)$ entonces no hay bombas en $(1,1)$ y $(1,0)$

  Esto implica que se cumplen estas dos ecuaciones

- $$P(0,0,1) \land P(0,1,9) \rightarrow \neg P(1,1,9)$$

  Esto es , si hay una bomba adyacente en $(0,0)$ y hay bomba en $(0,1)$, entonces no hay una bomba en $(1,1)$

- $$P(0,0,1) \land P(0,1,9) \rightarrow \neg P(1,0,9)$$

  Esto es , si hay una bomba adyacentes en $(0,0)$ y hay bomba en $(0,1)$, entonces no hay una bomba en $(1,0)$

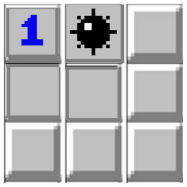

Figura 3: Esto es , si hay una bomba adyacente en $(0,0)$ y hay una bomba en $(0,1)$ entonces no hay bombas en $(1,1)$ y $(1,0)$

3.3. Regla reconocimiento bomba

posteriormente definimos las formulas para que el jugador sepa donde no hay una bomba o se encuentre una bomba utililizando la base de conocimiento.

- $$P(0,0,1) \land \neg P(0,1,9) \land \neg P(1,0,9) \rightarrow P(1,1,9)$$

  Esto es , si hay una bomba adyacente en $(0,0)$ y no hay bombas en $(0,1)$ y $(1,0)$, entonces hay una bomba en $(1,1)$

La ecuación de arriba es un caso simple, sin embargo suceden casos mas complicados cuando hay dos bombas adyacentes, pero la idea sigue siendo la misma por ejemplo

- $$P(0,0,2) \land \neg P(0,1,9) \rightarrow P(1,1,9)$$

  Esto es, si hay dos bombas adyacentes en $(0,0)$ y no hay bomba en $(0,1)$, entonces hay una bomba en $(1,1)$

- $$P(0,0,2) \wedge \neg P(0,1,9) \rightarrow P(1,0,9)$$

  Esto es, si hay dos bombas adyacentes en $(0,0)$ y no hay bomba en $(0,1)$, entonces hay una bomba en $(1,0)$

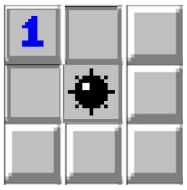

Figura 4: Si hay una bomba adyacente en $(0,0)$ y no hay bombas en $(0,1)$ y $(1,0)$, entonces hay una bomba en $(1,1)$

4. Resultados

El programa de agente se implementó para desde que se inicia con alguna casilla aleatoria descubierta, de tal manera el computador corre el algoritmo `and or search` sobre el tablero resultante y juega. Al comienzo del juego existen "pocas reglas", por lo que el computador debe generar un árbol relativamente pequeño Dado que el agente genera el árbol a medida que se va jugando, previendo todos las reglas posibles y sus acciones, puede suceder el caso donde suceda que el agente necesite volver a elegir una casilla aleatoriamente. Si aleatoriamente eligió una bomba pierde, si no, puede volver a obtener información y jugar mejor.

5. Discusión

El agente de conocimiento y el algoritmo de búsqueda en la base de conocimiento le falta una regla importante, pero que sin embargo aumenta de manera exponencial el numero de reglas necesarias, y es la regla que debe tener el numero adyacentes de cada casilla para poder determinar correctamente donde se encuentra o no una bomba, sin embargo, recursivamente crece mas que linealmente a medida que se destapa el problema, convirtiendo un problema NP-completo.

Seria interesante que pueda revisar primero si no es capaz de determinar una casilla libre usando las reglas en

- Se implementó un programa de agente para el juego de las mini-damas que permite hacer simulación.
- Es posible notar que a medida que aumenta la complejidad de decisión y espacio, es necesario a recurrir algoritmos más eficientes o heurísticas mejores.
- los computadores tiene mas poder de predicción que los humanos en general, si son entrenados para ello. Esto es mediante la capacidad que tienen de "mirar al futuro" en el sentido de predecir cualquier movimiento del adversario.
- Si se encontrara un algoritmo que permita solucionar un problema NP en tiempo polinomial o en un rango de tiempo considerable, el problema buscaminas se podría solucionar en tiempo récord.

6. Conclusiones

Del trabajo se obtienen las siguientes conclusiones:

- Las reglas tienden a aumentar a medida que la complejidad del juego crece,luego es necesario buscar mejores heurísticas o métodos de solucionarlo
- Con un poder de computación mas fuerte es posible solucionar tableros medianamente pequeños.
- Las reglas y clausulas de Horn son útiles al darle una guía al agente de que comienzc a "pensar" por si mismo.

Referencias

[1] Magnushoff Hovland Hoff. (2020). Solving Minesweeper. [ONLINE] Available at: https://magnushoff.com/articles/minesweeper/. [Accessed 20 October 2020].

[2] Richard Kaye. (2000). Minesweeper is NP-complete. Mathematical intelligences. Pag 9.