



# IA - TP02

## Buscaminas

Alumno:  
Castellano, Valentin

# Temas de la presentación

.....

**01** Marco teorico

**02** Buscaminas

**03** Sentencias  
logicas

**04** La IA



.....

**01**

# Marco teorico

# Agentes basados en conocimiento

Razonamiento basado en  
sentencias







# Sentencias

Cada sentencia lógica se refiere a una proposición. Ejemplo:

1- Hoy está soleado o está nublado.





**Una sola proposición no nos  
permite concluir nada, pero...**

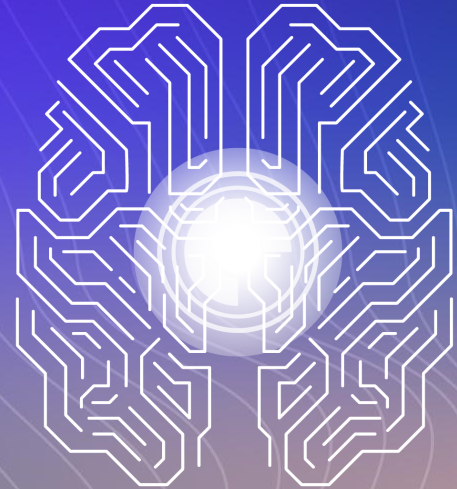
# Proposiciones

Si añadimos otra proposición...:

- 1- Hoy está soleado o está nublado.
- 2- Hoy no está nublado



Podemos inferir entonces:

- 3- Hoy está soleado





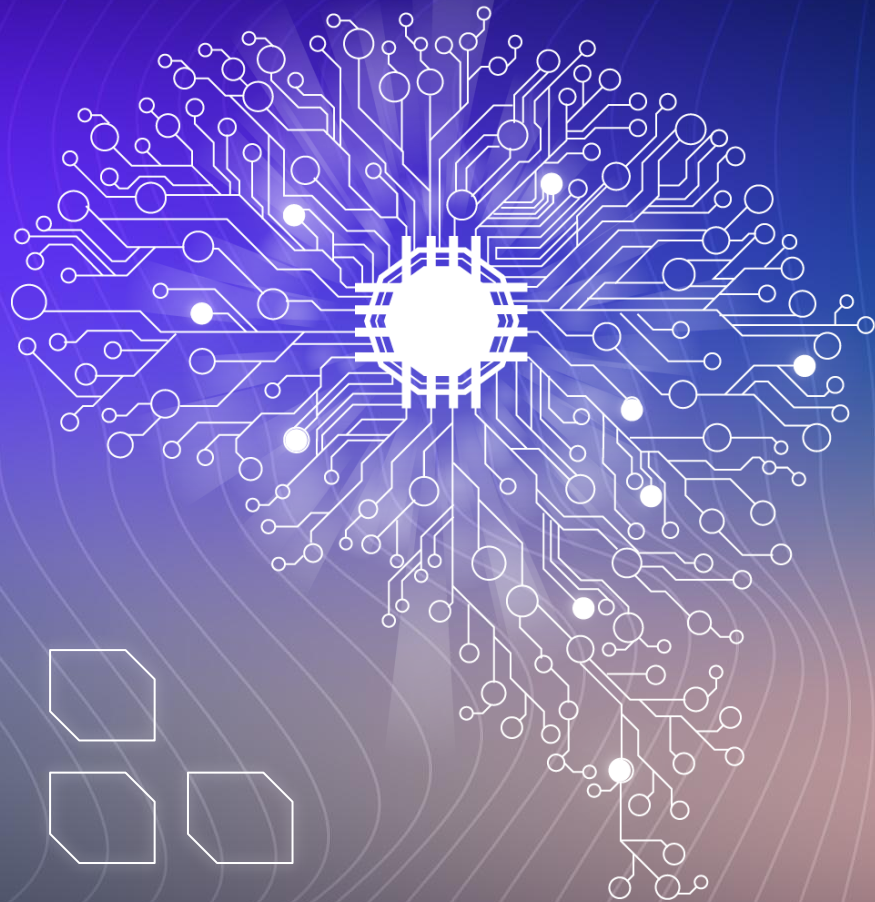
**Esto es lo que conocemos  
como lógica proposicional**





# Inferencia

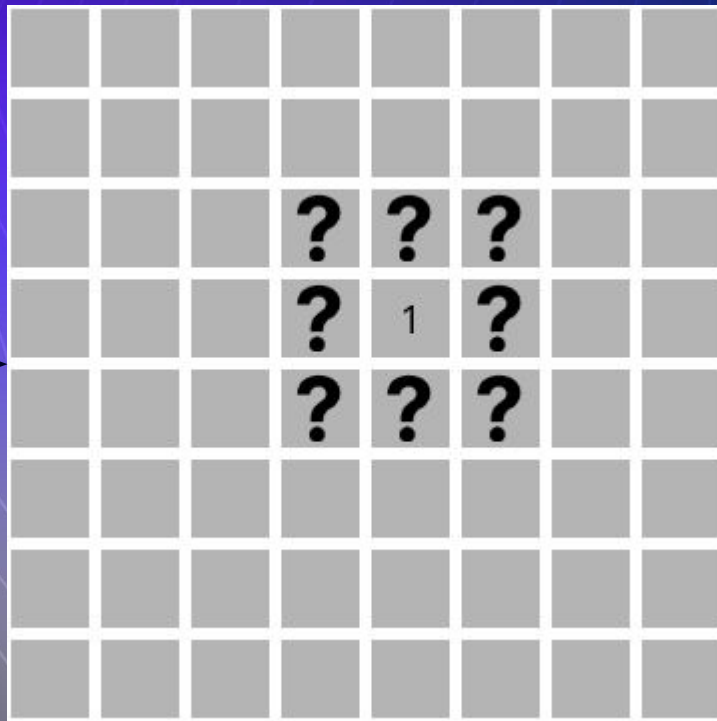
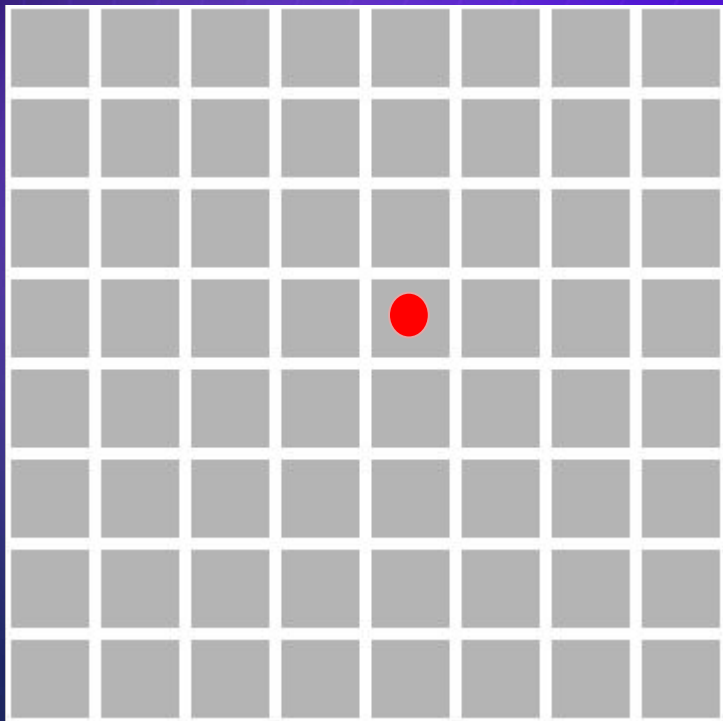
Derivar sentencias nuevas a partir de sentencias existentes.





**02**

# Buscaminas





.....

**03**

# Sentencias logicas



			H	A	B		
			G	1	C		
			F	E	D		

**$\{A, B, C, D, E, F, G, H\} = 1$**

*Cada celda es representada  
con su coordenada*

*$A = [2][4] = \text{Fila 3, columna 5}$*

*Dado que se empieza desde  
 $[0][0]$*

```
class Sentence():
```

```
    """
```

```
    Clase que representa una sentencia lógica sobre el juego del Buscaminas.  
    Una sentencia consta de un conjunto de celdas del tablero  
    y el número de minas en esas celdas.
```

```
    """
```

```
def __init__(self, cells, count):
```

```
    """
```

```
    Inicializa una nueva sentencia con el conjunto de celdas y el el número de minas.  
    Args:
```

```
        cells (set): Conjunto de coordenadas de celdas del tablero.
```

```
        count (int): Número de minas en las celdas dadas.
```

```
    """
```

```
    self.cells = set(cells)
```

```
    self.count = count
```

# Funciones de análisis

```
def known_mines(self):  
    """  
    Devuelve el conjunto de todas las celdas en la sentencia que se sabe que son minas.  
  
    Returns:  
    | set: Conjunto de celdas conocidas como minas.  
    """  
    #solo puedo estar seguro si las celdas contienen minas  
    # si la sentencia contiene el mismo numero de celdas que el contador de minas  
    #EJ: A, B, C = 1  
    #No puedo afirmar nada dado que cualquiera de las 3 podria contener la mina,  
    # por lo que devuelvo un set vacio  
    if len(self.cells) == self.count:  
        return self.cells.copy()  
    return set()
```



```
def known_safes(self):
```

```
    """
```

```
    Devuelve el conjunto de todas las celdas en la sentencia que se sabe que son seguras.
```

```
    Returns:
```

```
    | set: Conjunto de celdas conocidas como seguras.
```

```
    """
```

```
    #solo puedo estar seguro si las celdas son seguras si el contador de minas es 0
```

```
    if self.count == 0:
```

```
        return self.cells.copy()
```

```
    return set()
```

# Funciones de mercado

```
def mark_mine(self, cell):
```

```
    """
```

Actualiza la representación interna del conocimiento  
dado que puedo afirmar que una celda es una mina.

Args:

| cell (tupla): Coordenadas (fila, columna) de la celda conocida como mina.

```
    """
```

```
if cell in self.cells:
```

```
    self.cells.remove(cell)
```

```
    self.count -= 1
```

```
def mark_safe(self, cell):
```

```
    """
```

Actualiza la representación interna del conocimiento  
dado que puedo afirmar que una celda es segura.

Args:


| cell (tupla): Coordenadas (fila, columna) de la celda conocida como segura.

```
    """
```

```
if cell in self.cells:
```

```
    self.cells.remove(cell)
```






.....

**solo queremos que nuestras  
sentencias sean sobre  
objetos cells que aún no  
se sabe que sean seguros o  
minas.**

.....





04

La IA

```
class MinesweeperAI():
    """
    Clase que representa una IA que puede jugar al Buscaminas.
    """

    def __init__(self, height=8, width=8):
        """
        Inicializa la IA jugadora de Buscaminas.

        Args:
            height (int): Altura del tablero.
            width (int): Ancho del tablero.

        Atributos:
            height (int): Altura del tablero.
            width (int): Ancho del tablero.
            moves_made (set): Conjunto de celdas donde se ha hecho un movimiento.
            mines (set): Conjunto de celdas conocidas como minas.
            safes (set): Conjunto de celdas conocidas como seguras.
            knowledge (list): Lista de sentencias lógicas sobre el juego.
        """
```

# Funciones de mercado



```
def mark_mine(self, cell):  
    """  
    Marca una celda como mina y actualiza todo el conocimiento.  
  
    Args:  
    |   cell (tupla): Coordenadas (fila, columna) de la celda conocida como mina.  
    """  
    self.mines.add(cell)  
    for sentence in self.knowledge:  
        sentence.mark_mine(cell)
```

```
def mark_safe(self, cell):
```

```
    """
```

```
    Marca una celda como segura y actualiza todo el conocimiento.
```

```
    Args:
```

```
    | cell (tupla): Coordenadas (fila, columna) de la celda conocida como segura.
```

```
    """
```

```
    self.safes.add(cell)
```

```
    for sentence in self.knowledge:
```

```
    | sentence.mark_safe(cell)
```

# Función de conocimiento

```
def add_knowledge(self, cell, count):
```

```
    """
```

Se llama cuando el tablero Buscaminas nos dice, para una celda segura determinada, cuántas celdas vecinas tienen minas.

Esta función:

- 1) marca la celda como un movimiento que se ha realizado
- 2) marca la celda como segura
- 3) agrega una nueva oración a la base de conocimientos de la IA  
basado en el valor de "celda" y "recuento"
- 4) marca celdas adicionales como seguras o como minas  
si se puede concluir basándose en la base de conocimientos de la IA
- 5) agrega nuevas oraciones a la base de conocimientos de la IA  
si pueden inferirse del conocimiento existente

```
    """
```



```
# marca la celda como un movimiento que se ha realizado  
self.moves_made.add(cell)
```

```
# marca la celda como segura  
self.mark_safe(cell)
```

```
# agrega una nueva sentencia a la base de conocimientos de la IA
# basado en el valor de "cell" y "count"
nearby_cells = []
# obtengo todas las celdas alrededor de la celda a la que me movi
# (sin salir de los extremos del tablero)
for i in range(max(0, cell[0] - 1), min(self.width, cell[0] + 2)):
    for j in range(max(0, cell[1] - 1), min(self.height, cell[1] + 2)):

        current_cell = (i, j)

        if (i, j) == cell:
            continue

        if current_cell not in self.safes:
            nearby_cells.append(current_cell)

# genero una sentencia nueva la cual tenga
# todas las celdas alrededor de la cual me movi, con el contador de minas
generated_sentence = Sentence(nearby_cells, count)

# agrego la sentencia la base de conocimiento
self.knowledge.append(generated_sentence)
```

```
# marca celdas adicionales como seguras o como minas
# si se puede concluir basándose en la base de conocimientos de la IA
for sentence in self.knowledge:

    if len(sentence.known_mines()) > 0:
        for mine in sentence.known_mines():
            if mine not in self.mines:
                self.mark_mine(mine)

    if len(sentence.known_safes()) > 0:
        for safe in sentence.known_safes():
            if safe not in self.safes:
                self.mark_safe(safe)
```

```
# genero nuevas sentencias
# si pueden inferirse del conocimiento existente
new_sentences = []
#obtengo 2 sentencias a comparar de mi base de conocimientos
for subset_sentence in self.knowledge:
    for main_sentence in self.knowledge:

        if subset_sentence == main_sentence:
            continue

        #comparo si una sentencia es un subconjunto de la otra
        if subset_sentence.cells.issubset(main_sentence.cells):

            #genero una nueva sentencia inferida de los 2 sentencias
            new_sentence_cells = main_sentence.cells - subset_sentence.cells
            new_sentence_count = main_sentence.count - subset_sentence.count
            new_sentence = Sentence(new_sentence_cells, new_sentence_count)

            #verifico que esa sentencia no se encuentre ya
            # en la base de conocimiento y la agrego a mi nuevo conocimiento
            if new_sentence not in self.knowledge:
                new_sentences.append(new_sentence)
```



```
# Vuelvo a verificar si puedo marcar celdas como seguras o como minas
# en base a las nuevas sentencias y las agrego a mi base de conocimiento principal

for sentence in new_sentences:
    for sentence_cells in sentence.cells:
        if sentence_cells == cell:
            continue

        if sentence_cells not in self.mines and sentence_cells not in self.safes:
            if sentence.count == 0:
                self.mark_safe(sentence_cells)
            elif sentence.count == len(sentence.cells):
                self.mark_mine(sentence_cells)

    if sentence not in self.knowledge:
        self.knowledge.append(sentence)
```

# Funciones de movimiento

```
def make_safe_move(self):  
  
    """  
    Devuelve una celda segura para elegir en el tablero del Buscaminas.  
  
    Returns:  
    | tupla: Coordenadas de la celda segura o None si no hay celdas seguras disponibles.  
    """  
  
    #Devuelve la primera celda segura que encuentre comenzando en el (0,0)  
  
    for i in range(self.height):  
        for j in range(self.width):  
            cell = (i, j)  
            if cell not in self.moves_made and cell in self.safes:  
                return cell  
    return None
```

```
def make_random_move(self):  
    """  
    Devuelve un movimiento aleatorio para hacer en el tablero del Buscaminas.  
  
    Returns:  
    | tupla: Coordenadas del movimiento aleatorio o None si no hay movimientos.  
    """  
    cells = []  
  
    #Calcula todos los movimientos aleatorios posibles y luego elige uno.  
    for i in range(self.height):  
        for j in range(self.width):  
            cell = (i, j)  
            if cell not in self.moves_made and cell not in self.mines:  
                cells.append(cell)  
    if len(cells) > 0:  
        choice = random.choice(cells)  
        print("movimiento realizado a "+str(choice[0]+1)+str(choice[1]+1))  
        return choice  
  
    return None
```



**Gracias por  
su atención!**