Proyecto 2

Juan Pablo Ante - 2140132

Nicolas Garcés - 2180066

Alejandro Guerrero - 2179652

Alejandro Zamorano - 1941088

Universidad del Valle

Inteligencia Artificial

24 de mayo del 2024

Descripción del Juego

"Yoshi's World" es un juego de estrategia para dos jugadores donde cada participante controla un Yoshi, uno verde y el otro rojo. Estos personajes se mueven como un caballo en el ajedrez sobre un tablero de 8x8. Cada casilla que alcanza un Yoshi se pinta del color correspondiente y queda bloqueada, es decir, no puede ser utilizada nuevamente por ningún jugador. El juego concluye cuando ninguno de los dos jugadores puede realizar más movimientos. El ganador es aquel que haya pintado más casillas con su color al final del juego.

Niveles de Dificultad

El juego presenta tres niveles de dificultad:

- Principiante: Árbol Minimax de profundidad 2.
- Amateur: Árbol Minimax de profundidad 4.
- Experto: Árbol Minimax de profundidad 6.

Función Minimax

```python
def minimax(nodo, profundidad, es_max):
    if profundidad == 0 or not nodo.hijos:
        return heuristica(nodo.estado, nodo.estado.yoshi_verde if es_max else nodo.estado.yoshi_rojo)

    if es_max:
        mejor_valor = float("-inf")
        for hijo in nodo.hijos:
            valor = minimax(hijo, profundidad - 1, False)
            mejor_valor = max(mejor_valor, valor)
        return mejor_valor
    else:
        mejor_valor = float("inf")
        for hijo in nodo.hijos:
            valor = minimax(hijo, profundidad - 1, True)
            mejor_valor = min(mejor_valor, valor)
        return mejor_valor
```

- Esta función toma como argumento" nodo", "profundidad", y "es_max", que representan el nodo actual, la profundidad máxima a explorar y un booleano indicando si es el turno de maximizar.
- Crea un bucle que se ejecutará mientras haya nodos en la pila stack para explorar. Este bucle permite la exploración de nodos en el árbol de juego.
    - ✓ En el caso base del bucle; Si la profundidad es 0 o el nodo no tiene hijos (not nodo.hijos), retorna la heurística del nodo usando la función heuristica.En este caso la función retorna una evaluación heurística del estado actual del juego.
- Exploración de nodos hijos, esta funcion alterna entre maximizar y minimizar el valor de los nodos dependiendo de si es el turno de Yoshi o su oponente.
- Si "es_max es verdadero", inicializa "mejor_valor" a -inf y busca maximizar el valor entre los nodos hijos, para cada hijo en nodo.hijos, llama recursivamente a minimax con profundidad - 1 y False para minimizar y al terminar actualiza "mejor_valor" con el valor máximo encontrado.
- Si "es_max" es falso, inicializa "mejor_valor a inf" y busca minimizar el valor entre los nodos hijos. Para cada hijo en nodo.hijos, llama recursivamente a minimax con profundidad - 1 y "True" para maximizar y por último Actualiza mejor_valor con el valor mínimo encontrado.

- Al finalizar Retorna “mejor_valor” como el mejor valor encontrado en el árbol de decisiones donde este valor representa la mejor evaluación posible para el jugador en el estado actual

Función Obtener Mejor Movimiento:

```
def obtener_mejor_movimiento(ambiente, profundidad):
    mejor_valor = float("-inf")
    mejor_movimiento = None
    for movimiento in ambiente.obtener_casillas_disponibles(ambiente.yoshi_verde):
        copia_ambiente = copy.deepcopy(ambiente)
        copia_ambiente.realizar_movimiento(copia_ambiente.yoshi_verde, *movimiento)
        nodo = Nodo(copia_ambiente)
        generar_hijos(nodo, copia_ambiente.yoshi_rojo)
        valor = minimax(nodo, profundidad, False)
        if valor > mejor_valor:
            mejor_valor = valor
            mejor_movimiento = movimiento
    return mejor_movimiento
```

- Esta función toma como argumento ambiente y profundidad, que representan el estado inicial del ambiente y la profundidad máxima del árbol de búsqueda lo cual depende del nivel de dificultad seleccionado, donde se busca el mejor movimiento posible para Yoshi utilizando el algoritmo Minimax.
- Se inicializa mejor_valor a -inf y mejor_movimiento a None ya que estas variables almacenan el mejor valor y el movimiento correspondiente encontrados hasta el momento.
- Crea un bucle que itera sobre todos los movimientos posibles de Yoshi usando obtener_casillas_disponibles(ambiente.yoshi_verde) este bucle Evalúa todos los posibles movimientos de Yoshi y selecciona el que proporciona el mejor resultado según la evaluación heurística y el algoritmo Minimax.
- Para cada movimiento, se crea una copia del ambiente usando copy.deepcopy y se realiza el movimiento en la copia del ambiente (copia_ambiente.realizar_movimiento) creando un nuevo nodo con el “copia_ambiente” , tambien se generan los hijos correspondientes del “nodo” con “generar_hijos”

- Al finalizar se hace la evaluación del “nodo” usando “minimax” con la “profundidad” y si el valor obtenido es mejor que “mejor_valor” se hace la actualización correspondiente en “mejor_valor” y “mejor_movimiento” y se retorna “mejor_movimiento” como el mejor movimiento encontrado el cual sera el movimiento óptimo para Yoshi en el estado actual del juego.

Función Generar Hijos:

```python
def generar_hijos(nodo, yoshi):
    ambiente = nodo.estado
    casillas_disponibles = ambiente.obtener_casillas_disponibles(yoshi)
    for fila_destino, columna_destino in casillas_disponibles:
        ambiente_temporal = copy.deepcopy(ambiente)  # Asegurarse de hacer una copia profunda aquí también
        ambiente_temporal.realizar_movimiento(yoshi, fila_destino, columna_destino)
        nodo_hijo = Nodo(ambiente_temporal)
        nodo.hijos.append(nodo_hijo)
```

- Esta función toma como argumento nodo y yoshi, que representan el nodo actual y el Yoshi para el cual se generarán los movimientos. En esta funcion lo que se busca es generar todos los posibles movimientos del Yoshi y crear nodos hijos para cada uno de estos movimientos.
- Primero se obtienen las casillas disponibles para Yoshi usando ambiente.obtener_casillas_disponibles(yoshi), es decir las casillas en las que puede moverse.
- Luego se hace una simulación de movimientos para cada “fila_destino” y “columna_destino” en “casillas_disponibles” para ello se crea una copia del ambiente (ambiente_temporal) usando copy.deepcopy y se realiza el movimiento en esta copia del ambiente. Esto se hace con el fin de Simular el efecto de cada movimiento posible en una copia del estado actual del juego.
- Por último se Agrega cada nuevo estado resultante de los movimientos posibles como un nodo hijo del nodo actual en el árbol de búsqueda.

Explicación de la heurística

Diferencia en casillas pintadas entre Yoshi y el oponente:

```python
def heuristica(ambiente, yoshi):
    # Contar la cantidad de casillas pintadas por el yoshi y su oponente
    casillas_pintadas_yoshi = ambiente.casillas_pintadas_verde
    casillas_pintadas_oponente = ambiente.casillas_pintadas_rojo

    # Calcular la cantidad de casillas disponibles para el yoshi
    casillas_disponibles = len(ambiente.obtener_casillas_disponibles(yoshi))

    oponente = ambiente.yoshi_rojo
    mov_posibles_oponente = len(ambiente.obtener_casillas_disponibles(oponente))

    # Calcular la heurística combinando los aspectos anteriores
    return (casillas_pintadas_yoshi - casillas_pintadas_oponente) + (casillas_disponibles - mov_posibles_oponente)
```

La heurística comienza calculando la diferencia entre la cantidad de casillas pintadas por Yoshi y las casillas pintadas por el oponente.

Esta diferencia da una indicación directa de quién tiene una ventaja en términos de control del tablero. Un valor positivo indica que Yoshi tiene más casillas pintadas que su oponente.

Casillas disponibles para Yoshi y el oponente:

Luego, la heurística calcula la cantidad de casillas disponibles para moverse para Yoshi y su oponente.

Las casillas disponibles reflejan la libertad de movimiento de cada jugador. Una mayor cantidad de movimientos disponibles para Yoshi (en comparación con el oponente) muestra una ventaja ya que tiene mayor posibilidad de movimientos.

Calcular la heurística combinada:

Finalmente, la heurística combina la diferencia en casillas pintadas y la diferencia en movimientos disponibles en una única medida heurística. Esta combinación proporciona una evaluación más completa del estado del juego, ya que considera tanto el control del tablero como las opciones de movimiento futuro. Lo que da como resultado que la heurística será más alta cuando Yoshi tenga más casillas pintadas y más opciones de movimiento que su oponente, indicando un estado de juego favorable para Yoshi.

Admisibilidad de la Heurística

Subestimación

Casillas pintadas:

La diferencia entre las casillas pintadas por Yoshi y las casillas pintadas por el oponente. Este componente de la heurística mide la ventaja en el número de casillas pintadas y siempre proporciona una estimación conservadora, ya que no considera el costo total de todos los movimientos, sino solo la diferencia acumulada hasta el momento.

Casillas disponibles:

La diferencia en las casillas disponibles para moverse. Este componente mide la movilidad relativa de los Yoshis. La cantidad de casillas disponibles no sobreestima el costo real de los movimientos futuros, ya que solo contabiliza las posibilidades actuales sin prever todos los movimientos necesarios para llegar a una casilla específica.

Dado que ambos componentes son conservadores y no exceden el costo real del juego, la heurística es admisible.

Fórmula de Subestimación

$$h\,(n) \leq h\,*(n)$$

Donde:

h(n) es la heurística para el nodo

h∗(n) es el costo real mínimo desde el nodo n hasta el objetivo.

Consistencia de la Heurística

Para demostrar la consistencia de la heurística, necesitamos verificar que para cada nodo n y cada sucesor n′

$$h(n') \leq c\,(n,n') + h\,(n)$$

Donde:

- ***h(n')*** es la heurística del sucesor del nodo actual.
- ***c(n,n')*** es el costo real de moverse del nodo n al nodo n′ (en este caso, siempre 1, ya que cada movimiento tiene un costo de 1).
- ***h(n)*** es la heurística del nodo actual.

Explicación

Casillas pintadas:

La diferencia en casillas pintadas no se incrementa más allá del costo real del movimiento. Esto se debe a que cada movimiento incrementa o decrementa la cuenta en 1 casilla pintada, y la heurística simplemente ajusta la diferencia de casillas pintadas entre los dos jugadores.

Casillas disponibles:

La diferencia en las casillas disponibles refleja la movilidad de los jugadores después de cada movimiento. Dado que cada movimiento afecta solo las casillas adyacentes, la heurística se ajusta adecuadamente sin exceder el costo real del movimiento.

Fórmula de Consistencia

Para cada nodo ***n*** y su sucesor ***n'***: $h(n') \leq 1 + h(n)$

Esto asegura que la heurística cumple con la propiedad de triángulo, lo que significa que no sobreestima el costo adicional de moverse de un nodo a otro.

Conclusión

La heurística utilizada en Yoshi's World es tanto admisible como consistente:

- **Admisible** porque nunca sobreestima el costo real de alcanzar el objetivo.
- **Consistente** porque para cada nodo y su sucesor, la estimación de la heurística desde el nodo inicial hasta el sucesor más el costo de llegar al sucesor es menor o igual que la estimación de la heurística desde el nodo inicial hasta el sucesor.