Universidad Nacional de Colombia
Facultad de Ciencias - Departamento de Matemáticas
Introducción a la Inteligencia Artificial - 2025 II

Mateo Sebastian Ortiz Higuera mortizhi@unal.edu.co

Reporte Segundo Proyecto
Agente de Battle City basado en Búsqueda Adversaria

1. Introducción

La implementación de agentes sobre juegos de estrategia en tiempo real es un problema bastante interesante desde múltiples puntos de vista. El principal problema encontrado en la implementación de este tipo de agentes tiene que ver con el enorme espacio de búsqueda generado al utilizar algoritmos de planificación adversaria como Minimax y Expectimax [3], pues el factor de ramificación hace inviable el uso de estos algoritmos. Múltiples autores encuentran que simplificar el estado de juego y el uso de algoritmos menos costosos como el Monte Carlo Tree Search proporciona una metodología para hacer viable el uso de agentes de búsqueda adversaria en estos juegos [1][4][2].

En este reporte vamos a abordar un juego que es un shooter multidireccional clásico, pero en donde la estrategia en tiempo real es clave para lograr ganar; una versión simplificada del videojuego Battle City, desarrollado y publicado por Namco en 1985.

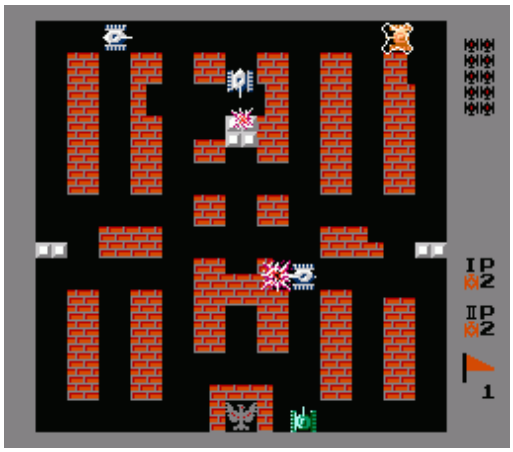

Figura 1: Ejemplo de una partida de Battle City.

En este videojuego el jugador se encuentra en un escenario de tamaño 13×13 en donde deberá defender una base de varios tanques enemigos que irán apareciendo. El jugador ganará al eliminar

todos los enemigos que tienen en reservas; por otra parte será derrotado si pierde todas las vidas o si la base es destruida. Existen diversos elementos que hacen más interesante el juego, como diverso tipo de paredes, de tanques enemigos, power-ups, entre otros.

En este trabajo vamos a implementar un agente que juega sobre algunos niveles extraídos utilizando los algoritmos de minimax y expectimax.

2. Definición de algoritmos

2.1 Estados del juego

En esta implementación vamos a simplificar aún más el juego, dada la naturaleza del juego:

- Solo hay dos tipos de paredes: de ladrillo que son destructibles como en el juego original y de metal que son indestructibles.
- No hay power-ups.
- Todos los tanques son del mismo tipo: todos son iguales en cuanto a salud, velocidad y capacidad de disparos. En particular se ha implementado un algoritmo para que los tanques sean bastante agresivos e intenten priorizar ir siempre a la base.
- No hay tanques en reserva y el jugador solo tiene 2 vidas.
- Una partida dura máximo 500 ticks.

Todo esto para que sea aún más abordable una implementación de los algoritmos y las partidas sean más rápidas.

En este sentido, los estados son la representación del escenario en cada instante, tomando toda la información de los elementos del juego:

- *Jugador*: posición, salud.
- *Enemigos*: posición, salud, está muerto.
- *Base*: posición, está destruida.
- *Paredes*: posición, tipo, salud, está destruida.
- *Balas*: posición, id del disparador, equipo al que pertenece.

Vamos a tomar cada tres estados expandidos como un instante de juego (tick)

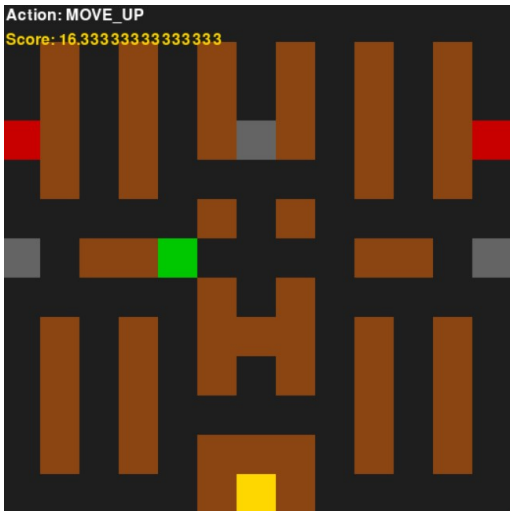

Figura 2: Un instante del juego en donde el jugador es el cuadro verde y los enemigos son los cuadros rojos.

2.2 Función Sucesora

Para la función sucesora vamos a considerar todos los posibles movimientos y disparos que tenga el agente en ese instante. En el peor de los casos un agente puede moverse hacia cualquiera de las cuatro direcciones y además tiene a los dos tanques enemigos en punto de mira (no hay paredes de por medio). En particular, las acciones disponibles para la función sucesora son:

- Movimiento a cualquiera de las cuatro direcciones (arriba,abajo,izquierda,derecha) si no hay paredes, tanques o demás obstáculos.
- Disparar a alguna dirección si hacia allí hay un tanque enemigo y no hay más de 1 obstáculo.
- No hacer nada.

En el peor de los casos, cuando los tres agentes tienen todas acciones disponibles, la función sucesora en un tick puede expandir un total de $\sum_{i=0}^{3} |\mathcal{A}|^i = 7^0 + 7^1 + 7^2 + 7^3 = 400$ nodos, en donde $\mathcal{A}$ es el conjunto de todas las acciones disponibles. Para nuestra implementación vamos a considerar 3 niveles de profundidad, luego en el peor de los casos se pueden expandir un total de $\sum_{i=0}^{3} 7^i = 47079208$ nodos.

2.3 Función de evaluación

Para la función de evaluación hacemos un equilibrio entre ser agresivo y cuidar la base. Formalmente el score se evalúa con la siguiente suma ponderada:

$$score = defensa + ataque + bonus - peligro - tiempo$$

en donde

$$
\begin{aligned}
defensa &= 50 \times \frac{1}{d_1(posJugador, posBase) + 1} \\
ataque &= 100 \times \frac{1}{\text{mín}_E\left(d_1(posJugador, posEnemigo)\right) + 1} \\
bonus &= 30 \times \frac{1}{d_1(posJugador, posEnemigo) + 1} \\
peligro &= \sum_E (10 + d_1(posEnemigo, posBase))^2 \\
tiempo &= 5 \times \text{tick actual}
\end{aligned}
$$

en donde E es el conjunto de enemigos vivos y d_1 es la distancia Manhattan. Además, $defensa$ solo se activa cuando algún enemigo está a distancia cercana a la base $d_1(posEnemigo, posBase) < 8$, $bonus$ se activa cuando solo queda un enemigo vivo y $peligro$ solo se activa cuando alguno de los dos enemigos están cerca a la base $d_1(posEnemigo, posBase) < 10$, si son los dos entonces se da la suma.

2.4 Algoritmos de Búsqueda

Para las pruebas se implementaron 4 agentes, uno para los bots enemigos, un agente reflejo con dos estrategias y Minimax con poda AlphaBeta y Expectimax.

- *Script para bots:* para el comportamiento de los bots se intentó que se comportara de forma similar a como los bots se comportan en el juego original. Este agente hace dos cosas: primero, con un 60 % de probabilidad elige disparar a cualquier dirección, si elige no disparar entonces calcula la distancia entre las posibles posiciones del siguiente tick y la posición de la base y con una probabilidad del 70 % elige esta acción.
- *Agente reflejo:* es bastante similar al comportamiento de los bots, pero en este tenemos dos estilos de juego *ofensivo* y *defensivo*. En el primero se verifica si hay enemigos hacia alguna dirección para elegir disparar, si no entonces se calcula la distancia hacia el enemigo más cercano y se elige esta acción. En la segunda se hace la misma verificación para disparar, pero si no se encuentra un enemigo entonces se calcula la distancia hacia la base y se elige la acción que haga que el agente se encuentra más cerca de la misma.
- *AlphaBeta y Expectimax:* se han implementado con profundización iterativa (iterative deepening), para que la búsqueda se realice sobre un máximo de niveles de profundidad, en este caso elegimos 3 niveles de búsqueda. Además, con un límite de tiempo que hemos elegido de 10 segundos, esto con el objetivo de mitigar el tiempo de ejecución y de simulación de estados que, en esta implementación se hace costoso[1]; se decide que el agente tome una decisión desde el agente reflejo, y con una probabilidad del 50 % se elige si es ofensiva o defensiva.

[1]Cuando hay demasiados elementos en el mapa, por los disparos, sumado al número de estados que se expanden, el cálculo de impactos y destrucción de paredes, agentes y base se vuelven costosos.

3. Resultados

Se experimentó con los tres algoritmos (reflejo, AlphaBeta y Expectimax) simulando 10 partidas. Para AlphaBeta y Expectimax utilizamos una profundidad de 3 y un tiempo de ejecución de 10 segundos por búsqueda. Los códigos de la implementación y el cuaderno de de los experimentos se encuentran en el repositorio.[2]

Medimos los resultados, el tiempo de ejecución de partida y el número medio de nodos expandidos por partida. En la figura 3 se puede ver los resultados de las diez simulaciones sobre los tres algoritmos, en donde se puede observar que con Expectimax el agente gana seis partidas de las diez, mientras que con los otros dos algoritmos gana tan solo cuatro. En la figura 4 se puede observar los tiempos de ejecución y el número promedio de nodos expandidos por simulación, en donde se puede observar que, como se espera, el agente reflejo tarda en ejecución bastante poco y el promedio de nodos expandidos es el también mínimo; con Expectimax se observa que el tiempo de ejecución ronda entre los 50 segundos y los 200 segundos, mientras que con Alpha Beta tarda entre 100 y 400 segundos; en el número promedio de nodos expandidos se observa que Expectimax expande entre 5000 y 25000 nodos por búsqueda, mientras que con Alpha Beta el número de nodos expandidos se acercan bastante más al agente reflejo.

Expectimax		
W	L	D
6	4	0

Alpha Beta		
W	L	D
4	6	0

Reflex Agent		
W	L	D
4	6	0

Figura 3: Comparación de resultados entre los algoritmos Expectimax, Alpha Beta y Reflex Agent. En donde W representa las victorias del jugador, L las derrotas y D los empates.

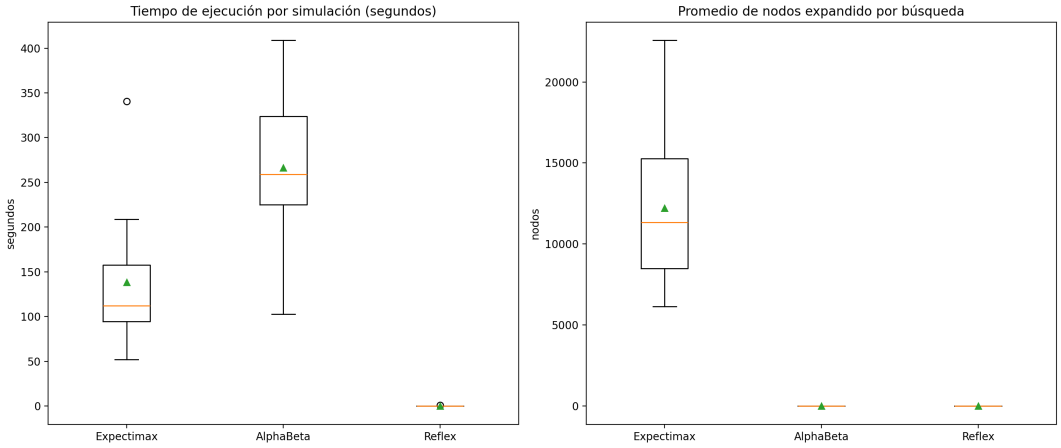

Figura 4: Comparación entre los algoritmos con el tiempo de simulación y número promedio de nodos expandidos por búsqueda.

[2] `https://github.com/MateoOrtiz001/BattleCity-Multi-Agent`.

4. Conclusiones

- De los experimentos realizados se puede observar que el algoritmo de Expectimax se adapta mejor para enfrentar estos escenarios en donde existen bots que toman decisiones de forma aleatorizada.
    - Sin embargo, en general se observa que este algoritmo es el que más memoria gasta al expandir más nodos, además de tener un tiempo de ejecución relativamente alto.
- Entre el tiempo de ejecución y los resultados se puede evidenciar que el algoritmo de Minimax no es tan bueno para modelar y realizar las simulaciones del juego.
- La implementación del problema no fue la más óptima, se podría mejorar los tiempos de ejecución y de rendimiento optimizando la forma de representar los estados y los objetos del juego.
    - Por esto mismo se podría considerar aumentar el número de simulaciones para obtener resultados más fiables.

Referencias

[1] N. A. Barriga, M. Stanescu, and M. Buro. Game tree search based on nondeterministic action scripts in real-time strategy games. *IEEE Transactions on Games*, 10(1):69–77, 2018.

[2] S. Ontañón. Experiments with game tree search in real-time strategy games. *arXiv preprint arXiv:1208.1940*, 2012.

[3] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[4] A. Uriarte and S. Ontanón. High-level representations for game-tree search in rts games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, pages 14–18, 2014.