

UCB1 y Regret Matching en el modelado de juegos RTS

Jesús Fermín Villar Ramírez

ESNE Escuela Universitaria, Madrid, Spain.

hello@pokoidev.com

Abstract

En los videojuegos de estrategia en tiempo real (posteriormente como 'RTS'¹), tiene un gran peso el papel de la Inteligencia Artificial para el disfrute de los jugadores. Además, los controladores de este tipo de juegos tienen que caracterizarse por una toma de decisiones rápida y que responda a las acciones del usuario, con un componente importante de aprendizaje automático y de predicción del comportamiento del adversario. Por ello se proponen los algoritmos de predicción de acciones UCB1 y Regret Matching para evaluar su funcionamiento en el modelado de este tipo de videojuegos.

I. INTRODUCCIÓN

UCB1 es un algoritmo de predicción de acciones que basa la toma de la siguiente decisión en los resultados obtenidos al usar previamente cada una de las acciones disponibles. En este sentido, UCB1 necesita conocer todas las posibles acciones para poder realizar su predicción. Esta característica, como veremos más adelante en la exposición de los resultados, es uno de

los mayores limitantes de este algoritmo en su uso en videojuegos RTS.

La decisión tomada por el algoritmo UCB1 es aquella que posea un mayor índice según la siguiente ecuación:

$$R = U_{(A)} + \sqrt{\frac{2 * \ln T}{T_{(A)}}}$$

Donde R es el resultado obtenido, $U_{(A)}$ es la recompensa media (más tarde denominaremos a este término “utilidad media”) de la acción A , T es la cantidad de veces que se ha ejecutado el algoritmo y $T_{(A)}$ la cantidad de veces que se ha ejecutado la acción A .

Por su parte, Regret Matching es un algoritmo que analiza, tras la toma de una decisión, el remordimiento de haber usado esa acción y no cualquiera de las demás una vez conocido el resultado del oponente. Por tanto, como el UCB1, Regret Matching también necesita conocer todas las posibles acciones y, lo que es más importante, la utilidad o ventaja de unas sobre otras.

Estos dos algoritmos de toma de decisiones, se presentan por tanto como algoritmos capaces de predecir una acción del usuario dentro de las conocidas. En nuestro modelo, los algoritmos UCB1 y Regret Matching tratarán de predecir las formaciones de

¹ Real Time Strategy

soldados del jugador para ejecutar la acción que mejor la contrarreste.

II. MÉTODOS

A. Modelado del entorno. Las unidades

Los juegos de estrategia destacan, en el ámbito de las unidades, por tener múltiples tipos con diferentes estadísticas cada uno. Esto hace a los juegos más complejos y con una mayor riqueza a la hora de tomar las decisiones estratégicas.

En nuestro modelo propuesto para evaluar la funcionalidad de los comentados algoritmos, las unidades están caracterizadas por:

- Una unidad con la información relativa su comportamiento dependiente de su tipo y su posición en el tablero.
- El componente de locomoción de la unidad que le permite desplazarse por el tablero.
- El componente de toma de decisiones de batalla, que le permite tomar decisiones con respecto a qué unidad atacar.
- Información relativa al equipo al que pertenece.

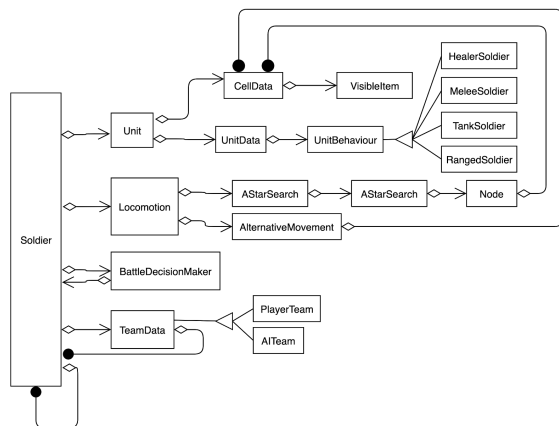


Diagrama del modelado de las unidades

Con respecto a qué parámetros o estadísticas maneja cada tipo de unidad, el modelo

mantiene un sistema simplificado que reúne lo esencial en este tipo de juegos: vida, daño, velocidad de ataque y rango.

Además, cada tipo de unidad posee un bonificador a las unidades contiguas en cruz (esto es, a aquellas unidades en las celdas contiguas tanto horizontal como verticalmente pero no en diagonal). Esto será importante y marcará diferencias en enfrentamientos en los cuales se presenten mismas unidades en diferentes posiciones. Las bonificaciones según el tipo son:

- Las unidades sanadoras aumentan en 20 puntos la vida máxima de sus unidades contiguas al comienzo de la batalla.
- Las unidades de ataque cuerpo a cuerpo aumentan en 5 el daño de sus unidades contiguas al comienzo de la batalla.
- Las unidades defensivas aumentan en 50 la vida máxima de sus unidades contiguas al comienzo de la batalla.
- Las unidades de ataque a distancia aumentan en 0.20 la velocidad de ataque de sus unidades contiguas.

B. Modelado del entorno. El tablero

El tablero del modelo propuesto está configurado por una serie de filas y columnas, perteneciendo la mitad de las columnas al terreno de trabajo de un jugador y la otra mitad al otro.

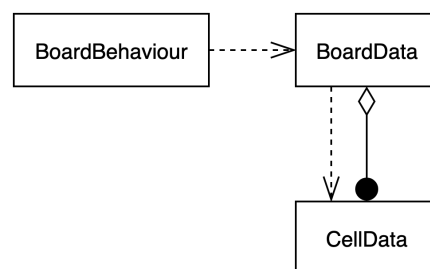


Diagrama de la construcción del tablero

El tablero crea las celdas que serán referenciadas por las unidades para establecer su posición. Hay, por tanto, solo una copia de cada celda en el juego,

evitando así la creación de celdas innecesarias.

C. Combinatoria. Posibilidades y escenarios

Las variables ya comentadas con respecto a la toma de decisiones de formaciones son: tipos de enemigos y posiciones. A estas hay que añadirles una nueva: cantidad de unidades por equipo. En la mayoría de juegos RTS, la cantidad de unidades militares posibles por equipo viene determinada por factores cambiantes a lo largo del tiempo de juego como poder militar o construcción de determinadas infraestructuras. En el modelo propuesto la cantidad es fija e invariable para una mejor toma de decisiones.

Con estas variables ya quedan definidos los componentes que determinan la cantidad de posibles acciones entre las que debe de tomar una decisión la IA para elegir una formación de unidades.

Por matemática y combinatoria, la fórmula que relaciona todos los componentes con la cantidad de posibles acciones queda:

$$D = \frac{(A * B)!}{((A * B) - C)! * (C!)} - B!$$

Donde A es el tamaño del tablero (entendido como filas por columnas), B la cantidad de tipos distintos de unidades, C la cantidad máxima de unidades en un equipo y D la cantidad de posibles acciones.

La justificación de este cálculo es la siguiente: nos interesan todas las posibles combinaciones de C unidades entre todos los tipos de unidades en cualquier posición del tablero, exceptuando aquellas posiciones en las que ya se encuentra una unidad (hay un máximo de una unidad por celda).

Las posibles acciones son calculadas por el controlador de la IA. La estructura de estas viene determinada por el siguiente diagrama.

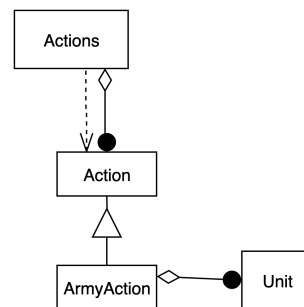


Diagrama de la información de las acciones

Una vez tenemos las posibles acciones, tanto la formación del jugador como la formación de la IA pertenecerán a dicha colección de acciones.

Es importante destacar la diferencia entre posibles acciones y posibles escenarios. Siguiendo la nomenclatura anterior, si D representa la cantidad de las posibles acciones, E (la cantidad de posibles escenarios) quedaría como:

$$E = D^F$$

Donde F es la cantidad de jugadores (tanto naturales como IA). Esto es porque dada una cantidad de posibles acciones, todos los posibles escenarios son todas las combinaciones de esas acciones por los jugadores.

En el caso del juego Piedra-Papel-Tijeras de dos jugadores, las posibles acciones son tres: (1) elijo piedra, (2) elijo papel, (3) elijo tijeras. Mientras que los posibles escenarios son nueve: (1) piedra-piedra, (2) piedra-papel, (3) piedra-tijeras, (4) papel-piedra, (5) papel-papel, (6) papel-tijeras, (7) tijeras-piedra, (8) tijeras-papel y (9) tijeras-tijeras.

Esta distinción es importante para entender el comportamiento de los algoritmos UCB1 y Regret Matching.

D. Locomoción

El algoritmo de A estrella (A^* posteriormente) es un algoritmo de búsqueda aplicado en multitud de ocasiones para búsqueda de caminos (posteriormente como *pathfinding*). El funcionamiento de este algoritmo se apoya en un cálculo heurístico que tiene en cuenta el coste de realizar una determinada acción. Este coste está determinado por el coste acumulativo en llegar hasta esa acción sumado al coste esperado en llegar a la acción final.

Trasladándolo a *pathfinding*, el algoritmo de A^* quedaría de la siguiente forma:

Si un agente en una posición A desea llegar hasta la posición B , el algoritmo expandirá cada una de las mejores posiciones desde su posición hasta alcanzar B , estableciendo esa “mejor posición” en relación al cálculo heurístico antes mencionado.

Este algoritmo implica que el movimiento o desplazamiento ha de realizarse nodo a nodo.

El algoritmo alternativo es una modificación de este algoritmo que, basándose en las mismas premisas, solo busca cuál es la posición óptima a la que debe llegar.

E. Toma de decisiones. Batalla

Una vez comenzada la batalla, cada uno de los soldados en el tablero ha de tomar la decisión de a qué unidad atacar (la acción de las unidades sanadoras se contempla como un ataque invertido hacia las unidades de su propio equipo).

Esta decisión se toma en relación a unos pesos. Estos pesos se asignan a: (1) la cantidad de vida restante del objetivo, (2) el daño total hecho por la unidad objetivo en la batalla y (3) la distancia a la unidad. Siguiendo la fórmula:

$$H = \alpha * A + \beta * B + \gamma * C$$

Donde H es el resultado de la heurística, A la cantidad de vida restante del objetivo, B el daño total hecho por la unidad objetivo en la batalla, C la distancia a la unidad y α, β, γ los pesos de cada una de las magnitudes.

Esta heurística funciona tanto en el caso de las unidades ofensivas, que buscan eliminar primero a las unidades más problemáticas del equipo del adversario, como en el caso de las unidades sanadoras que buscan mantener a las estrellas de su equipo.

F. Aprendizaje automático

Los algoritmos UCB1 y Regret Matching necesitan de una información previa que llamaremos “utilidad”. Esta utilidad es la que marca qué acciones son útiles frente a otras.

En el juego de Piedra-Papel-Tijera, la utilidad de elegir piedra sobre una posible elección de papel del adversario es negativa, ya que perdería; mientras que sería positiva en el caso contrario.

En nuestro modelo, esta utilidad se obtiene por la IA mediante un aprendizaje automático luchando contra sí misma, ejecutando cada uno de los escenarios posibles y guardando esos datos obtenidos en un fichero que poder consultar en partidas posteriores.

El resto de parámetros necesarios por los algoritmos, son obtenidos dentro de una misma partida durante el transcurso de las batallas.

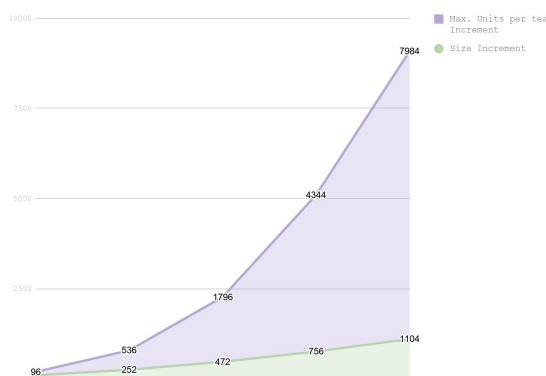
III. RESULTADOS

A. Combinatoria. Posibilidades y escenarios

La cantidad de posibilidades crece de manera muy rápida al modificar los parámetros, generando problemas de memoria.

Hemos de tener en cuenta que a cada posible acción le corresponde un índice de utilidad con el resto de posiciones, configurando una matriz bidimensional con las posiciones totales. Este índice ha de poder guardar cantidades negativas para reflejar los casos de pérdida y una gradación dentro de los valores positivos y negativos para conocer qué opciones son mejores y cuáles peores.

En las siguientes gráficas se muestra la evolución de la cantidad de posibilidades modificando el tamaño o la cantidad de unidades por equipo, siendo las condiciones iniciales de un tablero de cuatro columnas por dos filas, con cuatro tipos de unidades y un máximo de unidades por equipo inicial de dos unidades.



Evolución de la cantidad de posibles acciones

En cada iteración, en el caso del aumento de tamaño, todos los parámetros se mantienen estáticos salvo el tamaño del tablero, incrementándose en cuatro cada vez (el número de columnas inicial). En el caso del aumento del número máximo de unidades,

este valor solo aumenta en una unidad en cada iteración.

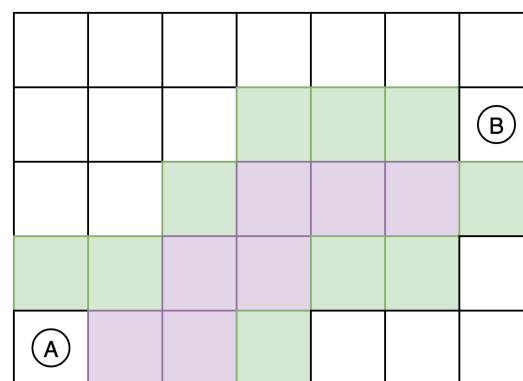
Por los resultados de la gráfica, observamos como pequeños cambios en la cantidad de unidades máximas por equipo generan un crecimiento exponencial de la cantidad de acciones posibles, lo que se traduce inmediatamente en una gran cantidad de memoria reservada por el crecimiento de posibles escenarios.

B. Locomoción

Se presentan varios diagramas para apoyar la exposición de los resultados de los algoritmos de locomoción.

En nuestro modelo, aparte de la heurística propia de los algoritmos, se ha añadido la condición de que la celda tiene que estar vacía para poder considerarse como relevante.

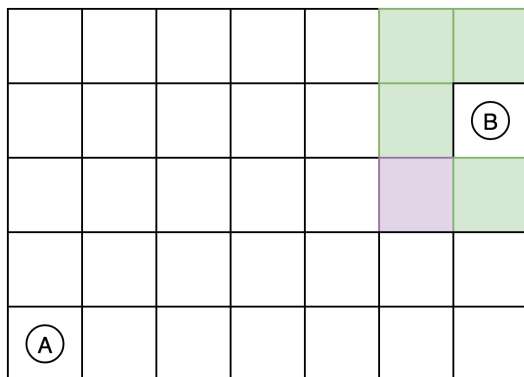
Puesto el escenario en el que una unidad tenga que ir del punto *A* al punto *B* con rango 1 (esto es, quedarse a una celda de diferencia con el objetivo), con el algoritmo *A** quedaría una solución como la mostrada en el siguiente diagrama:



*Algoritmo *A** con rango 1 al objetivo*

Donde las celdas verdes son los nodos expandidos y los morados los nodos elegidos. Vemos que se han expandido un total de 17 nodos en total.

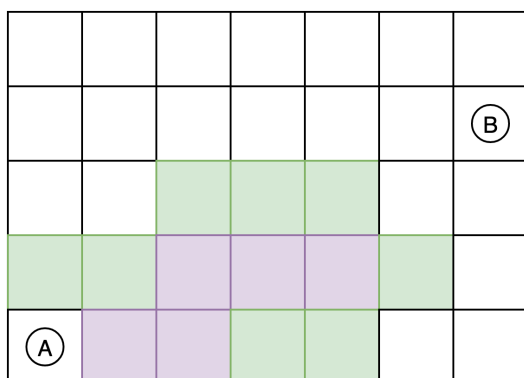
Puesto el mismo escenario, con el algoritmo alternativo, como muestra el diagrama siguiente, se han expandido solo 5 nodos.



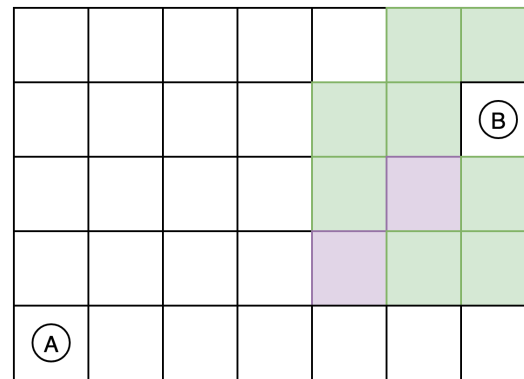
Algoritmo alternativo con rango 1 al objetivo

Esta simplificación del algoritmo supone un avance en la memoria usada por el algoritmo aun existiendo la diferencia de que el algoritmo alternativo expande todos los nodos o celdas contiguas mientras que el algoritmo de A* solo las contiguas en cruz.

Con respecto a los escenarios de rango mayor a uno (por ejemplo se propone un escenario de rango dos), la ventaja del algoritmo alternativo sigue existiendo con 10 nodos frente a los 13 nodos del algoritmo A*.



Algoritmo A con rango 2 al objetivo*



Algoritmo alternativo con rango 2 al objetivo

Pese a todo, estos algoritmos no son suficientes en nuestro modelo ya que generan escenarios inviables.

En nuestro modelo tenemos distintos agentes que están realizando estos mismos cálculos, en ocasiones de manera infinita por no encontrar una posible solución (ya que otras unidades han ocupado las posibles posiciones y ninguna está libre).

C. Aprendizaje automático

El aprendizaje automático y su coste viene de la mano del gran número de posibles acciones que conlleva un modelo como el propuesto.

La IA no solo tiene que replicar todos y cada uno de los escenarios, lo que lleva tiempo, sino que además tiene que trasladar los resultados obtenidos a un archivo exportable para su reutilización. Estos archivos, dependiendo de cuántas posibles acciones se estén manejando, tienden a ocupar bastante tamaño, llegando a alcanzar las decenas de gigas en casos medios.

D. Engañando a la IA

Dado que el aprendizaje es muy costoso, se ha procedido a introducir en la IA información errónea con respecto a la “utilidad” de las acciones para evaluar la capacidad de recapitación. Los resultados

han sido lo suficiente negativos como para inferir que, dado un tamaño tan grande de posibilidades, la IA necesita un margen de maniobra mayor que el esperado en un juego RTS.

IV. CONCLUSIONES

Los algoritmos UCB1 y Regret Matching tienen una limitación muy importante en entornos grandes con altas posibles acciones. Esto es debido a su necesidad de conocer todas las posibles acciones y cómo afectan cada una de ellas sobre el resto.

El crecimiento exponencial de las posibles acciones evidencia que en juegos como los juegos RTS donde hay multitud de acciones, es necesario un sistema de evaluación y de toma de las decisiones gradual que no tenga en cuenta todos los factores al mismo tiempo, a diferencia de los algoritmos presentados UCB1 y Regret Matching.

El aprendizaje automático es un recurso que siempre está disponible y que ahorra mucho cálculo de información en el tiempo de ejecución del juego frente al usuario final, aunque conlleva demasiados recursos en tiempo de desarrollo. Una forma rápida de agilizar el tiempo invertido por este aprendizaje automático es mediante el uso de varias unidades de computación que realicen aprendizajes paralelos, dividiendo el tiempo necesario.

Con respecto a la locomoción, en modelos donde hay tantos agentes ejecutando algoritmos de *pathfinding* como el modelo propuesto, se observa por los resultados lo necesario que es usar un sistema centralizado que administre las diferentes posiciones para garantizar un comportamiento de patrulla sin que unas unidades quiten el espacio necesario por otras.

V. BIBLIOGRAFÍA

- R.Lara-Cabrera, C.Cotta, A.J. Fernández-Leiva (2013). "A review of computational intelligence in RTS games", 2013/4, *Foundations of Computational Intelligence (FOCI)*, 2013 IEEE Symposium, 114-121.
- Robertson, G., & Watson, I. (2014). "A Review of Real-Time Strategy Game AI". *AI Magazine*, 35(4), 75-104.
- Weber, B. G.; Mateas, M.; and Jhala, A. 2011. Building Human-Level AI for Real-Time Strategy Games. In *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*, 329–336. AAAI Press
- Ontanon, S. (2013). . In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.
- J.Vasquez II "Implementing N-Grams for Player Prediction, Procedural Generation, and Stylized AI", *GameAIPro*, Chapter 48, 567-580
- S.C.J. Bakkes, P.H.M. Spronck, H.J. van der Herik (2009). "Opponent modelling for case-based adaptive game AI", *Entertainment Computing* 1 (1), 27-37.

VI. ANEXO

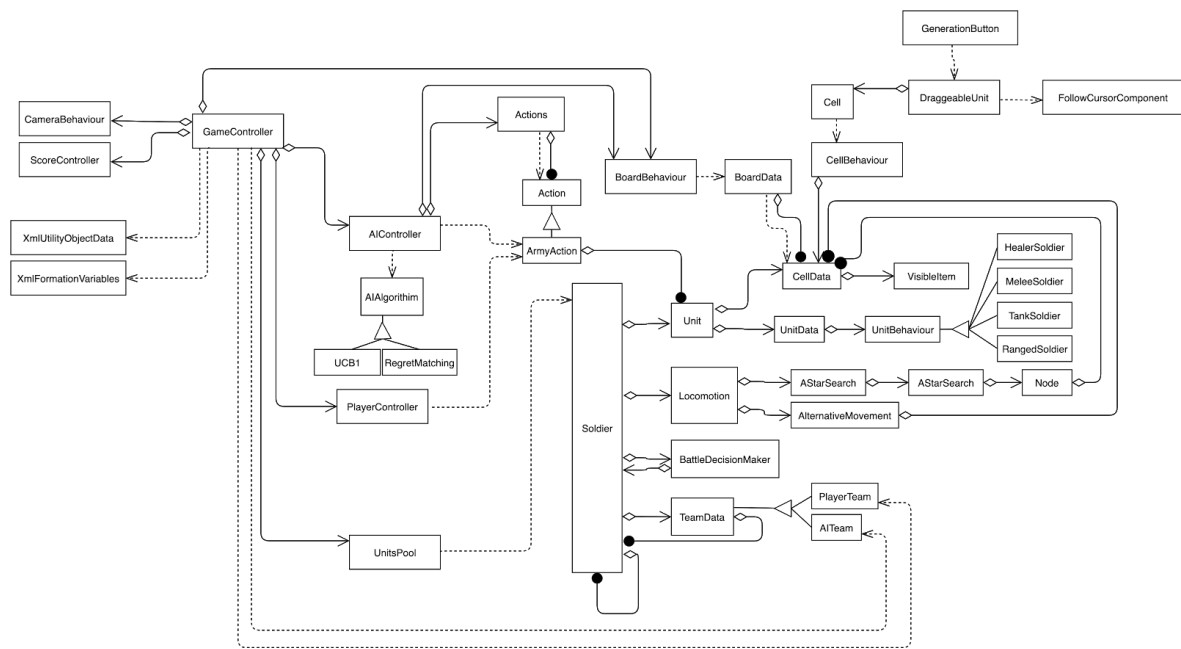


Diagrama de relación de las clases del modelo