



Reporte Tercer Proyecto

Agente de Battle City basado en Q learning

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 [2], 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 aprendizaje por refuerzo proporciona una metodología para desarrollar agentes en estos juegos [4][1].

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 diversos tipos de paredes, de tanques enemigos, power-ups, entre otros.

En este trabajo se implementa un agente que aprende a jugar una versión simplificada del videojuego BattleCity utilizando Q-learning y Q-learning Aproximado.

2. Definición de algoritmos

2.1 Estados del juego

Para hacer viable el uso de Q-learning se ha decidido definir estados con información básica y abstracta sobre el tiempo de juego actual. Si se tomara la información completa del tiempo de juego como estado tendríamos los siguientes datos:

- *Jugador*: posición (x,y), vida (1,2,3), si está vivo o no (1,0).
- *Bots enemigos*: posición (x,y), vida (1,2,3), si está vivo o no (1,0).
- *Base*: posición (x,y), si está en pie o no (1,0).
- *Todos los muros*: posición (x,y), vida (1,2,3), tipo de muro (1,2), si está en pie o no (1,0).
- *Balas*: posición (x,y), dirección (dx,dy), id del dueño (1,2,3), equipo (1,2).

En el mejor de los casos, que es cuando no hay balas disparadas, tenemos un total de:

$$(13^2 \times 3 \times 2) \times 2(13^2 \times 3 \times 2) \times (1 \times 2) \times 54(1 \times 3 \times 2 \times 2) \approx 2.6 \times 10^9 \text{ estados.}$$

Número de estados inmanejable para realizar Q-learning. Por esta razón es que se ha decidido simplificar la representación de los estados del juego.

La información que se guarda en los estados es la siguiente:

- *Salud del jugador*: Se tiene 0 si el jugador está muerto, si no entonces es la salud conocida (1,2,3).
- *Salud de los enemigos*: Similar a la salud del jugador, se tiene (0,1,2,3).
- *Posición de los enemigos*: Distancia relativa al jugador, si $d_1(\text{jugador}, \text{enemigo}) \leq 3$ entonces el enemigo está *cerca*; si $d_1(\text{jugador}, \text{enemigo}) > 3$ entonces el enemigo está *lejos*.
- *Base destruida*: Si la base está en pie o no (1,0).
- *Peligro en dirección*: Si en alguna de las direcciones hay una bala enemiga o un tanque enemigo (*arriba, abajo, izquierda, derecha*).
- *Peligro de base*: Si hay algún enemigo o bala cerca de la base, *cerca* si $d_1(\text{base}, \text{bala}) \leq 3$ o $d_1(\text{base}, \text{enemigo}) \leq 5$; si no se cumple ninguno de los dos entonces es *lejos*.

Con esta representación de los estados del juego capturamos la información más importante para una partida y tenemos

$$4 \times 2(4 \times 2) \times 2 \times 4 \times 2 = 1024 \text{ estados.}$$

Que hace mucho más manejable para el aprendizaje por refuerzo y para un estudio para las cadenas de Markov inducidas por las políticas π aprendidas en nuestros experimentos [3].



Figura 2: Los dos modelos de estados. (a) Estado que representa la configuración de todo el juego en el instante de partida; (b) Estado reducido que representa la información más relevante del juego en el instante de partida, las equis sobre los muros deben entenderse como que no son percibidas en los estados.

2.2 Función de evaluación

Para acelerar el proceso de aprendizaje, se ha decidido agregar una función de recompensa por situaciones ocurridas. Si se gana la partida entonces $r = 1000$, si se pierde $r = -500$. Esta función está dado por la siguiente suma ponderada:

$$r = enemigo_muerto + disparo + cerca_a_enemigo - vida_perdida - tiempo$$

en donde

$$enemigo_muerto = 100,$$

$$disparo = 5,$$

$$cerca_a_enemigo = 2.$$

$$vida_perdida = 20 \times vida,$$

$$tiempo = 1.$$

y $vida$ es la diferencia de vida entre el estado anterior y el estado actual.

De esta forma se guía al agente más rápido en aprender una política buena.

2.3 Modelos entrenados

Para las pruebas se entrenaron tres modelos, un modelo con el algoritmo Q-learning y dos modelos con Q-learning aproximado: el primero utilizando los estados reducidos, mientras que el segundo se entrenó utilizando estados completos. De aquí en adelante nos vamos a referir a cada uno como Modelo 1, Modelo 2 y Modelo 3 respectivamente.

3. Resultados

Los tres modelos se entrenaron con 100000 episodios. Los resultados de la tasa de victorias se puede observar en la Figura 3, mientras que la distribución de número promedio de ticks cada cien partidas se puede observar en la Figura 4. Lo primero que se puede observar es que el Modelo 1 obtiene, a lo

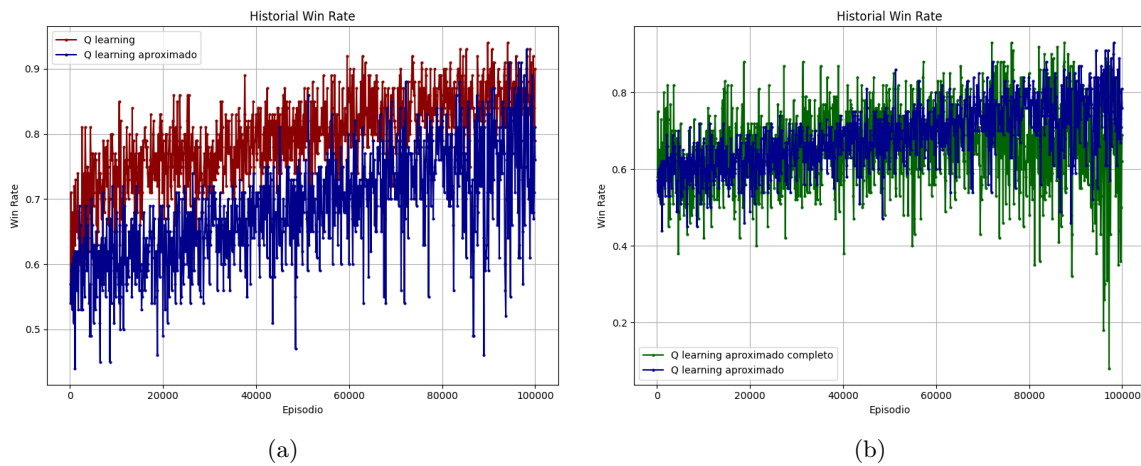


Figura 3: Tasa de victorias cada 100 episodios en el entrenamiento. (a) Entre el agente Modelo 1 y el Modelo 2. (b) entre el agente Modelo 2 y el Modelo 3.

largo de todo el entrenamiento, una tasa de victoria más alto que los otros dos modelos. Sin embargo, los dos modelos entrenados con los estados reducidos parecen converger a una tasa de victoria del 80 %. Por otro lado, el modelo entrenado con estados completos en un principio obtiene una tasa muy alta, sin embargo, con el paso de los episodios empieza a obtener resultados muy variables, hasta el punto de, al final del entrenamiento, obtener los peores puntajes entre los tres modelos. Por otra parte, el Modelo 2 parece tener, en promedio, las partidas más largas con una mediana de 236.345, mientras que los otros dos modelos presentan tiempos de partidas más cortas, con una mediana de 161.445 para el Modelo 1 y 146.675 el Modelo 3.

Después de esto se realizaron 1000 ejecuciones del juego con los tres modelos, en la Tabla 1 y en conjunto a la Figura 5 se pueden observar los resultados. Mientras tanto, en la Figura 6 se grafican las distribuciones de las puntuaciones obtenidas.

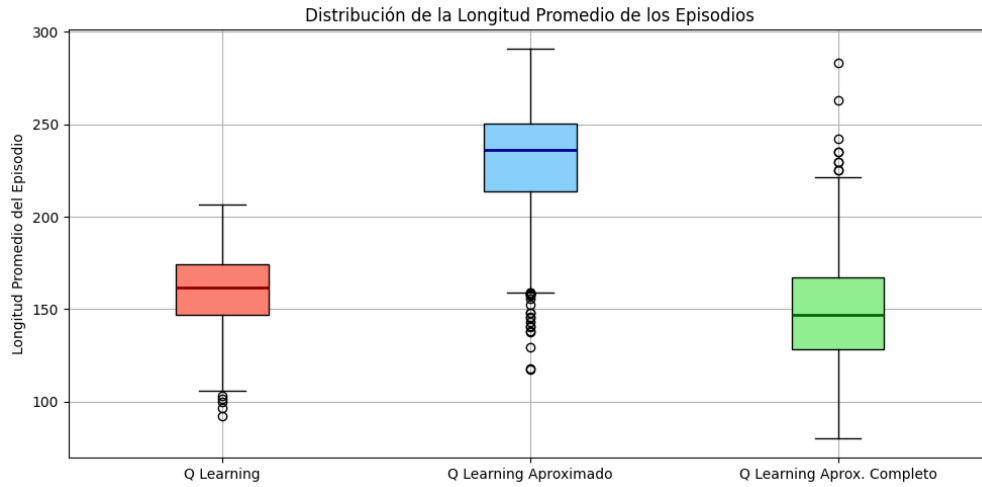


Figura 4: Distribución de la duración promedio de cada 100 partidas en el entrenamiento de los modelos.

Resultados			
Resultado	Modelo 1	Modelo 2	Modelo 3
Victoria	793	725	67
Derrota	127	213	871
Empate	50	62	62

Tabla 1: Número de victorias, derrotas y empates en las 1000 partidas simuladas en cada uno de los modelos.

Los resultados obtenidos son sorprendentes por el mal rendimiento del tercer modelo, aunque esperables por lo visto en el entrenamiento. El Modelo 1 obtiene más victorias y menos derrotas, seguido muy de cerca por el Modelo 2. Sin embargo, el Modelo 3 tiene un número de victorias deplorable, empata el mismo número de partidas que el Modelo 2, pero el número de derrotas es significativo. Asimismo, la puntuación obtenida deja también en claro esta información: se puede observar como el Modelo 1 y el Modelo 2 distribuye la mayor parte de su puntuación sobre el 1000, que es lo que obtiene el jugador al ganar, otra pequeña parte un poco más alto del 0 que es cuando hay empates y otra parte un poco más grande al rededor del -500 , que es cuando pierde. En contraste, el Modelo 3 concentra la mayor distribución sobre este último rango.

En general, el comportamiento observado en el Modelo 3 puede explicarse por la complejidad inherente al uso del estado completo del entorno. Al disponer de una representación más compleja y dimensionalmente más grande, el agente enfrenta un espacio de estados mucho más difícil de explorar y de generalizar, lo que parece producir políticas inestables. En contraste, los modelos entrenados con estados reducidos podrían actuar como si incorporaran una forma de regularización: al eliminar ruido y concentrar únicamente la información esencial, evidencian converger hacia estrategias más

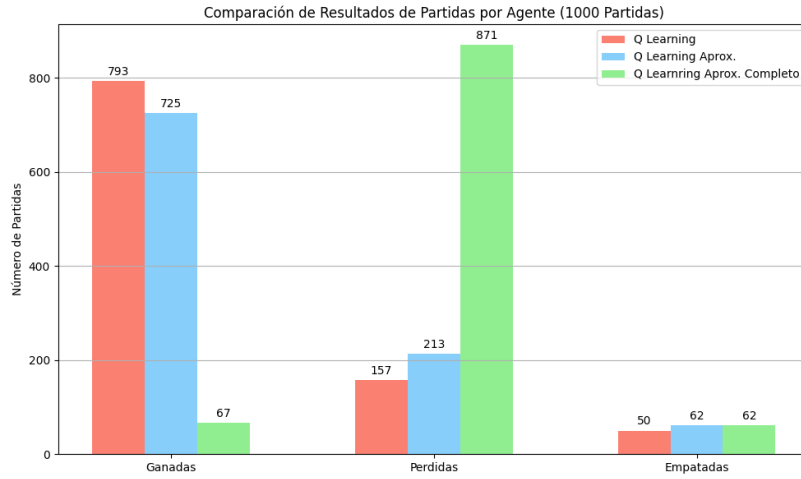


Figura 5: Comparación en el número de victorias, derrotas y empates en las 1000 partidas simuladas en cada uno de los modelos.

simples pero mucho más estables. Esto explica tanto la superioridad del Modelo 1 y el Modelo 2 en los experimentos finales como la caída drástica del rendimiento del Modelo 3, cuya distribución de recompensas se concentra mayoritariamente en valores asociados a derrotas.

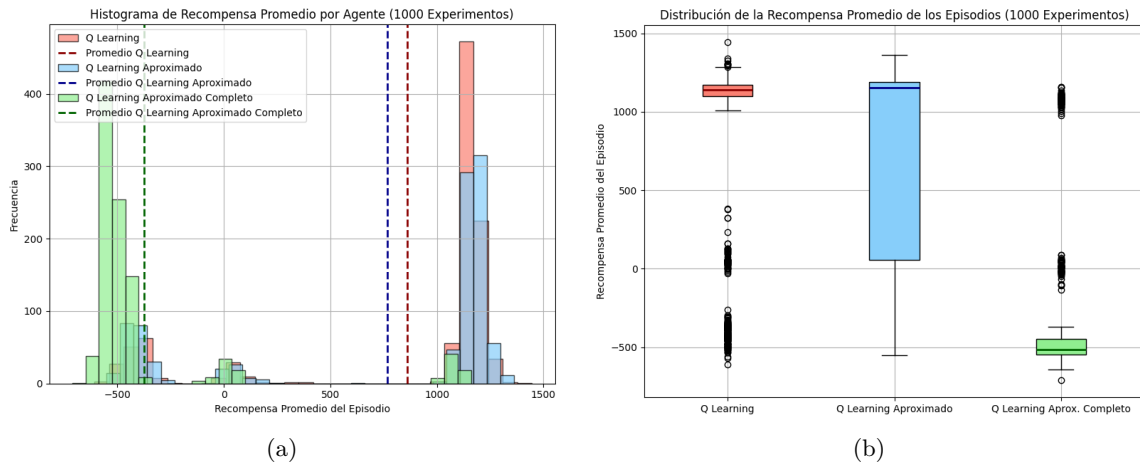


Figura 6: Distribución de las recompensas obtenidas de los 1000 experimentos de cada uno de los modelos.

4. Conclusiones

La aproximación para construir agentes utilizando aprendizaje por refuerzo para problemas en donde el espacio de estados se hace computacionalmente inviable, como lo es el videojuego Battle City, proporciona resultados bastante buenos y que

Se ha demostrado que la simplificación de los estados puede llegar a obtener mejores modelos en menos tiempo de entrenamiento.

La implementación y derivación matemática en este tipo de problemas ayuda a extender el análisis utilizando otro tipo de herramientas matemáticas.

Referencias

- [1] D. Isotelo. RL Battle City. https://github.com/danisotelo/RL_battle_city, 2023. Accedido: 2025-11-30.
- [2] 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.
- [3] M. Ortiz, S. Botero, and J. González. Análisis markoviano del proceso de aprendizaje por q-learning: Probabilidades terminales y tiempos esperados de las políticas inducidas. Technical report, Universidad Nacional de Colombia, 2025.
- [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.