

Proyecto Final, 2do Semestre 2025.

El Gato y el Ratón.

Algoritmos Avanzados de Búsqueda y Optimización.



Equipo I: Alexia Aurrecochea, Mercedes Barrutia,
Sofía Craigdallie y Paula Blasco.

Facultad de Ingeniería.
Prof; Ing. Michel Pedrera e Ing. Pío Dos Santos
2do Semestre, 2025

Índice

Introducción	2
Motivación	2
Marco Teórico	2
Criterios de Éxito	3
Juego Automatizado	3
Modelado	3
Algoritmos Implementados	5
A*	5
Minmax	6
Experimentos y Análisis de Resultados	8
Evaluación de Algoritmos	12
Conclusiones	13
Posibles Líneas Futuras	13
Referencias	13
Anexo: Manual de Usuario	14

Proyecto Final, 2do Semestre 2025.

Introducción

“En este juego, cada movimiento redefine la distancia entre captura y escape: dos estrategias opuestas en constante tensión.”

En un tablero limitado por obstáculos se encuentran dos agentes: un gato y un ratón. En cada turno se moverán alternadamente: el ratón, en busca evadir al gato y alcanzar una meta luego de haberse robado el queso y el gato, que intenta capturar al ratón en el menor tiempo posible, reduciendo la distancia entre ambos hasta ocupar la misma celda/posición.

Motivación

El problema del gato y el ratón constituye un entorno de alta complejidad estratégica que combina elementos de búsqueda, planificación, y toma de decisiones en escenarios adversos. Además, su estructura modular facilita la extensión hacia escenarios más complejos, como juegos multiagente (con más personajes), entornos parcialmente observables (como en videojuegos) o simulaciones con aprendizaje por refuerzo (en el que cada personaje posea distinta “skills”); de acuerdo al nivel de exigencia que el equipo docente sugiera.

Es un modelo desafiante por su complejidad, pero al mismo tiempo fácil de visualizar y evaluar por sus trayectorias, tiempo de captura y tiempo de escape. El enfrentarse a este modelo representa una prueba de comprensión de los algoritmos de búsqueda avanzada trabajados en el aula, y su implementación en escenarios competitivos y dinámicos. Así como resulta ser un proyecto de portfolio personal que brindará mayor comprensión de los mecanismos de decisión y dificultad en videojuegos y otros fenómenos gamificados.

Marco Teórico

El problema del gato y el ratón puede abordarse desde la perspectiva de la búsqueda y optimización en entornos dinámicos donde dos agentes con objetivos opuestos actúan sobre un mismo espacio. Este marco teórico desarrolla los fundamentos conceptuales que sustentan el modelado, los algoritmos seleccionados y la interpretación de los resultados obtenidos.

Los problemas de búsqueda consisten en encontrar una secuencia de acciones que lleve a un agente desde un estado inicial hasta un estado meta, dentro de un espacio de estados definido por reglas de transición y restricciones (Russell & Norvig, 2021). En este contexto, el tablero del juego se modela como un grafo donde cada vértice representa una celda y las aristas, las posibles transiciones entre posiciones que puede hacer tanto el gato como el ratón.

Existen dos enfoques principales:

- Búsqueda no informada, que explora el espacio sin conocimiento del problema, como Breadth-First Search (BFS) que garantiza soluciones óptimas en términos de costo uniforme, o Depth-First Search (DFS) que puede explorar más rápidamente pero sin garantía de optimalidad.

- Búsqueda informada, que utiliza heurísticas para estimar la distancia al objetivo. En este grupo se encuentran Dijkstra (búsqueda de costo mínimo acumulado) y A*, que combina el costo real y una heurística admisible para reducir el número de nodos explorados (Russell & Norvig, 2021).

El algoritmo Minimax busca la estrategia óptima, el gato intenta minimizar la distancia al ratón, mientras que el ratón intenta maximizarla. Cada nodo del árbol de decisiones representa un estado del tablero y cada arista, una acción posible. Para reducir el costo computacional, se aplica la poda Alfa-Beta, que elimina ramas cuya evaluación no altera el resultado final, manteniendo la optimalidad de la decisión (Knuth & Moore, 1975).

Criterios de Éxito

El desempeño de los algoritmos se evaluará mediante métricas cuantitativas:

1. Tasa de captura: porcentaje de partidas en las que el gato logra capturar al ratón.
2. Tiempo promedio de captura: número medio de pasos hasta la intersección de ambos jugadores.
3. Distancia promedio mantenida por el ratón: mide la eficiencia evasiva.
4. Eficiencia computacional: tiempo de ejecución y número de nodos explorados.
5. Comportamiento estratégico: coherencia y realismo de las trayectorias generadas.

Juego Automatizado

Modelado

A partir de una representación discreta del tablero mediante grafos, se desarrollaron dos configuraciones diferenciadas: una de tablero reducido y otro ampliado, ambos construidos mediante estructuras de datos que almacenan las coordenadas de cada nodo y las aristas que determinan sus conexiones posibles. Mapas en donde el ratón deberá obtener un “queso” antes de acceder a la ubicación de la meta final, lo que introduce una mecánica condicional en el escape del ratón y la búsqueda de la salida.

Si bien el modelo formal inicial planteaba un tablero con movimientos únicamente cardinales, durante la etapa de desarrollo se observó que esta estructura restringía la complejidad del entorno y las trayectorias posibles entre los agentes. Por esta razón, el equipo decidió reformular la representación espacial, agregando movimiento diagonal. De tal forma, en la versión reducida, el grafo se diseñó con una estructura ramificada y asimétrica, cómo puede visualizarse en la Figura 1.

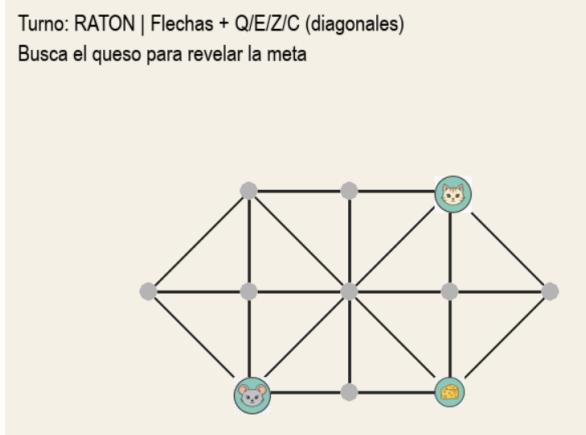


Figura 1: Modelado Inicial del Mapa con 11 Nodos

Mientras que la versión ampliada utiliza una rejilla 5×5 con conectividad ortogonal y diagonal, como se puede ver en la Figura 2. Lo que incrementa el número de trayectorias posibles y permite simular estrategias más complejas de evasión y persecución.

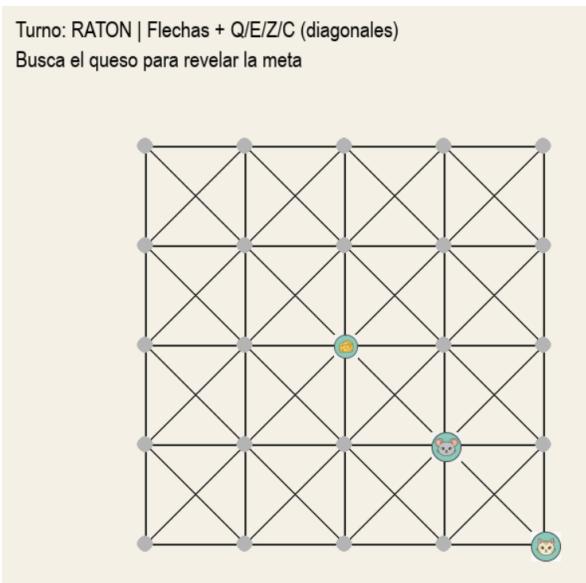


Figura 2: Modelado Inicial del Mapa con 25 Nodos

Ambas configuraciones fueron encapsuladas en módulos separados (`config_small.py` y `config_big.py`) para facilitar la experimentación con distintos entornos y tamaños de grafo. Y es en estos documentos en los que se encuentra el renderizado del juego: Se dibujan los nodos, aristas y piezas en pantalla utilizando Pygame, con diferenciación cromática para los estados y uso de imágenes escaladas para los agentes. Las funciones `dibujar_tablero()` y `dibujar_piezas()` gestionan esta visualización en cada frame.

El módulo principal (`modelado_juego.py`), desde donde se puede seleccionar la configuración del tablero, implementa la lógica dinámica del sistema, estructurada en un bucle

principal de actualización que controla los turnos alternos entre los agentes, la captura de eventos de teclado y la actualización gráfica en tiempo real. Cada iteración ejecuta los siguientes procesos:

- Entrada de usuario: para visualizar el modelado inicial se habilitó el registro de teclas pulsadas para mover al gato o al ratón en ocho direcciones posibles (flechas para movimientos cardinales y Q/E/Z/C para diagonales). El turno activo se almacena en la variable turno, alternando entre "ratón" y "gato", mientras que las posiciones actuales se guardan en pos_raton y pos_gato, respectivamente.
- Actualización del estado: se evalúan las condiciones de victoria o derrota mediante distintas variables. La variable booleana tiene_queso indica si el ratón ya ha recogido el queso (True o False), mientras que game_over y victoria determinan el final de la partida según el resultado. Los nodos relevantes se definen como queso (posición del queso) y final (nodo meta revelado).
- Control de flujo: al finalizar una partida se habilita la reinicialización automática, generando nuevas posiciones aleatorias para los tres objetos principales (pos_raton, pos_gato, queso) y restableciendo los estados (tiene_queso = False, game_over = False, victoria = False). El reloj interno RELOJ.tick(FPS) regula la frecuencia de actualización del juego, garantizando un rendimiento visual constante.

Algoritmos Implementados

Se implementaron algoritmos avanzados de búsqueda y optimización destinados a modelar la persecución y evasión entre dos agentes con objetivos opuestos. A través de A* se logró un comportamiento basado en rutas óptimas dentro del grafo, mientras que con Minimax, combinado con poda alfa-beta, se incorporó la capacidad estratégica de anticipación al movimiento del adversario. La comparación de ambos enfoques permite analizar no solo la eficiencia en la captura del ratón, sino también el realismo táctico emergente de la simulación.

A*

El algoritmo A* es un método de búsqueda informada que permite encontrar un camino óptimo entre dos nodos de un grafo, combinando el costo real acumulado desde el inicio con una heurística que estima el costo restante hasta la meta. Este algoritmo garantiza optimalidad siempre que la heurística sea admisible, nunca sobreestima el costo real es y consistente.

El algoritmo resulta adecuado en este problemas porque el espacio de estados está representado como un grafo y la heurística aporta información estructural del entorno. En este proyecto, el comportamiento del gato se modela como un problema de búsqueda con las siguientes características:

- Estado: nodo actual del gato
- Acciones: vecinos del nodo (a los que se puede avanzar)
- Meta: nodo ocupado por el ratón
- Costo de transición: 1 por cada arista (es un grafo no ponderado)

Cada nodo corresponde a una posición del mapa y las aristas representan pasos válidos. Se utilizó distancia Manhattan entre las coordenadas de los nodos, ya que es una heurística que nunca sobreestima el número mínimo de pasos (admisible) y que se aplica a un caso donde la distancia entre nodos vecinos siempre incrementa o reduce en 1 (consistente). Esto asegura que A* encuentre siempre un camino óptimo hacia el ratón. La implementación se hizo en dos partes, una función encargada de calcular un camino completo entre gato y ratón y la otra para el movimiento automático del gato que decide el siguiente paso en el turno del gato.

Una limitación de este algoritmo es que recalcula desde cero en cada turno, por lo que tiene un mayor costo computacional y un desafío al que nos enfrentamos fue el de ajustar la implementación para permitir pisar al ratón como nodo objetivo. Igualmente, nos permitió implementar un gato significativamente más eficiente, capaz de evaluar rutas completas garantizando la optimalidad de cada movimiento; mejorando la probabilidad de captura y reduciendo la cantidad de pasos promedio.

Minmax

Además del uso de estos algoritmos de búsqueda clásico se incorporó un enfoque con multiagente basado en el algoritmo minimax con poda alfa–beta. Mientras que A* busca rutas óptimas hacia un objetivo estático, Minimax modela el juego como una secuencia de turnos alternados entre dos agentes con objetivos opuestos: el gato que intenta maximizar sus chances de captura y el ratón que intenta minimizar esa ventaja, alejándose del gato.

De esta forma, Minimax permite analizar no solo la calidad de un camino, sino la calidad de la estrategia en un entorno donde el adversario también se mueve y reacciona. Para aplicarlo, el juego se modeló como un árbol de estados con las siguientes características:

- Cada nodo del árbol representa un estado del sistema.
- Cada arista representa una acción posible
- El movimiento del gato a un vecino del grafo y el movimiento del ratón a un vecino del grafo evitando el nodo ocupado por el gato.
- Los niveles del árbol alternan: MAX es el turno del gato y MIN es el turno del ratón.

A partir del estado actual, Minimax explora recursivamente los estados futuros hasta cierta profundidad, en este proyecto se fijó una profundidad pequeña para mantener el tiempo de cómputo razonable en el tablero de 11 nodos. Como el árbol se corta a una profundidad fija, se necesita una función de evaluación que asigne un valor numérico a cada estado no terminal. Esta función utilizada se basa en la distancia entre el gato y el ratón. Si el gato ya capturó al ratón se asigna un valor muy alto, indicando un estado favorable para el gato. Si no hay captura se calcula la distancia mínima en número de aristas entre el gato y el ratón utilizando BFS sobre el grafo de conexiones. Mientras más pequeña es la distancia, mejor es el estado para el gato, y cuanto más grande es la distancia, mejor para el ratón.

El algoritmo Minimax se implementó con la extensión de la poda alfa–beta para reducir el número de estados explorados sin alterar el resultado final. En los nodos MAX (turno del gato), el algoritmo elige el movimiento que maximiza la función de evaluación, y en los nodos MIN (turno del ratón), el algoritmo elige el movimiento que minimiza esa evaluación.

La poda alfa–beta introduce dos parámetros: alpha siendo este el mejor valor que MAX puede garantizar hasta ahora y beta siendo el mejor valor que MIN puede garantizar hasta ahora. Si en algún punto se cumple que beta es menor o igual que alpha se detiene la exploración de esa rama porque ya no puede mejorar el resultado. En el caso base se analiza si el gato ya capturó al ratón para ahí devolver utilidad máxima, pero si se alcanzó la profundidad límite se va a devolver la evaluación heurística.

En el turno del gato se va a explorar todos los vecinos posibles del gato y para cada movimiento se llama recursivamente a Minimax para el turno del ratón. Luego se actualiza alpha y se aplica poda si corresponde. De igual forma, en el turno del ratón se exploran todos los vecinos posibles del ratón y para cada movimiento se llama recursivamente a Minimax para el turno del gato. Luego se actualiza alpha y se aplica poda si corresponde.

Sobre esta base se definió la función de acción que se integra luego al loop del juego como movimiento automático del gato, de forma análoga a la versión basada en A*. En el código del juego se incorporó un parámetro que permite alternar entre el comportamiento basado en A* y el comportamiento basado en Minimax. Esto permite comparar de manera controlada el desempeño de ambos bajo las mismas condiciones de juego y en el mismo tablero.

En términos cualitativos A* es muy eficiente para perseguir al ratón por el camino más corto en el grafo y Minimax permite al gato anticipar no sólo el siguiente movimiento del ratón, sino varios turnos futuros; aunque requiere más cómputo y depende de la función de evaluación. Algunas limitaciones observadas del enfoque con Minimax fueron que la profundidad de búsqueda debe mantenerse baja y la función de evaluación utilizada es relativamente simple. Se podrían incorporar factores adicionales como la cercanía del ratón al queso o a la meta.

Experimentos y Análisis de Resultados

El objetivo de este estudio ha sido la evaluación, analizando su comportamiento, efectividad, fiabilidad y eficiencia temporal. Los pairings evaluados han sido los siguientes:

1. G_random vs R_astar
2. G_astar vs R_random
3. G_minmax vs R_astar
4. G_random vs R_minmax
5. G_astar vs R_minmax
6. G_minmax vs R_random
7. G_random vs R_random
8. G_astar vs R_astar
9. G_minmax vs R_minmax

Cada uno de estos se ha ejecutado 20 veces y a lo largo del análisis se han registrado las victorias del gato o ratón, las tasas de victoria, número promedio de pasos, el promedio y dispersión de duración y configuración (mapa pequeño o grande). Este código generó gráficos de tasas de victoria, de duración media y combinados de empate y pasos. También generó un CSV detallado y de resumen. Las gráficas proporcionadas son:

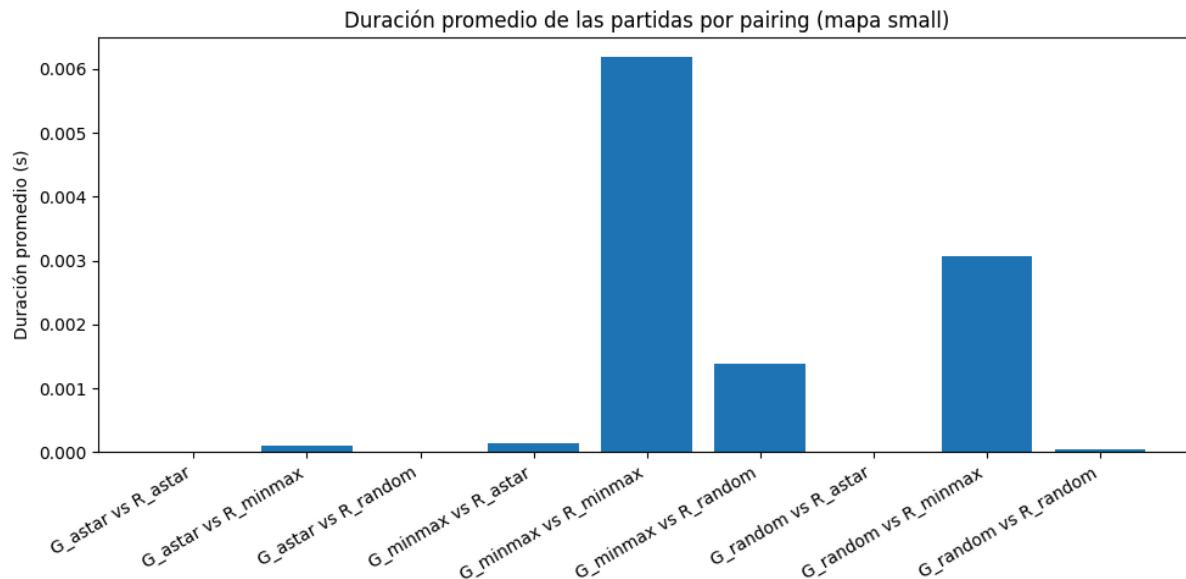


Figura 3: Duración Promedio de las Partidas por Pairing (mapa pequeño)

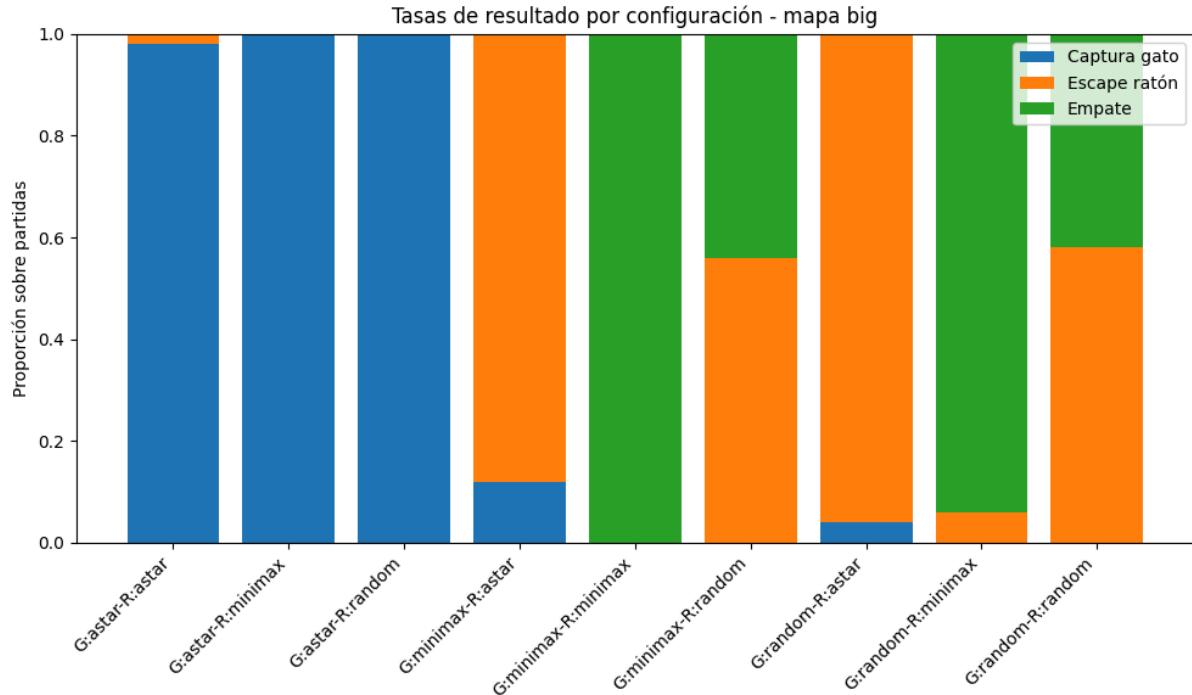


Figura 4: Tasa de Resultados por Configuración (mapa grande)

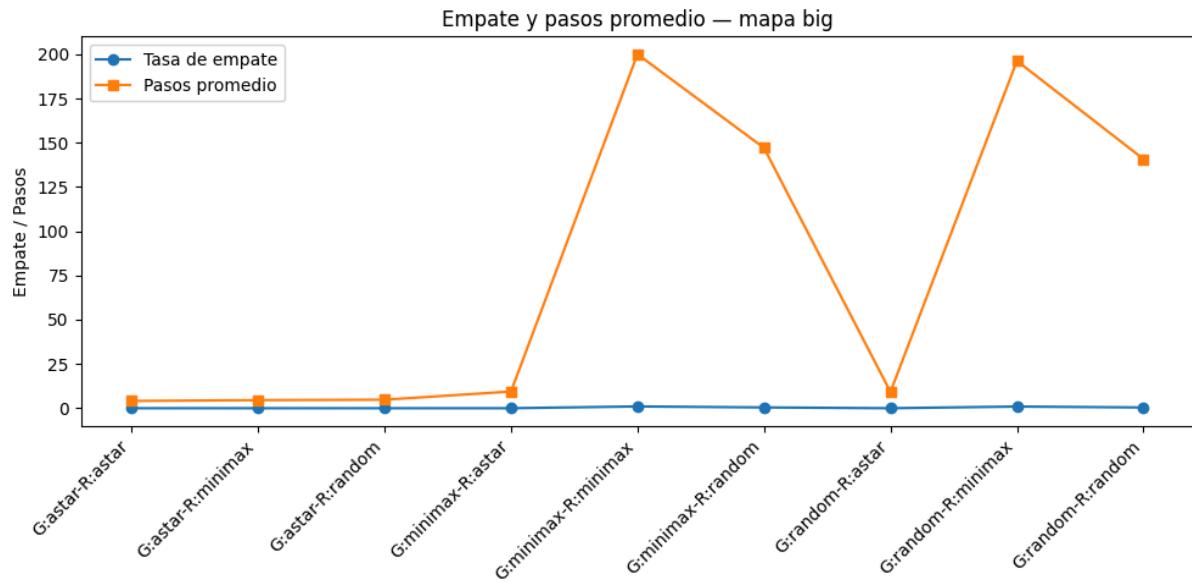


Figura 5: Empate y Pasos Medios (mapa grande)

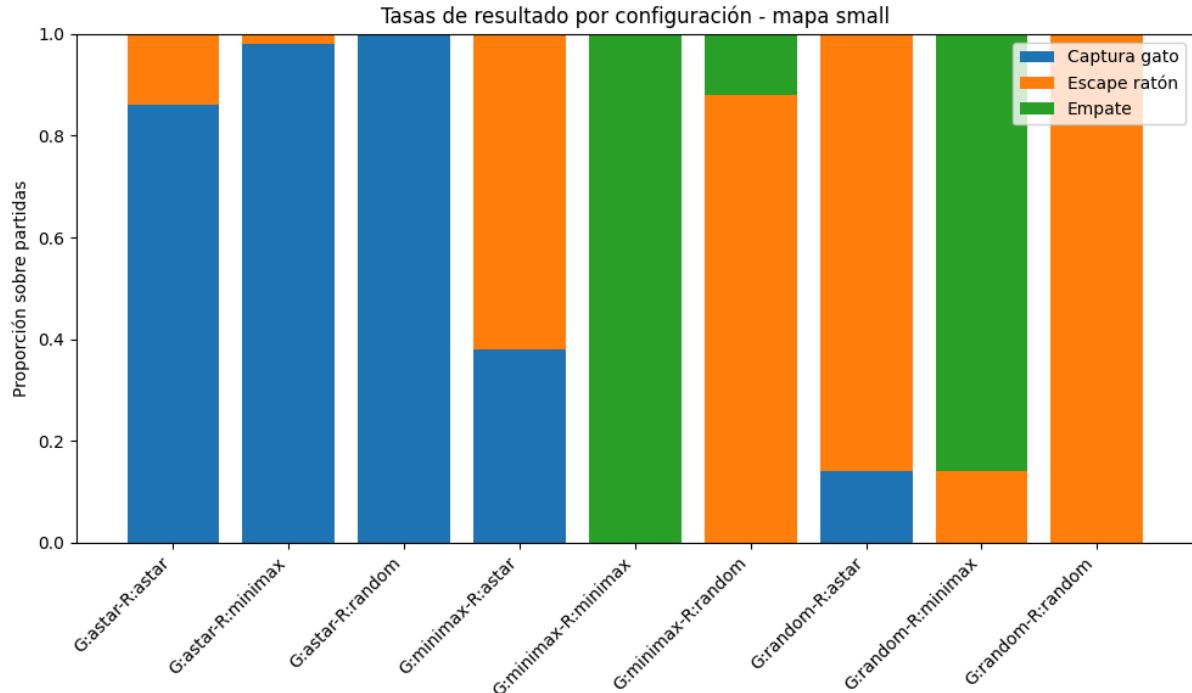


Figura 6: Tasas de Resultados por Configuración (mapa pequeño)

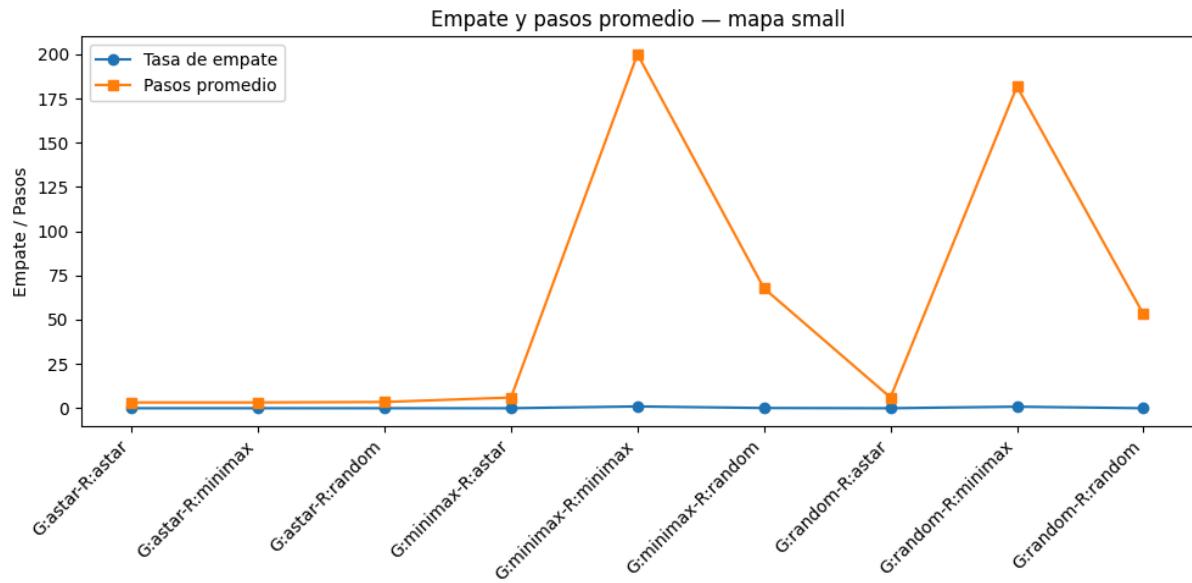


Figura 7: Empate y Pasos Medios (mapa pequeño)

Analizando estos resultados podemos comparar cada pareja, dentro del mapa small y big.

1. **G_random vs R_astar:** Como resultado general hay una victoria casi asegurada para el ratón con A*, el gato random no genera presión real y hay muy pocas capturas. Podemos observar que A* es excelente escapando y encontrando rutas óptimas hacia el queso y la meta. Y entonces el comportamiento aleatorio del gato no presenta amenaza.
2. **G_astar vs R_random:** Se observa que hay dominio total del gato con A* y el ratón muere casi siempre con partidas muy rápidas, ya que A* persigue óptimamente al ratón, el cuál se expone constantemente al no tener estrategia.
3. **G_minimax vs R_astar:** Se observa bastante empate, donde el ratón resiste muy bien y el gato no logra capturar con frecuencia. El algoritmo de minimax toma decisiones mirando estados futuros del juego pero no siempre encuentra caminos óptimos de persecución o no incorpora el objetivo del ratón (queso/meta). Por ello hay muchas partidas estancadas ya que minimax hace que la dinámica sea más lenta y menos resolutiva.
4. **G_random vs R_minimax:** El ratón minimax domina al gato random y hay muy pocas capturas, esto es porque el ratón aprovecha el mal movimiento del gato.
5. **G_astar vs R_minimax:** En ambos mapas el gato A* gana casi siempre, ya que el minimax ratón no es capaz de escapar, siendo incapaz de trazar rutas óptimas hacia la salida, porque no usa queso/meta por diseño.
6. **G_minimax vs R_random:** El gato minimax gana frecuentemente (donde el ratón random da muchas oportunidades) siendo mejor que random por poco, pero sigue siendo inferior que A*.
7. **G_random vs R_random:** En este enfrentamiento no hay estrategia por parte de ninguno de los dos, por lo que los resultados son completamente aleatorios. No se observa dominio claro: hay pocas capturas y pocas llegadas del ratón a la meta, ya que ambos se mueven sin intención ni planificación. Las partidas terminan más por coincidencias de movimiento que por decisiones significativas.
8. **G_astar vs R_astar:** Cuando ambos utilizan A*, las partidas se vuelven mucho más eficientes y cortas. Se observa que el gato suele ganar con bastante más frecuencia, ya que su objetivo es más simple (perseguir) mientras que el ratón debe dividir su estrategia entre queso y meta. El gato logra interceptar al ratón en muchos casos durante la fase previa a la obtención del queso. Aunque el ratón también toma rutas óptimas, la presión constante del gato hace que la mayoría de las partidas terminen en captura.

9. **G_minmax vs R_minmax:** En este caso no domina ninguno. Ambos algoritmos intentan anticipar jugadas, pero al no contar con una guía clara del mapa terminan tomando decisiones poco eficientes. Se observan muchos empates y partidas estancadas, donde el gato no logra perseguir de forma efectiva y el ratón tampoco logra encontrar rutas claras hacia la meta. El comportamiento de minimax en los dos agentes vuelve la dinámica más lenta, menos resolutiva y con muy pocas situaciones de ventaja decisiva para alguno de los lados.

Evaluación de Algoritmos

El análisis comparativo de todas las combinaciones de estrategias permite concluir que A* se posiciona como el algoritmo con mejor desempeño global. Su consistencia es evidente tanto cuando es implementado por el gato como por el ratón: logra sistemáticamente resultados favorables frente a oponentes más débiles, y en ambos roles demuestra una clara superioridad táctica. Además, produce partidas rápidas y resolutivas, minimizando la cantidad de pasos y evitando bucles improductivos. Al modelar correctamente todos los objetivos del juego (capturar, escapar y alcanzar la meta) logra una navegación óptima que se traduce en una ventaja decisiva.

En contraste, el enfoque Random se configura como la estrategia de menor desempeño. La ausencia total de planificación provoca que el gato casi nunca consiga capturar al ratón y que éste, a su vez, tenga pocas posibilidades de escapar con éxito. Las partidas tienden a desarrollarse sin sentido estratégico, dependiendo exclusivamente del azar. Si bien resulta útil como línea base experimental, queda claro que cualquier heurística mínimamente informada lo supera con amplitud.

Finalmente, el comportamiento de Minimax revela un posicionamiento intermedio dentro del espectro de rendimiento. Esta estrategia presenta una fortaleza apreciable: tiende a “jugar a no perder”, lo que incrementa la cantidad de empates y dificulta que el rival obtenga una victoria rápida. Sin embargo, también exhibe limitaciones importantes. En su implementación actual, el ratón no integra en la toma de decisiones los objetivos del queso y la meta, lo que le impide asegurar condiciones de victoria. Asimismo, el costo computacional se incrementa significativamente en mapas de mayor tamaño. En consecuencia, aunque Minimax no sobresale en ninguna de las métricas, tampoco puede considerarse un mal agente: promueve partidas más tácticas y pausadas, pero con una efectividad limitada para ganar.

Conclusiones

Este experimento evaluó un total de 180 partidas por mapa entre seis combinaciones de algoritmos. Los resultados muestran que A* es el algoritmo más eficaz, tanto para el gato como para el ratón, logrando los mejores porcentajes de victoria y las partidas más eficientes. En segundo lugar, minimax exhibe un comportamiento más defensivo que ofensivo, generando empates pero sin lograr superar a A*.

Por último, random sirve como base mínima y confirma que cualquier estrategia informada lo supera. Concluimos que se confirma que la navegación óptima marca una diferencia determinante en este tipo de juegos sobre grafos. Más adelante, la mejora del comportamiento estratégico (sobre todo de minimax) podría ser una gran mejora.

En síntesis, los resultados confirman que los agentes que modelan objetivos explícitos del entorno logran mejores desempeños. La inferencia posicional es condición necesaria pero no suficiente: debe complementarse con conocimiento del estado del juego para evitar estancamientos y maximizar posibilidades de victoria.

Posibles Líneas Futuras

Pensamos posibles mejoras para implementar a futuro, algunas de ellas son mejorando minimax, incorporando un estado completo del juego y una búsqueda limitada por profundidad más heurísticas. O también probando algoritmos alternativos como Best-first search o algoritmos híbridos como A* más heurísticas. De igual forma, técnicas de aprendizaje por refuerzo multi agente habilitarían la aparición de estrategias emergentes que no dependen de reglas programadas explícitamente, sino de la experiencia adquirida en el juego.

Por otro lado, podríamos extender la complejidad del entorno, incorporando reglas dinámicas, movilidad diferenciada o visión parcial entre agentes. Acercando la simulación original a un juego de mayor similitud a los de alto nivel.

Referencias

Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Knuth, D. E., & Moore, R. W. (1975). *An analysis of alpha-beta pruning*. Artificial Intelligence, 6(4).

Anexo: Manual de Usuario

Este manual describe el procedimiento que debe seguir el usuario para ejecutar y visualizar el funcionamiento del juego “Gato y Ratón”, desarrollado como entorno experimental para el análisis de algoritmos de búsqueda y optimización aplicados a un juego adversarial.

El sistema permite observar cómo un ratón intenta tomar un queso y escapar al objetivo final, mientras un gato trata de interceptarlo. Ambos agentes son autónomos y pueden utilizar diferentes algoritmos inteligentes.

El usuario podrá:

- ✓ Seleccionar el tablero de juego
- ✓ Definir el algoritmo de cada agente
- ✓ Visualizar la simulación en tiempo real
- ✓ Registrar el resultado de cada partida

Requisitos del sistema

Componente	Requisito
Sistema Operativo	Windows 10+, Linux o macOS
IDE	VSCode
Acceso	Git/Github
Python	Versión 3.8 o superior
Memoria RAM	4 GB mínimo
Librerías	pygame, numpy (instaladas automáticamente vía `requirements.txt`)

Instalación del software

1. Descargar el proyecto desde el repositorio GitHub:
https://github.com/aleaurre/Gato_raton.git
2. Ejecutar la terminal en la carpeta raíz y crear un entorno virtual:
`python -m venv .venv`
3. Activarlo:
 - a. Windows PowerShell: `.venv\Scripts\Activate.ps1`
 - b. Linux/MacOS: `source .venv/bin/activate`
4. Instalar dependencias:
`pip install -r requirements.txt`

Ejecución del juego

Para iniciar la simulación automática con visualización: `python -m scr.modelo_juego`

Se abrirá una ventana gráfica donde se mostrará el mapa y los agentes en movimiento.

La partida finaliza cuando:

- El ratón toma el queso y alcanza la meta → Victoria del ratón
- El gato alcanza al ratón → Captura / derrota
- Se alcanza el límite de pasos → Empate

Configuración de los algoritmos

El usuario puede elegir la estrategia de cada agente editando dos variables al inicio del archivo `simul_visual.py`:

```
MODO_GATO = "minimax" # opciones: astar | minimax | random
MODO_RATON = "astar" # opciones: astar | minimax | random
```

Descripción de modos

Modo	Requisito
random	Movimientos Aleatorios
astar	Búsqueda del Camino más Corto
minimax	Estrategia con poda Alfa-Beta

Selección del mapa

En `scr/simul_visual.py`, el usuario puede alternar entre:

Mapa	Archivo	Nodos	Dificultad
Chico	`config_small.py`	11	Ideal para el Inicio
Grande	`config_big.py`	25	Mayor Complejidad

Ejemplo de actualización: `from scr.config_big import conexiones, nodos, aristas, ...`

Registro y Visualización de resultados

Finalmente para el registro de todos los resultados y la evaluación de algoritmos, se podrá ejecutar el notebook `resultados.ipynb` mediante el botón “Run All”.

Contacto de soporte

Para asistencia técnica o reporte de errores:

 alexia.aurrecochea@correo.ucu.edu.uy, mercedes.barrutia@correo.ucu.edu.uy,
sofia.craigdallie@correo.ucu.edu.uy, paula.blasco@correo.ucu.edu.uy.

 Universidad Católica del Uruguay - Facultad de Ingeniería