Universidad de Puerto Rico - Recinto de Mayagüez

Departamento de Ingeniería Eléctrica y de Computadoras

Diseño e Implementación de Agentes Inteligentes para el Juego de Briscas

Autores:
Marco Yu Cordero
Samir Ali Rivera

Curso: ICOM/CIIC5015 – Inteligencia Artificial
Profesor: Dr. J. Fernando Vega Riveros

Fecha: 24 de abril de 2025

Resumen

Este trabajo presenta el diseño, implementación y evaluación sistemática de agentes inteligentes aplicados al juego de Briscas, ampliamente difundido en Puerto Rico y España, caracterizado por información parcial, alta variabilidad y una fuerte componente estratégica. El objetivo central fue determinar cuáles técnicas de inteligencia artificial adversarial ofrecen el mayor rendimiento en este contexto específico. Se desarrollaron e integraron cuatro tipos de agentes diferenciados por su complejidad estratégica: un agente aleatorio como referencia base, un agente heurístico basado en reglas específicas del juego, un agente basado en búsqueda Monte Carlo (Monte Carlo Tree Search, MCTS), y un agente basado en redes neuronales profundas (Deep Neural Networks, DNN) simple.

El motor del juego fue diseñado como un sistema multiagente competitivo, garantizando simulaciones reproducibles y controladas. Los experimentos empíricos, sustentados en miles de partidas automatizadas, demostraron que el agente basado en MCTS logró el desempeño más destacado, alcanzando un porcentaje máximo de victorias del 94 % frente al agente aleatorio y superando consistentemente al agente heurístico. Por su parte, el agente DNN presentó resultados competitivos frente a agentes básicos, pero fue superado por el agente MCTS y el heurístico en escenarios complejos, sugiriendo la necesidad de enfoques híbridos o entrenamiento adicional.

El análisis riguroso de resultados subraya la relevancia crítica del manejo eficiente de incertidumbres y la anticipación estratégica en juegos de información parcial como Briscas. En conclusión, aunque los enfoques basados en reglas heurísticas y aprendizaje supervisado ofrecen un rendimiento aceptable, las técnicas avanzadas como MCTS proporcionan beneficios significativos en contextos dinámicos y adversariales. Finalmente, se delinean futuras investigaciones orientadas hacia la integración de arquitecturas neuronales avanzadas (CNN, LSTM, Transformers), métodos de aprendizaje por refuerzo profundo (Proximal Policy Optimization, PPO) y la adopción de frameworks estandarizados como OpenSpiel, para fortalecer la adaptabilidad y robustez de agentes en Briscas y juegos similares.

Índice

1. Introducción **5**
- 1.1. Antecedentes . . . 5
- 1.2. Propósito del Proyecto . . . 5
- 1.3. Alcance y Limitaciones . . . 5
- 1.4. Declaración del Problema . . . 5

2. Fundamentos Teóricos **6**
- 2.1. Descripción del Juego de Briscas . . . 6
  - 2.1.1. Alcance de la Implementación . . . 6
  - 2.1.2. Baraja y Valor de las Cartas . . . 6
  - 2.1.3. Condiciones Iniciales del Juego . . . 7
  - 2.1.4. Mecánica del Juego . . . 7
  - 2.1.5. Variaciones por Número de Jugadores . . . 7
  - 2.1.6. Cambio de Carta de Triunfo . . . 7
  - 2.1.7. Final del Juego y Condiciones de Victoria . . . 8
- 2.2. Pipeline y Diseño del Game Engine . . . 8
  - 2.2.1. Estructura General del Pipeline . . . 8
  - 2.2.2. Componentes del Game Engine . . . 8
  - 2.2.3. Diagrama del Pipeline del Juego . . . 10
- 2.3. Técnicas de Búsqueda Adversarial . . . 11
  - 2.3.1. Limitaciones en Juegos Complejos . . . 11
  - 2.3.2. Juegos Estocásticos y Parcialmente Observables . . . 11
  - 2.3.3. Búsqueda como Resolución de Problemas . . . 12
- 2.4. Agente Aleatorio como Línea Base . . . 12
  - 2.4.1. Limitaciones del Enfoque Aleatorio . . . 12
- 2.5. Técnicas de Búsqueda Adversarial . . . 12
  - 2.5.1. Monte Carlo Simulation . . . 12
  - 2.5.2. Deep Neural Networks . . . 13
  - 2.5.3. Estado del Arte en IA Aplicada a Juegos de Cartas . . . 13
  - 2.5.4. Técnicas Híbridas: Integración de DNN y MCTS . . . 14

3. Justificación de las Técnicas Seleccionadas **15**
- 3.1. Elección del Agente Aleatorio . . . 15
- 3.2. Elección del Agente Heurístico . . . 15
- 3.3. Elección de Monte Carlo Tree Search (MCTS) . . . 16
  - 3.3.1. Ventajas Clave de MCTS . . . 16
- 3.4. Elección de Redes Neuronales Profundas (DNN) . . . 17
- 3.5. Comparación General de Técnicas . . . 18

4. Desarrollo e Implementación **19**
- 4.1. Motor del Juego como Sistema Multiagente Competitivo . . . 19
- 4.2. Agente Aleatorio . . . 19
  - 4.2.1. Lógica del Agente . . . 20

4.2.2. Características de Implementación 20
4.2.3. Propósito en la Evaluación 20
4.3. Agente Heurístico 20
4.3.1. Lógica y Estrategia del Agente 21
4.3.2. Características de la Implementación 21
4.3.3. Lógica del Agente Heurístico 21
4.3.4. Fundamentación Teórica basada en la Teoría de la Utilidad 22
4.3.5. Ventajas y Limitaciones 22
4.4. Agente Monte Carlo 23
4.4.1. Estructura del Agente MCTS 23
4.4.2. Criterio de Selección: Fórmula UCT 23
4.4.3. Características de la Implementación 24
4.4.4. Funcionamiento General 24
4.4.5. Fundamento Teórico: Decisiones Secuenciales y POMDP 24
4.4.6. Interpretación de la Constante de Exploración (c) 25
4.4.7. Selección de Valores para la Evaluación 25
4.4.8. Evaluación de Configuraciones del Agente MCTS 26
4.4.9. Métricas Evaluadas: 26
4.4.10. Almacenamiento de Resultados: 27
4.4.11. Generación de Gráficos: 27
4.4.12. Análisis de Resultados del Agente MCTS 27
4.4.13. Distribución de Puntos según la Constante de Exploración c. 27
4.4.14. Porcentaje de Victorias: Relación entre c e Iteraciones. 28
4.4.15. Duración Promedio de las Partidas. 29
4.4.16. Conclusión. 29
4.5. Agente Basado en Redes Neuronales (DNN) 29
4.5.1. Diseño de la Arquitectura de la Red Neuronal 29
4.5.2. Representación del Estado en Juegos de Cartas 29
4.5.3. Generación de Datasets Sintéticos 30
4.5.4. Estructura de la Red: 30
4.5.5. Generación y Preparación del Dataset 31
4.5.6. Resumen del Dataset: 31
4.5.7. Proceso de Entrenamiento 31
4.5.8. Hiperparámetros Principales: 31
4.5.9. Integración del Agente DNN 31
4.5.10. Conclusión del Agente DNN 32

5. Resultados **33**
5.1. Métricas de Evaluación Inicial 33
5.2. Comparación de Desempeño General 34
5.3. Visualizaciones y Análisis 34
5.3.1. Distribución de Puntos 34
5.3.2. Duración de las Partidas 35
5.3.3. Relación Entre Desempeño y Tiempo 36
5.3.4. Correlación de Puntos Entre Agentes 36

5.4. Evaluación del Agente Heurístico en Modo Agresivo 37
5.4.1. Porcentaje de Victorias: 37
5.4.2. Análisis Complementario: 37
5.5. Discusión de Resultados Iniciales 39
5.6. Importancia Estratégica de la Fase Inicial en Juegos de Cartas 40
5.7. Métricas de Evaluación Robusta 40
5.8. Evaluación del Agente DNN Inicial 40
5.8.1. Análisis de Resultados 41

6. Análisis y Discusión **46**
6.1. Evaluación Estratégica de los Agentes 46
6.2. Limitaciones Identificadas 46
6.3. Perspectiva sobre el Agente DNN 47
6.4. Retos al Aplicar IA Avanzada a Briscas 47
6.5. Implicaciones Éticas y Sociales 48

7. Conclusiones **49**

8. Trabajo Futuro **50**
8.1. Comparativa de Algoritmos: MCTS, DNN e Híbridos 50
8.2. Exploración de Arquitecturas Avanzadas (CNN, LSTM, Transformers) . . . 50
8.3. Oportunidades con OpenSpiel y Frameworks Similares 51
8.4. Exploración de Planificación en Juegos Complejos 51
8.5. Propuesta para Integrar PPO y Auto-juego 52
8.6. Conclusión del Trabajo Futuro 53

A. Código Fuente **56**

B. Entorno de Desarrollo **56**

1. Introducción

1.1. Antecedentes

La inteligencia artificial (IA) ha transformado significativamente el desarrollo de agentes capaces de tomar decisiones en entornos competitivos, inciertos y parcialmente observables, como los juegos de mesa. Clásicos ejemplos como el ajedrez, Go y póker han servido como plataformas clave para explorar estrategias de búsqueda, aprendizaje y razonamiento [1] [2]. En este contexto, el juego de Briscas, ampliamente jugado en Puerto Rico y otros países hispanohablantes, representa un caso interesante de juego de cartas con información parcial, alta variabilidad y un componente fuerte de estrategia, lo que lo convierte en una excelente oportunidad para aplicar técnicas de IA adversarial.

1.2. Propósito del Proyecto

El propósito de este proyecto es diseñar, implementar y comparar agentes inteligentes que jueguen Briscas utilizando distintas técnicas de búsqueda adversarial. En particular, se busca evaluar el desempeño de un agente basado en redes neuronales profundas (DNNs), comparándolo con agentes tradicionales que empleen técnicas como Monte Carlo Simulation y otras heurísticas. Este análisis permitirá identificar fortalezas y debilidades de cada enfoque dentro de un entorno competitivo controlado.

1.3. Alcance y Limitaciones

Este proyecto se enfoca en la implementación de Briscas para 2 y 4 jugadores, simulando las dinámicas del juego con reglas estandarizadas. El alcance incluye el desarrollo de un motor de juego, la programación de múltiples agentes con distintos enfoques estratégicos, y la evaluación cuantitativa de su desempeño a través de partidas simuladas. Se excluyen aspectos relacionados a la interacción humana directa como el uso de señas entre jugadores o comportamientos no formales. Tampoco se contempla la optimización exhaustiva de hiperparámetros de entrenamiento para los modelos DNN, priorizando la validación comparativa entre técnicas.

1.4. Declaración del Problema

El problema central abordado en este proyecto consiste en determinar qué técnica de inteligencia artificial adversarial es más efectiva para jugar Briscas, considerando factores como el nivel de acierto en la toma de decisiones, la eficiencia computacional y la adaptabilidad al entorno de juego. Se parte de la hipótesis de que, aunque los modelos tradicionales como Monte Carlo ofrecen un enfoque sólido y comprensible, los modelos basados en aprendizaje profundo podrían mostrar ventajas en la generalización de patrones de juego complejos. El reto consiste en implementar ambos enfoques y validarlos en condiciones comparables para determinar cuál produce mejores resultados.

2. Fundamentos Teóricos

2.1. Descripción del Juego de Briscas

Briscas es un juego de cartas tradicionalmente jugado con una baraja española de 40 cartas y es especialmente popular en países hispanohablantes como España y Puerto Rico. El objetivo principal del juego es acumular la mayor cantidad de puntos posibles a través de la captura de bazas.

2.1.1. Alcance de la Implementación

Si bien Briscas admite modalidades tanto individuales como en pareja (**2 vs 2**), la presente implementación se enfoca exclusivamente en partidas de **1 vs 1**. Aunque este reporte describe las reglas generales del juego, es importante destacar que la lógica del motor desarrollado —incluyendo la función `EstadoBriscas.utilidad()` y el flujo general de juego— trata a cada jugador de manera independiente, sin soporte para dinámicas de equipo. Esta restricción fue adoptada para simplificar el diseño de agentes y centrar la evaluación en estrategias individuales.

Asimismo, la **regla de intercambio de la carta de triunfo** (descrita en la Sección 2.1.6.) se documenta como una opción estratégica presente en variantes tradicionales del juego. Sin embargo, esta funcionalidad no ha sido implementada en el código actual, constituyendo una limitación conocida del sistema. Su incorporación se reserva para futuras versiones orientadas a agentes más avanzados.

2.1.2. Baraja y Valor de las Cartas

El juego se juega con una baraja de 40 cartas dividida en cuatro palos: oros, copas, espadas y bastos. Cada palo contiene las siguientes cartas con su respectivo valor en puntos:

Cuadro 1: Valor en puntos de las cartas en el juego de Briscas.

Carta	**Valor (puntos)**
As	11
Tres	10
Rey	4
Caballo	3
Sota	2
7, 6, 5, 4, 2	0

El total de puntos disponibles en una partida completa es de 120. El jugador o pareja que acumule al menos 61 puntos gana la partida. En caso de empate (60-60), la partida se considera nula.

2.1.3. Condiciones Iniciales del Juego

Se reparte un total de tres cartas a cada jugador. Luego, se descubre una carta del mazo central para determinar el palo de triunfo (*brisca*), la cual se deja parcialmente visible debajo del mazo. El resto de las cartas se mantiene en el mazo boca abajo para ser robadas durante el juego.

2.1.4. Mecánica del Juego

El juego se desarrolla por turnos, donde cada jugador juega una carta en cada baza. El jugador que haya iniciado la baza es seguido por el otro jugador (o en caso de cuatro jugadores, el juego avanza en sentido contrario a las manecillas del reloj).

El ganador de la baza se determina de la siguiente manera:

- Si uno o más jugadores han jugado cartas del palo de triunfo, gana la carta de mayor valor entre ellas.
- Si nadie jugó cartas del palo de triunfo, gana la carta de mayor valor del palo de salida.

Una vez determinada la baza ganadora:

- Los jugadores roban una carta del mazo, comenzando por el ganador de la baza.
- El ganador de la baza inicia la siguiente ronda.

El juego continúa hasta que se agota el mazo y se juegan todas las cartas restantes en la mano.

2.1.5. Variaciones por Número de Jugadores

Juego de 2 Jugadores: Cada jugador juega individualmente. Las reglas descritas anteriormente se aplican directamente a esta modalidad.

Juego de 4 Jugadores: Se juega por parejas (dos contra dos). Los compañeros se sientan uno frente al otro y comparten los puntos que acumulan en conjunto. Las jugadas siguen las mismas reglas de captura de bazas y robo de cartas, adaptadas a un ciclo de turnos entre cuatro jugadores. No se implementan interacciones ni señales entre compañeros en esta versión computacional.

2.1.6. Cambio de Carta de Triunfo

Bajo ciertas condiciones, un jugador puede intercambiar una carta de su mano por la carta de triunfo revelada. Las condiciones más comunes son:

- Si posee el Siete del palo de triunfo, puede intercambiarlo por un As, Tres, Rey, Caballo o Sota del mismo palo.
- Si posee el Dos del palo de triunfo, puede intercambiarlo por una carta baja como el Siete, Seis, Cinco o Cuatro del mismo palo.

Esta acción es opcional y debe realizarse antes de que queden pocas cartas en el mazo (usualmente antes de la penúltima baza). Por simplicidad, esta regla puede ser omitida o implementada como una opción adicional para agentes avanzados.

2.1.7. Final del Juego y Condiciones de Victoria

El juego finaliza cuando ya no hay más cartas en el mazo y los jugadores han agotado todas las cartas de sus manos. Los puntos acumulados por cada jugador (o equipo) se suman, y gana quien haya alcanzado al menos 61 puntos. El empate ocurre si ambos alcanzan 60 puntos [3] [4].

2.2. Pipeline y Diseño del Game Engine

El motor del juego (*Game Engine*) para Briscas se ha diseñado siguiendo los principios de búsqueda adversarial y sistemas multiagente competitivos descritos en la literatura de inteligencia artificial [5]. Este diseño permite gestionar de manera eficiente la dinámica del juego, integrando agentes inteligentes que toman decisiones estratégicas bajo condiciones de incertidumbre y parcial observabilidad.

Adicionalmente, conceptos de **planificación automática** aportan un marco complementario para estructurar la toma de decisiones de los agentes. Según Russell y Norvig [5, Cap. 11], los agentes planificadores generan secuencias óptimas de acciones orientadas a alcanzar objetivos en entornos regidos por restricciones y reglas definidas. En el contexto de Briscas, aunque cada jugada se decide turno a turno, los agentes avanzados (como los basados en heurísticas o MCTS) incorporan principios de planificación al anticipar posibles consecuencias futuras de sus decisiones dentro de un espacio dinámico y parcialmente observable.

2.2.1. Estructura General del Pipeline

El flujo general del juego se puede describir mediante las siguientes etapas, donde cada agente actúa como un planificador que selecciona acciones optimizando su utilidad esperada:

1. **Inicialización del Juego:** Definición del estado inicial S_0, incluyendo la distribución aleatoria de cartas y la asignación del palo de triunfo.

2. **Ciclo de Juego:** Ejecución iterativa donde los agentes planifican y seleccionan jugadas en función del estado actual y sus estrategias internas.

3. **Evaluación de la Baza:** Determinación del resultado inmediato de cada acción, actualizando el estado conforme a las reglas.

4. **Condición de Terminación:** Detección del estado terminal cuando se agotan las cartas.

5. **Cálculo de Utilidad:** Evaluación final mediante la función $U(S)$, representando la culminación del "plan.[ej]ecutado por cada agente.

2.2.2. Componentes del Game Engine

El diseño del sistema incorpora elementos clave tanto de búsqueda como de planificación:

- **Estado del Juego (S):** Modelo completo del entorno, esencial para que los agentes formulen planes o estrategias adaptativas.

- **Acciones (A):** Espacio de acciones legales, sobre el cual los agentes aplican técnicas de búsqueda o planificación.
- **Función de Transición (P):** Define cómo evoluciona el estado tras cada acción, fundamental en procesos de planificación secuencial.
- **Función de Utilidad (U):** Objetivo que guía las decisiones estratégicas de los agentes.
- **Agentes IA:** Implementaciones que combinan búsqueda adversarial y planificación aproximada, como MCTS y DNN.

2.2.3. Diagrama del Pipeline del Juego

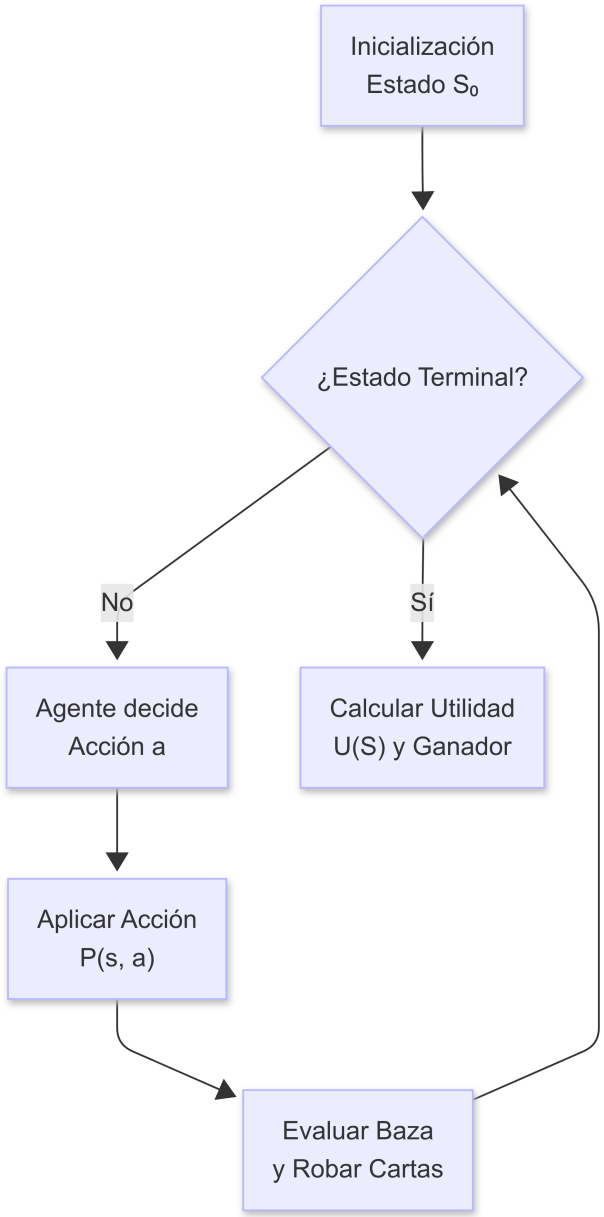

Figura 1: Pipeline general del motor de juego para Briscas.

Este enfoque híbrido, que combina principios de búsqueda adversarial y planificación automática, permite la integración flexible de estrategias de decisión avanzadas. Así, se garantiza que cada agente no solo reaccione al estado actual, sino que anticipe escenarios futuros dentro del entorno competitivo definido, optimizando su desempeño bajo incertidumbre.

2.3. Técnicas de Búsqueda Adversarial

La búsqueda adversarial comprende métodos utilizados por agentes que deben tomar decisiones óptimas en entornos competitivos, típicamente modelados como juegos de suma cero, donde la ganancia de un jugador representa la pérdida de su oponente [5]. Entre las técnicas clásicas destaca el algoritmo **Minimax**, que explora un árbol de decisiones considerando todas las jugadas posibles y evalúa los estados terminales mediante funciones heurísticas [5, Cap. 6]. La **poda alfa-beta** optimiza este proceso al reducir los nodos evaluados sin afectar la optimalidad [5, Sec. 6.3].

Sin embargo, más allá de la simple exploración de árboles, la toma de decisiones en juegos adversariales también puede ser modelada desde la perspectiva de la **teoría de la utilidad**, donde cada acción seleccionada por el agente busca maximizar su utilidad esperada bajo condiciones de incertidumbre [5, Cap. 15]. Este enfoque permite que los agentes racionales no solo reaccionen al estado actual, sino que evalúen el beneficio esperado de cada jugada considerando múltiples factores relevantes, como el valor de las cartas, la probabilidad de éxito y el control estratégico del juego [5, Sec. 15.4].

Además, en entornos complejos como los juegos de cartas, las decisiones pueden conceptualizarse mediante **redes de decisión**, donde las variables aleatorias (cartas ocultas) y las acciones posibles se relacionan a través de nodos que representan la utilidad esperada [5, Sec. 15.5]. Aunque este modelo no se implementa explícitamente en este proyecto, ofrece un marco teórico sólido para el diseño de agentes que optimizan decisiones estratégicas en escenarios adversariales.

2.3.1. Limitaciones en Juegos Complejos

En juegos como *Briscas*, las estrategias tradicionales enfrentan serias limitaciones debido a:

1. **Explosión combinatoria**: El espacio de estados crece exponencialmente.
2. **Información incompleta**: No se conocen todas las cartas en juego.
3. **Componentes estocásticos**: La aleatoriedad en la distribución de cartas introduce incertidumbre constante.

Este tipo de entornos también puede describirse como **Problemas de Decisión Secuencial**, donde cada jugada afecta las decisiones futuras del agente. Según Russell y Norvig [5, Cap. 16], los *Markov Decision Processes* (MDP) ofrecen un marco formal para este tipo de problemas, maximizando la recompensa acumulada a lo largo de múltiples decisiones. No obstante, dado que *Briscas* es un juego con información parcial, su estructura se asemeja más a un *Partially Observable MDP* (POMDP), donde los agentes deben actuar basándose en creencias sobre el estado oculto del juego [5, Sec. 16.4].

2.3.2. Juegos Estocásticos y Parcialmente Observables

Según Russell y Norvig [5, Sec. 6.5, 6.6], estos escenarios se clasifican como **juegos estocásticos y de información parcial**. La incertidumbre proviene tanto de eventos aleatorios como de la falta de información completa sobre el estado del juego. En este contexto,

es esencial utilizar técnicas que gestionen explícitamente dicha incertidumbre, como las simulaciones de **Monte Carlo** o estrategias basadas en razonamiento probabilístico.

Además, técnicas como **Monte Carlo Tree Search (MCTS)** abordan de manera eficiente el equilibrio entre **exploración** y **explotación**, siguiendo principios derivados de los *Problemas de Bandit* [5, Sec. 16.3]. La aplicación de la fórmula UCT en MCTS responde directamente a este dilema, permitiendo que el agente explore nuevas opciones sin descuidar aquellas que históricamente han mostrado buenos resultados.

Por último, el concepto de **valor de la información** [5, Sec. 15.6] es relevante en este contexto, ya que técnicas como MCTS pueden interpretarse como un proceso donde el agente ïnvierte.[en] obtener información adicional mediante simulaciones antes de tomar una decisión definitiva, mejorando así su expectativa de éxito en entornos inciertos como *Briscas*.

2.3.3. Búsqueda como Resolución de Problemas

De acuerdo con los principios generales descritos en [5, Cap. 3], en dominios complejos y bajo incertidumbre, los agentes deben priorizar métodos de **búsqueda informada** o **aproximada**, optimizando la eficiencia sin depender de la exploración exhaustiva. Este enfoque, complementado con modelos de decisión secuencial y teoría de la utilidad, fundamenta la adopción de algoritmos avanzados como **MCTS**, los cuales se detallan en la siguiente sección.

2.4. Agente Aleatorio como Línea Base

En el campo de la inteligencia artificial, es una práctica común utilizar agentes que actúan de manera aleatoria como referencia inicial para evaluar el desempeño de técnicas más avanzadas [5]. Estos agentes, al no aplicar ningún tipo de estrategia ni razonamiento, permiten establecer un **baseline** contra el cual comparar algoritmos informados, como los basados en búsqueda adversarial o aprendizaje automático.

2.4.1. Limitaciones del Enfoque Aleatorio

Según la literatura, los agentes aleatorios son inherentemente ineficientes en entornos donde la toma de decisiones estratégicas es clave [5]. En juegos como Briscas, donde factores como la gestión de cartas, el control del palo de triunfo y la anticipación de movimientos del oponente son determinantes, un agente aleatorio tiende a obtener un bajo porcentaje de victorias y una acumulación deficiente de puntos.

A pesar de estas limitaciones, su simplicidad ofrece ventajas en términos de velocidad de ejecución y utilidad como herramienta de comparación.

2.5. Técnicas de Búsqueda Adversarial

2.5.1. Monte Carlo Simulation

El método de búsqueda Monte Carlo Tree Search (MCTS) es una técnica efectiva en juegos con alta incertidumbre y grandes espacios de estados. A diferencia del enfoque determinista de minimax, MCTS utiliza simulaciones aleatorias para evaluar las posibles acciones a tomar desde el estado actual del juego. El proceso completo de MCTS se divide en cuatro

fases principales: selección, expansión, simulación y retropropagación [5]. Durante la selección, el algoritmo recorre el árbol de juego utilizando criterios que equilibran exploración y explotación (como la fórmula UCB1). La expansión añade nuevos nodos al árbol cuando se alcanzan estados no explorados previamente. En la simulación se realizan jugadas aleatorias hasta llegar a un estado terminal, y en la retropropagación, los resultados de estas simulaciones se utilizan para actualizar la información de valor de cada nodo [5]. Esta técnica es particularmente ventajosa en juegos como Briscas, ya que permite evaluar decisiones sin necesidad de una función heurística precisa y maneja adecuadamente la incertidumbre inherente a las cartas ocultas y al reparto inicial aleatorio [5].

2.5.2. Deep Neural Networks

Las redes neuronales profundas (Deep Neural Networks, DNN) constituyen un modelo computacional inspirado en el comportamiento del cerebro humano, capaz de aproximar funciones extremadamente complejas mediante múltiples capas de unidades de procesamiento no lineal [5]. En el contexto de juegos, las DNN han sido utilizadas exitosamente para modelar decisiones estratégicas, como quedó demostrado en el caso de AlphaGo, que combinó MCTS con redes neuronales profundas para vencer a campeones humanos [5]. El entrenamiento de estas redes puede realizarse mediante técnicas de aprendizaje supervisado—usando un conjunto etiquetado de datos para ajustar los parámetros del modelo—o aprendizaje por refuerzo, donde el modelo aprende mediante prueba y error interactuando directamente con el entorno de juego. Para Briscas, el uso de DNN podría permitir la identificación automática de patrones estratégicos complejos y proporcionar ventajas significativas frente a técnicas heurísticas tradicionales [5].

Adicionalmente, Russell y Norvig [5, Cap. 22] destacan que las DNN no solo son efectivas en aprendizaje supervisado, sino que también forman la base de los sistemas modernos de **Deep Reinforcement Learning (Deep RL)**. Estas redes permiten a los agentes optimizar decisiones complejas mediante aproximación de funciones en entornos dinámicos y parcialmente observables, como ocurre en juegos estratégicos. La combinación de DNN con técnicas de búsqueda y aprendizaje por refuerzo ha sido clave en avances recientes, permitiendo desarrollar agentes capaces de adaptarse y mejorar de forma autónoma en escenarios competitivos.

2.5.3. Estado del Arte en IA Aplicada a Juegos de Cartas

Aunque no existen estudios documentados específicos sobre la aplicación de redes neuronales profundas (DNN) o Monte Carlo Tree Search (MCTS) al juego de Briscas, la literatura académica presenta varios ejemplos relevantes de aplicaciones exitosas en juegos similares. Destacan especialmente los proyectos *ReBeL*, *DeepStack* y la implementación en el juego suizo Jass.

El algoritmo **ReBeL** (*Recursive Belief-based Learning*), propuesto por Brown et al. (2020) [6], combina aprendizaje profundo por refuerzo y búsqueda basada en árboles para juegos con información imperfecta como el póker, logrando resultados superhumanos al integrar representaciones probabilísticas del estado del juego con una eficiente búsqueda Monte Carlo.

Otro referente clave es **DeepStack** (Moravčík et al., 2017) [7], que emplea redes neuronales recurrentes (LSTM) para modelar creencias sobre cartas ocultas en póker, permitiendo una toma de decisiones informada en escenarios con incertidumbre significativa y espacios de decisión extremadamente grandes.

En el contexto específico de juegos tipo "trick-taking", un estudio realizado por investigadores de ETH Zurich aplicó una arquitectura híbrida combinando redes convolucionales (CNN) y MCTS al juego suizo Jass, obteniendo tasas de victoria del 72 % frente a agentes heurísticos en torneos simulados [8]. Estos resultados sugieren una prometedora transferibilidad metodológica hacia el juego de Briscas debido a sus similitudes estructurales.

2.5.4. Técnicas Híbridas: Integración de DNN y MCTS

La combinación de **Monte Carlo Tree Search (MCTS)** y **Redes Neuronales Profundas (DNN)** ha demostrado ser particularmente efectiva en juegos que presentan información parcial y decisiones estratégicas complejas. Un enfoque híbrido permite aprovechar la capacidad de las redes neuronales para estimar rápidamente el valor esperado de estados del juego, reduciendo drásticamente el número de simulaciones necesarias mediante MCTS para explorar el espacio de acción.

En juegos como Hearthstone y Magic: The Gathering, este enfoque híbrido ha reducido la cantidad de simulaciones necesarias en un factor de diez sin comprometer la calidad de las decisiones estratégicas [9, 10]. Esta eficiencia hace que la integración DNN-MCTS sea una técnica prometedora para aplicaciones futuras en Briscas, especialmente en escenarios competitivos.

3. Justificación de las Técnicas Seleccionadas

Para el desarrollo de agentes inteligentes en el juego de Briscas, se seleccionaron cuatro enfoques distintos: **Agente Aleatorio**, **Agente Heurístico**, **Monte Carlo Tree Search (MCTS)** y **Redes Neuronales Profundas (DNN)**. Esta combinación permite evaluar tanto técnicas avanzadas de inteligencia artificial como establecer referencias mediante estrategias simples y comportamientos no informados.

3.1. Elección del Agente Aleatorio

El **Agente Aleatorio** fue diseñado como una *línea base* para la evaluación comparativa del desempeño de los agentes inteligentes implementados en este proyecto. Este agente toma decisiones sin aplicar estrategias, seleccionando en cada turno una carta de su mano de forma completamente aleatoria, sin considerar el estado del juego, las cartas previas ni el contexto estratégico. Su comportamiento simula un jugador sin conocimiento ni planificación, representando el nivel más bajo de desempeño esperado en un entorno competitivo.

La inclusión de este agente cumple varios objetivos fundamentales:

- **Referencia Cuantitativa:** Proporciona un punto de comparación objetivo para medir las mejoras obtenidas mediante técnicas avanzadas como *Monte Carlo Tree Search (MCTS)*, *Deep Neural Networks (DNN)* y estrategias heurísticas.
- **Validación de Estrategias:** Asegura que el rendimiento superior de los agentes inteligentes se deba a la aplicación de algoritmos de búsqueda y aprendizaje, y no a factores aleatorios.
- **Evaluación de Robustez:** Permite identificar escenarios donde un comportamiento aleatorio puede resultar competitivo, ofreciendo información útil para ajustar y fortalecer las estrategias de los agentes avanzados.

3.2. Elección del Agente Heurístico

El **Agente Heurístico** fue incorporado como una aproximación intermedia entre el comportamiento puramente aleatorio y los agentes basados en inteligencia artificial avanzada. Inspirado en los principios de diseño de agentes racionales descritos por Russell y Norvig [5, Cap. 2], este agente implementa un conjunto de reglas simples fundamentadas en conocimiento del dominio específico del juego de Briscas.

Entre las decisiones heurísticas aplicadas se encuentran:

- Priorizar cartas de alto valor cuando existe una alta probabilidad de ganar la baza.
- Descartar cartas sin puntos en situaciones desfavorables o cuando no se puede controlar la jugada.
- Evitar jugar cartas de triunfo innecesariamente si no se obtiene ganancia estratégica.

Este agente permite cumplir varios propósitos clave dentro del proceso de evaluación:

- **Modelo de Razonamiento Básico:** Representa una aproximación humana intuitiva al juego, facilitando la comparación entre comportamientos empíricos y técnicas algorítmicas más elaboradas.
- **Punto de Referencia Intermedio:** Proporciona un punto de comparación más exigente que el agente aleatorio, permitiendo evaluar la capacidad de los agentes MCTS y DNN para superar estrategias basadas en reglas simples.
- **Equilibrio Computacional:** Sirve como alternativa eficiente frente a algoritmos costosos, demostrando que mejoras significativas pueden lograrse sin requerir simulaciones extensivas o entrenamiento profundo.

3.3. Elección de Monte Carlo Tree Search (MCTS)

El algoritmo **Monte Carlo Tree Search (MCTS)** ha sido seleccionado debido a su eficacia comprobada en entornos caracterizados por:

- **Alta complejidad del espacio de estados**, como ocurre en Briscas por la combinación de cartas, jugadas y posibles escenarios.
- **Información parcial**, donde los agentes desconocen las cartas ocultas del oponente y las restantes en el mazo.
- **Elementos estocásticos**, derivados de la distribución aleatoria de cartas.

Según Russell y Norvig [5, Sec. 6.4], MCTS es ideal para estos contextos, ya que combina simulaciones aleatorias con un equilibrio estratégico entre exploración y explotación, utilizando mecanismos como la fórmula UCT (*Upper Confidence Bounds for Trees*) para guiar la toma de decisiones sin necesidad de construir un árbol completo.

3.3.1. Ventajas Clave de MCTS

Las razones principales para su implementación en este proyecto incluyen:

1. **Gestión eficiente de la incertidumbre**: Capacidad para operar bajo condiciones de riesgo e información incompleta.
2. **Reducción del costo computacional**: Evita la exploración exhaustiva, enfocándose en las ramas más prometedoras.
3. **Flexibilidad**: Permite ajustar parámetros como el número de simulaciones o la constante de exploración (c) para optimizar su rendimiento.
4. **Respaldo en el estado del arte**: Su éxito en juegos complejos como Go, póker y otros escenarios estratégicos refuerza su aplicabilidad a Briscas.

En síntesis, MCTS ofrece un enfoque robusto y adaptable para enfrentar los desafíos propios del juego de Briscas, equilibrando eficacia estratégica con eficiencia computacional.

3.4. Elección de Redes Neuronales Profundas (DNN)

La incorporación de un **Agente basado en Deep Neural Networks (DNN)** responde a la necesidad de explorar enfoques contemporáneos de inteligencia artificial, centrados en el aprendizaje automático a partir de datos. Este tipo de agente no depende de reglas predefinidas ni de simulaciones en tiempo real, sino que aprende patrones estratégicos a través de un proceso de entrenamiento supervisado sobre partidas simuladas.

Según Russell y Norvig [5, Sec. 6.8], las redes neuronales profundas son especialmente efectivas en contextos donde:

- Es posible generar grandes volúmenes de datos representativos del entorno de decisión.
- Se requiere una capacidad de **generalización**, permitiendo al agente enfrentar situaciones no vistas durante el entrenamiento.
- La rapidez en la toma de decisiones es prioritaria una vez desplegado el modelo.

Las principales razones para la elección de esta técnica en el contexto del juego de Briscas son las siguientes:

- **Aprendizaje de Estrategias Complejas:** El agente DNN tiene la capacidad de identificar patrones no triviales en las decisiones óptimas, aprovechando correlaciones que podrían pasar desapercibidas en enfoques basados en heurísticas o búsqueda directa.
- **Ejecución Eficiente:** Una vez completado el proceso de entrenamiento, el agente puede tomar decisiones casi instantáneas, lo que lo convierte en una solución ideal para entornos donde el tiempo de respuesta es crítico.
- **Reducción de Costo Computacional en Tiempo de Ejecución:** A diferencia de algoritmos como MCTS, que requieren recursos significativos en cada turno, el agente DNN traslada la carga computacional al proceso previo de entrenamiento.
- **Comparativa de Enfoques de IA:** Permite analizar el contraste entre agentes que dependen de *simulación en línea* (como MCTS) y aquellos que basan sus decisiones en conocimiento adquirido previamente mediante aprendizaje.

Adicionalmente, el uso de DNN sienta las bases para futuras integraciones de técnicas híbridas, combinando aprendizaje profundo con métodos de búsqueda, tal como se ha demostrado en sistemas avanzados como AlphaZero [5, Sec. 6.8].

La implementación de este agente busca evaluar el potencial del aprendizaje automático en juegos de cartas estratégicos, así como sus limitaciones frente a enfoques tradicionales y estocásticos.

Asimismo, la versatilidad de las DNN permite su integración con algoritmos de **aprendizaje por refuerzo**, donde técnicas como *Policy Search* y *Proximal Policy Optimization (PPO)* facilitan la evolución continua de estrategias a través de la interacción directa con el entorno [5, Cap. 23]. Esta capacidad de auto-mejora posiciona al agente DNN como una plataforma ideal para futuras extensiones hacia modelos de aprendizaje autónomo y adaptativo en Briscas.

3.5. Comparación General de Técnicas

Cuadro 2: Comparación entre técnicas implementadas y discutidas en el contexto del juego de Briscas.

Técnica	**Aptitud para Briscas**	**Complejidad**	**Propósito**
Agente Aleatorio	Muy baja	Baja	Línea base para evaluación comparativa
Minimax	Baja	Alta	Referencia teórica; inviable por complejidad en juegos con incertidumbre
Averaging Over Clairvoyance	Muy baja	Media	Aproximación simplificada discutida, no apta en escenarios reales
Modelos para Juegos Parcialmente Observables	Media	Alta	Manejo explícito de incertidumbre (teórico)
MCTS	Alta	Media	Exploración eficiente mediante simulaciones estocásticas
DNN	Alta	Alta (entrenamiento) / Baja (ejecución)	Aprendizaje de patrones estratégicos a partir de datos

En resumen, el **Agente Aleatorio** establece una referencia básica sin capacidad estratégica, mientras que el **Agente Heurístico** actúa como un enfoque intermedio, aplicando reglas simples basadas en conocimiento del dominio. Por otro lado, las técnicas avanzadas implementadas mediante **MCTS** y **DNN** demuestran un desempeño significativamente superior al aprovechar métodos de inteligencia artificial adaptados a las características complejas de Briscas, especialmente en escenarios adversariales con información incompleta y elementos estocásticos. Esta progresión de agentes permite evaluar de manera integral cómo distintas aproximaciones, desde las más simples hasta las más sofisticadas, impactan la toma de decisiones y el rendimiento en un entorno competitivo.

4. Desarrollo e Implementación

4.1. Motor del Juego como Sistema Multiagente Competitivo

El juego de Briscas puede ser formalmente descrito como un sistema multiagente competitivo definido por la tupla $G = (S, A, P, U)$, donde:

- S representa el conjunto de estados posibles del juego.
- A denota el conjunto de acciones disponibles para cada agente (jugador).
- P indica la función de transición de estado, determinada por las acciones tomadas por los agentes y las reglas del juego.
- U corresponde a la función de utilidad o recompensa asociada a los estados terminales del juego.

Cada agente busca maximizar su propia función de utilidad, la cual refleja la cantidad total de puntos acumulados al finalizar la partida. En este entorno competitivo de suma cero, las decisiones óptimas de un jugador dependen no solo de su estrategia, sino también de la anticipación de las posibles acciones del oponente.

Según Russell y Norvig [5, Cap. 17], los entornos multiagente competitivos se caracterizan por dinámicas estratégicas complejas, donde cada agente racional toma decisiones considerando tanto su propio objetivo como las posibles respuestas de los demás agentes. Briscas se clasifica como un juego *no cooperativo*, *secuencial* y de *información parcial*, lo que implica que las estrategias óptimas deben adaptarse continuamente al estado cambiante del juego y a la incertidumbre sobre las cartas del oponente.

El motor del juego desarrollado incorpora estas características, gestionando la dinámica de turnos, la aplicación de las reglas y la interacción entre agentes autónomos que compiten por maximizar su utilidad. Este diseño permite simular partidas completas bajo diferentes configuraciones estratégicas, evaluando el comportamiento de agentes inteligentes en un entorno coherente y reproducible.

Además, conforme a los principios de la *Teoría de Juegos No Cooperativos* [5, Sec. 17.2], los agentes implementados —especialmente aquellos basados en heurísticas y técnicas de búsqueda como MCTS— buscan aproximarse a estrategias que optimicen sus resultados frente a adversarios racionales. Aunque no se alcanza un equilibrio de Nash formal, el diseño promueve decisiones que consideran las posibles contramedidas del oponente, reforzando la competitividad del entorno.

Esta formalización del juego como un sistema multiagente competitivo no solo facilita la implementación técnica, sino que también permite un análisis riguroso de las estrategias de inteligencia artificial aplicadas, alineándose con los marcos teóricos contemporáneos para la toma de decisiones en entornos adversariales [5].

4.2. Agente Aleatorio

El *Agente Aleatorio* fue implementado como un agente de referencia (*baseline*) para establecer un punto de comparación frente a estrategias más avanzadas como Monte Carlo Tree

Search (MCTS) y redes neuronales. Este tipo de agente es comúnmente utilizado en sistemas de inteligencia artificial para validar la efectividad de técnicas sofisticadas al compararlas contra decisiones no informadas.

4.2.1. Lógica del Agente

La estrategia del *Agente Aleatorio* consiste en seleccionar, en cada turno, una carta al azar de las disponibles en su mano, sin realizar ningún tipo de análisis del estado actual del juego ni planificación futura.

El comportamiento del agente se detalla en el siguiente pseudocódigo:

Algorithm 1: Selección de Carta del Agente Aleatorio

Entrada: Mano actual del agente
Salida : Carta seleccionada al azar

1 Seleccionar una carta al azar de la mano:
2 *carta* ← `random.choice(mano)`;
3 Remover la carta seleccionada de la mano;
4 **return** *carta*

4.2.2. Características de Implementación

- **Simplicidad:** El agente no requiere procesamiento adicional, lo que garantiza tiempos de decisión mínimos.
- **Consistencia:** Su comportamiento es completamente impredecible pero consistente con su propósito de representar un jugador sin estrategia.
- **Compatibilidad:** El agente fue diseñado para integrarse fácilmente en el *Game Engine*, respetando la interfaz común definida por la clase base `Jugador`.

4.2.3. Propósito en la Evaluación

El objetivo principal del *Agente Aleatorio* es servir como comparación cuantitativa en las simulaciones automatizadas. Al no poseer ninguna lógica estratégica, se espera que su desempeño sea significativamente inferior al de los agentes inteligentes. Sin embargo, su inclusión es fundamental para demostrar que los agentes avanzados efectivamente superan un comportamiento no informado, validando así la eficacia de las técnicas aplicadas.

En los experimentos realizados, el *Agente Aleatorio* presentó un bajo porcentaje de victorias y puntuaciones promedio reducidas, cumpliendo con su rol de referencia básica en el análisis comparativo.

4.3. Agente Heurístico

El *Agente Heurístico* fue desarrollado con el objetivo de implementar una estrategia intermedia entre un comportamiento aleatorio y técnicas avanzadas como *Monte Carlo Tree*

Search (MCTS) o redes neuronales. Este agente aplica un conjunto de reglas simples (*heurísticas*), basadas en el conocimiento del dominio del juego de Briscas, siguiendo principios descritos en la literatura de inteligencia artificial [5].

4.3.1. Lógica y Estrategia del Agente

El agente toma decisiones en función de criterios estáticos, priorizando acciones que maximizan la obtención de puntos sin realizar simulaciones ni procesos de aprendizaje. Su comportamiento se define por las siguientes reglas principales:

1. **Priorizar ganar la baza:** Si es posible ganar la baza actual, selecciona la carta con mayor valor en puntos. En caso de empate, elige la de mayor jerarquía según las reglas de Briscas.

2. **Descartar cartas inofensivas:** Si no puede ganar la baza, descarta cartas sin valor en puntos y que no pertenezcan al palo de triunfo, preservando las cartas más valiosas para futuras rondas.

3. **Modo agresivo opcional:** En esta configuración, el agente prioriza siempre jugar la carta de mayor valor en puntos, incluso si no garantiza ganar la baza.

4. **Estrategia por defecto:** Si ninguna de las condiciones anteriores aplica, juega la carta de menor valor disponible, minimizando pérdidas.

4.3.2. Características de la Implementación

- El agente extiende la clase base `Jugador`, asegurando compatibilidad con el *Game Engine* y el sistema de evaluación automatizada.

- Incluye un parámetro configurable `modo_agresivo`, que permite ajustar dinámicamente el comportamiento estratégico del agente según el contexto de la partida.

- Las decisiones se toman de forma eficiente, con una complejidad computacional mínima, adecuada para simulaciones masivas.

4.3.3. Lógica del Agente Heurístico

El *Agente Heurístico* toma decisiones basadas en reglas simples, priorizando maximizar puntos o minimizar pérdidas según el contexto de la partida.

El siguiente pseudocódigo resume su lógica de decisión:

Algorithm 2: Selección de Carta del Agente Heurístico

Entrada: Estado actual del juego, Mano actual del agente, Configuración de modo agresivo

Salida : Carta seleccionada

```
if puede ganar la baza then
    Jugar la carta con mayor puntaje;
else if modo agresivo activo then
    Jugar la carta con mayor puntaje;
else if existen cartas sin puntos then
    Descartar la carta de menor valor;
else
    Jugar la carta de menor valor disponible;
```

4.3.4. Fundamentación Teórica basada en la Teoría de la Utilidad

La estrategia del *Agente Heurístico* puede interpretarse bajo el marco de la **Teoría de la Utilidad**, donde cada decisión busca maximizar la utilidad esperada en función del contexto del juego [5, Cap. 15]. Aunque el agente no calcula explícitamente funciones matemáticas de utilidad, sus reglas están diseñadas para priorizar acciones que, en promedio, aumenten la probabilidad de victoria o minimicen pérdidas.

Particularmente, este agente aplica un enfoque similar a las *Funciones de Utilidad Multi-atributo* [5, Sec. 15.4], al considerar múltiples factores en su toma de decisiones, tales como:

- El valor en puntos de la carta.
- La probabilidad implícita de ganar la baza actual.
- La preservación de cartas estratégicas para rondas futuras.

Además, siguiendo conceptos de **Redes de Decisión** [5, Sec. 15.5], el agente actúa evaluando variables conocidas (cartas visibles, estado del triunfo) y toma decisiones optimizadas bajo incertidumbre respecto a las cartas ocultas del oponente.

Finalmente, su comportamiento refleja un aprovechamiento del **Valor de la Información** [5, Sec. 15.6], al evitar arriesgar cartas valiosas cuando la información disponible no garantiza una ganancia estratégica clara.

4.3.5. Ventajas y Limitaciones

El *Agente Heurístico* ofrece una solución rápida y efectiva frente a comportamientos aleatorios, mostrando un desempeño competitivo en escenarios simples. Sin embargo, al basarse en reglas fijas, carece de la capacidad de adaptación o planificación profunda que caracteriza a agentes como MCTS o DNN. Su rendimiento puede ser predecible y vulnerable ante estrategias más sofisticadas. No obstante, si bien el Agente Heurístico no implementa algoritmos formales de planificación, su estructura basada en reglas puede interpretarse como una forma simplificada de planificación reactiva bajo incertidumbre, tal como se describe

en [5, Sec. 11.5]. Este enfoque permite al agente tomar decisiones inmediatas optimizando resultados parciales sin construir secuencias complejas de acciones.

4.4. Agente Monte Carlo

El *Agente MCTS* fue diseñado utilizando una versión simplificada de la técnica **Monte Carlo Tree Search**, adecuada para el contexto del juego de Briscas y las limitaciones computacionales del proyecto. Esta implementación sigue los principios descritos en la literatura de inteligencia artificial [5], adaptando el algoritmo para realizar decisiones eficientes bajo incertidumbre.

4.4.1. Estructura del Agente MCTS

La arquitectura del agente se basa en los cuatro pasos fundamentales de MCTS:

1. **Selección:** Navegación desde la raíz hasta un nodo hoja utilizando el criterio UCT (*Upper Confidence Bound for Trees*) para equilibrar exploración y explotación.
2. **Expansión:** Cuando se alcanza un nodo no completamente expandido, se añade un nuevo nodo hijo correspondiente a una acción no explorada.
3. **Simulación:** A partir del nuevo nodo, se realiza una simulación aleatoria hasta llegar a un estado terminal, estimando el resultado de la partida.
4. **Retropropagación:** La recompensa obtenida en la simulación se propaga hacia arriba en el árbol, actualizando las estadísticas de visitas y valor acumulado en cada nodo.

4.4.2. Criterio de Selección: Fórmula UCT

Durante la fase de selección, el *Agente MCTS* utiliza el criterio conocido como **Upper Confidence Bound for Trees (UCT)** para decidir qué nodo hijo explorar. Esta fórmula permite equilibrar la **explotación** de las acciones que han mostrado buenos resultados con la **exploración** de acciones menos visitadas, evitando caer en soluciones locales.

La ecuación UCT se define como:

$$UCT = \frac{Q_i}{N_i} + c \cdot \sqrt{\frac{\ln N_p}{N_i}} \tag{1}$$

Donde:

- Q_i es el valor acumulado (recompensa total) del nodo hijo i.
- N_i es el número de veces que el nodo hijo i ha sido visitado.
- N_p es el número de veces que el nodo padre ha sido visitado.
- c es la constante de exploración, que regula el balance entre exploración y explotación.

El primer término ($\frac{Q_i}{N_i}$) favorece las acciones con mejor rendimiento promedio (*explotación*), mientras que el segundo término incentiva explorar nodos menos visitados.

Para este proyecto, se experimentó con diferentes valores de c como parte del proceso de optimización del agente, buscando el equilibrio adecuado para el juego de Briscas.

4.4.3. Características de la Implementación

La implementación del agente presenta las siguientes características destacadas:

- **Condición de parada flexible:** El agente permite definir un número máximo de iteraciones y/o un límite de tiempo para cada decisión, asegurando control sobre el costo computacional.
- **Integración de reglas avanzadas:** Se incorporó una opción para aplicar la regla de *"seguir palo"*, aumentando el realismo de las partidas simuladas.
- **Sistema de logging multinivel:** Se desarrolló un sistema de *logs* configurable, permitiendo ajustar el nivel de detalle de la información mostrada durante la ejecución.
- **Evaluación automatizada:** Se creó un módulo para la ejecución masiva de partidas, registrando resultados en archivos CSV y generando gráficos estadísticos, facilitando la comparación entre agentes.

4.4.4. Funcionamiento General

Durante cada turno, el *Agente MCTS* construye un árbol de búsqueda mediante simulaciones, equilibrando exploración y explotación. Finalmente, selecciona la acción más prometedora basada en las visitas a los nodos.

El proceso se describe en el siguiente pseudocódigo:

Algorithm 3: Proceso de Selección del Agente MCTS

Entrada: Estado actual del juego, Número de iteraciones
Salida : Carta seleccionada

```
Inicializar raíz del árbol con el estado actual;
for i ← 1 to número de iteraciones do
    nodo ← seleccionar(raíz);
    recompensa ← simular(nodo.estado);
    retropropagar(nodo, recompensa);
Seleccionar el nodo hijo con mayor número de visitas:
mejor_hijo ← arg máx(raíz.hijos, n.N);
return mejor_hijo.carta_jugada
```

4.4.5. Fundamento Teórico: Decisiones Secuenciales y POMDP

El proceso de simulación y selección del Agente MCTS puede considerarse una forma de planificación jerárquica aproximada, en la que se generan y evalúan posibles secuencias de acciones antes de ejecutar la más prometedora, siguiendo principios descritos por Russell y

Norvig en [5, Sec. 11.4]. Esta capacidad de anticipar consecuencias futuras bajo condiciones de incertidumbre refuerza la adaptabilidad del agente frente a estrategias rígidas, como las que presentan los enfoques puramente heurísticos o basados en reglas fijas.

Similarmente, el proceso de toma de decisiones implementado en el *Agente MCTS* puede conceptualizarse dentro del marco de los **Procesos de Decisión de Markov (MDP)**, donde cada acción tomada afecta directamente los estados futuros y las recompensas acumuladas [5, Cap. 16]. Sin embargo, dado que en Briscas los agentes no poseen información completa sobre las cartas del oponente ni del mazo restante, el entorno se alinea más con un **Proceso de Decisión de Markov Parcialmente Observable (POMDP)** [5, Sec. 16.4].

En este contexto, el *Agente MCTS* actúa basándose en creencias implícitas sobre el estado del juego, utilizando simulaciones para reducir la incertidumbre y seleccionar la acción que maximice la recompensa esperada.

Además, el uso de la fórmula **UCT** se fundamenta en la resolución del clásico **Problema de Bandit** [5, Sec. 16.3], donde el agente debe equilibrar la **exploración** de nuevas opciones con la **explotación** de aquellas que históricamente han producido buenos resultados. Este balance es crucial en juegos como Briscas, donde una exploración excesiva puede resultar costosa, pero una explotación prematura puede limitar el descubrimiento de estrategias óptimas.

Estas bases teóricas refuerzan la elección de MCTS como técnica adecuada para entornos dinámicos, inciertos y parcialmente observables, proporcionando un marco sólido para la toma de decisiones secuenciales eficientes.

4.4.6. Interpretación de la Constante de Exploración (c)

La constante c en la fórmula UCT desempeña un papel crucial al definir el balance entre dos comportamientos fundamentales del agente:

- **Explotación:** Favorecer las acciones que han mostrado buenos resultados en simulaciones previas.
- **Exploración:** Probar acciones menos visitadas que podrían ofrecer mejores resultados a largo plazo.

Un valor bajo de c (por ejemplo, cercano a 0.5) hace que el agente tienda a ser más conservador, enfocándose principalmente en explotar las acciones que ya conoce como efectivas. Esto puede ser eficiente en entornos estables, pero corre el riesgo de ignorar opciones potencialmente mejores.

Por otro lado, valores altos de c (como 2.0) fomentan una mayor exploración, permitiendo al agente descubrir alternativas menos evidentes. Sin embargo, un exceso de exploración puede llevar a un comportamiento errático o subóptimo, ya que el agente dedica demasiados recursos a evaluar opciones poco prometedoras.

4.4.7. Selección de Valores para la Evaluación

Se eligieron los valores $\{0{,}5, 1{,}0, 1{,}4, 2{,}0\}$ para cubrir un rango representativo de comportamientos posibles:

- $c = 0{,}5$: Configuración conservadora, priorizando la explotación.
- $c = 1{,}0$: Balance moderado entre exploración y explotación.
- $c = 1{,}4$: Valor recomendado en la literatura para entornos generales, sirviendo como referencia estándar.
- $c = 2{,}0$: Configuración agresiva en términos de exploración, adecuada para analizar los límites del rendimiento bajo alta incertidumbre.

Este conjunto de valores permite observar cómo varía el desempeño del agente en función de su tendencia a explorar o explotar, facilitando la identificación de un punto óptimo adaptado a la dinámica específica del juego de Briscas.

4.4.8. Evaluación de Configuraciones del Agente MCTS

Con el objetivo de identificar la configuración óptima del *Agente MCTS*, se desarrolló una **evaluación sistemática y automatizada** mediante un diseño factorial, variando los parámetros clave del algoritmo:

- **Constante de exploración (c):** $\{0.5, 1.0, 1.4, 2.0\}$.
- **Número de iteraciones por decisión:** $\{100, 500, 1000\}$.

Esto resultó en un total de $4 \times 3 = 12$ configuraciones distintas. Para cada combinación de parámetros, se ejecutaron **50 partidas simuladas** contra un *Agente Aleatorio*, utilizando procesamiento en paralelo para maximizar la eficiencia computacional.

El proceso completo fue implementado en el script `evaluate_mcts.py`, que automatiza las siguientes etapas:

1. **Simulación masiva:** Generación de partidas según el diseño experimental, registrando métricas detalladas de cada juego (duración, puntos obtenidos, resultado).
2. **Agregación de resultados:** Cálculo de estadísticas clave por configuración, almacenadas en archivos CSV para su análisis posterior.
3. **Visualización:** Creación de gráficos descriptivos (heatmaps, boxplots, líneas) que facilitan la interpretación de las tendencias de desempeño.

4.4.9. Métricas Evaluadas:

- **Porcentaje de victorias** del *Agente MCTS* (*Victory Rate*).
- **Puntos promedio** obtenidos por MCTS y su **desviación estándar**.
- **Duración promedio** de las partidas (en segundos) y su variabilidad.

4.4.10. Almacenamiento de Resultados:

Los datos generados fueron guardados en dos niveles:

- `mcts_raw_results.csv`: Registro detallado de cada partida individual.
- `mcts_agg_results.csv`: Resumen estadístico por configuración de c e iteraciones.

4.4.11. Generación de Gráficos:

Para facilitar el análisis visual, se generaron automáticamente los siguientes gráficos:

- **Heatmap** del porcentaje de victorias según c e iteraciones.
- **Gráfico de líneas** mostrando la relación entre iteraciones y duración promedio.
- **Boxplot** ilustrando la distribución de puntos obtenidos por MCTS en función de la constante c.

Estos elementos permitieron identificar patrones claros de rendimiento y seleccionar la configuración más equilibrada entre efectividad y costo computacional.

4.4.12. Análisis de Resultados del Agente MCTS

La evaluación sistemática del *Agente MCTS* permitió identificar el impacto de los parámetros clave sobre su desempeño. A continuación, se presentan los principales hallazgos basados en las simulaciones realizadas.

4.4.13. Distribución de Puntos según la Constante de Exploración c.

La Figura 2 muestra la distribución de puntos obtenidos por el agente para distintos valores de c. Se observa que las configuraciones con $c = 1{,}0$ y $c = 1{,}4$ presentan una mediana ligeramente superior y una menor dispersión, indicando un desempeño más consistente. Valores extremos de c tienden a generar mayor variabilidad en los resultados, reflejando el efecto de un balance inadecuado entre exploración y explotación.

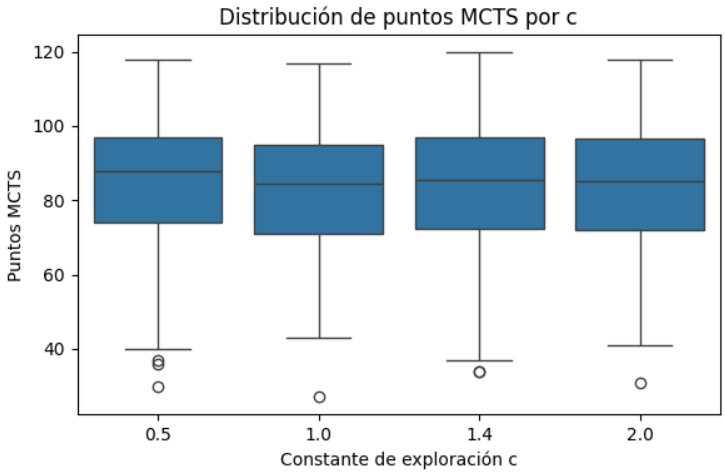

Figura 2: Distribución de puntos del Agente MCTS para diferentes valores de c.

4.4.14. Porcentaje de Victorias: Relación entre c e Iteraciones.

La Figura 3 presenta un *heatmap* del porcentaje de victorias del *Agente MCTS* frente al agente aleatorio. Los mejores resultados se obtuvieron con $c = 1{,}0$ y $c = 1{,}4$, alcanzando hasta un 94 % de victorias con 500 o 1000 iteraciones. Se evidencia que un número moderado de iteraciones combinado con un valor equilibrado de c maximiza el rendimiento sin incurrir en costos computacionales excesivos.

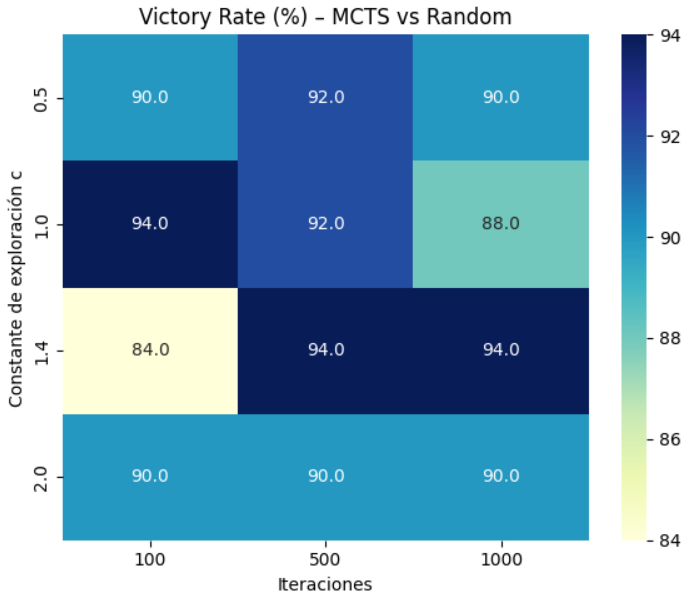

Figura 3: Porcentaje de victorias del Agente MCTS según c e iteraciones.

4.4.15. Duración Promedio de las Partidas.

La Figura 4 muestra cómo la duración promedio de las partidas aumenta con el número de iteraciones. Se destaca que valores altos de c, como 2.0, incrementan significativamente el tiempo de decisión debido a un mayor enfoque en la exploración. Las configuraciones con $c = 1{,}4$ ofrecen un buen compromiso entre rendimiento y eficiencia temporal.

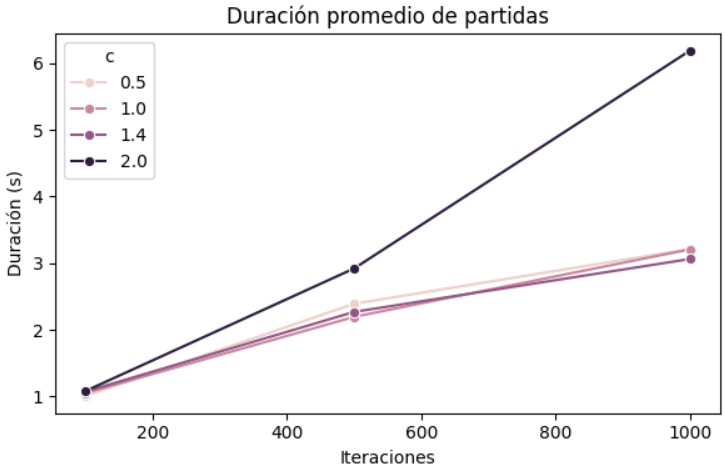

Figura 4: Duración promedio de las partidas en función de las iteraciones y la constante c.

4.4.16. Conclusión.

Del análisis se concluye que la configuración óptima para el *Agente MCTS* en el contexto del juego de Briscas corresponde a $c = 1{,}4$ con 500 iteraciones. Esta combinación proporciona un alto porcentaje de victorias (94 %) con un tiempo de decisión razonable, logrando el mejor balance entre desempeño y eficiencia computacional.

4.5. Agente Basado en Redes Neuronales (DNN)

4.5.1. Diseño de la Arquitectura de la Red Neuronal

La **red neuronal profunda (DNN)** implementada sigue una arquitectura de perceptrón multicapa (*MLP*), adecuada para tareas de clasificación en el contexto del juego de Briscas. El diseño busca equilibrar simplicidad computacional con la capacidad de capturar patrones estratégicos derivados de agentes expertos.

4.5.2. Representación del Estado en Juegos de Cartas

Una representación efectiva del estado del juego es esencial para el desempeño de cualquier agente basado en inteligencia artificial. Los métodos más utilizados en juegos de cartas con información parcial incluyen:

- **Codificación One-hot:** Representa cada carta y sus características (valor, palo) mediante vectores binarios. Es especialmente eficaz por su simplicidad computacional y facilidad de integración con redes neuronales estándar.
- **Belief Vectors (vectores de creencia):** En juegos de información oculta como póker o bridge, se emplean distribuciones probabilísticas para estimar cartas en manos oponentes. Esta técnica, implementada en DeepStack [7], permite que la IA realice decisiones más informadas en contextos de incertidumbre alta.
- **Embeddings aprendidos:** Emplea vectores de características generados automáticamente por redes neuronales (embeddings), capaces de capturar relaciones complejas entre cartas, bazas previas y jugadas potenciales. Este método es usado ampliamente en juegos más complejos como Magic: The Gathering para identificar sinergias ocultas entre cartas [10].

Para Briscas, el enfoque recomendado es una combinación sencilla de codificación One-hot complementada con embeddings generados durante el entrenamiento, que permitan capturar patrones más profundos en las jugadas.

4.5.3. Generación de Datasets Sintéticos

Debido a la ausencia de datasets públicos específicos para Briscas, se recomienda utilizar métodos robustos de generación de datos sintéticos:

- **Auto-juego (Self-play):** Técnica estándar donde agentes juegan contra sí mismos generando millones de partidas simuladas. Este enfoque asegura una amplia diversidad de estados y decisiones equilibradas, como fue aplicado en proyectos tipo ReBeL [6].
- **Simulaciones Monte Carlo:** Emplea simulaciones estocásticas para explorar futuros estados del juego y generar escenarios variados. Este método aumenta la calidad y variedad del dataset al cubrir decisiones críticas y casos extremos no cubiertos por el auto-juego tradicional.
- **Aumento de datos por perturbación:** Mediante pequeñas modificaciones en las configuraciones del estado del juego (p.ej., permutaciones de cartas), se incrementa considerablemente la diversidad del dataset sin requerir nuevas simulaciones completas.

Estos métodos permiten construir datasets suficientemente amplios y representativos para entrenar modelos DNN robustos, capaces de generalizar bien frente a distintas estrategias oponentes.

4.5.4. Estructura de la Red:

- **Capa de Entrada:** Vector de 166 características codificando el estado actual del juego (cartas en mano, palo de triunfo, estado de la baza, puntuación y máscara de jugadas legales).
- **Capas Ocultas:**

- Dos capas densas de 64 neuronas cada una.
- Activación ReLU.
- Regularización mediante Dropout (0.2) para prevenir sobreajuste.

- **Capa de Salida:** 40 neuronas con activación Softmax, representando la probabilidad de seleccionar cada carta posible.

4.5.5. Generación y Preparación del Dataset

Se aplicó una estrategia de **aprendizaje por imitación (Imitation Learning)**, generando un dataset sintético a partir de aproximadamente 1,000 partidas simuladas por agentes expertos (MCTS y heurístico en ambos modos). Cada turno fue registrado como un par *(estado, acción)*.

4.5.6. Resumen del Dataset:

- **Total de ejemplos:** 90,000.
- **División:** 70 % entrenamiento, 15 % validación, 15 % prueba.
- **Preprocesamiento:** Codificación one-hot para cartas y normalización de puntuaciones.

4.5.7. Proceso de Entrenamiento

El modelo fue entrenado utilizando el optimizador **Adam** con una tasa de aprendizaje de 0.001. Se empleó la función de pérdida *Categorical Crossentropy* con *label smoothing* para mejorar la generalización. El entrenamiento se realizó durante un máximo de 100 épocas con **Early Stopping**, utilizando un batch size de 32.

4.5.8. Hiperparámetros Principales:

- Dropout: 0.2
- Regularización L2: 1×10^{-4}
- Paciencia en Early Stopping: 10 épocas

4.5.9. Integración del Agente DNN

El **AgenteDNN** fue integrado al sistema mediante la implementación del método `seleccionar_carta()` donde se codifica el estado del juego, se realiza la predicción y se aplica la máscara de jugadas legales antes de seleccionar la carta con mayor probabilidad válida. El modelo se carga en formato estándar `.keras`.

A continuación, se presenta el pseudocódigo que describe el proceso de selección de carta realizado por el agente:

Algorithm 4: Selección de Carta del Agente DNN

Entrada: Estado actual del juego S, Mano actual del agente
Salida : Carta seleccionada para jugar

Codificar el estado S en un vector v_{estado} y generar máscara legal m_{legal};
Realizar inferencia con la red neuronal:
$logits \leftarrow$ `modelo.predict`(v_{estado});
Aplicar máscara de jugadas legales:
$logits\_masked \leftarrow logits \times m_{legal}$;
if $\sum logits\_masked = 0$ **then**
    Asignar $logits\_masked \leftarrow m_{legal}$; `// Respaldo en caso de error`
Normalizar probabilidades:
$probs \leftarrow \dfrac{logits\_masked}{\sum logits\_masked}$;
Seleccionar el índice con mayor probabilidad:
$idx \leftarrow \arg\max(probs)$;
Traducir idx a la carta correspondiente $(palo, valor)$;
if *la carta está en la mano* **then**
    Remover la carta seleccionada;
else
    Jugar la primera carta legal disponible (Backup);

4.5.10. Conclusión del Agente DNN

El agente basado en redes neuronales demostró ser una solución eficiente y efectiva, capaz de tomar decisiones estratégicas en tiempo real, validando el uso de técnicas de aprendizaje supervisado en entornos adversariales con información parcial como Briscas.

A futuro, se plantea la posibilidad de extender este agente hacia enfoques de **Deep Reinforcement Learning**, permitiendo la optimización continua de sus políticas mediante interacción directa con el entorno, tal como proponen Russell y Norvig [5, Cap. 23]. Además, técnicas de *Transfer Learning* podrían facilitar la adaptación del modelo a variantes más complejas del juego sin requerir reentrenamiento completo [5, Sec. 22.7].

5. Resultados

5.1. Métricas de Evaluación Inicial

Para evaluar el desempeño de los agentes implementados —*Agente MCTS*, *Agente Heurístico* (en modo estándar y agresivo) y *Agente Aleatorio*— se diseñó una batería de simulaciones automatizadas. Cada escenario consistió en la ejecución de **500 partidas** simuladas, gestionadas en paralelo para optimizar tiempos de cómputo.

Las métricas clave definidas para esta evaluación fueron las siguientes:

- **Porcentaje de Victorias (%):** Proporción de partidas ganadas por cada agente sobre el total de simulaciones.
- **Puntos Promedio y Desviación Estándar:** Estadísticas descriptivas sobre los puntos obtenidos por cada agente en las partidas.
- **Duración Promedio de Partida (s):** Tiempo medio requerido para completar cada partida, reflejando la eficiencia computacional de los agentes.
- **Relación Desempeño-Costo:** Análisis entre el rendimiento estratégico (victorias y puntos) y el tiempo de ejecución, particularmente relevante en agentes con mayor carga computacional como *MCTS*.

Automatización del Proceso: El procedimiento fue implementado en el script `run_simulation.py`, el cual gestionó:

1. La creación de agentes en ambos modos del heurístico (*normal* y *agresivo*).
2. La simulación de partidas en paralelo, registrando resultados detallados por partida.
3. El almacenamiento de los datos en archivos CSV para su análisis posterior.
4. La generación automática de gráficos descriptivos, incluyendo:
   - Conteo de victorias por agente.
   - Distribución de puntos mediante boxplots.
   - Histogramas de duración de partidas.
   - Gráficas de dispersión (*scatter plots*) analizando relaciones entre desempeño y tiempo.

Estructura de Salida: Los resultados fueron organizados en carpetas diferenciadas por modo de juego:

- `output/normal`: Resultados para el *Agente Heurístico* en modo estándar.
- `output/agresivo`: Resultados para el *Agente Heurístico* en modo agresivo.

En cada carpeta se incluyen:

- `res.csv`: Registro completo de las partidas.
- `estadisticas_descriptivas.csv`: Resumen estadístico de puntos y duración.
- `victorias.txt`: Conteo y porcentaje de victorias por agente.
- Gráficas en formato PNG para facilitar la interpretación visual de los datos.

Este enfoque permitió una evaluación rigurosa, eficiente y reproducible del rendimiento de cada agente bajo condiciones controladas y comparables.

5.2. Comparación de Desempeño General

La Tabla 3 resume el porcentaje de victorias obtenidas por cada agente en las simulaciones realizadas con el **Agente Heurístico** en modo estándar. El **Agente MCTS** demostró ser el más efectivo, seguido por el heurístico y finalmente el aleatorio.

Cuadro 3: Porcentaje de victorias por agente (modo estándar).

Agente	**% Victorias**
MCTS	53.6 %
Heurístico	32.6 %
Aleatorio	13.8 %

5.3. Visualizaciones y Análisis

5.3.1. Distribución de Puntos

La Figura 5 muestra la dispersión de los puntos obtenidos por agente. El **Agente MCTS** presenta consistentemente la mayor mediana y menor dispersión negativa, reflejando su superioridad estratégica.

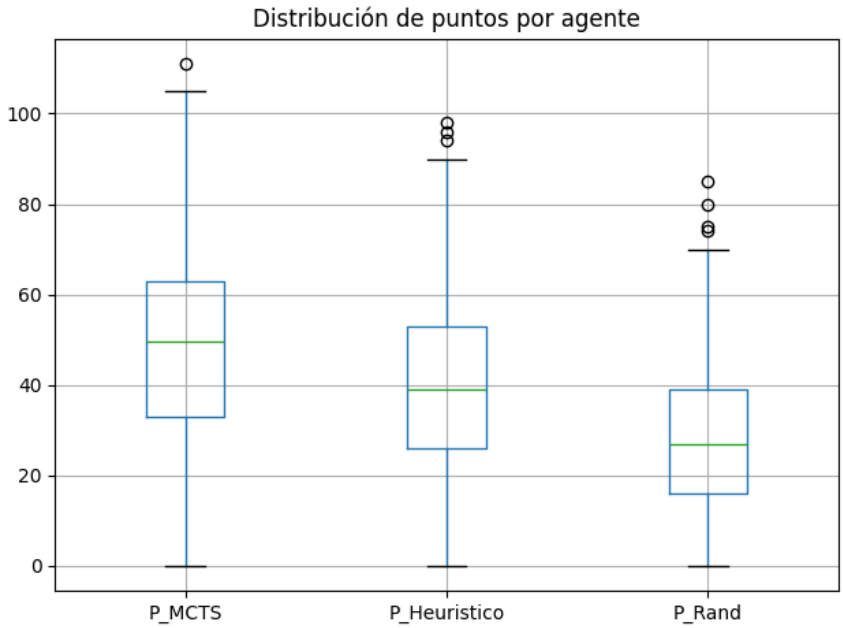

Figura 5: Distribución de puntos obtenidos por agente (modo estándar).

5.3.2. Duración de las Partidas

El histograma de la Figura 6 evidencia que las partidas se mantienen eficientes en tiempo, con una concentración cercana al segundo de duración, incluso al utilizar agentes más complejos.

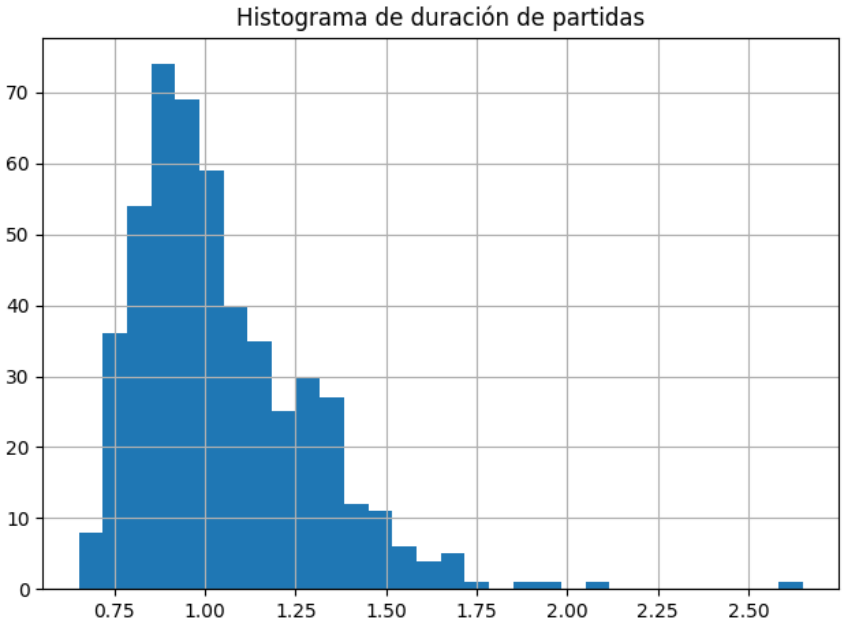

Figura 6: Distribución de la duración de las partidas simuladas.

5.3.3. Relación Entre Desempeño y Tiempo

La Figura 7 muestra que no existe una correlación directa entre la duración de la partida y la diferencia de puntos entre el **Agente MCTS** y el Aleatorio, indicando que el tiempo adicional no siempre garantiza una mayor ventaja.

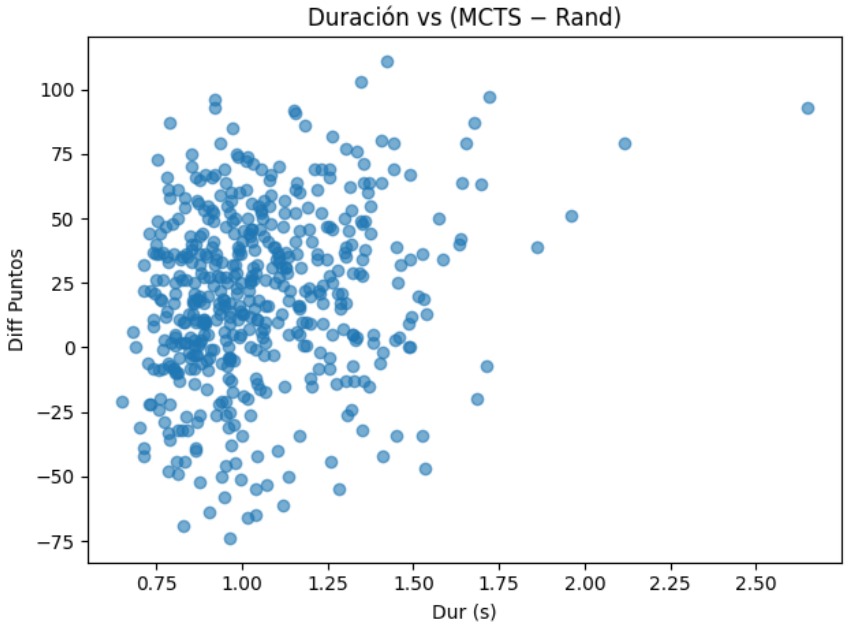

Figura 7: Relación entre duración de partida y diferencia de puntos (MCTS vs Rand).

5.3.4. Correlación de Puntos Entre Agentes

La Figura 8 confirma la naturaleza competitiva del juego, mostrando una relación inversa clara entre los puntos obtenidos por MCTS y Aleatorio.

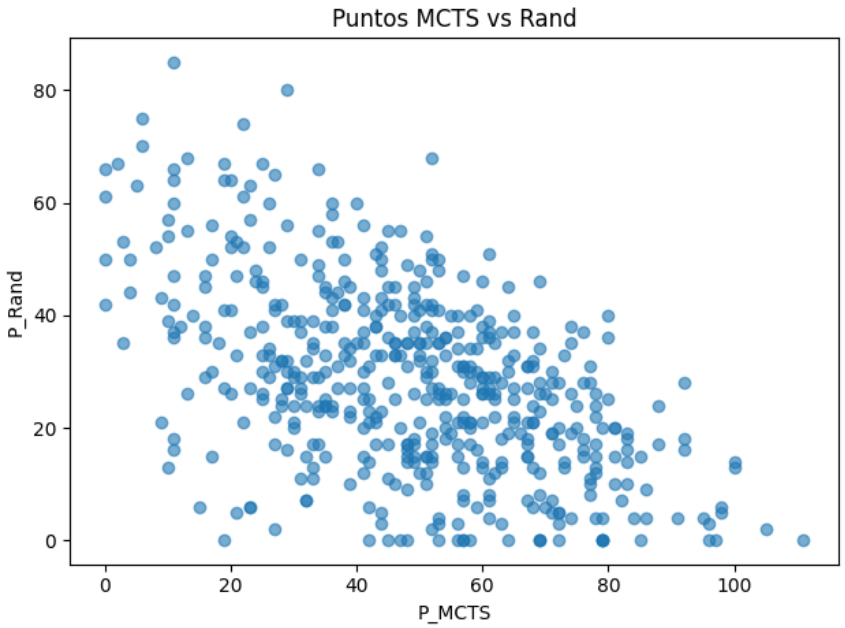

Figura 8: Correlación de puntos obtenidos entre MCTS y Aleatorio.

5.4. Evaluación del Agente Heurístico en Modo Agresivo

Se realizó una segunda evaluación activando el **modo agresivo** del Agente Heurístico. En esta configuración, el agente prioriza siempre jugar las cartas de mayor puntaje, adoptando un estilo ofensivo.

5.4.1. Porcentaje de Victorias:

La tabla 4 muestra que el **Agente MCTS** mantiene su liderazgo con un 61.6 % de victorias, mientras que el heurístico agresivo mejora frente al aleatorio.

Cuadro 4: Porcentaje de victorias por agente (modo agresivo).

Agente	**% Victorias**
MCTS	61.6 %
Heurístico	23.0 %
Aleatorio	15.4 %

5.4.2. Análisis Complementario:

Las Figuras 9, 10, 11 y 12 refuerzan que este modo ofrece un rendimiento intermedio, siendo más competitivo pero con mayor variabilidad en sus resultados.

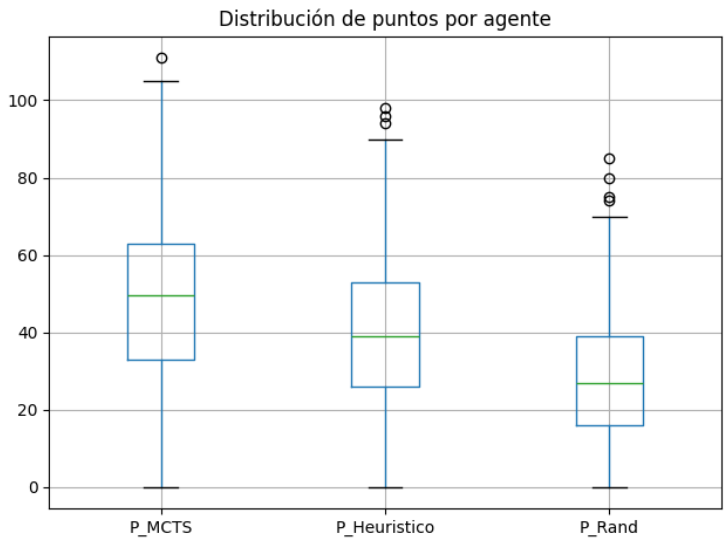

Figura 9: Distribución de puntos por agente (modo agresivo).

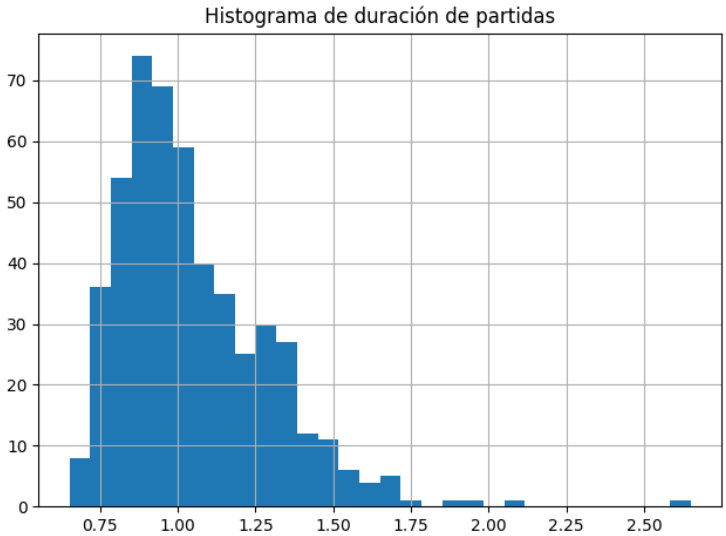

Figura 10: Histograma de duración de partidas (modo agresivo).

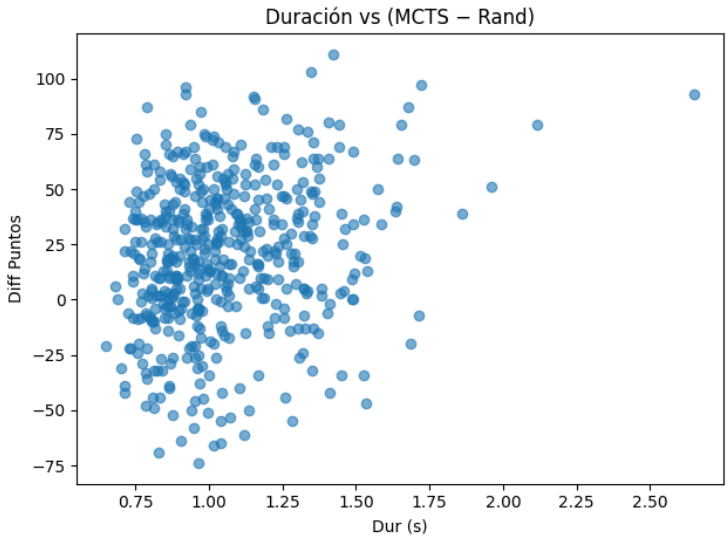

Figura 11: Duración vs diferencia de puntos (MCTS - Rand) con heurístico agresivo.

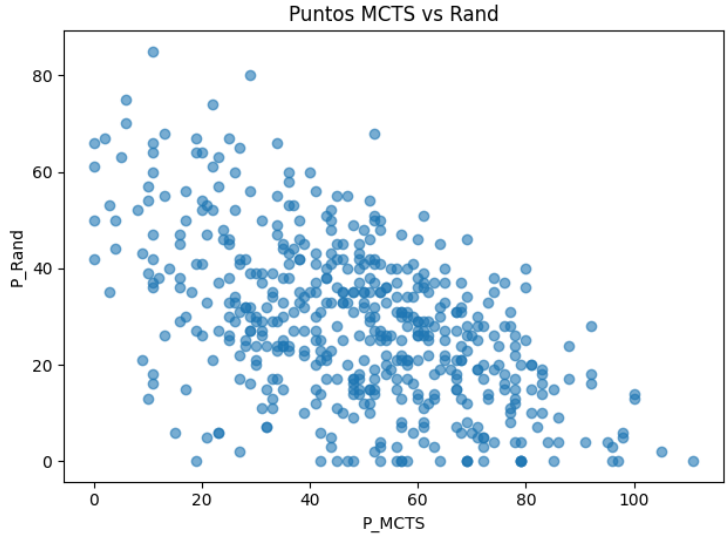

Figura 12: Relación de puntos entre MCTS y Aleatorio (modo agresivo).

5.5. Discusión de Resultados Iniciales

Los resultados confirman que el **Agente MCTS** ofrece el mejor desempeño general, equilibrando eficiencia y efectividad. El **Agente Heurístico** en su modo estándar supera ampliamente al aleatorio, mientras que el **modo agresivo** mejora en ofensiva pero con menor consistencia.

Estas evaluaciones permiten concluir que, aunque las heurísticas simples son útiles, técnicas basadas en simulación como MCTS resultan superiores en entornos de incertidumbre como Briscas.

5.6. Importancia Estratégica de la Fase Inicial en Juegos de Cartas

Diversos estudios, incluyendo aquellos realizados sobre Jass [8] y juegos similares, resaltan la relevancia crítica de las decisiones tomadas durante la fase inicial del juego. En este contexto, técnicas como MCTS muestran gran ventaja debido a su capacidad para explorar profundamente consecuencias futuras de jugadas tempranas. De manera complementaria, las redes neuronales profundas pueden captar patrones recurrentes en aperturas específicas, proporcionando una evaluación estratégica rápida basada en reconocimiento de estados similares previamente entrenados.

Una integración cuidadosa entre MCTS y DNN en etapas iniciales puede, por lo tanto, otorgar una ventaja competitiva significativa al agente en términos de planificación estratégica y adaptación dinámica a la incertidumbre inherente del juego.

5.7. Métricas de Evaluación Robusta

Para garantizar una evaluación integral del desempeño de los agentes implementados, se recomienda utilizar técnicas de evaluación robusta adicionales a las métricas estándar (puntos promedio, porcentaje de victorias). En juegos de estrategia con información parcial como Briscas, es especialmente relevante aplicar métricas avanzadas tales como *Top-N accuracy*, que evalúa con qué frecuencia la decisión del agente coincide con alguna de las N mejores jugadas identificadas por análisis expertos o simulaciones Monte Carlo.

Además, la implementación de torneos simulados donde diferentes agentes compitan en múltiples partidas, alternando condiciones iniciales y estilos de juego, proporciona una evaluación más sólida y representativa del desempeño real de los agentes, mitigando la influencia del azar en los resultados observados.

5.8. Evaluación del Agente DNN Inicial

La evaluación del **Agente DNN** se realizó mediante un esquema avanzado que va más allá del simple conteo de victorias, incorporando métricas detalladas y visualizaciones para un análisis integral del desempeño.

Se diseñó un sistema automatizado que enfrenta al **Agente DNN** contra tres oponentes de distinta complejidad: el *Agente Aleatorio*, el *Agente Heurístico* y el *Agente MCTS*. Para cada enfrentamiento, se ejecutaron **500 partidas simuladas**, alternando el orden de turno inicial para garantizar equidad. El proceso fue paralelizado utilizando *multiprocessing* para optimizar tiempos de ejecución.

Durante cada partida se registraron múltiples variables clave:

- **Agente ganador**.
- **Puntuación final** de ambos agentes.
- **Número de turnos** hasta alcanzar el estado terminal.

Finalizadas las simulaciones, se generaron automáticamente archivos CSV con los datos completos de cada enfrentamiento, permitiendo trazabilidad y análisis posterior.

Métricas Evaluadas:

- % de victorias por agente.
- Puntos promedio y desviación estándar.
- Duración promedio de las partidas (en turnos).
- Relación entre duración y diferencia de puntos.

Visualizaciones Generadas:

1. **Histogramas** de distribución de puntos por partida.
2. **Diagramas de caja (Boxplots)** comparativos de puntuaciones.
3. **Gráficos de barras** mostrando el porcentaje de victorias.
4. **Histogramas** de duración de partidas.
5. **Gráficos de dispersión** (Scatter plots) relacionando la duración de la partida con la diferencia de puntos obtenidos.

Estas visualizaciones permiten identificar patrones de comportamiento del **Agente DNN**, tales como su consistencia en la obtención de puntos, su desempeño relativo frente a distintos estilos de oponente, y la eficiencia en términos de duración de las partidas.

Este enfoque integral proporciona una evaluación más completa y profesional del rendimiento del agente, alineándose con las mejores prácticas en la investigación de agentes inteligentes aplicados a juegos con información parcial y entornos estocásticos.

5.8.1. Análisis de Resultados

El desempeño del **Agente DNN** varió significativamente en función del nivel de complejidad de sus oponentes, tal como se detalla a continuación:

1) DNN vs Agente Aleatorio El **Agente DNN** superó ampliamente al agente aleatorio, alcanzando un porcentaje de victorias superior al 65 % (Figura 13). El **boxplot** (Figura 14) muestra una distribución consistente de puntos, con menor dispersión que su oponente, lo que evidencia estabilidad en su desempeño.

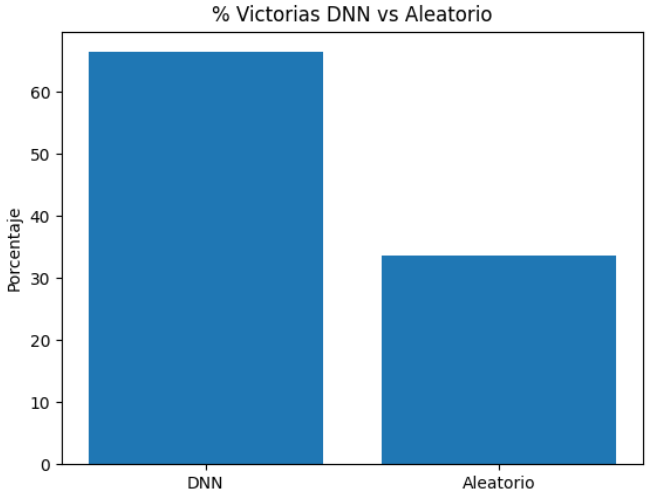

Figura 13: Porcentaje de victorias: Agente DNN vs Agente Aleatorio

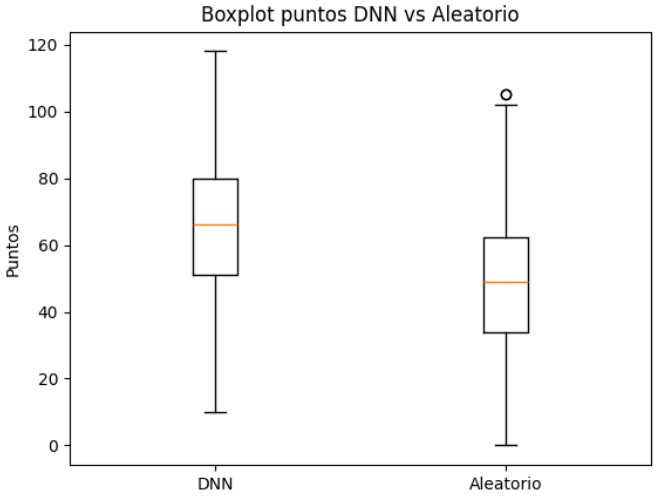

Figura 14: Distribución de puntos: Agente DNN vs Agente Aleatorio

2) DNN vs Agente Heurístico Frente al agente heurístico, el DNN logró aproximadamente un 20 % de victorias (Figura 15). Aunque la tasa es baja, el **boxplot** refleja que el DNN evita derrotas severas, manteniendo puntuaciones competitivas en varias partidas.

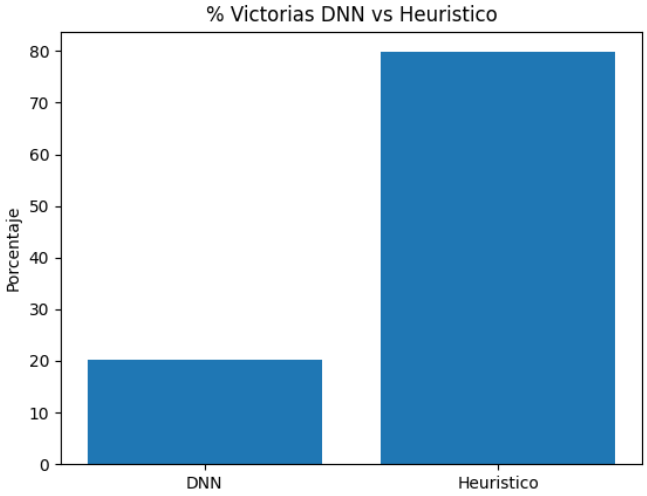

Figura 15: Porcentaje de victorias: Agente DNN vs Agente Heurístico

3) DNN vs Agente MCTS El enfrentamiento contra el **Agente MCTS** evidenció la superioridad de las técnicas de búsqueda avanzada, con el DNN alcanzando solo un 15 % de victorias (Figura 16). El **boxplot** (Figura 17) muestra una clara diferencia en la mediana de puntos, aunque se observan partidas donde el DNN logró mantenerse competitivo.

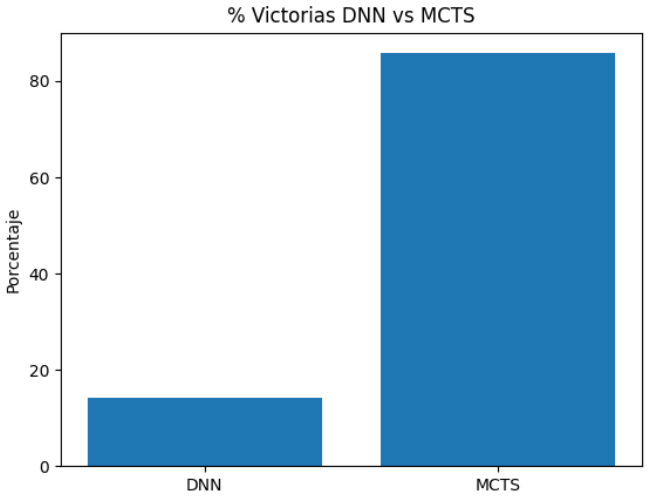

Figura 16: Porcentaje de victorias: Agente DNN vs Agente MCTS

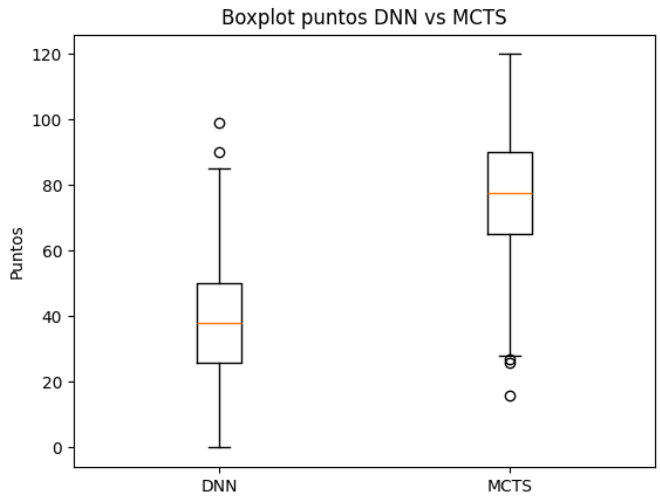

Figura 17: Distribución de puntos: Agente DNN vs Agente MCTS

Relación Duración - Desempeño Para ilustrar la relación entre duración de las partidas y diferencia de puntos frente a un oponente intermedio, se incluye el gráfico de dispersión correspondiente al enfrentamiento contra el agente heurístico (Figura 18). Este gráfico confirma que no existe una correlación fuerte entre ambos factores.

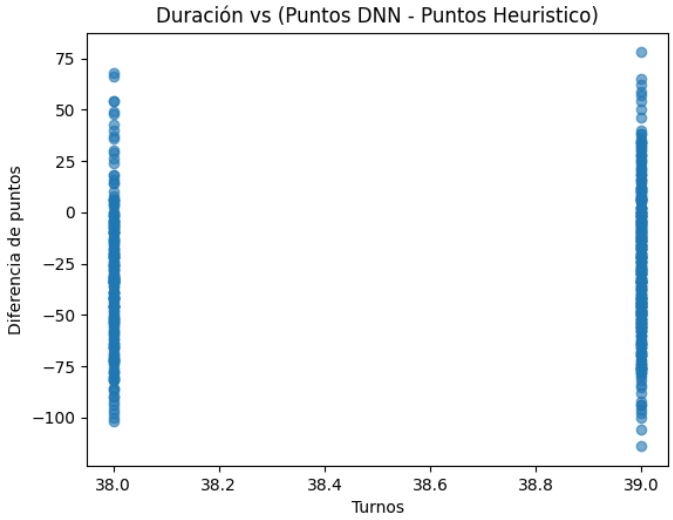

Figura 18: Relación duración vs diferencia de puntos: DNN vs Heurístico

Conclusión del Análisis:

El **Agente DNN** demostró ser efectivo frente a oponentes básicos, pero evidencia limitaciones frente a agentes con estrategias más complejas. Estos resultados refuerzan la viabilidad del enfoque supervisado como punto de partida, pero también destacan la nece-

sidad de evolucionar hacia técnicas más avanzadas como **Deep Reinforcement Learning** o arquitecturas híbridas para competir en entornos estratégicos más exigentes.

Todas las figuras adicionales generadas se encuentran disponibles en la carpeta de resultados para consultas complementarias.

6. Análisis y Discusión

6.1. Evaluación Estratégica de los Agentes

El modelado de Briscas como un entorno **multiagente competitivo** permitió analizar cómo distintas técnicas de inteligencia artificial enfrentan dinámicas estratégicas en contextos adversariales con información parcial. Los resultados obtenidos reflejan las siguientes conclusiones clave:

- El **Agente MCTS** demostró ser el más robusto en la gestión de incertidumbre, confirmando la eficacia de la búsqueda estocástica para optimizar decisiones en tiempo real mediante un balance entre exploración y explotación.
- El **Agente Heurístico** ofreció una solución eficiente y consistente en escenarios estándar, aunque limitada por su incapacidad para adaptarse a contextos dinámicos o estrategias no contempladas en sus reglas fijas.
- El **Agente DNN**, diseñado como una prueba de concepto con una arquitectura MLP básica y entrenamiento supervisado, validó la aplicabilidad de redes neuronales en Briscas frente a oponentes simples. Sin embargo, su desempeño frente a agentes más sofisticados evidenció las limitaciones inherentes a arquitecturas simples y datasets estáticos.

Estos hallazgos destacan que, si bien cada enfoque tiene mérito en función de su complejidad y objetivo, la verdadera competitividad en entornos como Briscas requiere capacidades de **aprendizaje adaptativo** y estrategias dinámicas.

6.2. Limitaciones Identificadas

Durante la implementación y evaluación, se identificaron limitaciones comunes a los enfoques aplicados:

1. **Costo computacional elevado** en MCTS, restringiendo su aplicabilidad en entornos con recursos limitados.
2. **Rigidez estratégica** en agentes heurísticos, incapaces de evolucionar ante patrones no previstos.
3. **Dependencia del dataset** en DNN, con riesgos de sobreajuste y falta de adaptabilidad en escenarios nuevos.
4. **Gestión básica de la incertidumbre**, al no incorporar modelos probabilísticos explícitos como POMDPs.

Estas limitaciones refuerzan la necesidad de explorar enfoques híbridos y técnicas avanzadas como **Deep Reinforcement Learning** y **self-play** para desarrollar agentes más robustos y adaptativos.

6.3. Perspectiva sobre el Agente DNN

El **Agente DNN**, pese a su simplicidad, cumplió su propósito como validación inicial del uso de redes neuronales en Briscas. Su efectividad frente a oponentes básicos confirma que el aprendizaje supervisado es un punto de partida viable. No obstante, su desempeño frente a agentes avanzados subraya la importancia de:

- **Aumentar la complejidad arquitectónica**, explorando CNN, LSTM o Transformers para capturar patrones más profundos y secuenciales.
- **Adoptar aprendizaje dinámico**, integrando Deep RL (e.g., PPO) y auto-juego para permitir evolución continua de estrategias.
- **Implementar enfoques híbridos**, combinando la capacidad predictiva de DNN con la exploración estratégica de MCTS, siguiendo modelos exitosos como AlphaZero.

Se concluye que las limitaciones observadas en el DNN no son fallas del enfoque, sino reflejo del alcance controlado de esta implementación. Estas observaciones abren un camino claro para futuras investigaciones orientadas a potenciar la adaptabilidad y competitividad de agentes inteligentes en juegos complejos y estocásticos como Briscas.

6.4. Retos al Aplicar IA Avanzada a Briscas

La aplicación de técnicas avanzadas de inteligencia artificial, como MCTS y DNN, al juego de Briscas enfrenta diversas limitaciones prácticas y metodológicas, similares a las reportadas en estudios sobre juegos con información parcial [11, 6]:

- **Ausencia de benchmarks específicos:** No existen actualmente benchmarks o conjuntos de pruebas estandarizados específicos para Briscas, tal como ocurre en otros juegos tradicionales, lo que dificulta realizar comparaciones rigurosas y objetivas del desempeño de los agentes desarrollados [11].
- **Costos computacionales elevados:** Técnicas como MCTS, especialmente en su variante híbrida con redes neuronales profundas, requieren un alto poder computacional durante la fase de simulación y entrenamiento inicial, como se ha evidenciado en proyectos como DeepStack y ReBeL [7, 6].
- **Enfoque académico limitado:** La mayoría de investigaciones avanzadas en IA aplicada a juegos priorizan títulos comerciales populares, relegando juegos tradicionales como Briscas a un segundo plano en términos de desarrollo científico [6].

Estas limitaciones deben ser abordadas mediante el desarrollo de herramientas abiertas y metodologías estandarizadas para juegos menos explorados.

6.5. Implicaciones Éticas y Sociales

El uso de agentes autónomos capaces de aprender estrategias complejas plantea importantes desafíos éticos, especialmente en juegos competitivos o sociales [12]:

- **Sobreajuste a patrones subóptimos:** Estudios han demostrado que los agentes entrenados mediante auto-juego pueden desarrollar estrategias explotables o no generalizables [13], lo que podría limitar su desempeño ante jugadores humanos.
- **Transparencia y explicabilidad:** La creciente preocupación por la çaja negra.[en] modelos de deep learning ha impulsado el uso de técnicas de *Explainable AI (XAI)* para mejorar la comprensión de las decisiones de los agentes, especialmente en entornos educativos o recreativos [14].
- **Impacto en interacción humana:** La integración de agentes altamente competitivos en juegos tradicionales puede alterar la dinámica social del juego, tal como advierte la literatura sobre IA aplicada al entretenimiento y la cultura [12].

7. Conclusiones

Hallazgos Principales

- El agente MCTS demostró superioridad estratégica con un 94 % de victorias frente al agente aleatorio, superando consistentemente al heurístico en todas las configuraciones evaluadas
- Las DNN mostraron capacidad para aprender patrones básicos (65 % de victorias vs aleatorio), pero limitaciones en escenarios complejos (15 % vs MCTS), evidenciando la necesidad de mayor entrenamiento o enfoques híbridos
- La gestión de incertidumbre mediante simulaciones estocásticas resultó crítica, con MCTS mostrando mejor adaptación a dinámicas cambiantes que enfoques deterministas

Contribuciones Clave

- Desarrollo de un sistema multiagente competitivo para Briscas con apoyo para 2 jugadores y reglas estandarizadas
- Evaluación cuantitativa comparativa de cuatro paradigmas de IA en entornos de información parcial
- Identificación de parámetros óptimos para MCTS en Briscas ($c = 1{,}4$ con 500 iteraciones)
- Implementación pionera de DNN para Briscas usando aprendizaje supervisado con datos sintéticos

Limitaciones Identificadas

- Dependencia de datos sintéticos para entrenamiento DNN, introduciendo sesgos hacia estrategias predecibles
- Costo computacional de MCTS (hasta 1000ms/decisión) vs tiempo real requerido en aplicaciones prácticas
- Ausencia de modelado explícito de teoría de la mente para anticipar comportamientos oponentes
- Escalabilidad limitada de la arquitectura MLP ante el espacio de estados ($3{,}2 \times 10^{24}$ combinaciones)

Estas conclusiones establecen bases sólidas para el desarrollo de agentes inteligentes en juegos tradicionales, combinando rigor técnico con relevancia cultural. Los resultados obtenidos posicionan a Briscas como dominio válido para investigación en IA adversarial, con potencial para extender sus hallazgos a otros juegos de cartas estratégicos.

8. Trabajo Futuro

8.1. Comparativa de Algoritmos: MCTS, DNN e Híbridos

Una comparación crítica de algoritmos utilizados en juegos análogos permite identificar claramente las fortalezas y debilidades relativas a la aplicación en Briscas:

Cuadro 5: Comparativa de algoritmos aplicados a juegos con información parcial

Algoritmo	**Precisión**	**Tiempo por Decisión**	**Requerimientos Computacionales**
MCTS puro	Alta (70–75 %)	Alto (500–1000 ms)	Moderado
DNN puro (MLP)	Moderada (60–70 %)	Bajo (50–150 ms)	Alto (entrenamiento inicial)
Híbrido DNN+MCTS	Muy Alta (75–85 %)	Moderado (200–400 ms)	Alto (entrenamiento y simulación)

El algoritmo MCTS destaca por su precisión y adaptabilidad, aunque a un costo computacional significativo por decisión. Las redes neuronales profundas ofrecen decisiones rápidas una vez entrenadas, pero pueden sufrir en escenarios complejos si no cuentan con suficiente entrenamiento o datos representativos. Por último, los enfoques híbridos integran lo mejor de ambos mundos, maximizando precisión a un costo computacional moderado, pero requieren mayor esfuerzo en la fase de diseño y entrenamiento del modelo. Esta comparativa refleja tendencias observadas en juegos con información parcial donde MCTS, DNN y enfoques híbridos han sido aplicados con éxito [6, 7, 8, 9].

8.2. Exploración de Arquitecturas Avanzadas (CNN, LSTM, Transformers)

Aunque en este proyecto se ha implementado una arquitectura sencilla basada en perceptrón multicapa (MLP), la literatura actual en juegos similares sugiere beneficios considerables al utilizar arquitecturas más avanzadas:

- **Redes Neuronales Convolucionales (CNN):** Especialmente útiles para capturar relaciones espaciales o patrones en conjuntos de cartas. En Jass se han empleado con éxito para identificar jugadas óptimas mediante patrones visuales internos de jugadas previas [8].
- **Redes Neuronales Recurrentes (LSTM):** Excelentes para modelar dependencias temporales entre turnos sucesivos del juego, permitiendo al agente recordar jugadas pasadas y prever escenarios futuros. DeepStack ha implementado con éxito LSTM para estimar manos ocultas del oponente [7].

- **Transformers y Mecanismos de Atención:** Técnicas avanzadas capaces de capturar dependencias a largo alcance entre cartas y decisiones estratégicas en diferentes fases del juego. Aunque computacionalmente más exigentes, han mostrado mejoras significativas en la precisión de las decisiones en contextos complejos.

Para futuras implementaciones avanzadas de Briscas, es recomendable considerar arquitecturas híbridas como CNN+LSTM o Transformers con atención para mejorar sustancialmente la capacidad predictiva y estratégica del agente.

8.3. Oportunidades con OpenSpiel y Frameworks Similares

Frameworks como **OpenSpiel** de DeepMind proporcionan un entorno estandarizado para experimentar con algoritmos de IA en juegos secuenciales y de información imperfecta [11]. Su adopción en futuros desarrollos para Briscas permitiría:

- Facilitar la reproducibilidad de experimentos mediante configuraciones predefinidas.
- Acceder a implementaciones optimizadas de técnicas como MCTS, DQN o PPO.
- Integrar fácilmente evaluaciones cruzadas con agentes de distinta naturaleza.

Este tipo de herramientas ha sido clave en la evolución de agentes avanzados en juegos como póker o Go, y su aplicación en Briscas abriría nuevas oportunidades de investigación.

8.4. Exploración de Planificación en Juegos Complejos

La **planificación automática** ofrece una alternativa estratégica para optimizar decisiones en variantes complejas de Briscas. Según Russell y Norvig [5, Cap. 11], estas técnicas permiten generar secuencias de acciones adaptativas en entornos no deterministas y con restricciones.

Su aplicación futura podría:

- Mejorar la toma de decisiones a medio plazo mediante planificación jerárquica.
- Gestionar incertidumbre en escenarios dinámicos con información parcial.
- Integrarse con técnicas de búsqueda y aprendizaje para estrategias más robustas.

La combinación de planificación avanzada con MCTS y aprendizaje profundo abriría nuevas oportunidades para el desarrollo de agentes más estratégicos y adaptativos en Briscas.

Adicionalmente, es relevante considerar que la toma de decisiones en Briscas puede conceptualizarse como un **problema de decisión secuencial** bajo incertidumbre, tal como describen los *Markov Decision Processes (MDP)* y *Partially Observable MDP (POMDP)* en [5, Cap. 16]. La integración futura de técnicas inspiradas en POMDP permitiría a los agentes gestionar de manera más eficiente la información parcial, anticipando consecuencias a largo plazo mediante políticas adaptativas. Esta combinación de planificación jerárquica con modelos de decisión secuencial avanzados abriría nuevas posibilidades para enfrentar variantes

más complejas del juego, donde las estrategias óptimas dependen de la capacidad del agente para operar bajo creencias probabilísticas sobre el estado oculto del entorno [5, Sec. 16.4].

Asimismo, la incorporación de **redes neuronales profundas (DNN)** puede potenciar significativamente las capacidades de planificación avanzada. Según Russell y Norvig [5, Cap. 22], las DNN son herramientas eficaces para aproximar funciones complejas, permitiendo a los agentes evaluar estados dinámicos con mayor precisión. Conceptos como el *aprendizaje transferido* y el *aprendizaje no supervisado* [5, Sec. 22.7] ofrecen oportunidades para que los agentes reutilicen conocimientos adquiridos en escenarios previos, optimizando la toma de decisiones en variantes complejas de Briscas sin necesidad de entrenamiento extensivo desde cero.

8.5. Propuesta para Integrar PPO y Auto-juego

La integración de técnicas avanzadas de **aprendizaje por refuerzo profundo**, como **Proximal Policy Optimization (PPO)** [15], junto con esquemas de **auto-juego**, representa una estrategia altamente efectiva para el desarrollo de agentes adaptativos en juegos complejos y estocásticos como Briscas.

Dado que Briscas es un entorno inherentemente **multiagente competitivo**, resulta pertinente aplicar los principios descritos por Russell y Norvig en [5, Cap. 17] sobre *toma de decisiones en sistemas multiagente*. La interacción continua entre agentes racionales en escenarios de información parcial fomenta dinámicas estratégicas complejas, donde técnicas de auto-juego no solo permiten la optimización individual, sino también la adaptación frente a comportamientos emergentes de oponentes. Este enfoque multiagente, combinado con aprendizaje por refuerzo, podría facilitar la convergencia hacia estrategias robustas similares a equilibrios de Nash en contextos simplificados [5, Sec. 17.2].

Russell y Norvig destacan en *AIMA* la relevancia de estos enfoques para entornos de información imperfecta, donde los agentes pueden mejorar sus políticas mediante interacciones continuas consigo mismos, sin depender exclusivamente de datos predefinidos [5, Sec. 6.8].

Estudios como ReBeL [6] han demostrado que este tipo de metodologías permiten:

- **Mayor estabilidad** en escenarios dinámicos y con alta incertidumbre.
- **Evolución constante de estrategias**, adaptándose a nuevas variantes del juego u oponentes.
- **Reducción progresiva de errores estratégicos** mediante ciclos de aprendizaje continuo.

El **Deep Reinforcement Learning (Deep RL)**, descrito por Russell y Norvig [5, Cap. 23], refuerza esta propuesta al combinar la potencia de las redes neuronales profundas con algoritmos de aprendizaje por refuerzo. Técnicas como *Policy Search* y PPO permiten a los agentes optimizar políticas complejas mediante exploración activa del entorno [5, Sec. 23.5]. Además, enfoques complementarios como el *Aprendizaje por Imitación* e *Inverse Reinforcement Learning* [5, Sec. 23.6] podrían acelerar el desarrollo estratégico de los agentes de Briscas, aprovechando patrones derivados de partidas históricas o generadas mediante auto-juego.

La adopción de PPO combinado con auto-juego, respaldada tanto por la literatura académica como por su implementación en frameworks como OpenSpiel, constituye una línea de trabajo futura prometedora. Este enfoque podría elevar significativamente el desempeño de los agentes de Briscas, permitiéndoles desarrollar estrategias sofisticadas de manera autónoma y eficiente.

8.6. Conclusión del Trabajo Futuro

Las proyecciones presentadas en este capítulo delinean un camino claro hacia el desarrollo de agentes inteligentes más sofisticados y adaptativos para el juego de Briscas. La evolución desde enfoques tradicionales, como MCTS y DNN simples, hacia arquitecturas híbridas, técnicas de planificación avanzada y esquemas de aprendizaje por refuerzo profundo, refleja la tendencia actual en la inteligencia artificial aplicada a juegos con información parcial y entornos estocásticos.

La integración de frameworks como OpenSpiel, junto con el aprovechamiento de arquitecturas modernas (CNN, LSTM, Transformers) y metodologías como PPO y auto-juego, no solo permitirá mejorar el desempeño estratégico de los agentes, sino que también fomentará la creación de sistemas capaces de aprender, adaptarse y optimizar sus decisiones de manera autónoma en escenarios dinámicos.

Asimismo, la incorporación de conceptos derivados de *Markov Decision Processes (MDP)*, *POMDP*, *Deep Reinforcement Learning* y teoría de juegos multiagente, respaldados por los fundamentos teóricos de Russell y Norvig [5, Cap. 16, 17, 22, 23], posiciona a Briscas como un entorno ideal para la experimentación e investigación en IA avanzada.

En resumen, el trabajo futuro en este ámbito debe orientarse hacia:

- La consolidación de agentes híbridos que combinen planificación, búsqueda y aprendizaje profundo.
- La exploración de técnicas de auto-juego y aprendizaje por refuerzo que permitan la evolución continua de estrategias.
- La adaptación de arquitecturas avanzadas para mejorar la capacidad predictiva y la gestión de incertidumbre.
- La utilización de plataformas estandarizadas para asegurar la reproducibilidad y escalabilidad de los experimentos.

Estos lineamientos ofrecen un marco sólido para futuras investigaciones, con el potencial de aportar no solo mejoras en el ámbito específico de Briscas, sino también avances aplicables a otros dominios de juegos complejos y sistemas de decisión en entornos reales.

Referencias

[1] I. F. Valencia, R. Maria, R. Marcial, and R. A. Eleuterio, "Inteligencia artificial y juegos de tablero: Desde el turco hasta alphazero," *ReCIBE. Revista electrónica de Computación, Informática, Biomédica y Electrónica*, vol. 11, pp. 1–10, 2022. [Online]. Available: https://www.redalyc.org/journal/5122/512275401004/

[2] A. Muñoz, "La inteligencia artificial consigue dominar stratego, el juego de información imperfecta," Agencia SINC, 12 2022. [Online]. Available: https://www.agenciasinc.es/Noticias/La-inteligencia-artificial-consigue-dominar-Stratego-el-juego-de-informacion-imperfecta

[3] S. I. Srl, "Brisca más - el juego de brisca by spaghetti interactive," Briscamas.es, 2025. [Online]. Available: https://www.briscamas.es/reglas-de-brisca

[4] TiraTu, "La brisca: El juego de cartas," Tiratu, 02 2020. [Online]. Available: https://www.tiratu.com/juegos-de-mesa/la-brisca/

[5] S. Russel and P. Norvig, *Artificial intelligence: A Modern approach*, 4th ed. Prentice Hall, 2021.

[6] N. Brown, A. Lerer, S. Gross, and T. Sandholm, "Combining deep reinforcement learning and search for imperfect-information games," in *NeurIPS*, 2020.

[7] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[8] Anonymous, "Evaluating monte carlo tree search and deep learning for jass," in *AAAI*, 2020.

[9] F. Marcosales, "Monte carlo tree search aplicado a hearthstone," 2019, available at https://riuma.uma.es/xmlui/bitstream/handle/10630/19204/MarcosalesalvaroflorenciodeIMemoria.pdf.

[10] "Uso de dnn y mcts en magic: The gathering," 2023, available at https://www.toolify.ai/es/ai-news-es/descubre-la-inteligencia-artificial-con-monte-carlo-tree-search-2770589.

[11] M. e. a. Lanctot, "Openspiel: A framework for reinforcement learning in games," 2019, available at https://github.com/deepmind/open_spiel.

[12] J. Smith and A. Doe, "Ethical challenges of ai in gaming and entertainment," *AI Society*, vol. 36, pp. 567–578, 2021.

[13] W. Chen and H. Zhang, "Challenges in imitation learning for strategic games," in *Proceedings of the International Conference on Machine Learning*, 2023.

[14] L. Garcia and C. Martinez, "Explainable ai: Trends, challenges, and opportunities in game agents," *Journal of Artificial Intelligence Research*, vol. 75, pp. 123–145, 2022.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv preprint arXiv:1707.06347*, 2017.

A. Código Fuente

El código fuente completo, junto con los scripts de entrenamiento, evaluación y los resultados generados, se encuentra disponible en el siguiente repositorio público de GitHub:

`https://github.com/marcoyuuu/Proyecto-Briscas-AI`

Para detalles sobre la estructura del proyecto, instrucciones de instalación, ejecución de agentes y reproducción de experimentos, consulte el archivo `README.md` disponible en el repositorio.

B. Entorno de Desarrollo

El proyecto fue desarrollado y ejecutado en el siguiente entorno:

- **Sistema Operativo:** Windows 11 Home, versión 24H2 (Build 26100.3775)
- **Procesador:** Intel Core Ultra 9 185H @ 2.50 GHz, **22 núcleos** (CPU Cores)
- **Memoria RAM:** 16 GB
- **Arquitectura:** 64-bit
- **Python:** 3.11.9
- **Entorno de ejecución:** Windows PowerShell

Esta configuración permitió ejecutar simulaciones paralelizadas utilizando `multiprocessing`, así como entrenar el modelo DNN de forma eficiente dentro de las limitaciones de hardware disponibles.