

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



**JUGADOR ARTIFICIAL DE DOMINÓ BASADO EN  
MÉTODOS DE MONTE CARLO**

**TESIS**

**QUE PARA OBTENER EL TÍTULO DE  
INGENIERO EN COMPUTACIÓN**

**P R E S E N T A**

*Andrés Cruz y Vera*

**ASESOR: Dr. Marco Antonio Morales Aguirre**

**TABLA DE CONTENIDO**

**1. INTRODUCCIÓN**

1.1 Contexto

1.2 Identificación del problema

1.3 Objetivo

1.4 Metodología

1.5 Organización del documento

**2. ANÁLISIS DE REQUISITOS DE SOFTWARE**

2.1 Requerimientos funcionales

2.2 Requerimientos no funcionales

2.3 Restricciones

2.4 Trabajos relacionados

**3. EL JUEGO DE DOMINÓ**

**4. DISEÑO Y ARQUITECTURA**

**5. IMPLEMENTACIÓN**

**6. RESULTADOS**

**7. CONCLUSIONES**

**REFERENCIAS**

## **Introducción**

En el primer capítulo se presentará el problema que aborda el presente trabajo. Se dará el contexto histórico de la relación entre los videojuegos y los jugadores artificiales para luego identificar el problema y definir tanto los objetivos como la metodología a seguir.

### **1.1 Contexto**

La historia de los juegos por computadora inicia desde la década de 1950 en el ámbito académico y en los años setenta y ochenta gana popularidad para el público en general. Los videojuegos han tenido un gran impacto en la cultura popular, así como en grandes figuras de la computación que tuvieron su primer acercamiento a los ordenadores por medio de estos y del lenguaje BASIC

Asimismo, los juegos de mesa han tenido un papel importante en el desarrollo del área de inteligencia artificial siendo una área muy fructífera de investigación como en el caso del ajedrez y la famosa contienda entre Deep Blue y Garry Kasparov

### **1.2 Identificación del problema**

El desarrollo de videojuegos es un ámbito multidisciplinario en donde se utilizan técnicas de inteligencia artificial para complementar la experiencia de juego del usuario. Los jugadores artificiales (o *bots*) cumplen un papel importante como contrincantes o personajes secundarios dentro del juego.

Con miras a desarrollar una versión online del juego de dominó con un modelo de monetización basado en anuncios, se tiene como uno de los objetivos maximizar el número de impresiones de los anuncios en el usuario. Así, es necesario proveer una experiencia atractiva que tenga como efecto que el usuario pase un largo tiempo activo en la página.

El lanzamiento a mercado de un juego multijugador online presenta distintos retos. Entre ellos, existe la necesidad de crear una base mínima de usuarios que permita tener un tiempo razonable de espera para poder encontrar una partida a la cual unirse. Una forma de solventar parcialmente este obstáculo, particularmente en las primeras fases del lanzamiento, es contar con jugadores artificiales que suplementen la falta de contrincantes humanos.

Así, es deseable contar con un jugador artificial que permita a los usuarios iniciar una partida aun en las circunstancias en que no cuenten con suficientes personas para completar los equipos. Al momento en que se realiza este escrito, no se ha encontrado una implementación de código abierto de un jugador artificial para el juego de dominó (con las reglas que se usan en latinoamérica) que cuente tanto con una licencia que permita su uso comercial así como una interfaz de programación diseñada para su integración a un juego de tiempo real con usuarios humanos.

### 1.3 Objetivos

Implementar un programa de computadora que sea capaz de jugar en una partida de dominó como parte de un equipo de dos participantes que compiten con dos contrincantes.

### 1.4 Metodología

Para la implementación del bot, se decidió utilizar la metodología de cascada debido a que el alcance y la funcionalidad del proyecto es relativamente pequeña.

### 1.5 Organización del documento

1. Introducción
2. Análisis de requisitos del software
3. Las reglas del dominó
4. Juegos estocásticos con información imperfecta
5. El método de Árbol de Búsqueda Montecarlo
6. Diseño del programa
7. Implementación
8. Validación
9. Conclusiones

## **2. Análisis**

Una vez que se ha identificado el problema, se formulará una primera aproximación a la solución por medio de los requerimientos funcionales que debe cumplir, así como por las restricciones que debe satisfacer. También se mostrarán trabajos relacionados.

### **2.1 Requerimientos funcionales**

El programa generará una jugada a partir del estado actual del juego. Es decir, el programa recibirá como entrada una representación de sus fichas asignadas así como de las fichas tiradas por los otros participantes y como salida indicará cual de sus fichas debe jugarse.

También será posible elegir distintos niveles de juego para el programa

### **2.2 Requerimientos no funcionales**

El programa debe generar las jugadas en un tiempo razonable. Debe siempre terminar la ejecución antes de un lapso predeterminado para poder utilizarse en un juego de tiempo real contra contrincantes humanos y no debe poseer información sobre las manos de sus contrincantes ni de su pareja de equipo.

En segundo lugar, el desempeño del programa tiene que ser mejor que la estrategia más sencilla posible (el jugador *greedy*): de entre las fichas que se pueden bajar siempre se elige la de mayor puntaje. Una jugada *greedy* es sumamente barata de calcular. De no cumplirse con este requisito no tendría sentido utilizar una estrategia más compleja y costosa en tiempo y recursos computacionales.

Así, se pone como meta que un equipo de los jugadores artificiales debe vencer a un equipo de jugadores *greedy* en al menos 70% de las partidas. Se considera que este margen es el mínimo para justificar el uso de un algoritmo distinto a la estrategia *greedy*.

Por último, debe exponer una API sencilla para ser integrado a distintas interfaces, tanto aplicaciones web como móviles.

### **2.5 Restricciones**

El software a desarrollar cuenta con dos restricciones principales. En primer lugar, en cuanto a los recursos para implementar la solución, se cuenta con un periodo aproximado de 6 meses para completar el desarrollo del sistema así como de un solo desarrollador (el autor de este trabajo).

En segundo lugar, el sistema debe cumplir con los requerimientos funcionales y no funcionales dentro de un ambiente de ejecución en la nube con un costo razonable. Es difícil estimar el costo de los recursos computacionales que consumirá la solución, pues depende de la cantidad de usuarios del sistema así como de su comportamiento de uso. No se cuenta con los datos necesarios para estimar la naturaleza de la carga a la que el sistema debe hacer

frente pero se puede definir unas características mínimas del ambiente de ejecución en el cual la solución debe correr.

Como un punto de referencia, se ha elegido la instancia más modesta de la categoría de servidores de proposito general de Digital Ocean. Dicho servidor cuenta con ocho gigabytes de memoria RAM y con dos procesadores virtuales. La máquina virtual corre sobre procesadores Intel Xeon Skylake con una velocidad base de 2.7 ghz y con máxima velocidad de 3.7 ghz. El costo del servidor es de sesenta dólares al mes. Se eligió Digital Ocean por los credits que regala para probar los servidores.

Si el sistema no puede correr en un servidor de esta naturaleza, es muy probable que en una escala más grande el costo de la solución sea prohibitivo para su uso.

## 2.4 Trabajos relacionados

Uno trabajo importante en el ambito de algoritmos para juegos de información imperfecta lo realiza Ginsberg (2001). Con esta metodología logra implementar un jugador de Bridge de nivel experto.

Por otra parte, un jugador artificial en un contexto de incertidumbre puede estudiarse desde la perspectiva de procesos de decisión de markov como en la disciplina de aprendizaje por refuerzo. En este campo es importante el trabajo de Mnih et al. (2013) que es uno de los primeros en integrar aprendizaje profundo a los algoritmos de aprendizaje por refuerzo para la creación de agentes en el juego de atari.

Long et al. (2010) realizaron un trabajo en donde , a partir de árboles de juego sintéticos, definen indicadores estadísticos que les permiten identificar propiedades importantes de juegos de información imperfecta en los que el método de *Perfect Information Monte Carlo* (PIMC) se puede adaptar exitosamente. Dicho trabajo extiende la línea de investigación sobre las limitaciones de PIMC en el contexto de información imperfecta que inician Frank y Basin (1998)

Asimismo, se recuperó de la web un proyecto de licenciatura sobre un jugador artificial para dominó (en el texto se le refiere como Latin-American dominoes) desarrollado por Angeris y Li (2016) de la universidad de Stanford. El proyecto consiste en simulaciones para contrastar distintas algoritmos pero no tiene la finalidad de ser consumido como una API.

### **3. Diseño**

Ya que se ha identificado el problema, su contexto y se ha delimitado los requerimientos funcionales y restricciones que debe satisfacer la solución se pasará a definir el diseño de ésta así como las posibles alternativas.

#### 3.1 Elección de algoritmo

El principal reto en la implementación del jugador artificial es que el juego de dominó es de información imperfecta. Existen dos alternativas principales de algoritmos que pueden utilizarse en este tipo de juegos.

En primer lugar, existen algoritmos de aprendizaje por refuerzo que han sido utilizados exitosamente en juegos con incertidumbre e información escondida. El algoritmo de *Counterfactual Regret Minimization* ha sido utilizado para crear jugadores de póker. Dicho algoritmo utiliza un enfoque iterativo en el que se busca aproximar una función que para cada estado del juego defina una acción óptima.

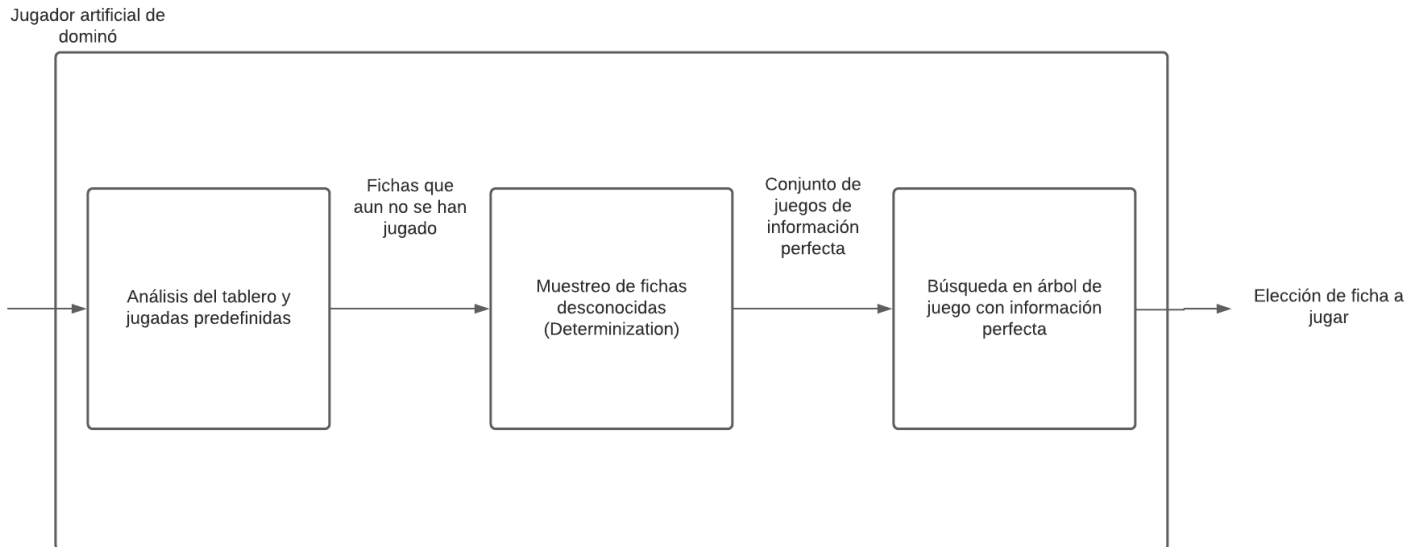
Por otro lado, existe el algoritmo *Perfect Information Monte Carlo* (PIMC). En este algoritmo se busca transformar el juego de información imperfecta en un conjunto de juegos de información perfecta congruentes con el estado actual del juego. El conjunto es una muestra aleatoria de los posibles escenarios en los que el jugador se puede encontrar. En cada uno de estos escenarios se realiza búsqueda en árbol y al final se toma la acción que en el mayor número de escenarios llevó a una victoria.

Se considera que PIMC muestra una ventaja sobre la primera alternativa por su simpleza y facilidad de implementación. Así mismo, permite aprovechar la experiencia previa que el desarrollador ha tenido implementando jugadores basados en búsqueda en árbol. Dadas las restricciones del desarrollo, se llegó a la conclusión que PIMC sería la mejor alternativa para los fines de este trabajo.

## 3.2 Arquitectura

El programa consta de los siguientes módulos:

1. Análisis del tablero y jugadas predefinidas
2. Muestreo de fichas desconocidas (*Determinization*)
3. Búsqueda en árbol de juego con información perfecta



*Figura 1 Diagrama de bloques del sistema. El sistema se descompone en tres módulos internos.*

El sistema recibe la representación del estado actual del juego externamente. Cada módulo procesa su entrada y tiene como salida la entrada del siguiente módulo. La salida del último módulo es la respuesta final del sistema completo.

### 3.2.1 Análisis de tablero y jugadas predefinidas

En la primera etapa del sistema, se realiza un análisis del estado del juego que se recibe como parámetro. En este análisis se busca extraer información necesaria para el muestreo de la siguiente etapa, así como determinar si existe una jugada predefinida para el estado actual del juego.

Las circunstancias para utilizar una jugada predeterminada son dos. La primera es si sólo se tiene una jugada posible, en cuyo caso se realiza dicha jugada sin iniciar el proceso costoso de búsqueda. La segunda es en el primer turno del juego, en donde puede ser deseable incorporar conocimiento experto del dominio para elegir una acción cuando no se cuenta con suficiente información para que el sistema calcule una jugada efectiva en el límite de tiempo establecido.

Si existe una jugada predefinida se omitirá la ejecución de los demás módulos y la jugada será la salida del sistema. En otro caso, a partir de la historia del juego se calcularán las



fichas que aun no han sido tiradas y el número de fichas que tiene cada jugador. Este resultado se pasará al siguiente módulo.

### 3.2.2 Muestreo de fichas desconocidas

Siguiendo el algoritmo de PIMC, es necesario generar un conjunto de posibles escenarios en los que se puede encontrar el jugador. En este caso, los escenarios se diferencian por las fichas que poseen los otros jugadores. Ya que se cuenta con el conjunto de fichas que aun no se han jugado, se puede simular el proceso de repartir las fichas aleatoriamente a los otros jugadores y así generar los escenarios. Un escenario se conforma de un arreglo en donde se guardan las fichas que se le repartieron aleatoriamente a cada jugador.

El número de simulaciones que se realizan es un parámetro del sistema que impacta su comportamiento en dos formas. Entre más simulaciones se realizan, aumenta el tiempo de ejecución. Entre menos simulaciones, el sistema toma en cuenta menos escenarios y el desempeño del jugador empeora.

Al final, se recopilan los distintos escenarios en un arreglo y se pasan al siguiente módulo.

### 3.2.3 Búsqueda en árbol de juego con información perfecta

Una vez que se ha transformado el juego en un conjunto de escenarios en los que se conocen las fichas de los oponentes, es posible calcular una acción óptima para cada uno de los escenarios. La ficha que el jugador artificial tirará será aquella que en el mayor número de escenarios fue óptima.

Para hacer el cálculo de la jugada óptima hay distintas alternativas. *Negamax*, *Alpha-Beta Pruning* y *Monte Carlo Tree Search* (MCTS) son los principales candidatos para realizar búsqueda en árbol de juego con información perfecta

Se eligió el algoritmo MCTS debido a dos características que no comparte con las dos primeras opciones. En primer lugar, es posible correr el algoritmo sin necesidad de una heurística, es decir, de una función que estime la utilidad de un estado del juego. En segundo lugar, MCTS es un algoritmo *anytime*, lo que significa que la ejecución puede detenerse en un intervalo arbitrario de tiempo y el algoritmo regresará la mejor jugada que ha encontrado hasta ese momento.

La segunda propiedad es particularmente apropiada para este sistema ya que permite controlar el tiempo de ejecución total modificando el tiempo que se invierte en calcular la acción óptima en cada uno de los escenarios.

## 3.3 Estándares utilizados

Como parte de la API que el sistema expondrá para ser integrado con otras plataformas se decidió que la comunicación de información se haga con el estándar JSON (ECMA-404)

debido a su flexibilidad y facilidad de uso. Asimismo, se utilizará el estándar PEP8 que define una guía de estilo y mejores prácticas para escribir código en Python.

## **Referencias**

Angeris, G., & Li, L. (2016). CS 221 Project Final : DominAI. Recuperado de <https://web.stanford.edu/~guillean/papers/dominai.pdf>

Frank, I., and Basin, D. 1998. Search in games with in- complete information: A case study using bridge card play. *Artificial Intelligence* 87–123.

Ginsberg, M. L. (2001). GIB: Imperfect Information in a Computationally Challenging Game. *Journal of Artificial Intelligence Research*, 14, 303-358. <https://doi.org/10.1613/jair.820>

Long, Jeffrey Richard et al. "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search." AAAI (2010).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. , .