Facultad 4

Componente de Inteligencia Artificial en Unity para videojuegos de Dominó

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Alexander Rodríguez Saavedra
Tutores: Ing. Rafael Iglesias Miranda
Dr.C PA Yuniesky Coca Bergolla

La Habana, noviembre de 2022
Año 63 de la Revolución

DECLARACIÓN DE AUTORÍA

El autor del trabajo de diploma con título ***"Componente de Inteligencia Artificial en Unity para videojuegos de Dominó"***, concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara como único autor de su contenido. Para que así conste firma la presente a los 16 días del mes de noviembre del año 2022.

Alexander Rodríguez Saavedra

_____

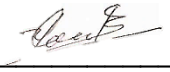
Firma del Autor

Ing. Rafael Iglesias Miranda

_____

Firma del Tutor

Dr.C PA Yuniesky Coca Bergolla



_____

Firma del Tutor

Datos de contacto

Autor:

Alexander Rodríguez Saavedra

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: alexanderrs@estudiantes.uci.cu

Tutor:

Ing. Rafael Iglesias Miranda

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: riglesias@uci.cu

Co-tutor

Dr.C PA Yuniesky Coca Bergolla

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: ycoca@uci.cu

Agradecimientos

A mi familia que me ha apoyado en todo momento, en especial a mi hermana, a mi papá y a mi mamá, hoy estoy aquí gracias a ella.

A mi novia (casi esposa) y Evian por siempre hacernos reír.

A mis tutores, que fueron pieza clave en la realización de esta tesis, a Rafael que a pesar de estar pasando por un momento complicado nunca dejó de brindarme su apoyo, y al Profe Coca, que siempre pudo hacer un espacio para conversar y ayudarme.

A mis amigos de la UCI, a Yaylé y Lili por el café y los chismes.

Dedicatoria

Dedico mi esfuerzo a mi mamá que le ha tocado ser madre y padre este último año y medio, y ha aguantado mi mal humor y mis peleas.

A mi papá, para quien sobran las palabras, mil veces podría borrar y reescribir este párrafo y aún no encontraría las adecuadas, sé que le hubiera encantado estar aquí, a él va dedicado todo lo que soy.

Resumen

En la actualidad existen altos niveles de consumo de videojuegos, estos se han venido perfeccionando a lo largo de los años. Uno de los pilares fundamentales en este desarrollo ha sido la Inteligencia Artificial (IA), que le agrega cierto grado de realismo a los mismos, haciéndolos más atractivos al usuario. En el presente trabajo se plantea como objetivo, desarrollar un componente de IA para videojuegos de dominó en Unity, que permita a la computadora enfrentarse a un usuario. Para guiar el desarrollo de la propuesta de solución se siguen los pasos que plantea la metodología de desarrollo ágil Programación Extrema (XP, por sus siglas en inglés). Se definen como motor de videojuego Unity y C# como lenguaje de programación. El componente dotará al ordenador de un comportamiento inteligente que, siguiendo una serie de estrategias puede llegar a ganar partidas tal como lo haría un jugador humano. Se empleó en la solución el algoritmo Árboles de Búsqueda de Monte Carlo (MCTS, por sus siglas en inglés). Se obtuvieron buenos resultados y se comprobó que mientras más profundo se explorara el árbol de búsqueda mejor sería la respuesta.

Palabras clave: Algoritmo, Árbol de búsqueda, Dominó, Inteligencia Artificial, Videojuegos

Abstract

Currently there are high levels of video game consumption, these have been perfected over the years. One of the fundamental pillars in this development has been Artificial Intelligence (AI), which adds a certain degree of realism to them, making them more attractive to the user. In the present work, the objective is to develop an AI component for domino video games in Unity, which allows the computer to face a user. To guide the development of the solution proposal, the steps proposed by the Extreme Programming (XP) agile development methodology are followed. Unity is defined as the game engine and C# as the programming language. The component will provide the computer with an intelligent behavior that, following a series of strategies, can win games just as a human player would. The Monte Carlo Search Trees (MCTS) algorithm was used in the solution. Good results were obtained and it was found that the deeper the search tree was explored, the better the response would be.

Keywords: *Algorithm, Artificial Intelligence, Dominoes, Search Tree, Videogames*

Índice

Introducción 1

Capítulo 1 Fundamentación Teórica 5

1.1 Inteligencia Artificial en los videojuegos 5

1.2 Análisis del estado del arte 6

1.3 Técnicas de IA 8

1.3.1 Sistemas expertos: 8

1.3.2 Máquinas de Estados Finitos: 8

1.3.3 Árboles de comportamiento: 9

1.3.4 Aprendizaje Automático 10

1.3.5 Árboles de Búsqueda de Montecarlo (MCTS): 10

1.4 Selección de la técnica a aplicar 10

1.5 Dominó 11

1.6 Estrategias en un juego de dominó 12

1.7 Análisis de homólogos 13

1.8 Metodologías y Herramientas para llevar a cabo el desarrollo de la solución 14

1.8.1 Metodología de desarrollo de software 14

1.8.2 Metodologías de desarrollo de software tradicionales 15

1.8.3 Metodologías de desarrollo de software ágiles 15

1.8.4 Motor gráfico 17

1.8.5 Entorno de desarrollo integrado (IDE) 17

1.8.6 Herramienta de modelado 17

1.8.7 Lenguaje de programación 18

1.8.8 Control de Versiones 18

Conclusiones del capítulo 18

Capítulo 2 Caracterización y diseño de la propuesta de solución 19

2.1 Propuesta de solución 19

2.2 Estructura de la IA propuesta 21
2.3 Fase 1 Planificación 22
2.3.1 Historias de Usuarios 23
2.4 Fase 2 Diseño 27
2.4.1 Tarjetas CRC 27
2.5 Arquitectura del software 28
2.6 Patrones de diseño 28
Conclusiones del capítulo 30
Capítulo 3 Implementación y pruebas realizadas al componente de IA 31
3.1 Estándares de codificación 31
Convenciones de nomenclatura 31
Convenciones de diseño 31
Convenciones de comentarios 32
3.2 Videojuego Pixel Dominoes 32
3.3 Fase 3 Codificación 34
3.3.1 Tareas de implementación Iteración 1 34
3.3.2 Tareas de implementación Iteración 2 35
3.3.3 Tareas de implementación Iteración 3 36
3.4 Fase 4 Pruebas 37
3.4.1 Validación del componente 37
3.4.2 Pruebas de aceptación 39
3.4.3 Análisis de los resultados de las pruebas de aceptación 40
Conclusiones del capítulo 40
Conclusiones 42
Recomendaciones 43
Referencias bibliográficas 44
Bibliografía 47

Anexos..... 48

Historias de Usuario: 48

Pruebas de Aceptación:..... 50

ÍNDICE DE TABLAS

Tabla 1 HU 1: 23
Tabla 2. HU 2:..... 23
Tabla 3.HU 3: 24
Tabla 4. HU 4:..... 24
Tabla 5. HU 5..... 24
Tabla 6. HU 6:..... 25
Tabla 7. HU 7:..... 25
Tabla 8. HU 8:..... 25
Tabla 9. Estimación de esfuerzo por Historia de Usuario..... 26
Tabla 10. Plan de duración de las iteraciones..... 26
Tabla 11. Planificación de entrega..... 27
Tabla 12. Tarjeta CRC # 1 27
Tabla 13. Tarjeta CRC # 2 28
Tabla 14. Estimaciones por HU en la Iteración 1. 34
Tabla 15. Tarea de implementación 1. 34
Tabla 16. Tarea de implementación 2. 34
Tabla 17. Estimaciones por HU en la Iteración 2. 35
Tabla 18. Tarea de implementación 3. 35
Tabla 19. Tarea de implementación 4. 35
Tabla 20 Tarea de implementación 5..... 36
Tabla 21. Estimaciones por HU en la Iteración 3. 36
Tabla 22. Tarea de implementación 6. 36
Tabla 23. Tarea de implementación 7. 36
Tabla 24. Tarea de implementación 8. 37
Tabla 25..... 38
Tabla 26..... 38
Tabla 27. Prueba de Aceptación 1. 39
Tabla 28. Prueba de Aceptación 2. 39
Tabla 29. Prueba de Aceptación 3. 40
Tabla 30. Prueba de Aceptación 4..... 40
Tabla 31 HU 1:..... 48
Tabla 32. HU 2:..... 48
Tabla 33.HU 3:..... 48

Tabla 34. HU 4:..... 49
Tabla 35. HU 5..... 49
Tabla 36. HU 6:..... 49
Tabla 37. HU 7:..... 50
Tabla 38. HU 8:..... 50

ÍNDICE DE FIGURAS

Ilustración 1. Fases del algoritmo [Adaptado de (Chaslot, 2010)] 21
Ilustración 2. Pantalla de inicio..... 32
Ilustración 3. Desarrollo de una partida..... 33

Introducción

La historia del juego data de mucho antes de lo que se imagina. Los juegos son una parte integral de todas las culturas y es una de las formas más viejas de interacción social humana. Las características comunes de juegos incluyen incertidumbre de resultado, estado de acuerdo a reglas, competición, tiempo y sitio separados, elementos de ficción, elementos de posibilidad, objetivos prescritos y diversión personal (Radoff, 2010).

Un videojuego es una aplicación interactiva, donde los usuarios deben involucrarse activamente con el contenido, orientada al entretenimiento que, a través de ciertos mandos o controles, permite simular experiencias en la pantalla de un televisor, una computadora u otro dispositivo electrónico. El concepto se refiere a cualquier juego digital interactivo, independientemente de su soporte físico. Pueden ser muy distintos entre sí, tanto en complejidad como en calidad gráfica y en temática (Rebour).

En este campo, la Inteligencia Artificial (IA) se define como la simulación de comportamientos de los personajes no controlados por el jugador: *Non Playable Character* (NPC) y objetos del entorno que no están bajo el control directo de los jugadores (Hmong).

Se podría pensar que el primer videojuego con cierto nivel de Inteligencia Artificial fue el famoso Pong, pero existen evidencias que data de incluso 20 años atrás antes de que este surgiera. En concreto, el primer videojuego que utilizó algún algoritmo de Inteligencia Artificial se llamaba Nim. Era un juego sencillo, trataba de ir eligiendo diferentes elementos en una estructura (los elementos clásicos son las cerillas) y ser el último en retirar un objeto. La IA implementaba teoría combinatoria de juegos utilizando operaciones lógicas binarias (Sistemas & Telefónica, 2020).

En la actualidad se evidencia el protagonismo que ha ganado la IA en el campo del entretenimiento. Incluso en el área de los juegos de mesa, los cuales se han llevado al mundo digital mediante el uso de las nuevas tecnologías. Se puede mencionar varios ejemplos de aplicaciones de IA que se han destacado en juegos como el

Ajedrez, Shogi y Go, tal es el caso de Deep Blue de IBM, Stockfish y AlphaZero, una IA que fue capaz de aprender a jugar por si sola recurriendo a una red neuronal profunda, a través de constantes partidas aleatorias y sin más información que las reglas del juego.

Esto supuso una ruptura con respecto al enfoque adoptado por las aplicaciones mencionadas anteriormente, las cuales se basaban en miles de reglas y heurísticas creadas por fuertes jugadores humanos que tratan de explicar cada eventualidad en un juego.

La Universidad de Ciencias Informáticas (UCI) posee varios proyectos que persiguen además de fomentar el aprendizaje, entretener a los usuarios. La UCI cuenta con varios centros de desarrollo, uno de ellos es el Centro de Entornos Interactivos 3D (Vertex), responsable de la creación de disímiles videojuegos que forman parte del entretenimiento y disfrute del público en general. Entre sus proyectos se encuentra la plataforma de videojuegos en línea Cosmox, que permite a usuarios de diferentes localizaciones del país converger en el mismo espacio digital, promoviendo la recreación, la interacción social y el aprendizaje. Esta plataforma dispone de diversos videojuegos, entre ellos el Dominó, tanto en la versión doble seis como la versión doble nueve. Sucede que para comenzar una partida debe haber al menos dos personas conectadas, pero en ocasiones la cantidad de jugadores no alcanza para comenzar dicha partida, ni siquiera para jugar uno contra uno, trayendo consigo la pérdida de usuarios para estos juegos y plataformas que aún no incorporan un oponente controlado por el ordenador con cierto nivel de inteligencia

De esta **situación problemática,** surge el siguiente **problema:** ¿Cómo lograr que la computadora se enfrente a un usuario en un videojuego de Dominó elaborado en Unity?

Se define como **objetivo** desarrollar un módulo de IA para videojuegos de dominó en Unity, que permita a la computadora enfrentarse a un usuario.

Teniendo como **objeto de estudio** las técnicas de IA aplicadas en videojuegos, se define como **campo de acción** de la investigación: las técnicas de IA aplicadas en videojuegos de Dominó.

Para dar cumplimiento al objetivo planteado se trazan las siguientes tareas de investigación:

- Realización de análisis y síntesis de los elementos relacionados con el campo de acción: concepto de videojuegos, las diferentes técnicas de IA que más se usan en la actualidad en los videojuegos y selección de una de estas para su implementación.

- Caracterización de la IA en los videojuegos.

- Identificación de la metodología y herramientas para el desarrollo de la propuesta de solución.

- Elaboración de los artefactos según la metodología seleccionada y descripción de la propuesta de solución.

- Implementación del módulo de IA propuesto teniendo en cuenta los resultados de las tareas previas.

- Validación del cumplimiento de los requerimientos de la IA implementada.

A continuación, se mencionan los métodos teóricos y empíricos que fueron empleados durante el desarrollo de la investigación.

Métodos Teóricos

- **Histórico - Lógico:** método utilizado para realizar un resumen del desarrollo de la IA a través de los años enfocado en el progreso de los videojuegos, haciendo énfasis en las principales técnicas usadas en esta área.

- **Analítico - Sintético:** utilizado con el propósito de estudiar y resumir la bibliografía necesaria para la resolución del problema, relacionada con los videojuegos y la IA aplicada a estos.

Métodos Empíricos

- **Consultas bibliográficas:** utilizado para la consulta de las fuentes bibliográficas durante la investigación para elaborar el marco teórico de la investigación.

Estructura del documento

El presente trabajo de diploma, está compuesto por tres capítulos, que incluyen los procedimientos desarrollados en relación con el trabajo investigativo, así como la propuesta de solución y validación de la investigación. A continuación, se presenta su descripción:

Capítulo 1: Fundamentación teórica: Este capítulo contiene los conceptos fundamentales relacionados con el campo de acción.

Capítulo 2: Caracterización y diseño de la propuesta de solución: En este capítulo se describe la propuesta de solución, así como los requerimientos que debe cumplir la aplicación mediante las Historias de Usuario y las tarjetas CRC.

Capítulo 3: Implementación y pruebas realizadas al módulo de IA: En este capítulo queda evidenciado el proceso de implementación de la solución, así como los estándares utilizados en la codificación de la misma. También se muestran los resultados de las pruebas de validación para verificar su correcto funcionamiento.

Capítulo 1 Fundamentación Teórica

Durante el presente capítulo se lleva a cabo un estudio abordando conceptos relacionados a la Inteligencia Artificial (IA).

Se realiza un recorrido desde su surgimiento y primeros referentes históricos, que aunque se remontan a los años 30 con Alan Turing, numerosos investigadores e historiadores consideran que el punto de partida de la moderna inteligencia artificial fue el año 1956, cuando los padres de la IA moderna, John McCarty, Marvin Misky y Claude Shannon acuñaron formalmente el término durante la conferencia de Darmouth, como: «la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cálculo inteligente»

1.1 Inteligencia Artificial en los videojuegos

En informática la IA se refiere a la inteligencia expresada por máquinas, sus procesadores y sus softwares, que serían los análogos al cuerpo, el cerebro y la mente, respectivamente, a diferencia de la inteligencia natural demostrada por humanos y ciertos animales con cerebros complejos. Se considera que el origen de la IA se remonta a los intentos del hombre desde la antigüedad por incrementar sus potencialidades físicas e intelectuales, creando artefactos con automatismos y simulando la forma y las habilidades de los seres humanos (Gaudio, 2021).

En ciencias de la computación, una máquina inteligente ideal es un agente flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea.

En los videojuegos la IA se concibe como el comportamiento de los *Non Playable Character* (NPC) y otros objetos no controlados por el jugador, y tienen la capacidad de percibir su entorno y reaccionar de una forma similar a los humanos. El desarrollo de la IA en esta área está enfocado en hacer que los NPC tengan un comportamiento más humano, no más perfecto. Es precisamente este enfoque lúdico lo que hace que sea diferente respecto a otros campos (IAT, 2020).

1.2 Análisis del estado del arte

A continuación, se hará un repaso a la historia de la IA en los videojuegos. Utilizar algoritmos para conseguir que los personajes de un videojuego tengan algún tipo de comportamiento inteligente, viene prácticamente desde los comienzos de la Informática, durante los años 50 con videojuegos como Nim, que se menciona anteriormente, y el Pong.

En los años 70 y 80 se comienza a observar juegos en el mercado donde ya se tomaba más en serio el desarrollo de la IA. Uno de ellos es Pac-Man, revolucionando la utilización de patrones en el mundo de la IA en los videojuegos que existían hasta ese momento. Se evidenciaba en los personajes del juego, los 4 fantasmas usaban diferentes patrones, se podría decir que cada uno tenía su propia personalidad. El fantasma azul era el que tenía posiblemente el comportamiento más especial, intentaba predecir, basándose en algo de aleatoriedad, cálculos para encontrar el camino más corto y un poco de vectores, hacia donde era posible que se moviera Pac-Man.

Ya en los años 90, con ordenadores cada vez más potentes, el estilo de las clásicas IAs fueron pasando de moda y comenzaron a utilizarse otro tipo de algoritmos e incluso se mejoraron los ya existentes. Aparecieron los juegos de estrategia en tiempo real, como Dune II o Warcraft, los cuales básicamente se basan en Finite State Machine o Máquina de Estados Finitos (FSM), algoritmos para la resolución de laberintos o cálculos para hallar el camino más corto. Pero claro, utilizar FSM puede llevar a un estado de repetición donde el ordenador siempre realizará los mismos pasos.

En esta etapa entran los Árboles de Búsqueda de Monte Carlo (MCTS), algoritmo capaz de visualizar los posibles movimientos disponibles en un momento dado de la partida. En cada paso se ejecuta un análisis completo para considerar si es o no factible realizarlo (Sistemas & Telefónica, 2020).

Un ejemplo del uso de la inteligencia artificial en videojuegos para crear enemigos con un comportamiento más humano fue Metal Gear Solid (PSX, 1998). En este juego de infiltración, los soldados enemigos no solo seguían una rutina predefinida.

Además, eran capaces de saltársela si escuchaban un ruido extraño, veían pisadas en la nieve, etc. Incluso algunos soldados eran más perspicaces que otros cuando veían al jugador escondido dentro de una caja (IAT, 2020).

Lo último en inteligencia artificial es el perfeccionamiento de los sistemas procedurales. De las técnicas básicas de generación de antaño se ha pasado a otras capaces de generar terrenos y escenarios casi infinitos, por ejemplo, en juegos como Minecraft, No Man´s Sky o Spelunky (IAT, 2020).

Los juegos de mesa son un terreno de entrenamiento útil para la inteligencia artificial porque se basan en unas reglas muy determinadas y en información que está totalmente disponible en el tablero (La Vanguardia, 2018).

En la actualidad existen varios ejemplos de algoritmos de IA que han sido capaces de derrotar incluso, a grandes maestros del ajedrez y del Go. Entre ellas se encuentran: Stockfish, que es un motor de ajedrez UCI (Interfaz de Ajedrez Universal) de código abierto para múltiples plataformas desarrollado por Tord Romstad, Joona Kiiski, Marco Costalba y Gary Linscott, En abril de 2016, Stockfish goza de la primera o segunda posición en los rankings de motores de ajedrez compitiendo contra programas fuertes como Houdini, Komodo, Rybka, Critter, Gull, entre otros (Álvarez Alba, 2021). Otra sería Alpha Zero, un programa que había demostrado ser capaz de aprender a jugar desde cero, ajedrez, shogi y Go**,** y terminar ganando a todas las aplicaciones de IA que se habían proclamado campeonas en cada uno de esos juegos. Comienza a aprender de forma autónoma recurriendo a una red neuronal profunda, a través de constantes partidas aleatorias y sin más información que las reglas del juego. Es un ejemplo de aprendizaje reforzado, muestra cómo es posible aprender desde cero y alcanzar un rendimiento sobrehumano en varios juegos de gran complejidad.

El “aprendizaje por refuerzo” utiliza en este caso en una red neuronal que juega millones de partidas contra sí misma en un proceso de prueba y error, de tal modo que va tomando nota de la clase de jugadas que contribuyen a alcanzar el objetivo de ganar la partida. Una vez entrenada, la red se usa para guiar un algoritmo de búsqueda llamado “Árbol de búsqueda de Monte-Carlo” que permite que, en lugar

de analizar todos los movimientos posibles, AlphaZero se centre únicamente en aquellos más prometedores según su experiencia previa.

1.3 Técnicas de IA

Como se expresa anteriormente, la IA en videojuegos se refiere al comportamiento de los NPC y otros objetos no controlados por el jugador. A continuación, se describen las principales técnicas o métodos más usados en la actualidad en la industria de los videojuegos.

1.3.1 Sistemas expertos:

Los sistemas expertos son un tipo de sistema basado en reglas que definen el conocimiento para que los agentes autónomos se comporten de manera similar a un jugador experto.

El desarrollo de sistemas expertos ha sido esencial a partir de la utilización de IA, que imita los mecanismos y la forma de pensar de un experto en cierta materia para resolver problemas de su campo de aplicación. Esto lo hace adecuado para un juego como el dominó, cuyas estrategias ganadoras son conocidas. Estos sistemas tienen una gran implantación en diversas ramas de la ciencia, como medicina, ingeniería o sistemas de decisiones para negocios (Riley, 1989).

1.3.2 Máquinas de Estados Finitos:

Las máquinas de estados finitos (FSM) son un modelo computacional que realiza cálculos automáticamente sobre una entrada para producir una salida. Están fundamentadas en la teoría de autómatas y han sido ampliamente estudiadas y formalizadas. Para el desarrollo de videojuegos, donde históricamente han sido una de las técnicas más utilizadas para implementar comportamientos, se ven como un conjunto de estados en los que puede encontrarse la entidad y una serie de transiciones o condiciones de cambio de estado. Eso permite una representación visual simple e intuitiva en forma de grafo donde los nodos son los estados y las transiciones son las aristas.

Cada nodo/estado se describe mediante la acción o acciones primitivas que la entidad ejecutará cuando se encuentre en ese estado. Cada una de las aristas/transiciones son una descripción de la condición del mundo que dispara esa transición. La ventaja de las máquinas de estados frente a otros mecanismos de creación de comportamientos es su simplicidad, que no requieren conocimientos de programación para entenderlas y crearlas, su determinismo y rapidez de ejecución. Además, como tienen una representación visual simple, se pueden construir editores gráficos.

El tradicional problema de las máquinas de estado estriba en que estas no escalan demasiado bien cuando el número de transiciones es muy grande, ya que comienza a incrementar su número rápidamente cuando la complejidad del problema comienza a crecer, por lo que comienza a ser difíciles de mantener, de seguir y controlar por parte del diseñador (Sagredo-Olivenza et al., 2014).

1.3.3 Árboles de comportamiento:

Los árboles de comportamiento (*Behaviour Trees* o BTs) son utilizados en videojuegos para modelar la inteligencia artificial de los NPCs de una forma similar a las máquinas de estado. La diferencia principal radica en que los BTs eliminan las transiciones de las máquinas de estado, lo que permite subsanar los problemas derivados del crecimiento del número de transiciones típicos de las FSMs. De esta forma, los árboles de comportamiento definen un mecanismo para decidir cuál es la siguiente acción a ejecutar. Para conseguirlo, incorporan información adicional de control a las acciones que determine cuál de ellas debe ser ejecutada en un momento dado. Esa información de control es modelada normalmente mediante nodos intermedios que toman decisiones sobre el orden en el que deben ejecutarse las acciones primitivas. Estas decisiones se toman utilizando los datos del entorno mediante nodos especiales que los chequean y utilizando el valor de retorno de las acciones (Sagredo-Olivenza et al., 2014).

1.3.4 Aprendizaje Automático

El aprendizaje automático es una rama de la inteligencia artificial (IA) y la informática que se centra en el uso de datos y algoritmos para imitar la forma en que los humanos aprenden, mejorando gradualmente su precisión (IBM, 2020).

El *machine learning* o aprendizaje automático es una disciplina científica del ámbito de la inteligencia artificial cuyo principal objetivo consiste en desarrollar técnicas que permitan a las máquinas aprender de manera autónoma. Para lograr esta labor el algoritmo se encarga de construir un modelo matemático usando como base unos datos de ejemplo. Por lo tanto, se puede definir como un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados que describen casos particulares (Labena, 2020).

1.3.5 Árboles de Búsqueda de Montecarlo (MCTS):

Monte Carlo Tree Search (MCTS) es el primer método de búsqueda que no requiere una función de evaluación de posición en contraste con la búsqueda αβ. Está basado en una exploración aleatoria del espacio de búsqueda, pero usa los resultados de previas exploraciones. Para ello MCTS construye gradualmente un árbol en memoria, que mejora sucesivamente estimando los valores de los movimientos más prometedores. Gracias a la estructura de árbol y un alto número de simulaciones aleatorias, el método MCTS puede estimar a largo plazo el potencial de cada movimiento (Nasarre Embid, 2012).

El Método de Monte Carlo proporciona soluciones a una gran variedad de problemas matemáticos haciendo experimentos con conjuntos de datos estadísticos en una computadora. El método es aplicable a cualquier tipo de proceso, sea estocástico o determinístico (Moreno Montiel et al., 2020).

1.4 Selección de la técnica a aplicar

Luego de realizado un análisis de las distintas técnicas de IA existentes más utilizadas en la actualidad en el mundo de los juegos electrónicos, teniendo en cuenta sus características, sus ventajas y desventajas se decide emplear MCTS para el desarrollo de la propuesta de solución, pues estos permiten un control de la

dificultad del juego al limitar el tiempo de búsqueda, además permite visualizar los movimientos posibles en un determinado momento de la partida. El algoritmo es de fácil implementación y se han obtenido muy buenos resultados en videojuegos como Go.

1.5 Dominó

El dominó es un juego de estrategia que ha atravesado por diversos cambios y evoluciones desde sus orígenes. Consiste en la unión de fichas que corresponden a un sentido lógico para garantizar una puntuación superior al resto de los jugadores y resultar ganador. Al igual que ocurre en muchos juegos, el dominó no se inventa en una fecha concreta ni es la idea de una sola persona. Más bien es la evolución o modificación de juegos más antiguos.

Se deduce que el dominó actual proviene del continente asiático, específicamente en China donde las personas tenían un juego similar al dominó moderno. Hace más de 1.500 años este juego se empezó a documentar en China los escritos de Zhou Mi (1232-1298) en la dinastía Yuan (Lanza Digital, 2020).
A continuación, se muestra el flujo de una partida de dominó:

Iniciar ronda:

- Comienza con las fichas boca abajo en la mesa y se mezclan o como se dice, darle agua.
- Cada jugador toma 7 o 10 fichas aleatoriamente.
- Para comenzar el juego sale el jugador que tenga el doble 6/doble 9.

Desarrollo:

- En su turno cada jugador podrá jugar una de sus fichas, con la condición de que dos fichas solo pueden estar juntas cuando los puntos en ambos lados de estas tengan el mismo valor.
- Los dobles se colocan de forma transversal.
- Si el jugador no puede jugar ninguna ficha debe pasar el turno al otro jugador

Fin:

- La ronda puede finalizar si ningún jugador, teniendo todos fichas en mano, pueda jugarlas, esto se conoce como tranque. Gana el jugador cuyas fichas tengan menos puntos.
- Si un jugador colocó todas sus fichas en la mesa, gana el juego o ronda.

1.6 Estrategias en un juego de dominó

En el dominó existen dos tipos de estrategias, unas son para abrir el juego llamadas estrategias de salida y las otras son para el desarrollo del juego o estrategias durante el juego. Para jugar de forma correcta se deben combinar estas. La estrategia que se va a utilizar se decide a partir de las combinaciones de fichas que se tiene al momento del inicio del juego y durante su desarrollo.

Estrategias de Salida:

1. Salir con la ficha más acompañada: Esta estrategia de salida se considera el hecho de tener o no dobles y también se considera la configuración inicial de fichas con la que cuenta el usuario. Se revisa también la situación en el que se pueda formar una pareja de fichas.
2. Salir con la ficha de mayor valor: Se sale con la mayor de las fichas que se tenga en la mano sin importar el resto.
3. Salir con el mayor doble: Se sale con el mayor de los dobles, aunque no se cuente con más fichas con ese valor.
4. Salir con la ficha que nos “estorba”: Se sale con la ficha que nos moleste para el desarrollo del juego.

Estrategias para el desarrollo del juego:

1. Estrategia básica: Esta estrategia es la que cualquier jugador novato utilizaría si es que está jugando por primera vez dominó, y consiste en tirar la primera ficha válida que encaje con el desarrollo del juego, sin importar si es un doble o no.

2. Tirar la ficha más acompañada: Se aplica la misma estrategia de salida “Salir con la ficha más acompañada”.
3. Tirar la ficha de mayor valor: Se aplica la misma estrategia de salida “Salir con la ficha de mayor valor”.
4. Tirar el mayor doble: Consiste primeramente en jugar los dobles sin importar el resto de las fichas y además elegir de los dobles disponibles el mayor.
5. Pasar al oponente: Jugadores más experimentados mediante el análisis del tablero puede crear situaciones para pasar al oponente, evitando que este coloque alguna ficha en el tablero, y tratar de repetir esta actividad las veces que sea posible.
6. Cerrar el juego: Esta estrategia es buena para seguir cuando se cuenta con fichas de valores bajos, consiste en contar el total de fichas que han salido de determinado valor, si han salido cuatro o cinco de estas, ver si se pueden poner ambos extremos del juego al mismo valor procurando así, que todos los jugadores pasen y no puedan hacer otra jugada, cuando el juego se “cierra” ganará el juego aquel que cuente con la menor suma de los puntos de las fichas en su mano.

1.7 Análisis de homólogos

En la web existen variados sitios donde se pueden establecer partidas de dominó, ya sea contra otros jugadores reales, o contra jugadores controlados por la computadora, a continuación, se describen algunos de ellos.

Dominó en línea:

Es un juego muy sencillo de dominó, se juega contra un oponente controlado por la computadora, las fichas se colocan automáticamente y la puntuación es calculada por el programa. Solo tiene un nivel bastante fácil, y se informa de cuando el jugador está haciendo una jugada ilegal.

Devworks Dominó:

Es un juego que cuenta con geniales animaciones y música. Es muy simple, consiste en efectuar partidas entre el jugador y el ordenador. También cuenta con

una sección de ayuda por si el jugador quiere conocer un poco más acerca de las reglas de este juego de mesa.

En el artículo llamado: “Simulación de Monte Carlo para el juego de dominó” de una serie de autores publicado en el 2010, persiguen como objetivo, demostrar mediante la Simulación de Monte Carlo cual es la mejor estrategia a seguir para ganar una partida de dominó. Para ello realizaron un análisis tanto de las estrategias de salida como de desarrollo del juego que utilizan los jugadores. En ese trabajo, luego de un largo número de simulaciones de partidas de dominó, utilizando la simulación estadística o Método de Monte Carlo se arribó a la conclusión de que la estrategia usada en la salida del juego no influye en el resultado obtenido, y que la mejor estrategia para el desarrollo del juego es jugar siempre con la ficha más acompañada (Moreno Montiel et al., 2020).

Luego de investigar la existencia tanto de otras aplicaciones, como de sitios online, donde es posible realizar partidas de dominó contra un componente no jugador, se puede concluir que ninguno cumple los requisitos necesarios para dar solución a la problemática, pues solo cuentan con funcionalidades básicas, no se pudo determinar un patrón en ellas, además de no tener acceso al código fuente por lo que no se pudo determinar si las jugadas realizadas por el ordenador son realmente inteligentes o son al azar.

No obstante, a través del estudio del artículo “Simulación de Monte Carlo para el juego de dominó”, se logró confirmar la eficacia del Método de Monte Carlo, pues se obtuvieron muy buenos resultados.

1.8 Metodologías y Herramientas para llevar a cabo el desarrollo de la solución

1.8.1 Metodología de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de técnicas y métodos organizativos que se aplican para diseñar soluciones de software informático. El objetivo de las distintas metodologías es el de intentar organizar los equipos de trabajo para que estos desarrollen las funciones de un programa de la mejor manera posible (Santander Universidades, 2020).

Cuando se trata de desarrollar productos o soluciones para un cliente o mercado concreto, es necesario tener en cuenta factores como los costes, la planificación, la dificultad, el equipo de trabajo disponible, los lenguajes utilizados, etc. Todos ellos se engloban en una metodología de desarrollo que permite organizar el trabajo de la forma más ordenada posible. El trabajo con una metodología de desarrollo de software permite reducir el nivel de dificultad, organizar las tareas, agilizar el proceso y mejorar el resultado final de las aplicaciones a desarrollar.

En la actualidad se pueden diferenciar dos grandes grupos de metodologías de desarrollo de software: las ágiles y las tradicionales. A continuación, se explican las características de cada una de ellas:

1.8.2 Metodologías de desarrollo de software tradicionales

Las metodologías de desarrollo de software tradicionales se caracterizan por definir total y rígidamente los requisitos al inicio de los proyectos de ingeniería de software. Los ciclos de desarrollo son poco flexibles y no permiten realizar cambios, al contrario que las metodologías ágiles; lo que ha propiciado el incremento del uso de las segundas.

La organización del trabajo de las metodologías tradicionales es lineal, lo que significa que las etapas se suceden una tras otra y no se puede empezar la siguiente sin terminar la anterior. Tampoco se puede volver hacia atrás una vez se ha cambiado de etapa, por lo que no adapta bien a los cambios.

1.8.3 Metodologías de desarrollo de software ágiles

Estas son las más utilizadas hoy en día debido a su alta flexibilidad y agilidad. Los equipos de trabajo que las utilizan son mucho más productivos y eficientes, ya que saben lo que tienen que hacer en cada momento. Además, la metodología permite adaptar el software a las necesidades que van surgiendo por el camino, lo que facilita construir aplicaciones más funcionales.

Las metodologías ágiles se basan en la metodología incremental, en la que en cada ciclo de desarrollo se van agregando nuevas funcionalidades a la aplicación final. Sin embargo, los ciclos son mucho más cortos y rápidos, por lo que se van agregando pequeñas funcionalidades en lugar de grandes cambios.

Luego de un análisis se decide usar una metodología ágil por todas las ventajas que ofrece con respecto a una tradicional, se elimina la burocracia de una metodología tradicional y al ser el equipo de desarrollo de una sola persona es necesario aumentar la velocidad y acortar el tiempo de entrega del producto, trabajando con plazos y entregas parciales.

Las principales metodologías ágiles son:

Kanban: Consiste en un método visual de gestión de proyectos que permite dividir las tareas en porciones mínimas y organizarlas en un tablero de trabajo dividido en tareas pendientes, en curso y finalizadas. De esta forma, se crea un flujo de trabajo muy visual basado en tareas prioritarias y carga de trabajo (Asana, 2020).

Scrum: es también una metodología incremental que divide los requisitos y tareas de forma similar a Kanban. Se trata de una metodología de trabajo ágil que tiene como finalidad la entrega de valor en períodos cortos de tiempo y para ello se basa en tres pilares: la transparencia, inspección y adaptación. Las etapas son: planificación de la iteración (planning sprint), ejecución (sprint), reunión diaria (daily meeting) y demostración de resultados (sprint review). Cada iteración por estas etapas se denomina también sprint.

Lean: está configurado para que pequeños equipos de desarrollo muy capacitados elaboren cualquier tarea en poco tiempo. Los activos más importantes son las personas y su compromiso, relegando así a un segundo plano el tiempo y los costes. El aprendizaje, las reacciones rápidas y potenciar el equipo son fundamentales.

Programación extrema (XP): Es un conjunto de técnicas que dan agilidad y flexibilidad en la gestión de proyectos. Se centra en crear un producto según los requisitos exactos del cliente. De ahí, que le involucre al máximo durante el método de gestión del desarrollo del producto (Sinnaps, 2020).
Luego de abordadas las principales metodologías de software ágil se decide emplear XP para el desarrollo de la propuesta de solución. Tiene sus bases en la comunicación constante y la retroalimentación. Es adaptable a los cambios,

pudiendo actuar de forma rápida frente a cualquier inconveniente. Se adoptan sus 4 fases: Planificación, Diseño, Desarrollo y Prueba.

1.8.4 Motor gráfico

Se utiliza Unity como motor de desarrollo, en su versión 2019.1.0f2. Unity es un motor de juegos multiplataforma desarrollado por Unity Technologies, anunciado y lanzado por primera vez en junio de 2005 en la Conferencia Mundial de Desarrolladores de Apple Inc. como un motor de juegos exclusivo de Mac OS X. Desde entonces, el motor se ha ampliado gradualmente para admitir una variedad de plataformas de escritorio, móviles, consolas y realidad virtual. Es particularmente popular para el desarrollo de juegos móviles iOS y Android y se utiliza para juegos como Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Saber y Cuphead. Se considera fácil de usar para desarrolladores principiantes y es popular para el desarrollo de juegos independientes (Hmong, 2021b).

1.8.5 Entorno de desarrollo integrado (IDE)

Se utiliza Rider de JetBrains en la versión 2021.3.3, permite desarrollar de forma productiva una amplia variedad de aplicaciones, incluidas aplicaciones de escritorio .NET, servicios y bibliotecas, juegos Unity y Unreal Engine, aplicaciones Xamarin, ASP.NET y aplicaciones web ASP.NET Core. JetBrains Rider es un editor de C# rápido y potente para Unity que funciona en Windows, Mac y Linux. Con más de 2500 inspecciones y refactorizaciones de código inteligente. Rider tiene compatibilidad con Unity incorporada, gracias a la comunicación bidireccional integrada, puede entrar y salir del modo Play, y hacer una pausa y pasar un marco sin abandonar Rider. La barra de herramientas contiene los botones de vista de juego Play, Pause y Step, que corresponden a los mismos botones en el editor de Unity y funcionan de la misma forma que él (Jetbrains, s. f.).

1.8.6 Herramienta de modelado

Se emplea como herramienta de modelado Visual Paradigm, en su versión 8.0, es una herramienta CASE: Ingeniería de Software Asistida por Computación. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde

la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación (Pressman, 2010).

1.8.7 Lenguaje de programación

C# es un lenguaje de programación multiparadigma de propósito general. C# abarca disciplinas de programación de tipado estático, tipado fuerte, de ámbito léxico, imperativo, declarativo, funcional, genérico, orientado a objetos (basado en clases) y orientado a componentes (Hmong, 2021a).

Es el lenguaje que prima el propio motor de juegos Unity, ya que le da un soporte especial y basa toda su documentación en él. Es que es el más utilizado, con mucha diferencia, por la comunidad de usuarios de Unity, lo que facilita luego el aprovechamiento de scripts creados por terceros en nuestros propios videojuegos.

1.8.8 Control de Versiones

Para el control de versiones se usa GitHub. Es una plataforma de alojamiento perteneciente a Microsoft, que ofrece a los desarrolladores crear repositorios de código y guardarlos de forma segura en la nube, para ello usa un sistema de control de versiones llamado Git. Hace más fácil la organización de los proyectos y permite la colaboración de varios desarrolladores en tiempo real (Camacho, 2021). Usamos la versión web y la aplicación de escritorio en su versión 3.0.2.

Conclusiones del capítulo

Durante la realización del presente capítulo se estudiaron las distintas técnicas de IA más usadas en esta área y soluciones semejantes que se desarrollaron antes. Se definió el algoritmo MCTS para el desarrollo de la propuesta de solución, pues existen evidencias de su correcto funcionamiento y eficacia específicamente en este juego. Se escogieron como herramientas para la implementación, Unity como motor de desarrollo, C# como lenguaje de programación y el IDE Rider de JetBrains. También se escogió XP luego de una comparativa entre las metodologías tradicionales y ágiles para realizar una correcta ingeniería del software.

Capítulo 2 Caracterización y diseño de la propuesta de solución

En este capítulo se dará una descripción de la propuesta de solución, así como el funcionamiento del algoritmo de Monte Carlo. Se desarrollarán las fases de planificación y diseño que plantea la metodología XP junto a sus artefactos ingenieriles, las historias de usuario que fueron definidas y las tarjetas CRC.

2.1 Propuesta de solución

Luego de abordados los elementos necesarios referentes a la problemática que se plantea, técnicas de IA, seleccionada las herramientas y metodología a usar se propone la implementación de un módulo de IA para ajustarlo posteriormente al videojuego Pixel Dominoes, dicho módulo será capaz de dotar de un comportamiento más humano e inteligente a un NPC del juego. Para ello se decide emplear en la solución el algoritmo MCTS.

Fases del algoritmo propuesto

Antes de continuar, se hace necesario explicar las fases del algoritmo a emplear en la solución, así como el funcionamiento y estructura de la IA. MCTS consiste fundamentalmente en 4 pasos, que se repetirán en función del tiempo disponible y la dificultad que se le quiera dar a la IA. Las fases son las siguientes:

- **Selección:** Se recorre el árbol desde la raíz hasta alcanzar un nodo hoja, se toma una rama u otra en función de la estrategia de selección empleada y la información guardada en el nodo en ese momento.

  En nuestro caso usaremos *Upper Confidence bounds apllied to Trees* (UCT) debido a su simplicidad y eficiencia. Esta estrategia es independiente del juego y no usa ningún dominio del conocimiento, calcula para cada movimiento posible una combinación de dos valores, la tasa de éxito de ese nodo y un número asociado a la relación del número de veces que se ha visitado el nodo en relación a un nodo padre. El valor de la tasa de éxito está relacionado con la explotación y el valor del número asociado está relacionado con la exploración.

- **Expansión:** En este paso se añaden nodos al árbol MCTS. En aquellos problemas en los que no es posible almacenar en memoria el juego completo es necesario poseer una estrategia de expansión. Se decide seleccionar la de no expandir el nodo hoja N_i hasta que no se alcance un número mínimo de visitas, logrando con esto un ahorro de memoria evitando que se creen ramas innecesarias. En el caso de expandirse, se puede crear un solo hijo o todos de golpe, se decide emplear esta última pues ocupa más memoria, pero solo se realiza el cálculo de los movimientos posibles alcanzables desde el nodo actual una vez. El nodo raíz se trata como un caso especial, al reflejar la situación de partida no es útil realizar simulaciones directamente sobre él, por lo que siempre se expande.
- **Simulación:** A partir del nodo hoja N_i dado por la fase anterior, se realiza una partida simulada, donde el programa juega solo, realizando movimientos de todos los jugadores que intervienen de forma aleatoria hasta que la partida finalice y obtenga un resultado. Las estrategias que se utilizan consisten o bien utilizar los movimientos aleatorios o combinar la aleatoriedad con una heurística asociada al problema concreto. En estos casos es necesario buscar un equilibrio entre la exploración, que da la aleatoriedad, y la explotación, que dirige hacia un movimiento más prometedor.
- **Retropropagación:** En este paso se realiza la actualización de los valores de los nodos, primero el nodo hoja, luego el nodo padre de éste y así sucesivamente hasta alcanzar la raíz del árbol. La actualización de cada nodo consiste en incrementar en uno el número de visitas y actualizar su valor usando el resultado R de la simulación.

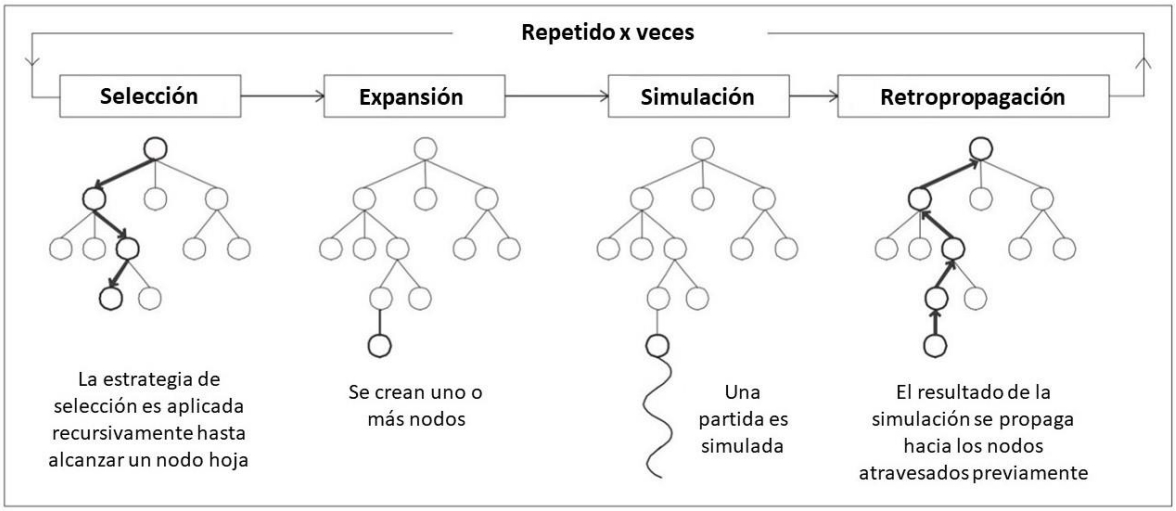


Ilustración 1. Fases del algoritmo [Adaptado de (Chaslot, 2010)]

Selección de la jugada final

Para elegir el movimiento final se considera el mejor hijo del nodo raíz, es decir, el movimiento más prometedor de acuerdo a la información obtenida. Existen varios criterios a la hora de elegir qué nodo es el mejor:

- Más robusto: El hijo con mayor contador de visitas.
- Valor máximo: El hijo donde más simulaciones resultaron ganadoras.
- Robusto – Valor máximo: Es el hijo que tiene tanto el mayor número de visitas como el máximo valor.

Se ha seleccionado para nuestra implementación el criterio “Robusto – Valor máximo” pues puede darse el caso que en ocasiones se visiten la misma cantidad de veces los nodos disponibles, entonces entre ellos se selecciona el del máximo valor.

2.2 Estructura de la IA propuesta

A continuación, se explica con elementos más técnicos el funcionamiento de la IA haciendo uso del algoritmo MCTS, así como una conceptualización de los elementos que este debe poseer para su correcto funcionamiento (Nasarre Embid, 2012).

Una partida se representa como un árbol, donde cada nodo corresponde a un estado particular. El nodo raíz representa la posición de inicio de partida. Los hijos de cada nodo son estados alcanzables en un movimiento legal.

Cada nodo i del árbol MCTS representa un estado alcanzado en una partida, este contendrá la siguiente información:

- *wins*: es el valor actual de la posición, representa el número de partidas ganadas desde el nodo.
- *visitedTimes*: es el contador de visitas que ha tenido ese nodo.
- C_i es la información referente a la partida.

Cada estado alcanzado de la partida contendrá la siguiente información:

- *fieldTokens*: listado de las fichas jugadas hasta el momento. El primer elemento de esta lista corresponde a la última ficha colocada en la izquierda, y el último elemento de la lista corresponde al último elemento colocada por la derecha.
- *fichasOponente*: entero que contiene la cantidad de fichas en posesión del oponente.
- *hand*: listado de las fichas disponibles que posee la IA para realizar la jugada en ese turno.
- *jugadasLegales*: listado de jugadas legales que puede realizar la IA en un momento determinado.

Para el correcto funcionamiento del algoritmo y obtener la siguiente jugada se debe proporcionar a la IA en cada turno el estado actual de la partida, así como la cantidad de fichas del oponente. En la primera iteración que se parte de la situación inicial de la partida, se construye el nodo raíz a partir del estado actual. Al finalizar el tiempo de simulación, se escogerá el movimiento más prometedor teniendo en cuenta la información resultante.

2.3 Fase 1 Planificación

Esta actividad comienza escuchando al cliente permitiendo que los miembros técnicos del equipo XP entiendan el negocio para el software y comprendan las

características principales y funcionalidades que se requieren. Como resultado se llega a la creación de algunas historias, conocidas también como Historias de Usuario (HU), que describen la salida necesaria, características y funcionalidad del software que se va a elaborar (Pressman, 2010).

2.3.1 Historias de Usuarios

Las HU son escritas por el cliente y se colocan en una tarjeta indizada. El cliente le asigna un valor o prioridad a la historia. Luego el equipo XP evalúa cada una de las historias y le asignan un costo, medido en semanas de desarrollo. Si alguna historia se estima que requiere más de tres semanas, se le pide al cliente que la divida en historias más chicas. En cualquier momento se pueden escribir historias nuevas (Pressman, 2010).

A continuación, se describen las HU definidas para desarrollar la solución:

Tabla 1 HU 1:

Historia de Usuario	
Número: 1	**Usuario**: Usuario
Nombre: Realizar jugadas cercanas a la óptima	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 2	**Iteración**: 1
Descripción: Los NPC deben ser capaces de incrementar la dificultad de la partida al efectuar jugadas legales que tengan impacto a largo plazo.	

Tabla 2. HU 2:

Historia de Usuario	
Número: 2	**Usuario**: Usuario
Nombre: Aplicar estrategia de salida: Salir con la más acompañada	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 0.5	**Iteración**: 1
Descripción: La IA debe poder emplear la estrategia Salir con la más acompañada. De jugar la ficha que más se le repita en la mano.	

Tabla 3.HU 3:

Historia de Usuario	
Número: 3	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Estrategia Básica	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 2
Descripción: La IA debe poder emplear la estrategia básica. Consiste en hacer la primera jugada legal que encuentre, sin importar que sea doble o no.	

Tabla 4. HU 4:

Historia de Usuario	
Número: 4	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar la ficha más acompañada	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 1	**Iteración**: 2
Descripción: La IA debe poder emplear la estrategia: Tirar la ficha más acompañada. Usa el mismo principio de Salir con las más acompañada. Para ello la IA analiza las fichas que tiene en la mano y juega de ellas la que más se repita. Esto se puede usar en combinación con otra estrategia, como Pasar al oponente.	

Tabla 5. HU 5.

Historia de Usuario	
Número: 5	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar la ficha de mayor valor	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 2
Descripción: La IA juega la ficha de mayor valor, sin importar el resto, siempre que sea una jugada legal.	

Tabla 6. HU 6:

Historia de Usuario	
Número: 6	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar el mayor doble	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 3
Descripción: La IA debe jugar los dobles sin importar el resto de las fichas, y en caso de tener varios disponibles, elegir el mayor de ellos.	

Tabla 7. HU 7:

Historia de Usuario	
Número: 7	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Pasar al oponente	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 3
Descripción: La IA, en combinación con otras técnicas o no, debe tener la capacidad de realizar jugadas que provoquen que el jugador no pueda colocar fichas en su turno.	

Tabla 8. HU 8:

Historia de Usuario	
Número: 8	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Cerrar el juego	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 3
Descripción: La IA debe, cuando cuente con fichas de un valor bajo, realizar jugadas que lleven a cerrar el juego, causando que en ocasiones gane la partida.	

Estimación de esfuerzo por Historia de Usuario

En este momento el equipo de desarrollo realiza una estimación del esfuerzo que requerirá la implementación de cada HU para brindar al cliente un valor aproximado de la duración del proyecto, a continuación, se muestra la estimación realizada:

Tabla 9. Estimación de esfuerzo por Historia de Usuario

	Historias de Usuario	**Puntos estimados**
1	Realizar jugadas cercanas a la óptima	2
2	Aplicar estrategia de salida: Salir con la más acompañada	0.5
3	Aplicar estrategia de juego: Estrategia Básica	0.5
4	Aplicar estrategia de juego: Tirar la ficha más acompañada	1
5	Aplicar estrategia de juego: Tirar la ficha de mayor valor	1
6	Aplicar estrategia de juego: Tirar el mayor doble	0.5
7	Aplicar estrategia de juego: Pasar al oponente	0.5
8	Aplicar estrategia de juego: Cerrar el juego	1
	Total	7

Desarrollo del plan de iteraciones

Luego de definidas las HU y estimado el esfuerzo para cada una, y de comprobar cómo se mencionaba anteriormente que su duración no excediera las 3 semanas, el equipo de desarrollo agrupa las historias por iteraciones para su implementación: quedando de la siguiente forma:

Tabla 10. Plan de duración de las iteraciones

Iteración	**Historia de Usuario**	**Duración**
1	Realizar jugadas cercanas a la óptima	2,5
	Aplicar estrategia de salida: Salir con la más acompañada	
2	Aplicar estrategia de juego: Estrategia Básica	2,5
	Aplicar estrategia de juego: Tirar la ficha más acompañada	
	Aplicar estrategia de juego: Tirar la ficha de mayor valor	
3	Aplicar estrategia de juego: Tirar el mayor doble	2

	Aplicar estrategia de juego: Pasar al oponente	
	Aplicar estrategia de juego: Cerrar el juego	
Total		7

Planificación de entrega

A continuación, se presenta el plan de entrega, que plantea una propuesta de las versiones del sistema resultantes de cada iteración:

Tabla 11. Planificación de entrega

Entregable	Iteración 1	Iteración 2	Iteración 3
Versiones	Versión 0.1	Versión 0.2	Versión 0.3
% de entrega	30 %	60%	100%
Fechas	20/05/22	1/05/22	17/06/22

2.4 Fase 2 Diseño

Esta fase XP sigue el principio Mantenlo Sencillo. El diseño guía la implementación de una historia conforme se escribe: nada más y nada menos. XP estimula el uso de las tarjetas Clase, Responsabilidad y Colaborador (CRC) como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos (Pressman, 2010).

2.4.1 Tarjetas CRC

Como su nombre lo indica son tarjetas que relacionan una clase con sus responsabilidades y los colaboradores que ayudan a realizar dichas actividades. Las tarjetas CRC son el único producto del trabajo del diseño que se genera como parte del proceso XP y su utilidad radica en que facilita al equipo XP entender de manera fácil y concisa. A continuación, se describen las tarjetas CRC definidas para la implementación de la solución:

Tabla 12. Tarjeta CRC # 1

Tarjeta CRC	
Clase: AlgoritmoMCTS	
Responsabilidad	**Colaboración**
• Simular partidas para encontrar jugadas óptimas.	• NodoMCTS

• Utilizar árbol resultante de estas simulaciones.	

Tabla 13. Tarjeta CRC # 2

Tarjeta CRC	
Clase: NodoMCTS	
Responsabilidad	**Colaboración**
• Se encarga de almacenar información de un estado de la partida necesaria para el funcionamiento de la IA.	

2.5 Arquitectura del software

Como arquitectura del sistema se propone una basada en componentes. Esta arquitectura plantea que los componentes son independientes y son unidades de composición fundamentales de un sistema. Uno de sus principios fundamentales es la reusabilidad (Pressman, 2010).

A continuación, se describe el componente de inteligencia artificial que se propone:

Componente de IA: El componente proporciona la interfaz *IAlgoritmoMcts*, para el demo implementado, esta interfaz se comunica con la clase *Player,* que es el encargado de jugar una ficha. Se tuvo en cuenta para su diseño, otro de los fundamentos básicos de los componentes, y es, que debe existir una clara separación entre la interfaz de los componentes y su implementación para que una implementación de un componente pueda reemplazarse por otro sin cambiar el sistema.

2.6 Patrones de diseño

Los patrones de diseño son soluciones habituales a problemas comunes en el diseño del software. Se encargan de identificar: clases, instancias, roles, colaboraciones y distribución de responsabilidades. Son descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto (Gamma et al., 1994).

Para la propuesta de solución se emplearon los Patrones generales de asignación de responsabilidades (GRASP, por sus siglas en inglés) que describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones, y los patrones definidos en “Patrones de Diseño: Elementos de software orientado a objetos reutilizable”, conocido generalmente como Gang of Four Book (GOF, por sus siglas en inglés). A continuación, se mencionan algunos ejemplos de patrones usados:

Patrones GRASP usados:

- **Experto:** la clase que cuenta con la información necesaria para cumplir la responsabilidad. Un ejemplo de su utilización es la clase *GameManager*, encargada de todo el flujo de una partida, desde repartir las fichas a los jugadores hasta gestionar sus turnos.
- **Creador:** una instancia de un objeto tiene que ser creada por el objeto que tiene la información para ello. Se evidencia con la clase *GameManager,* que actúa como creadora.
- **Controlador:** es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. La clase *GameManager* también actúa como controlador pues hace de intermediaria para el manejo en eventos
- **Alta cohesión:** se aplica en la mayoría de las clases, pues en cada una se implementan solo las funcionalidades que le corresponden

Patrones GOF utilizados:

Según el libro GOF existen tres tipos de patrones, creación, estructurales y comportamiento. A continuación, mencionamos algunos ejemplos de su uso en la propuesta de solución.

- **Factory Method (método de fabricación):** define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clases instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

- **Prototype (prototipo):** crea nuevos objetos clonándolos de una instancia ya existente.
- **Singleton (instancia única):** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
- **Proxy (Apoderado):** proporciona un sustituto o representante de otro objeto para controlar el acceso a este.

Conclusiones del capítulo

Con la realización del presente capítulo quedaron definidas las HU que guiarán el proceso de desarrollo, se describió la propuesta de solución, la cual queda concretada al darle cumplimiento a las dos primeras fases que plantea XP. Se define un plan de iteraciones y un plan de entrega que servirán de guía en la fase de implementación. Queda definida la arquitectura a emplear, que aprovechará las características de Unity.

Capítulo 3 Implementación y pruebas realizadas al componente de IA

En este capítulo queda evidenciado el proceso de implementación de la solución, así como los estándares utilizados en la codificación de la misma. También se muestran los resultados de las pruebas de validación de la propuesta solución para verificar su correcto funcionamiento. Quedando así, completadas las dos siguientes fases propuestas en la metodología seleccionada.

3.1 Estándares de codificación

El objetivo de los estándares de codificación de software es inculcar prácticas de programación probadas que conduzcan a un código seguro, confiable, comprobable y mantenible. Además de crear una apariencia coherente en el código, de esta forma los lectores se centran en el contenido y no en el diseño del mismo. Permite una mejor comprensión del código (BillWagner, 2022). Estas convenciones de código también evitan a la larga, costosos procesos de revisión y solución de errores en el código, a continuación, se mencionan las convenciones adoptadas en el desarrollo de la propuesta de solución:

Convenciones de nomenclatura

- Se usa Pascal case: al nombrar clases, ejemplo: “PascalCasing”.
- Al asignar un nombre a interface, se usa la grafía Pascal además de agregar el prefijo I al nombre. Esto indica claramente a los consumidores que es un elemento interface.
- Se usa CamelCase al nombrar campos internos o privados y al nombrar parámetros de métodos, ejemplo: “camelCasing”.

Convenciones de diseño

- Escribir solo una instrucción por línea.
- Escribir solo una declaración por línea.
- Usar tabulación (cuatro espacios) para identación.
- Agregue al menos una línea en blanco entre las definiciones de método y las de propiedad.
- Utilizar paréntesis para que las cláusulas de una expresión sean evidentes.

Convenciones de comentarios

- Comenzar el texto del comentario con una letra mayúscula.
- Finalizar el texto del comentario con un punto.
- Inserte un espacio entre el delimitador de comentario (//) y el texto del comentario.

3.2 Videojuego Pixel Dominoes

Se realizó la implementación de un videojuego de dominó, en su variante Dominó Doble Nueve, llamado Pixel Dominoes, el cual cuenta con las funcionalidades necesarias para probar el módulo de IA.

Ilustración 2. Pantalla de inicio.

Una vez dentro del juego se tiene la posibilidad de reiniciar la partida en todo momento, así como la posibilidad de salir del juego completamente, el juego es desarrollado por dos jugadores, los cuales harán uso de la IA una vez implementada.

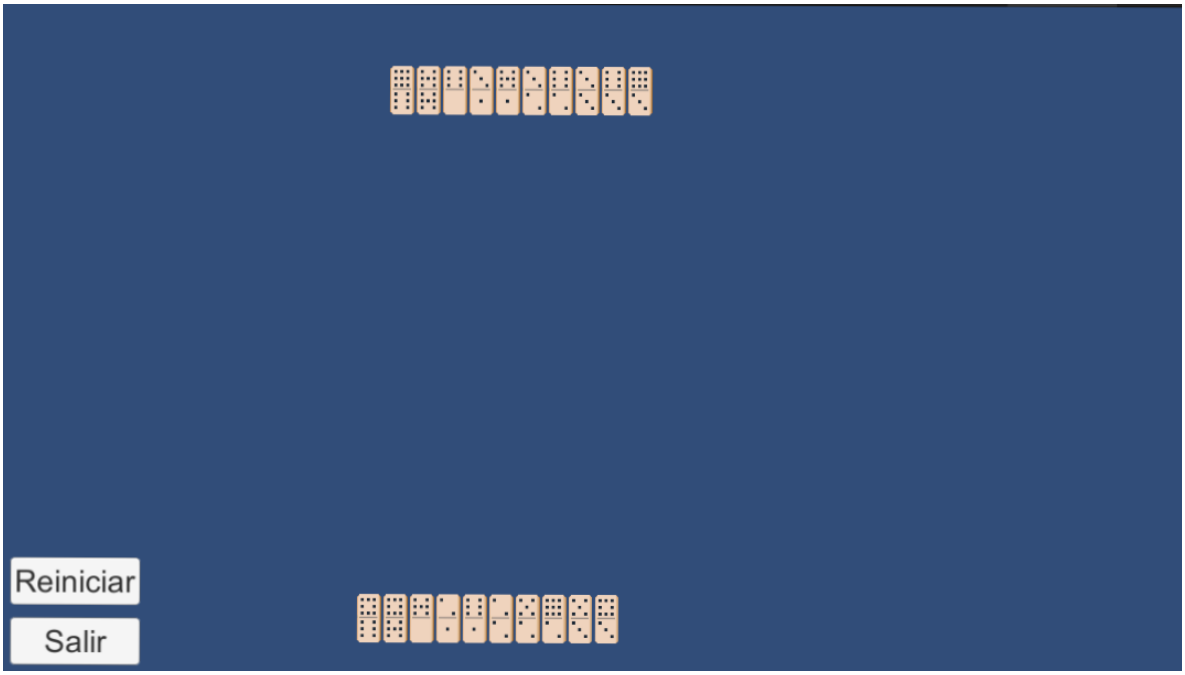


Ilustración 3. Desarrollo de una partida.

Mecanismos:

En el juego se implementan una serie de mecanismos, los cuales empleará la IA una vez implementada:

Mecanismo Iniciar Partida:

- Propiedades: El jugador 1 es al que le corresponde salir.
- Comportamiento: Debido a que es un videojuego que sirve de base para acoplar la IA siempre va a salir el primer jugador.

Mecanismo Colocar Ficha:

- Propiedades: Siempre que existan movimientos legales y el jugador lleve, podrá colocar la ficha en el tablero.
- Comportamientos: Se reparten las fichas. Sale el jugador 1. El usuario que está en turno selecciona la ficha que lleva en el tablero, en caso de seleccionar una ficha que no lleva se pasa el turno al otro jugador. El usuario gana la partida al pegarse o al virarse con la menor cantidad de puntos en las fichas.

3.3 Fase 3 Codificación

Luego de terminadas las dos fases anteriores, el equipo no procede inmediatamente a la codificación, sino que desarrolla una serie de pruebas unitarias a cada una de las historias que se van a incluir en la entrega en curso. Al hacer esto al desarrollador le queda más claro en qué debe enfocarse para pasar dicha prueba, se mantiene lo más simple posible no agregando nada extraño a la solución. A medida que los programadores terminan su trabajo, el código desarrollado se integra con el trabajo de los demás. En algunos casos esto lo realiza a diario un equipo de integración, en otros son los mismos programadores los responsables. Esta estrategia de integración continua ayuda a evitar problemas de compatibilidad y ayuda a descubrir a tiempo los errores (Pressman, 2010).

3.3.1 Tareas de implementación Iteración 1

A continuación, se definen para cada HU y por cada iteración, tareas de implementación asociadas a cada una de estas.

Tabla 14. Estimaciones por HU en la Iteración 1.

HU	Historia de usuario	Duración
1	Realizar jugadas cercanas a la óptima	2
2	Aplicar estrategia de salida: Salir con las más acompañada	0.5

Tabla 15. Tarea de implementación 1.

Tarea de implementación	
Número de tarea: 1	**Número de Historia de Usuario:** 1
Nombre de la tarea: Realizar jugadas cercanas a la óptima	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 2
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementa el algoritmo MCTS para la obtención de jugadas óptimas o cercanas en un tiempo aceptable.	

Tabla 16. Tarea de implementación 2.

Tarea de implementación

Número de tarea: 2	**Número de Historia de Usuario:** 2
Nombre de la tarea: Aplicar estrategia de salida: Salir con la más acompañada.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementan los mecanismos necesarios para que el jugador controlado por la PC salga con la ficha más acompañada que tenga en el momento del inicio de la partida.	

3.3.2 Tareas de implementación Iteración 2

Tabla 17. Estimaciones por HU en la Iteración 2.

HU	Historia de usuario	Duración
1	Aplicar estrategia de juego: Estrategia Básica	0.5
2	Aplicar estrategia de juego: Tirar la ficha más acompañada	1
3	Aplicar estrategia de juego: Tirar la ficha de mayor valor.	1

Tabla 18. Tarea de implementación 3.

Tarea de implementación	
Número de tarea: 3	**Número de Historia de Usuario:** 3
Nombre de la tarea: Implementar el mecanismo para aplicar la estrategia básica durante el juego	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementa los mecanismos necesarios para que el jugador controlado por la PC aplique la estrategia básica a seguir durante el desarrollo de una partida.	

Tabla 19. Tarea de implementación 4.

Tarea de implementación	
Número de tarea: 4	**Número de Historia de Usuario:** 4
Nombre de la tarea: Implementar los mecanismos necesarios para aplicar la estrategia de juego: Tirar la ficha más acompañada.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Responsable: Alexander Rodríguez Saavedra	

Descripción: Se implementan los mecanismos necesarios para que el jugador controlado por la PC juegue la ficha más acompañada que tenga en la mano.

Tabla 20 Tarea de implementación 5

Tarea de implementación	
Número de tarea: 5	**Número de Historia de Usuario:** 5
Nombre de la tarea: Implementar mecanismos necesarios para aplicar la estrategia de juego: Tirar la ficha de mayor valor.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementa los mecanismos necesarios para que el jugador controlado por la PC juegue la ficha de mayor valor.	

3.3.3 Tareas de implementación Iteración 3

Tabla 21. Estimaciones por HU en la Iteración 3.

HU	Historia de usuario	Duración
1	Aplicar estrategia de juego: Tirar el mayor doble	0.5
2	Aplicar estrategia de juego: Pasar al oponente	0.5
3	Aplicar estrategia de juego: Cerrar el juego	1

Tabla 22. Tarea de implementación 6.

Tarea de implementación	
Número de tarea: 6	**Número de Historia de Usuario:** 6
Nombre de la tarea: Implementar mecanismos necesarios para aplicar la estrategia de juego: Tirar el mayor doble.	
Tipo de tarea: Desarrollo	**Puntos Estimados: 0.5**
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementa los mecanismos necesarios para que el jugador controlado por la PC juegue los dobles de mayor valor primero.	

Tabla 23. Tarea de implementación 7.

Tarea de implementación	
Número de tarea: 7	**Número de Historia de Usuario:** 7

Nombre de la tarea: Implementar mecanismos necesarios para aplicar estrategia de juego: Pasar al oponente.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementan los mecanismos necesarios para que el jugador controlado por la PC, durante el desarrollo del juego, haciendo uso de las otras estrategias, logre pasar al contrario.	

Tabla 24. Tarea de implementación 8.

Tarea de implementación	
Número de tarea: 8	**Número de Historia de Usuario:** 8
Nombre de la tarea: Implementar mecanismos necesarios para aplicar la estrategia de cierre de juego.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Responsable: Alexander Rodríguez Saavedra	
Descripción: Se implementan los mecanismos necesarios para que el jugador controlado por la PC realice acciones favorables para ganar la partida.	

Luego de definidas las tareas de implementación se establecen un conjunto de pruebas para comprobar la calidad de la propuesta desarrollada.

3.4 Fase 4 Pruebas

Esta fase es fundamental en la metodología XP, se divide en Unitarias y de Aceptación. Están diseñadas para garantizar el correcto funcionamiento del código implementado y así su calidad, además de evitar el surgimiento de nuevos errores al realizar alguna modificación. También se realizan pruebas para analizar los resultados arrojados por el algoritmo y validar el correcto funcionamiento del componente implementado.

3.4.1 Validación del componente

Para conocer el comportamiento de la IA se desarrollaron una serie de partidas por dos jugadores con conocimientos básicos sobre dominó, variando la profundidad con que se exploraba el árbol.

A continuación, se muestra la tabla que recoge la relación entre la profundidad y el tiempo de respuesta:

Tabla 25.

Profundidad	Simulaciones	Tiempo (s)
3	115	0
4	676	0,023
5	3066	0,6
6	12654	2,54
7	20945	5,46
8	38358	7,88

Para los 3 primeros niveles de profundidad con 115 simulaciones realizadas por cada turno del ordenador los tiempos de respuesta son casi nulos, lo mismo sucede con el nivel 4 y 5, donde se simulan 676 y 3066 partidas respectivamente, demorando hasta 0.6 (s), tiempo casi imperceptible para el usuario. Ya en el nivel 6 se realizan más de 12500 simulaciones, un número considerablemente mayor, tardando unos 2.5 (s) y en ocasiones hasta 3 o 4 (s), un valor aún aceptable, especialmente en un juego como el dominó donde lejos de incomodar al usuario lo que le crearía la sensación de que el ordenador está pensando. Sin embargo, en el nivel 7 de profundidad ya comienza a ser más prolongado el tiempo de respuesta pues simula más de 20000 juegos, llegando a ser molesto para el usuario esperar durante 5.4 (s) o más, de igual forma sucede con el nivel 8 de profundidad, donde se generan más de 35000 simulaciones y el tiempo de espera es de más de 7.5 (s).

Tabla 26.

Profundidad	Partidas Realizadas	Partidas Ganadas
3	50	14
4	50	12
5	50	20
6	50	23
7	50	23
8	50	24

En la tabla se muestra cómo a medida que aumenta la profundidad con la que se explora el árbol aumenta el número de partidas ganadas, algo lógico si se tiene en cuenta que el ordenador basa su selección en simular miles de situaciones posibles

en el juego, sin embargo, llegado un punto, aunque se explore el árbol con más profundidad, el promedio de partidas ganadas es similar. Basado en el análisis del tiempo que demora el algoritmo y en estos resultados se decide explorar el árbol de búsqueda solo hasta el nivel 6.

3.4.2 Pruebas de aceptación

Las pruebas de aceptación también llamadas pruebas del cliente, son especificadas por el cliente y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Estas pruebas son pruebas de sistema de caja negra y se derivan de las historias de los usuarios que se han implementado como parte de la liberación del software (Pressman, 2010). A continuación, se muestra una representación de las pruebas de aceptación a realizar en cada iteración.

Tabla 27. Prueba de Aceptación 1.

Caso de prueba de aceptación
Historia de Usuario: 1
Nombre: Realizar jugadas cercanas a la óptima.
Descripción: Comprueba que el jugador controlado por la PC selecciona jugadas prometedoras.
Condiciones de ejecución: Partida desarrollada entre el ordenador y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Tabla 28. Prueba de Aceptación 2.

Caso de prueba de aceptación
Historia de Usuario: 3
Nombre: Aplicar estrategia de juego: Estrategia básica.
Descripción: Comprueba que el jugador controlado por la PC realiza jugadas legales y básicas durante el desarrollo de la partida.
Condiciones de ejecución: Partida desarrollada entre el ordenador y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad

Resultado: Satisfactorio

Tabla 29. Prueba de Aceptación 3.

Caso de prueba de aceptación
Historia de Usuario: 5
Nombre: Aplicar estrategia de juego: Tirar la ficha de mayor valor.
Descripción: Comprueba que el jugador controlado por la PC selecciona las fichas de mayor valor.
Condiciones de ejecución: Partida desarrollada entre el ordenador y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Tabla 30. Prueba de Aceptación 4

Caso de prueba de aceptación
Historia de Usuario: 7
Nombre: Aplicar estrategia de juego: Cerrar el juego.
Descripción: Comprueba que el jugador controlado por la PC selecciona una jugada que impida que la partida continúe.
Condiciones de ejecución: Partida desarrollada entre el ordenador y el jugador humano. Se da la situación de que el algoritmo bloquea la partida con un movimiento.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

3.4.3 Análisis de los resultados de las pruebas de aceptación

Se realizaron las pruebas propuestas de forma organizada por cada iteración definida. Se obtuvieron resultados satisfactorios en las pruebas, comprobando el correcto funcionamiento del sistema.

Conclusiones del capítulo

En este capítulo se definen los estándares de codificación usados, también se definió el proceso de implementación de la solución, desglosado en tareas de implementación. Además, quedan definidas las pruebas de aceptación que

permitieron encontrar fallas e inconformidades en el sistema y corregirlas. Luego de realizado el análisis del tiempo de respuesta y las partidas ganadas por el componente variando la profundidad con que se exploraba el árbol se decide solo explorarlo hasta el nivel 6.

Conclusiones

El empleo de las herramientas, metodologías y técnicas seleccionadas, permitieron el desarrollo de un componente de Inteligencia Artificial capaz de realizar partidas de dominó contra un usuario.

El estudio de las técnicas de IA aplicada a videojuegos permitió seleccionar e implementar el algoritmo MCTS para darle solución a la problemática del presente trabajo, desarrollando el componente mencionado con anterioridad, el cual tuvo un buen desempeño en el juego.

Se demostró que, para lograr un mayor grado de inteligencia y efectividad, la profundidad con que se explore el árbol debe ser la mayor posible. Además, resulta de utilidad asociar estas simulaciones con una heurística pues evita tener que crear completamente el árbol de juego y buscar en todas las soluciones posibles por lo tanto se acerca a la mejor jugada con más rapidez.

Recomendaciones

Incorporar la solución a videojuegos del centro Vertex.

Implementar el comportamiento del algoritmo para juegos en pareja.

Referencias bibliográficas

Álvarez Alba, D. (2021, febrero). *Stockfish 13 ya disponible para descargar*. https://www.peonelectrico.com/post/stockfish-13-ya-disponible-para-descargar

Asana. (2020). *¿Qué es la metodología Kanban y cómo funciona? • Asana*. Asana. https://asana.com/es/resources/what-is-kanban

BillWagner. (2022). *Convenciones de código de C#*. https://docs.microsoft.com/es-es/dotnet/csharp/fundamentals/coding-style/coding-conventions

Camacho, D. (2021, septiembre). *Qué es GitHub y cómo usarlo para aprovechar sus beneficios*. Platzi. https://platzi.com/blog/que-es-github-como-funciona/

Chaslot, M. J.-B. (2010). *Monte-Carlo Tree Search*.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Patrones de Diseño. Elementos de software orientado a objetos reutilizable.*

Gaudio, N. (2021). *¿Qué es la INTELIGENCIA ARTIFICIAL?* https://www.ceupe.cl/blog/que-es-la-inteligencia-artificial.html

Hmong. (s. f.). *Inteligencia artificial en videojuegos Descripción generalyHistoria*. Recuperado 15 de octubre de 2022, de https://hmong.es/wiki/Artificial_intelligence_(video_games)

Hmong. (2021a). *C Sharp (lenguaje de programación)*. https://hmong.es/wiki/C_Sharp_language

Hmong. (2021b). *Unidad (motor de juego) HistoriayDescripción general*. https://hmong.es/wiki/Unity_(game_engine)

IAT. (2020, marzo 19). ▷ Inteligencia artificial en videojuegos: Origen y evolución. *IAT*. https://iat.es/tecnologias/inteligencia-artificial/videojuegos/

IBM. (2020, julio). *IBM Cloud Education*. https://www.ibm.com/cloud/learn/machine-learning

Jetbrains. (s. f.). *Rider. Editor C# multiplataforma para Unity*. JetBrains: Developer Tools for Professionals and Teams. Recuperado 5 de mayo de 2022, de https://www.jetbrains.com/lp/dotnet-unity

La Vanguardia, E. (2018, diciembre). *La inteligencia artificial ya no necesita a las personas para aprender*. La Vanguardia. https://www.lavanguardia.com/ciencia/20181206/453396865826/inteligencia-artificial-alphazero-deepmind-juegos-mesa-ajedrez-go-shogi.html

Labena, D. S. (2020). *Deep Learning en videojuegos*. 39.

Lanza Digital. (2020, diciembre 9). *La fascinante historia del Dominó—Lanza Digital—Lanza Digital*. https://www.lanzadigital.com/general/la-fascinante-historia-del-domino/

Moreno Montiel, B., Moreno Montiel, C. H., Alfaro Pérez, J., Domínguez Sánchez, G. F., & MacKinney Romero, R. (2020). *Simulación de Monte Carlo para el juego de dominó*.

Nasarre Embid, B. (2012). *Método de Monte-Carlo Tree Search (MCTS) para resolver problemas de alta complejidad: Jugador virtual para el juego del Go*.

Pressman, R. S. (2010). *Ingenieria del Software. Un Enfoque Practico*. 810.

Radoff, J. (2010). *History of Social Games*.

Rebour, M. (s. f.). *¿Qué es un videojuego? | Plan Ceibal – Formación*. Recuperado 3 de mayo de 2022, de https://blogs.ceibal.edu.uy/formacion/faqs/que-es-un-videojuego/

Riley, G. (1989). *Sistemas Expertos Principios y Programación 3ra Edicion*.

Sagredo-Olivenza, I., Gomez-Martın, M. A., & Gonzalez-Calero, P. A. (2014). *Un modelo integrador de m´aquinas de estados y ´arboles de comportamiento para videojuegos*. 12.

Santander Universidades. (2020, diciembre). *Metodologías de desarrollo de software: ¿qué son?* https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

Sinnaps. (2020). *Metodología XP o Programación Extrema: ¿Qué es y cómo aplicarla?* Gestor de proyectos online. https://www.sinnaps.com/blog-gestion-proyectos/metodologia-xp

Sistemas, F. R. I. en I. de, & Telefónica, T. S. en E. D. y M. en S. de las T. I. de seguridad informática en el equipo de I. L. C. de. (2020, agosto 19). *Breve historia de la IA en los videojuegos—Think Big Empresas*. Think Big. https://empresas.blogthinkbig.com/breve-historia-de-la-ia-en-los-videojuegos/

Bibliografía

«Monte Carlo Tree Search - About». Accedido 7 de septiembre de 2022. https://www.cs.swarthmore.edu/~mitchell/classes/cs63/f20/reading/mcts.html.

GeeksforGeeks. «ML | Monte Carlo Tree Search (MCTS)», 14 de enero de 2019. «Minimax and Monte Carlo Tree Search - Philipp Muens». Accedido 7 de septiembre de 2022. https://philippmuens.com/minimax-and-mcts.

https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/.

«Monte Carlo Tree Search». En *Wikipedia, La Enciclopedia Libre*, 6 de septiembre de 2022. https://en.wikipedia.org/w/index.php?title=Monte_Carlo_tree_search&oldid=1108779094.

Daniel Whitehouse. «MCTS for Games with Hidden Information and Uncertainty». PhD, University of York, 2014.

«El papel de la inteligencia artificial en los videojuegos». Accedido 5 de octubre de 2022. https://www.pontegeek.com/revista/agosto2021/inteligencia-artificial-videojuegos/.

rascalselfies. «Training an AI to play Dominoes». Reddit Post. *r/MLQuestions*, 15 de agosto de 2018. www.reddit.com/r/MLQuestions/comments/97m47l/training_an_ai_to_play_dominoes/.

«Método de Monte Carlo - programador clic». Accedido 1 de noviembre de 2022. https://programmerclick.com/article/7245488668/.

MortenGR. «Monte Carlo Tree Search: Implementation for Tic-Tac-Toe». Forum post. *Stack Overflow*, 22 de mayo de 2014. https://stackoverflow.com/q/23803186.

Anexos

Historias de Usuario:

Tabla 31 HU 1:

Historia de Usuario	
Número: 1	**Usuario**: Usuario
Nombre: Realizar jugadas cercanas a la óptima	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 2	**Iteración**: 1
Descripción: Los NPC deben ser capaces de incrementar la dificultad de la partida al efectuar jugadas legales que tengan impacto a largo plazo.	

Tabla 32. HU 2:

Historia de Usuario	
Número: 2	**Usuario**: Usuario
Nombre: Aplicar estrategia de salida: Salir con la más acompañada	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 0.5	**Iteración**: 1
Descripción: La IA debe poder emplear la estrategia Salir con la más acompañada. De jugar la ficha que más se le repita en la mano.	

Tabla 33.HU 3:

Historia de Usuario	
Número: 3	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Estrategia Básica	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 2
Descripción: La IA debe poder emplear la estrategia básica. Consiste en hacer la primera jugada legal que encuentre, sin importar que sea doble o no.	

Tabla 34. HU 4:

Historia de Usuario	
Número: 4	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar la ficha más acompañada	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 1	**Iteración**: 2
Descripción: La IA debe poder emplear la estrategia: Tirar la ficha más acompañada. Usa el mismo principio de Salir con las más acompañada. Para ello la IA analiza las fichas que tiene en la mano y juega de ellas la que más se repita. Esto se puede usar en combinación con otra estrategia, como Pasar al oponente.	

Tabla 35. HU 5.

Historia de Usuario	
Número: 5	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar la ficha de mayor valor	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 2
Descripción: La IA juega la ficha de mayor valor, sin importar el resto, siempre que sea una jugada legal.	

Tabla 36. HU 6:

Historia de Usuario	
Número: 6	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Tirar el mayor doble	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 3
Descripción: La IA debe jugar los dobles sin importar el resto de las fichas, y en caso de tener varios disponibles, elegir el mayor de ellos.	

Tabla 37. HU 7:

Historia de Usuario	
Número: 7	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Pasar al oponente	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración**: 3
Descripción: La IA, en combinación con otras técnicas o no, debe tener la capacidad de realizar jugas que provoquen que el jugador no pueda colocar fichas en su turno.	

Tabla 38. HU 8:

Historia de Usuario	
Número: 8	**Usuario**: Usuario
Nombre: Aplicar estrategia de juego: Cerrar el juego	**Programador:** Alexander Rodríguez Saavedra
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 3
Descripción: La IA debe, cuando cuente con fichas de un valor bajo, realizar jugadas que lleven a cerrar el juego, causando que en ocasiones gane la partida.	

Pruebas de Aceptación:

Caso de Prueba de Aceptación 1

Caso de prueba de aceptación
Historia de Usuario: 1
Nombre: Realizar jugadas cercanas a la óptima.
Descripción: Comprueba que el jugador controlado por la PC selecciona jugadas prometedoras.
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Caso de Prueba de Aceptación 2

Caso de prueba de aceptación
Historia de Usuario: 2
Nombre: Aplicar estrategia de salida: Salir con la más acompañada.
Descripción: Comprueba que el jugador controlado por la PC selecciona la ficha de la cual posea más en la mano al inicio de la partida
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano. La IA juega de primero.
Pasos de ejecución: • Inicia la partida • Permitir a la IA realizar el primer movimiento
Resultado: Satisfactorio

Caso de Prueba de Aceptación 3

Caso de prueba de aceptación
Historia de Usuario: 3
Nombre: Aplicar estrategia de juego: Estrategia básica.
Descripción: Comprueba que el jugador controlado por la PC realiza jugadas legales y básicas durante el desarrollo de la partida.
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Caso de Prueba de Aceptación 4

Caso de prueba de aceptación
Historia de Usuario: 4
Nombre: Aplicar estrategia de juego: Tirar la ficha más acompañada.
Descripción: Comprueba que el jugador controlado por la PC selecciona la ficha de la cual posea más en la mano.
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Caso de Prueba de Aceptación 5

Caso de prueba de aceptación

Historia de Usuario: 5
Nombre: Aplicar estrategia de juego: Tirar la ficha de mayor valor.
Descripción: Comprueba que el jugador controlado por la PC selecciona las fichas de mayor valor.
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Caso de Prueba de Aceptación 6

Caso de prueba de aceptación
Historia de Usuario: 6
Nombre: Aplicar estrategia de juego: Pasar al oponente.
Descripción: Comprueba que el jugador controlado por la PC selecciona jugadas que hacen que el otro jugador no puede jugar en ese turno
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio

Caso de Prueba de Aceptación 7

Caso de prueba de aceptación
Historia de Usuario: 7
Nombre: Aplicar estrategia de juego: Cerrar el juego.
Descripción: Comprueba que el jugador controlado por la PC selecciona una jugada que impide que la partida continúe.
Condiciones de ejecución: Partida desarrollada entre la IA y el jugador humano. Se da la situación de que la IA bloquea la partida con un movimiento.
Pasos de ejecución: • Se desarrolla la partida con normalidad
Resultado: Satisfactorio