UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y programación de bots de inteligencia artificial para juegos de gestión de recursos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Almenar Vicente, Borja

Tutor/a: Sánchez Anguix, Víctor

Cotutor/a: Alberola Oltra, Juan Miguel

CURSO ACADÉMICO: 2023/2024

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño y programación de bots de inteligencia artificial para juegos de gestión de recursos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Borja Almenar Vicente

Tutores: Víctor Sanchez Anguix, Juan Miguel Alberola Oltra

2023-2024

Resumen

En este proyecto, se ha utilizado el proyecto de final de grado “PyCatan” realizado por Adrián Heras como base y marco teórico para el diseño y elaboración de sistemas no controlados o agentes inteligentes en un juego de gestión de recursos complejo. Después de un análisis sobre las distintas metodologías y usos de cada modelo heurístico de toma de decisiones, se ha decidido optar por crear un agente inteligente o agente basado en una máquina de estados finitos (FSM), y un Bot con un sistema basado en reglas. Además, se ha generado también un marco para desarrollar un sistema capaz de aprender cómo jugar de manera más eficiente a través de partidas anteriores.

En este documento, se expondrán las distintas metodologías de inteligencia artificial a tener en cuenta en el desarrollo de los agentes, se diseñarán estrategias de juego que se puedan implementar en código, así como que sean óptimas para ganar partidas y se analizará cómo cada estrategia se puede aplicar al juego de mesa Colonos del Catán. Además, una vez desarrollados los agentes, se han analizado los datos de miles de partidas con el fin de establecer unas conclusiones que puedan servir a la hora del desarrollo de nuevas estrategias dentro de esta herramienta.

Palabras clave: Inteligencia Artificial, juego de mesa, Catan, simulación, Python

Abstract

In this project, the final degree project "PyCatan" by Adrián Heras has been used as the basis and theoretical framework for the design and development of uncontrolled systems or intelligent bots in a complex resource management game. After analyzing various methodologies and the use of each heuristic decision-making model, it was decided to create a bot or intelligent agent based on a Finite State Machine (FSM) and a Bot with a rule-based system. Additionally, a framework has been generated to develop a system capable of learning how to play more efficiently through previous games.

In this document, different artificial intelligence methodologies to be considered in the development of the bots will be presented, game strategies that can be implemented in code and are optimal for winning games will be designed, and how each strategy can be applied to the board game Settlers of Catan will be analyzed. Furthermore, once the bots were developed, data from thousands of games were analyzed to draw conclusions that could be useful in the development of new strategies within this tool.

Keywords : Artificial intelligence, boardgame, Catan, simulation, Python

Tabla de contenidos

Listado de figuras

Listado de tablas

1. Introducción

En los últimos años, la inteligencia artificial (IA) ha experimentado un crecimiento exponencial en términos de desarrollo e implementación. Este fenómeno ha sido impulsado por avances significativos en el aprendizaje automático, la capacidad de procesamiento y la disponibilidad de grandes cantidades de datos. La IA está revolucionando múltiples sectores, desde la medicina hasta las finanzas, y se está convirtiendo en un componente esencial en la infraestructura tecnológica moderna.

Las aplicaciones de la IA son diversas y abarcan una amplia gama de industrias. En la medicina, por ejemplo, los algoritmos de IA están mejorando los diagnósticos y personalizando tratamientos (Galdames, I. S., 2023). Otro ejemplo es la industria financiera, en la cual la IA está optimizando las estrategias de inversión y detectando fraudes (Cao, L., 2020).

El ámbito de los videojuegos es uno de los sectores donde la IA está teniendo un impacto particularmente notable. Los videojuegos, desde su creación, han buscado ofrecer experiencias más inmersivas y realistas. La IA, con su capacidad para aprender y adaptarse, está llevando esta búsqueda a nuevos horizontes, revolucionando la forma en que los juegos se desarrollan, se juegan y se experimentan.

Por otro lado, un gran aspecto de los videojuegos radica en las pruebas previas al lanzamiento. Antes, se utilizaban a lo que se denominaban “beta-testers”, o simplemente “testers”, personas que se encargaban de probar de forma exhaustiva todos los aspectos del videojuego con el fin de encontrar vulnerabilidades o errores, así como de evaluar la experiencia general jugando. Sin embargo, existen determinadas pruebas que pueden llevar horas a una persona de realizar, así como que estos “testers” no abarquen a todos los estilos de juegos que pueden adoptar los jugadores. Es por ello que, recientemente, se está utilizando la IA en materia de pruebas en videojuegos, con el fin de conseguir pruebas más extensas y eficientes hasta ahora nunca vistas (Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., & Lucas, S. M., 2019).

En el caso del presente trabajo, se va a utilizar el entorno de pruebas controlado desarrollado por Adrián Heras, PyCatan (2023). Se trata de un entorno de desarrollo de agentes inteligentes, capaces de realizar partidas a demanda del juego de mesa Catan. Con ello, es posible implementar multitud de estrategias a la hora de programar estos agentes, con el fin de probar cuál se adapta mejor al juego y consigue unos mejores resultados.

Utilizando el entorno de PyCatan, se van a desarrollar una serie de agentes basados en distintas estrategias de juego, con el fin de comprobar que tipos de estrategias son mejores o peores, así como la viabilidad del desarrollo de distintas metodologías basadas en heurística dentro de la plataforma.

Este trabajo se enmarca en un Proyecto de Innovación y Mejora Educativa que persigue mejorar las competencias del alumnado de Grado y Máster relacionadas con la IA, haciendo hincapié en el desarrollo de sus habilidades prácticas y técnicas, así como incrementar su motivación para su aprendizaje, mediante la utilización de esta plataforma de simulación. En este sentido, este proyecto sirve como base para que otros alumnes utilicen los agentes desarrollados para competir entre ellos y también de base para desarrollar los suyos propios.

1.1 Objetivos

El objetivo de este trabajo final de grado es desarrollar una serie de agentes basados en inteligencia artificial sobre la aplicación PyCatan desarrollada por Adrián Heras en su proyecto del año 2023. Estos agentes serán utilizados en distintas asignaturas universitarias a forma de ejemplo de las distintas técnicas de inteligencia artificial que existen. En concreto, se centrará en la parte de la gestión de recursos, eludiendo toda la parte de negociación. Con esto, se obtendrá un entorno más controlado, basado en tomar las mejores decisiones dependiendo del estado de la partida y del azar de los dados.

Con ello, se obtienen una serie de subobjetivos, necesarios para cumplimentar el objetivo principal del proyecto. Estos son:

- Analizar las carencias actuales de la plataforma PyCatan.
- Desarrollar nuevas funcionalidades que suplan las limitaciones actuales de la plataforma.
- Desarrollar agentes basados en heurística determinista.
- Desarrollar estrategia avanzada.
- Realizar un análisis de datos con resultados de un número elevado de partidas.

1.2 Etapas de realización

Para lograr alcanzar los objetivos definidos anteriormente, a continuación, se detallan los pasos que se han seguido:

1- Análisis y estudio del PyCatan, así como del juego de mesa Catán. Con esto, se pretenden extraer las necesidades que presenta el simulador PyCatan, y las posibles estrategias a seguir en el juego de mesa Catán.

2- Investigación sobre distintos métodos de IA válidos para la implementación de agentes inteligentes en juegos de mesa como Catan. Con esta información, se podrán plantear distintas estrategias a la hora del desarrollo de los agentes.

3- Desarrollo de los agentes en el entorno PyCatan con las técnicas de IA seleccionadas. Con los puntos anteriores, se completará el desarrollo de los bots.

4- Análisis de resultados, comparando todos los agentes utilizados. Utilizando distintos escenarios, se realizará un análisis de partidas para comprobar la efectividad de los métodos y estrategias aplicadas.

1.3 Estructura del documento

El presente documento se encuentra dividido en 5 apartados principales:

- Apartado 2 (Marco teórico): En él, se establecerán las bases teóricas con las cuales se trabajarán a la hora de desarrollar los agentes.
- Apartado 3 (Propuesta): Se realiza la elección de que agentes se van a desarrollar y con qué metodologías.
- Apartado 4 (Desarrollo): Se explican los desarrollos realizados en los distintos ámbitos del proyecto.
- Apartado 5 (Resultados): Se realiza un análisis de las partidas jugadas por los agentes desarrollados.
- Apartado 6 (Conclusiones): Una vez terminado el proyecto, se evalúa si se ha cumplido el objetivo y los subobjetivos propuestos.

2. Marco teórico

A continuación, se expondrán las distintas investigaciones llevadas a cabo para evaluar los métodos de IA a implementar, así como el análisis del Juego de mesa Catán y la plataforma PyCatan. En base a estos análisis, vendrá condicionada la etapa de desarrollo de los agentes.

2.1 Inteligencia artificial

La inteligencia artificial (IA, o AI por sus siglas en inglés) es un campo de la informática que estudia y desarrolla sistemas autónomos capaces de realizar tareas que requieren la inteligencia humana. Estas tareas pueden incluir el aprendizaje, el razonamiento, la resolución de problemas, la compresión del lenguaje natural o la interacción con el entorno. En los últimos años, ha sido un campo que ha crecido exponencialmente, debido principalmente a los trabajos en las IAs generativas, así como las que son capaces de comprender el lenguaje humano en todos sus sentidos. Empresas como OpenAI, Google entre otras, han contribuido a una carrera por el programa más útil basado en inteligencia artificial.

No se puede hablar de inteligencia artificial sin mencionar a Alan Turing. Conocido como uno de los padres de la computación moderna, Alan Turing aportó la gran mayoría de semillas que acabarían germinando en todo lo que conocemos hoy en día como inteligencia artificial (S. Muggleton, 2014). Fue el primero en idear una máquina programable, la máquina universal de Turing. Todos sus estudios son los que han dado pie al estado actual de este campo.

En el ámbito que ocupa este proyecto, la inteligencia artificial adaptada a juegos de mesa tiene un amplio recorrido, ya que el juego de mesa que más se ha utilizado en este ámbito ha sido el ajedrez. Este clásico juego ha sido objeto de innumerables estudios y artículos, basándose en encontrar los mejores movimientos para una posición concreta. Ya desde 1950, Claude Shannon, uno de los padres de la teoría de la información escribía un artículo titulado *Programming a computer for playing ches (*C. E. Shannon, 1950*)*, dónde establecía los dos conceptos básicos a la hora de desarrollar algoritmos para la búsqueda de jugadas: La profundidad y la importancia de una buena función de evaluación.

Sin embargo, no todos los juegos de mesa son tan deterministas como el ajedrez en cuanto a decisiones y estados disponibles. En el caso del Catán, existen muchos más

elementos que escapan al jugador, como puede ser la tirada de un dado o qué carta robaran. Es por ello por lo que se utilizarán métodos basados en Inteligencia artificial más complejos a nivel de decisiones y estados.

2.3.1 Inteligencia artificial basada en la heurística

La definición más clásica del término “heurístico” radica en el conjunto de métodos y técnicas para encontrar soluciones a un problema. En informática, se puede definir mejor como aquellos algoritmos o metodologías capaces de, dado un estado inicial y un objetivo final, consiga encontrar una solución óptima al problema, sin asegurar que sea la mejor.

A este respecto, en el presente trabajo se enfocarán dichos métodos en la toma de decisiones estratégicas, y a la búsqueda de tácticas dentro del propio juego para ganar con una alta probabilidad.

Dentro de estos algoritmos, existen distintas metodologías a la hora de desarrollarlos y aplicarlos. Para poder establecer que estrategia o algoritmo es el más apropiado para un juego de mesa como el catán, se debe estudiar su comportamiento y aplicaciones. Dentro de la heurística, los algoritmos más interesantes para estas aplicaciones son:

Algoritmo A*: El algoritmo A* es un algoritmo de búsqueda de rutas, en concreto, es capaz de encontrar la ruta más corta entre el nodo inicial y final sobre un grafo, ya sea con o sin costes en los caminos. Dentro de un contexto similar al que se está utilizando en este trabajo, en los videojuegos, se suele utilizar en los personajes no jugables (NPCs), para que estos encuentren dentro de un mundo abierto el camino más corto entre dos puntos A y B. Además, se puede utilizar en juegos de mesa cuyo objetivo sea llegar más rápido a la meta, tratando de optimizar el camino utilizado. (Candra, A., Budiman, M. A., & Pohan, R. I.,2021)

Sistemas basados en reglas: Estos algoritmos se basan en reglas lógicas para tomar decisiones en base a un estado inicial. Estas reglas son del estilo: “si pasa A entonces B, si no C”.

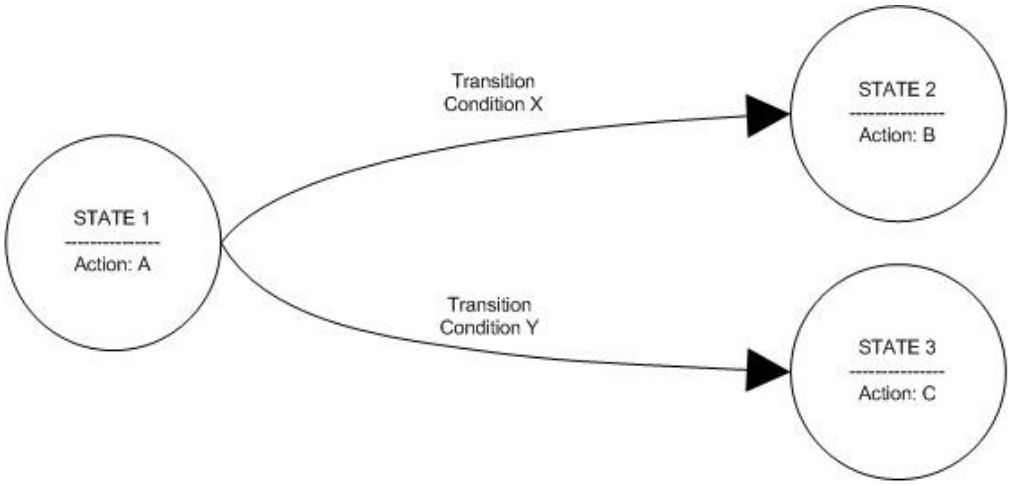

Figura 1: Esquema básico de una FSM

Máquinas de estados finitos FSM (Figura 1): Se trata de un modelo de computación basado en un número finito de estados, y lo que más cobra relevancia de este son las transiciones entre estos estados. Cada estado almacena los disparadores que permiten llegar a otros estados. Con esto, se consiguen agentes que se comportan de manera distinta dependiendo de en qué estado se encuentren. (Adeniyi, A. E., Brahma, B., Adebiyi, M. O., Awotunde, J. B., Jimoh, R. G., Olasinde, E., & Bandyopadhyay, A., 2024)

Árboles de decisiones: Funcionan de manera similar a las FSM, sin embargo, los nodos de este tipo de sistemas se comportan de una manera jerárquica, teniendo en cuenta sucesiones de varios nodos en adelante. Este tipo de algoritmos permiten mucha profundidad en juegos donde las acciones que se realizan tienen relevancia en las acciones posteriores.

Algoritmos Minimax: Este tipo de algoritmos se basan en juegos con adversario, asumiendo un escenario y decisiones perfectos de ambos jugadores. Se suele utilizar en juegos como el ajedrez o el tres en raya, y se basa en esperar que el oponente siempre vaya a hacer la mejor jugada.

Dentro de los algoritmos o métodos que se han explicado, se pueden extraer las siguientes conclusiones:

- El algoritmo A*, en el caso del Catán, no es un algoritmo interesante, ya que, en este juego, no se realizan movimientos con las piezas sobre el tablero. Por este motivo, no se utilizará en el desarrollo de nuevos agentes.
- Los sistemas basados en reglas para el uso en juegos de mesa, es un buen candidato para formar parte del proyecto, dado que permite mucha complejidad dentro de sus reglas sin una complicación elevada.

- Las máquinas de estados finitos o FSM son muy útiles en sistemas como el presentado en este trabajo, ya que permite reducir el juego a estados y transiciones entre los mismos, haciendo más sencilla la toma de decisiones y los costes de computación a la hora de evaluar las decisiones a tomar.
- Los sistemas basados en árboles de decisiones, en juegos de mesa como el catán, son interesantes, ya que algunas decisiones tomadas con anterioridad pueden llevar consecuencias asociadas a las mismas.
- Los algoritmos minimax son perfectos para juegos deterministas, ya que ofrecen soluciones óptimas. Sin embargo, como se basa en juegos sin nada de azar, no es una buena opción para el catán, ya que la tirada de dados iniciales depende del azar.

Vistos los algoritmos que se han tenido en cuenta para el desarrollo de los agentes, los más interesantes y que se tendrán en cuenta son:

- Máquinas de estados finitos
- Sistemas basados en reglas
- Árboles de decisiones

2.3.2 Inteligencia artificial avanzada adaptada a juegos

La algorítmica avanzada basada en IA radica en métodos no estáticos capaces de evolucionar sus rutas hacia la solución óptima. Estos algoritmos son un paso más allá a los ya explicados anteriormente, ya que sus resultados no tienen por qué ser los esperados, y en etapas jóvenes de evolución no tienen por qué ser los óptimos. Existen una gran variedad de estrategias a la hora de desarrollar este tipo de metodologías. En el mundo de los videojuegos, que es el que más se acerca al ámbito de este proyecto, se han hecho muchos avances en los últimos años.

En los videojuegos existen muchos aspectos que, al tratarlos con métodos relacionados con la inteligencia artificial, se vuelven mucho más sencillos y eficientes (Sharma, S., & Sharma, V). Algunos de estos aspectos son los siguiente:

- **Personajes No Jugables** (NPCs) y Comportamiento Realista: Los personajes no jugables (NPCs) en los videojuegos se están volviendo más inteligentes y realistas gracias a la IA. Anteriormente, los NPCs seguían patrones de comportamiento predefinidos, lo que limitaba su capacidad de interacción. Con la IA, estos personajes pueden adaptarse a las acciones del jugador, aprender de su comportamiento y ofrecer respuestas más realistas y variadas. Esto se traduce en experiencias de juego más dinámicas y desafiantes.

- **Generación Procedural de Contenidos**: La generación procedural, asistida por IA, permite la creación de mundos de juego vastos y únicos sin la necesidad de diseñar cada elemento manualmente. Algoritmos de IA pueden generar paisajes, niveles y escenarios que son coherentes y estéticamente agradables, ofreciendo a los jugadores una experiencia fresca cada vez que juegan.
- **Personalización y Adaptación de la Experiencia de Juego**: La IA puede analizar el estilo de juego de un usuario y adaptar la dificultad y los desafíos del juego en tiempo real. Esto asegura que el juego sea siempre atractivo y accesible, manteniendo un equilibrio adecuado entre desafío y entretenimiento.

Con esto, el aspecto más avanzado del proyecto se va a centrar en generar un sistema capaz de aprender de la experiencia en anteriores partidas y adaptar su jugabilidad dependiendo de sus resultados.

2.2 Juego de mesa Catan

El juego de mesa Catan[1] se considera uno de los pilares de los juegos de mesa modernos. En él, cada jugador debe intentar convertirse en el mejor comerciante de la isla, combinando su habilidad para comerciar, así como gestionar los recursos que se van obteniendo durante la partida. Gana el primero en llegar a 10 puntos de victoria.

Figura 2: Tablero del juego de mesa Catán

El juego se desarrolla sobre un tablero hexagonal, dividido en 19 hexágonos más pequeños, como se puede observar en la figura 2. Al comienzo de la partida, se dispone el tablero siguiendo una configuración aleatoria en los números encima de cada hexágono. Estos, determinan que losetas producen recursos en cada ronda dependiendo de la tirada de dados inicial de la misma. Con esto, se decide el jugador inicial mediante una tirada de dados. Previo a ninguna acción del juego, cada jugador coloca una ciudad y una carretera en un nodo no costero en orden de turno. Una vez llegado al último jugador, comenzando por él, se vuelve a hacer lo mismo, pero en orden inverso, quedando al final 2 ciudades y carreteras de cada jugador en el tablero, y con ello, comenzaría la partida. El desarrollo del turno de cada jugador es el siguiente:

[1] https://es.wikipedia.org/wiki/Los_colonos_de_Cat%C3%A1n

Primero, el jugador debe tirar los dos dados. Su suma determinará, con las tiradas entre 2 y 12 excluyendo el 7, que casillas producirán materiales esa ronda. En caso de que el resultado no sea un 7, los jugadores que tengan un poblado o ciudad en un nodo que sea adyacente a una loseta que haya producido ese material, obtendrán un material de ese tipo por cada pueblo o ciudad adyacente. En caso de que salga el 7, está relacionado con el ladrón. Cuando salga este número, si algún jugador tiene más de 8 cartas en mano, debe descartarse de la mitad, redondeando hacia arriba, y posteriormente, el jugador activo mueve al ladrón un espacio. El ladrón es una ficha que se encuentra en el tablero, y cuando un jugador la mueve a otra loseta, ese jugador puede robarle a un jugador que tenga un poblado o ciudad en la loseta en la que acaba el ladrón un material de su mano.

Después de lanzar los dados, comenzará una fase de negociación. En este momento, cada jugador puede elegir si quiere negociar esa ronda o no. En el caso de que quiera, los demás jugadores pueden proponerle ofertas de cualquier cantidad de materiales por los suyos. A partir de ahí, cada jugador es libre de aceptar o rechazar cualquiera de estas ofertas. Si algún jugador lo desea, es posible hacer una negociación marítima, sin necesidad de intercambiar materiales con otro jugador. Esto se puede realizar siempre en un ratio de 4:1, es decir, devolviendo 4 materiales iguales a su montón para intercambiarlo por el que el jugador elija. Si el jugador posee algún poblado o ciudad en un hexágono con puerto, puede beneficiarse de negociaciones más atractivas, como puede ser 3:1 incluso 2:1. Una vez finalizada esta fase, se pasa a la fase de construcción.

En esta fase, cada jugador puede construir acorde a los materiales que tengan en ese momento. No hay límite en cuanto a construcciones por turno. En esta fase, es posible construir:

- Carreteras: Son los caminos que recorren los lados de los hexágonos, que permiten expandir terreno, además de ser uno de los objetivos del final de partida. Para construirla, es obligatorio que esté conectada con otra carretera del mismo jugador o con un poblado/ciudad de este.
- Poblados: Los poblados se colocan en los vértices de los hexágonos. Para poder construirlos, es necesario que estén conectados por una carretera, y al menos a dos casillas de distancia del siguiente poblado. Estos otorgan un punto al final de la partida.

- Ciudades: Las ciudades funcionan como mejora del poblado, ya que su funcionamiento es el mismo, pero otorgan 2 puntos de victoria al final de la partida.
- Cartas de desarrollo: Durante esta fase, también es posible construir cartas de desarrollo. Si se hace está acción, se toma la primera carta del mazo de desarrollo, y se mantiene en secreto al resto de jugadores. En cualquier turno que no sea el mismo en el que se construyó, se puede hacer uso de la carta, realizando su efecto y descartándola en caso de que no se trate de un ejercito o puntos de victoria, en cuyo caso se dejará delante del jugador como recordatorio. Las cartas disponibles son las siguientes:
    - Caballeros: Permiten mover al ladrón y robar un material a un jugador con un poblado/ciudad en la casilla.
    - Monopolio: El jugador que la juega dice un material. El resto de los jugadores deben darle todas las cartas de recurso de ese material a ese jugador.
    - Año de cosecha: Permite coger dos cartas de cualquier material de la banca.
    - Construcción de carreteras: Permite la construcción gratuita de dos carreteras en posiciones legales.
    - Puntos de victoria: Estas cartas deben mantenerse en secreto hasta que el jugador esté seguro de que al desvelarlas puede llegar a los 10 puntos para ganar la partida.

Además de estas cartas, existen dos cartas que están visibles a todos los jugadores, ya que se tratan de objetivos de final de partida. Estas son, la carretera más larga, y el ejército más grande.

- Carretera más larga: Una vez un jugador consiga un camino de 5 de distancia, sin ser interrumpido por ningún poblado o ciudad de otro jugador, obtendrá la carta “Carretera más larga”, que le otorgará 2 Puntos de victoria. Sin embargo, en cualquier momento, si otro jugador consigue una carretera de mayor distancia, podrá quitarle esta carta.
- Mayor ejercito: Cuando un jugador posea 3 cartas de caballero jugadas, obtendrá la carta “Mayor ejército”, que le otorgará 2 puntos al final de la partida. Al igual que la de carretera más larga, si otro jugador supera al primero, le quitará esta carta.

El jugador que consiga primero 10 Puntos de victoria será el vencedor.

2.3 Entornos de simulación en materia de IA

El uso de entornos de simulación para pruebas de programación de inteligencias artificiales (IA) en el contexto universitario es una práctica cada vez más relevante y efectiva en la formación de estudiantes en áreas como la informática, la ingeniería y la ciencia de datos. Estos entornos proporcionan un espacio controlado y seguro donde los alumnos pueden desarrollar, experimentar y evaluar algoritmos de IA sin las limitaciones o riesgos asociados a aplicaciones en el mundo real.

Los entornos de simulación permiten a los estudiantes interactuar con escenarios complejos que imitan el comportamiento de sistemas reales, como la conducción autónoma, la robótica, o la gestión de recursos en redes inteligentes. Al utilizar simulaciones, los estudiantes pueden probar y depurar algoritmos en condiciones controladas, evitando las consecuencias adversas de errores en entornos reales, además de observar y analizar de manera detallada el comportamiento de los sistemas de IA bajo diferentes condiciones y parámetros, facilitando una comprensión más profunda de los algoritmos y modelos.

Existen distintas plataformas que ofrecen un entorno de simulación para estrategias con IA, como NetLogo[2], Repast[3], CoppeliaSim[4] o PyGomas[5] que proporcionan una experiencia de aprendizaje interactiva y atractiva, donde los estudiantes pueden aprender haciendo, y aplicar sus conocimientos para resolver problemas del mundo real.

En el caso de este proyecto, se utilizará la plataforma PyCatan, ya que, con ello, se terminará de probar la herramienta desarrollada por Adrián en su trabajo de fin de grado, y se terminará con una versión más estable y de adaptación sencilla a las clases de las distintas asignaturas que quieran explicar de forma más prácticas conocimientos y técnicas relacionadas con IA.

[2] https://ccl.northwestern.edu/netlogo/
[3] https://repast.github.io/
[4] https://www.coppeliarobotics.com/
[5] https://github.com/javipalanca/pygomas

2.4 PyCatan

Para la realización de este TFG, se ha hecho indispensable un entorno para el desarrollo y prueba de los agentes a desarrollar. Gracias al trabajo de fin de grado de Adrián Heras del año 2023 ha sido posible.

PyCatan, que fue desarrollado por Heras (2023), es una plataformaque permite simular partidas del juego de mesa Catán en un entorno que es capaz de enfrentar a agentes en partidas automáticas. Esto permite comparar las distintas metodologías de IA aplicadas entre ellas, puesto que cada jugador puede basarse en un comportamiento distinto.

La aplicación tiene dos componentes principales:

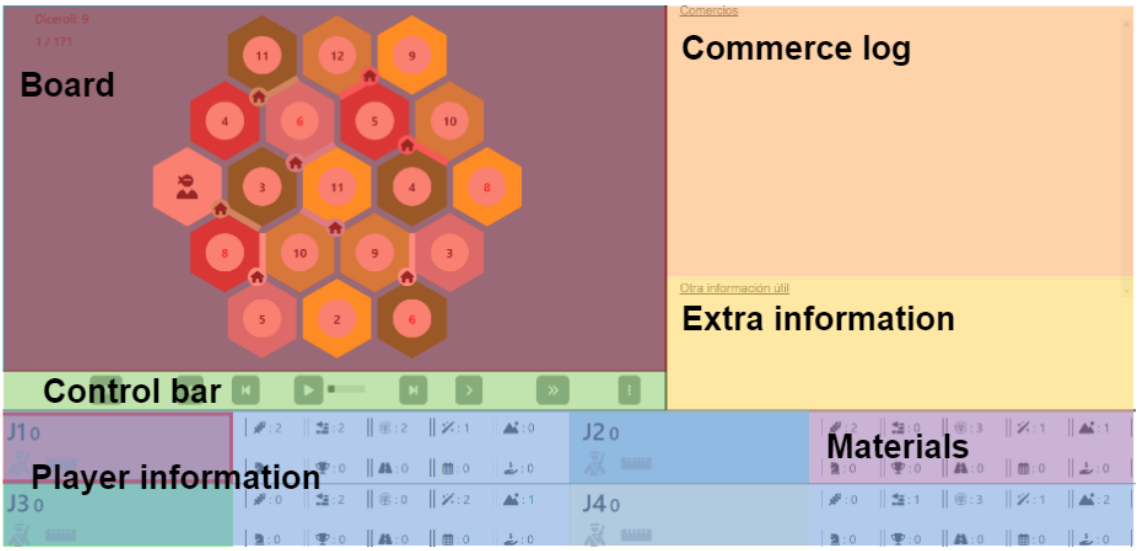

Figura 3: Visualizador del PyCatan

- El visualizador: Se trata del frontend de la aplicación. Como se puede ver en la Figura 3, su función es permitir al usuario visualizar y examinar la partida de una manera cómoda, ya que se dispone de toda la información relevante en una misma pantalla. En él, se pueden cargar partidas generadas por el entorno que simula las partidas, y avanzar o retroceder el tiempo en la partida, siendo capaces de ver las distintas decisiones que toman los jugadores en cada momento de la partida. Además de esto, también se obtiene información de los materiales que posee cada jugador, así como una representación gráfica del tablero.

- Backend: Es el encargado de simular las partidas. Contiene toda la lógica de la partida, y mediante clases que se denominan “masters”, van orquestando la partida, dejando que los agentes decidan en los momentos en los que les es posible hacerlo.

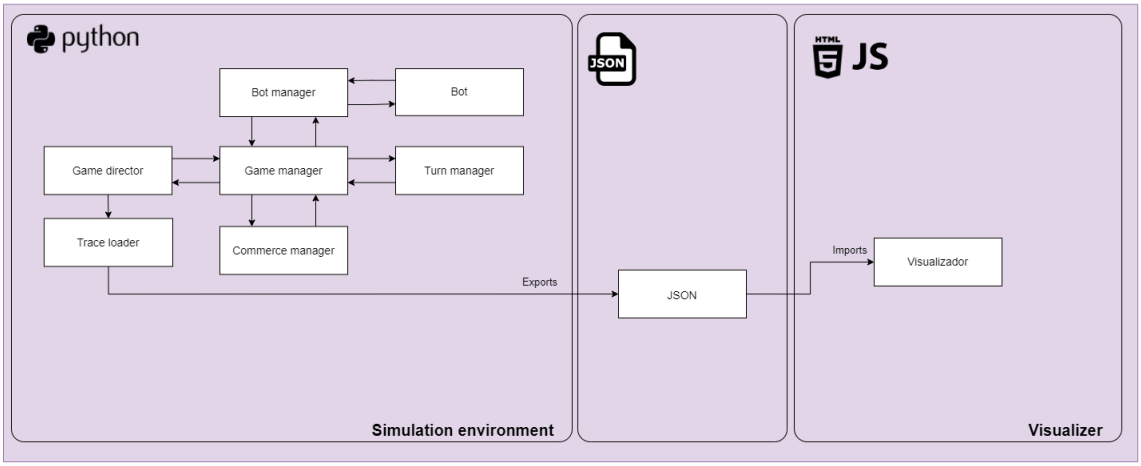

Figura 4: Estructura aplicación PyCatan

Como se puede observar en la Figura 4, en el apartado del entorno de simulación, existen una serie de clases clave para el funcionamiento de la aplicación. En el caso del presente TFG, las clases más relevantes a la hora del desarrollo de los agentes incluyen:

- La clase **Bot**: Es la que contiene toda la lógica a realizar en cada fase dentro de la partida. El GameManager pide una decisión al Bot, el cual dependiendo de su lógica realiza de una manera o de otra.
- La clase **Bot Manager**: Se encarga de guardar y administrar los Agentes que hay en la partida
- **GameManager**: Es el core de la aplicación. Se trata de un director de orquesta, que va dando paso a los distintos actores dentro de la partida dependiendo de la fase y turno en el que se encuentren.

2.4.1 Interfaz BOT

La interfaz bot está diseñada para el desarrollo de los distintos agentes que se pueden implementar en la aplicación. En ella, existen métodos para la toma de decisiones en cada punto de la partida, además de variables para ver el estado del tablero y mano del jugador.

Las funciones relevantes dentro de esta interfaz se hayan en el manual del desarrollador del PyCatan, y son las siguientes:

- **on_game_start**: Como bien se ha explicado anteriormente, al comienzo de la partida se deben de colocar dos poblados y carreteras anexas a los mismos. Con esta clase, el bot recibe la información de donde hay poblados y carreteras colocadas en el momento en el que tiene que decidir, y devuelve en que nodo coloca el poblado y hacia donde se dirige la carretera que coloca.
- **on_turn_start**: El GameManager da la opción de jugar alguna carta de desarrollo al comienzo del turno.
- **on_trade_offer**: Se llama cuando existe un intercambio propuesto por otro jugador. Puede aceptarse, rechazarse, o realizar una contraoferta.
- **on_build_phase**: Función que se llama cuando comienza la fase de construcción. Es la función que permite construir cualquier construcción, así como cartas de desarrollo. Obtiene como valores de entrada la instancia del tablero, para saber los nodos disponibles para construir. Como valores de salida, puede devolver none si elige no construir, DevelopmentCard si elige carta de desarrollo, o la construcción que quiere construir y el nodo donde quiere hacerlo. Si se trata de una carretera, también se especifica hacia que nodo va dicha carretera.
- **on_turn_end**: Similar al on_turn_start, el GameManager da la opción de jugar una carta de desarrollo.
- **on_commerce_phase**: El GameManager permite al bot lanzar una oferta a otro jugador o a la banca. También se pueden jugar cartas de desarrollo.

- **on_moving_thief**: Cuando el jugador saca un 7, permite mover al ladrón y a que jugador con poblado/ciudad adyacente robar un material.

- **on_having_more_than_7_materials**: Función que es llamada cuando sale un 7 en el dado y el bot tiene más de 7 cartas en mano. El GameManager devuelve una instancia de la mano una vez descartadas la mitad de las cartas.

- Además, existen una serie de funciones que son llamadas cuando se juega una carta de desarrollo.

    - **on_road_building_card_use**: Tiene como valores de entrada que dos carreteras se desean construir
    - **on_monopoly_card_use**: El bot elige de que material quiere tener monopolio.

- **on_year_of_plenty_use**: Elige dos materiales, y se obtiene un material de cada (puede ser el mismo)

3. Propuesta

Una vez realizado el trabajo de análisis de la herramienta y de las distintas metodologías que se pueden aplicar a la hora de desarrollar algoritmos y programas no controlados, se va a realizar una primera aproximación para ver cómo afrontar el proyecto.

Como se ha explicado en el apartado anterior, las metodologías heurísticas que van a emplearse para elaborar las clases que van a definir el comportamiento de los agentes son la máquina de estados finitos y un sistema de decisiones basado en reglas. Estos presentan la suficiente profundidad para poder elaborar lógicas suficientemente avanzadas para la toma de decisiones en un juego de mesa de gestión de recursos como es el catán. Antes de comenzar el desarrollo de los agentes, se han detectado algunas limitaciones debidas al alcance en el proyecto del PyCatan, que deberán abordarse antes de comenzar el desarrollo de los agentes.

3.1 Metodología

A continuación, se van a definir los pasos que se han seguido para la elaboración de la propuesta final, los cuales han sido claves para el resultado final del mismo. Estos, se podrían definir en tres grandes bloques.

Primero, el análisis de la herramienta y los agentes ya desarrollados sirve para entender cómo funciona la herramienta, encontrar carencias o requisitos necesarios para conseguir alguno de los objetivos, así como entender cómo fueron programados los agentes anteriores, ya que esto será clave para diseñar y programar los agentes de este proyecto. Con este paso, se obtendrán una lista de requisitos previos al desarrollo de los agentes.

Después, se aplicarán los cambios pertinentes en la herramienta debido a las limitaciones, así como el diseño y desarrollo de los nuevos agentes. Así, se podrán probar los cambios y comprobar si se ha facilitado la tarea del desarrollo de los agentes.

Por último, será necesario analizar como juegan los agentes desarrollados, para evaluar si es posible analizar la efectividad de los agentes frente a otros agentes inteligentes, o contra uno que tome decisiones de manera aleatoria.

3.2 Análisis PyCatan y sus Agentes

Previo al desarrollo del código para el desarrollo de los nuevos agentes, se ha realizado un trabajo de enfoque sobre las distintas ideas, así como un estudio del programa PyCatan en profundidad.

Siguiendo la metodología elegida, lo primero que se realizó fue un análisis de los agentes ya implementados por Adrián. Cuando se comenzó con este proyecto, existían dos: el RandomBot y el AdrianHerasBot.

El RandomBot, como su propio nombre indica, es un agente puramente aleatorio, que escoge si construir o no con una probabilidad definida, y lo hace en un a posición elegida de manera completamente aleatoria. Este agente servirá como base de comparación o elemento de control en las pruebas que se realizarán en etapas posteriores en este proyecto.

El AdrianHerasBot ha sido de gran ayuda durante la realización de este proyecto. Con unas reglas muy sencillas, Adrián consiguió un agente que derrotaba al RandomBot en más de un 60% de las ocasiones. El agente de Adrián ha servido de base para el desarrollo de los agentes de este proyecto, pues que, aunque sus reglas sean sencillas, utiliza todas las variables que hay que tener en cuenta a la hora de tomar decisiones.

Para poder analizar los resultados, Adrián utilizó unas sencillas tablas para comprobar que su Bot ganaba un porcentaje alto de ocasiones. Sin embargo, la idea de este proyecto es generar una gran cantidad de partidas, que serán difícilmente analizables sin alguna herramienta externa. Es por ello por lo que, se ha decidido realizar un nuevo desarrollo en Python, que permita, recibiendo el json de salida de las partidas, establecer el porcentaje de victorias, así como otras métricas de interés a analizar.

En el caso del PyCatan, presentaba carencias no contempladas en el alcance inicial. Durante la tarea de análisis, se enumeraron para su posterior validación y desarrollo. Estas eran:

- JSON de salida demasiado complejo para un análisis exhaustivo tras muchas partidas
- Una vez seleccionados los agentes de cada jugador y el número de partidas, estas se ejecutaban sin modificaciones en las estrategias. En el alcance inicial del PyCatan, esta función se implementó para probar los agentes uno por uno. Sin embargo, para este proyecto, se desean implementar agentes que se

adapten dependiendo de las partidas anteriores, así que habrá que plantear un nuevo sistema que se adapte a esta necesidad.

- No existe límite en el número de carreteras de cada jugador, lo que podía dar pie a que un jugador pueda pasarse la partida construyendo carreteras, sin dejar avanzar al resto de oponentes.
- No existe un límite en el número de rondas a jugar. En la implementación inicial con el comercio incluido no era un problema, ya que los agentes intercambiaban los materiales que necesitaban. Sin embargo, con esta nueva versión sin comercio es necesario establecer un límite de rondas antes de que la partida dure de manera indeterminada hasta que salgan los números correctos en los dados.

3.3 Requisitos

Siempre que se realiza un desarrollo, es necesario establecer una base de requisitos. En este caso, existen una serie de requisitos que permitirán desarrollar los agentes y que sirvan de utilidad como ejemplo para nuevos desarrolladores, así como que establezcan una buena base para comparar distintos tipos de estrategia en el juego de mesa Catán.

- Corregir los errores de la herramienta dado el nuevo alcance y las nuevas necesidades
- Desarrollar un modo avanzado, que permita evaluar estrategias más avanzadas.
- Desarrollar agentes que basados heurística, como FSM, sistemas basados en reglas, o árboles de decisiones..
- Comprobar que estos agentes pueden vencer al agente aleatorio con un porcentaje superior al 70%.
- Desarrollar un modo de jugadores adaptativos, los cuales puedan buscar la mejor estrategia, y que elijan estrategias desarrolladas antes que la aleatoria en más de un 80% de las ocasiones.
- Integrar una funcionalidad en la herramienta que permita un correcto análisis de resultados masivos.
- Integrar una funcionalidad en la herramienta que permita jugar una gran cantidad de partidas y almacenarlas.

3.4 Alcance

En este proyecto, debido a la complejidad de elaborar desarrollos de heurística avanzada, y la profundidad que hay dentro de las negociaciones, se ha decidido dejar fuera del alcance del proyecto el aspecto de negociación entre jugadores. Con ello, toma mucha más relevancia dónde comienza cada jugador, y hacia dónde se expande, ya que los materiales que posee dependerán de los nodos de terreno que tiene adyacentes a sus ciudades, las tiradas de los dados, y de las cartas de desarrollo que pueden llevar a que reciba materiales (año de cosecha y monopolio).

Este aspecto está siendo desarrollado en otro proyecto paralelo, y no se cierra la posibilidad de que se puedan unir las dos metodologías para crear una IA capaz de jugar de forma óptima al juego.

3.5 Diseño de la solución

Una vez estudiados los objetivos, el alcance, y los requisitos del proyecto, en este apartado se va a hacer una primera aproximación a lo que será el desarrollo de los agentes. Para ello, se deben diseñar unas pautas que se seguirán durante el desarrollo de los mismos.

3.5.1 Limitaciones debidas al alcance

Debido al alcance, existen una serie de limitaciones inherentes a la falta de la negociación. Con ella, los jugadores tienen fácil acceso al material que necesitan para continuar con la estrategia que están desarrollando. Sin embargo, a la hora de afrontar una partida sin poder negociar, puede que durante las partidas se den situaciones dónde alguno de los jugadores se quede en un estado “en bucle”, dónde necesita construir un cierto tipo de construcción para poder avanzar en su estrategia, pero no es capaz porque no posee los materiales necesarios ni acceso a ellos. Esto puede ocurrir si, por ejemplo, no comienza con sus dos poblados con acceso a los 5 materiales disponibles en el juego.

3.5.2 Estrategias de juego

Dado que se van a implementar varios agentes basados en heurística, se ha decidido optar porque cada uno de estos agentes adopte una estrategia distinta dentro de su estilo de juego, para así aportar variedad en el diseño de los mismos.

Siendo Catán un juego de gestión de recursos con elementos de azar y distintas formas de ganar, existen una gran variedad de estrategias posibles para intentar alcanzar los

10 puntos de victoria. Como ya se ha explicado en el punto anterior, existen X elementos que otorgan puntos de victoria:

- Los poblados: 1 PV
- Las ciudades: 2 PV
- Cartas de desarrollo: 1 PV
- Carretera más larga: 2 PV
- Mayor ejército: 2PV

Con esto, es posible confeccionar distintas estrategias, centrándose en una o varias de las opciones anteriores como principal para poder optar a una estrategia más consistente. Al margen de qué estrategia se siga, la etapa de colocación inicial de la partida es muy importante, ya que definirá como de rápidos y eficientes serán los primeros turnos. Es por ello por lo que, cabe definir una estrategia común para esta primera etapa.

En las etapas iniciales del juego lo más efectivo es poseer el acceso a la mayor cantidad de recursos distintos posible. Es por ello por lo que, el nodo inicial debería estar adyacente a 3 recursos distintos. Además, al tratarse de un juego donde se tiran dos dados al inicio de cada ronda, los números cercanos al 7 son los números con más probabilidades de salir. Colocar los poblados adyacentes a casillas con números como el 6 y el 8 tendrá un efecto muy positivo, ya que esos terrenos proporcionarán recursos con una mayor frecuencia.

Una vez definidas las mejores ubicaciones para los poblados, es necesario establecer una estrategia para colocar las carreteras de los mimos. Estas carreteras definirán hacia donde se expandirá el jugador posteriormente en la partida. Una primera aproximación sería construir la carretera en dirección al siguiente nodo que tenga al menos un terreno adyacente de un material que no se encuentre entre los materiales accesibles en ese momento. Con ello, se ampliarían la variedad de recursos que puede conseguir el jugador. Sin embargo, esta estrategia es poco óptima, ya que durante la partida se va consiguiendo esta variedad de recursos. Es por ello por lo que la mejor estrategia sería construir la carretera en un camino cuya distancia a un oponente sea la menor disponible. Con ello, se conseguirá una estrategia de bloqueo, ya que, una vez construida la ciudad en el nodo al que apunta, el oponente no podrá avanzar por ese camino. También se puede seguir una estrategia más conservadora, enfocada en no enfrentarse por controlar ciertas zonas. Colocando las carreteras en dirección a la costa o bordeándola, se consigue una estrategia más pasiva y con menos complicaciones para el avance.

Además, el orden de turno también es importante, ya que el jugador inicial tiene menos complicaciones a la hora de colocar sus fichas en el tablero. Sin embargo, el orden de turno es un aspecto que se suele dejar al azar en los juegos de mesa, para que ningún jugador se vea favorecido por el mismo.

Siguiendo estas estrategias, se consigue que el comienzo de la partida sea lo óptimo posible. A partir de este momento, se han de definir distintas estrategias, que posteriormente serán implementadas en forma de distintos agentes.

La primera estrategia es una estrategia **de control y agresiva**. Sigue la misma dinámica de lo explicado anteriormente en la colocación de la primera carretera. Esta estrategia se centra en ganar construyendo ciudades y poblados principalmente, pero siguiendo una estrategia agresiva en cuanto a que construye siempre hacia el centro del tablero, lo que consigue delimitar las zonas donde los oponentes pueden expandirse y construir. Este tipo de estrategias es muy común en los juegos de mesa de gestión de recursos, ya que, aunque un jugador vaya con buen camino a ganar la partida, si no se le impide ganar a otro que vaya mejor, ganará antes. Las mecánicas de esta estrategia son las siguientes:

- Construye ciudades siempre que puede
- Construye poblados si no puede construir ciudades
- Construye carreteras, siempre hacia un nodo adyacente al poblado de un oponente, siempre que no pueda construir ciudades o poblados.
- Construye carta de desarrollo si no tiene materiales suficientes para otra construcción

Otra estrategia común entre los jugadores de los juegos de mesa es una **estrategia pacifista o pasiva**. Esta estrategia se centra en un estilo de juego más solitario, construyendo carreteras alejándose de los oponentes, para poder avanzar sin complicaciones. Esta estrategia, además de ganar puntos a base de ciudades y de poblados, también trata de conseguir la carretera más larga, ya que, al construir por la costa, no suelen impedirle su avance. Además, se ha añadido que este tipo de estrategias no pueda construir cartas de desarrollo, para evitar la competitividad y robar cartas a los oponentes. Las directrices para esta estrategia son las siguientes:

- Construye ciudades siempre que puede
- Construye poblados si no puede construir ciudades
- Construye carreteras, siempre la costa o siguiéndola.

La última estrategia que se va a implementar en el proyecto es una **estrategia siguiendo** la mecánica que parece más secundaria al principio, que son **las cartas de desarrollo**. En esta estrategia, el jugador centrará sus materiales y sus turnos en construir cartas de desarrollo, tratando de conseguir las máximas cartas de puntos para el final de la partida, así como el ejército más grande. Además, para evitar desperdiciar materiales que necesita para construir ese tipo de cartas, no construye ciudades. Las reglas básicas de esta estrategia son las siguientes:

- Construye cartas de desarrollo siempre que puede
- Construye poblados si no puede construir ciudades
- Construye carreteras, de manera aleatoria

3.5.3 Diseño de agentes

Siguiendo el marco teórico, los algoritmos o **técnicas de heurística simple** que mejor se adaptan a los juegos de mesa como Catán, son:

- La máquina de estados finitos
- Los sistemas basados en reglas
- Los árboles de decisiones

De estos tres, se ha decidido trabajar con la máquina de estados finitos y con el sistema basado en reglas. Esto es porqué los sistemas basados en reglas y los árboles de decisiones tiene mucho en común. Ambos se basan a nivel de implementación en secuencias if-else para obtener la decisión correspondiente. Es por ello que se ha decidido prescindir de uno de ellos dos.

La máquina de estados finitos tendrá como base un esquema de relaciones entre estados, el cual permitirá cambiar entre ellos mediante unas condiciones específicas de cada relación.

El sistema basado en reglas irá evaluando distintas reglas hasta que alguna se cumpla, y cuando lo haga, efectuará la acción implementada en esa regla.

En el caso de la **heurística más avanzada**, se ha optado por que los jugadores decidan en base a un vector de pesos que estrategias seguir ,ya que se pretende generar estadísticas y decisiones basadas en las experiencias en partidas anteriores. Con esto, serán necesarios cambios a nivel de aplicación. La implementación de este marco se basará en varios agentes como los anteriores, y radicará en los jugadores qué estrategia elegir dependiendo de algún parámetro.

- Se ha diseñado una Máquina de estados finitos o FSM, capaz de que, cada vez que vaya a tomar una decisión dentro de la partida, analice si le es más atractivo cambiar el modo en el que juega en ese momento, o continuar con lo que estaba haciendo. Este desarrollo, hará uso de una función de estado, donde se evalúe el estado actual, y devuelva el estado nuevo si es que ha cambiado.
- Crear un bot basado en un sistema basado en reglas, donde cada vez que le llegue una decisión, tenga que evaluar distintas condiciones de manera jerárquica para obtener la decisión indicada en este momento. Para este desarrollo, será necesario enumerar una serie de reglas en cada tipo de decisión que se puede tomar dentro del juego.
- Crear una serie de agentes, cada uno con una estrategia distinta, y que el sistema pueda ir evaluando estas estrategias entre partida y partida, y que cada jugador sea capaz de ir iterando sobre las distintas estrategias de manera libre o controlada, para así obtener ellos mismos la mejor estrategia. Gracias a esto, se obtendrá un marco de trabajo más basado en el aprendizaje, ya que la idea es extraer que tipo de estrategia o bot es mejor dependiendo del jugador que se sea en la partida.
- Crear un sistema capaz de evaluar y analizar los resultados obtenidos en las simulaciones, para poder obtener que tipo de Bot o estrategia son las más efectivas y contra quien.

3.5.3.1 Diseño de Agente basado en FSM

Para poder comenzar con el desarrollo de este agente, es necesario comenzar decidiendo qué estrategia seguir a la hora de establecer los distintos estados que pueden acontecer, así como cómo se cambian entre ellos. En este caso, se ha optado por utilizar una estrategia agresiva para desarrollar este bot, ya que el bot de AdrianHerasBot utiliza una estrategia más pasiva a la hora de expandirse con carreteras.

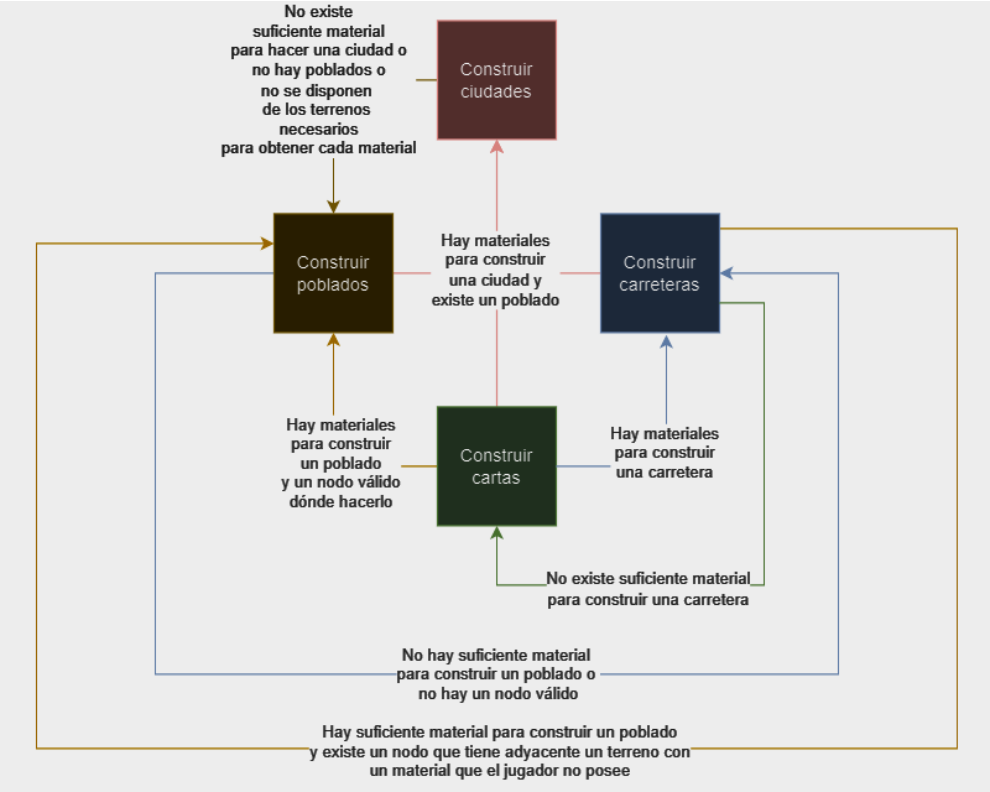

Figura 5: Esquema de la FSM implementada

Decidida la estrategia, hay que comenzar , como se puede observar en la figura 5, definiendo los distintos estados en los que se puede encontrar la máquina. Siguiendo la estrategia agresiva, podemos hacer una aproximación a las acciones que se pueden realizar a la hora de comenzar la fase de construcción. Además, siguiendo la estrategia elegida, estos tendrán una estructura jerárquica, siendo, de mejor a peor:

- Construir Ciudades
- Construir Poblados
- Construir Carreteras
- Construir cartas de desarrollo

Una vez definidos los estados, a nivel de implementación, se barajaron distintas opciones:

- Establecer las reglas dentro de cada una de las funciones donde se fueran a implementar
- Crear una función que evalúe únicamente las relaciones entre estados y actualice el estado en una variable global

La primera aproximación es poco eficiente y se aleja de lo que es una máquina de estados finitos. Sin embargo, la segunda es justo lo que se desea, ya que con ella, la máquina evaluará sus condiciones y su estado una vez por ronda.

3.5.3.2 Diseño de Agente basado en Reglas

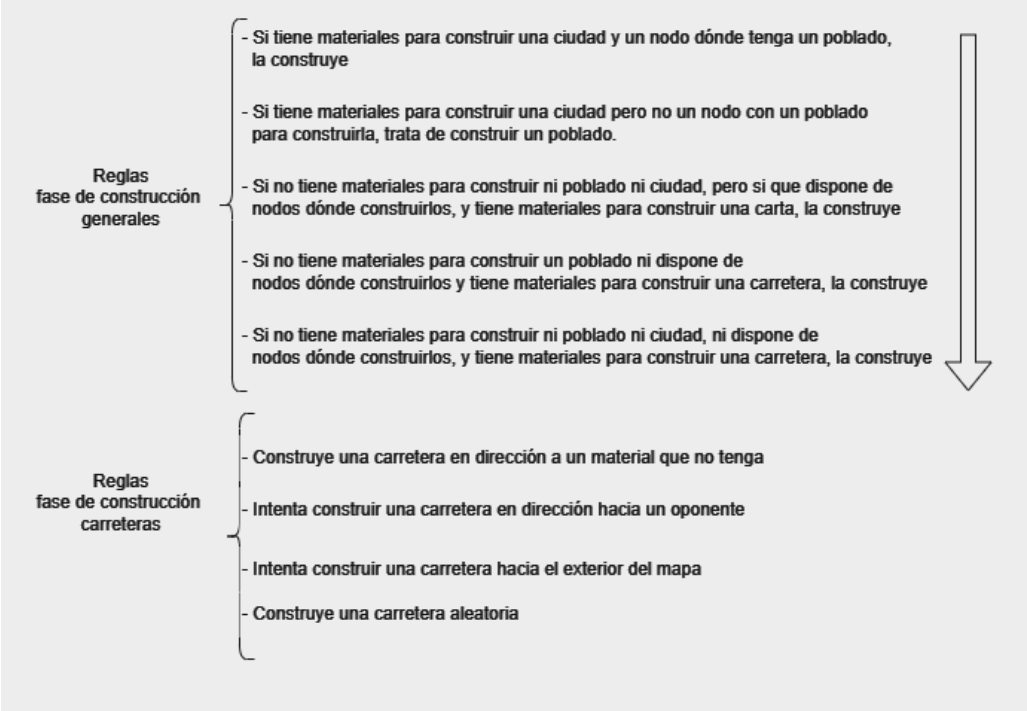

Figura 6: Conjunto de reglas para agente basado en reglas

Al igual que con el agente anterior, lo primero que hay que definir es la estrategia que va a seguir este agente a la hora de jugar. Se ha decidido utilizar una estrategia más pacífica o pasiva para este agente, con el fin de diferenciarse suficiente de la estrategia seguida por el agente basado en FSM, y se han establecido unas reglas base, como se puede observar en la figura 6

Para comenzar a definir unas reglas para controlar el sistema, se comenzará exponiendo en qué momentos o fases de la partida el agente debe seguir unas reglas

para actuar, y cuando hará una acción predeterminada. Las fases claves a considerar serían las siguientes:

- Comienzo de la partida: Dónde colocar el poblado y hacia dónde dirigir la carretera inicial
- En fase de construcción: Qué priorizar, dónde construir cada tipo de construcción
- Jugando cartas de desarrollo: Que materiales elegir, cuándo jugarlas

Además de estos momentos, existen otros momentos dónde el bot podría seguir unas reglas para establecer sus decisiones. Sin embargo, como bien se ha explicado, cuanto más complejo es un problema, más complejo se vuelve establecer un sistema basado en reglas que lo rija. Es por ello por lo que se trabajará sobre esta base.

Como bien se ha explicado en el apartado de estrategias, la estrategia de la colocación inicial de poblados es común a los agentes para facilitar el acceso a todos los recursos.

Como ya se ha comentado, los agentes desarrollados comienzan igual al inicio de la partida, ya que es un comienzo suficientemente óptimo. Sin embargo, en esta estrategia pacifista, se va a modificar todo lo relacionado con la construcción de carreteras, haciendo que el jugador trate de construir carreteras siempre hacia la costa. A continuación, se van a definir de forma funcional las reglas que se van a seguir a la hora de construir carreteras.

Si quiere construir una carretera:

- Construye una carretera en dirección a un material que no tenga
- Intenta construir una carretera hacia el exterior del mapa
- Construye una carretera aleatoria

Durante la fase de construcción, para saber que priorizar en cada momento, se seguirán las siguientes reglas:

- Si tiene materiales para construir una ciudad y un nodo dónde tenga un poblado, la construye
- Si tiene materiales para construir una ciudad, pero no un nodo con un poblado para construirla trata de construir un poblado.
- Si no tiene materiales para construir ni poblado ni ciudad, pero si que dispone de nodos dónde construirlos, y tiene materiales para construir una carta, la construye
- Si no tiene materiales para construir un poblado ni dispone de nodos dónde construirlos y tiene materiales para construir una carretera, la construye

- Si no tiene materiales para construir ni poblado ni ciudad, ni dispone de nodos dónde construirlos, y tiene materiales para construir una carretera, la construye

3.5.3.3 Diseño de modelo avanzado

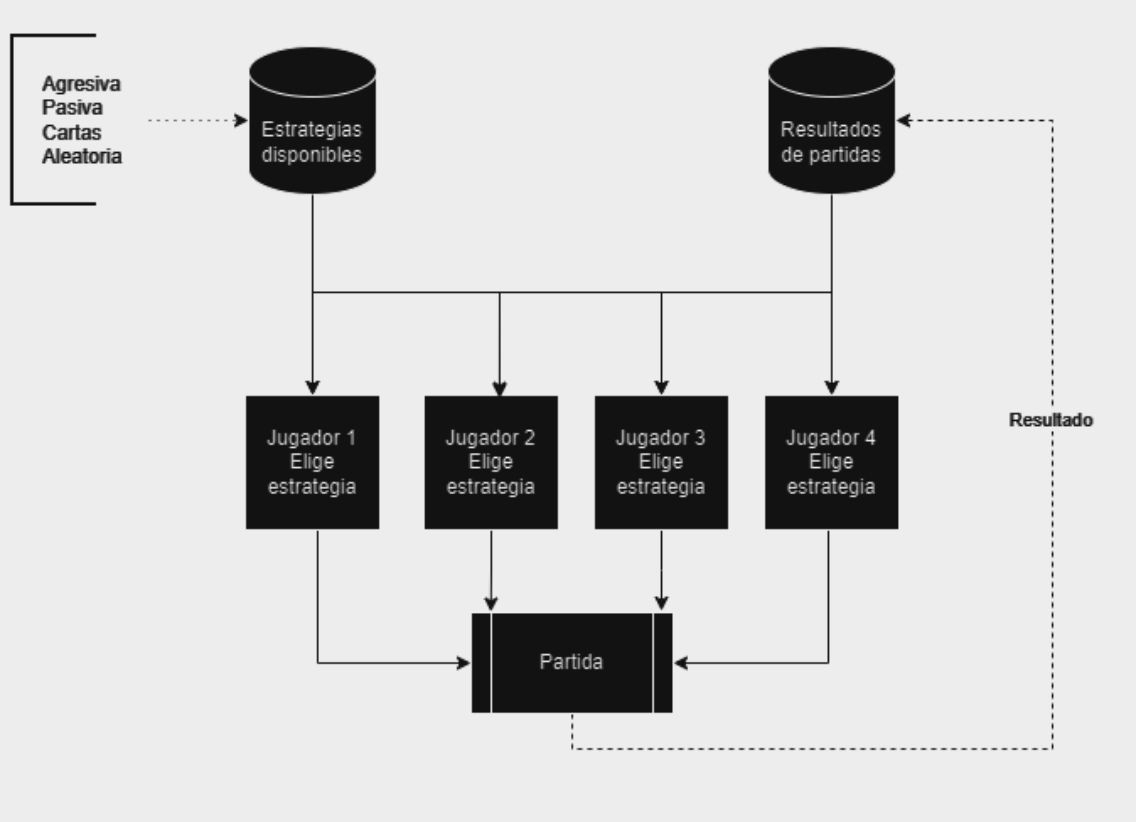

Figura 7: Esquema de partidas para estrategias avanzadas

Para el desarrollo de este marco, se han decidido aplicar estrategias avanzadas a nivel de jugador, y no de estrategia. Como se puede observar en la figura 7, la idea es que cada jugador elija de entre las estrategias disponibles, una estrategia para jugar la siguiente partida. Es por ello por lo que era necesario desarrollar nuevos agentes que siguieran estrategias distintas, para que cada jugador fuera capaz de elegir que estrategia seguir según ciertos criterios. Se ha decidido implementar dos nuevas estrategias, y para hacerlo lo más uniforme posible, se van a desarrollar basándose en FSM, debido a la experiencia que se ha adquirido tras el desarrollo del primer agente.

La primera estrategia que se usará, la agresiva, ya ha sido explicada en el 3.5.3.1.

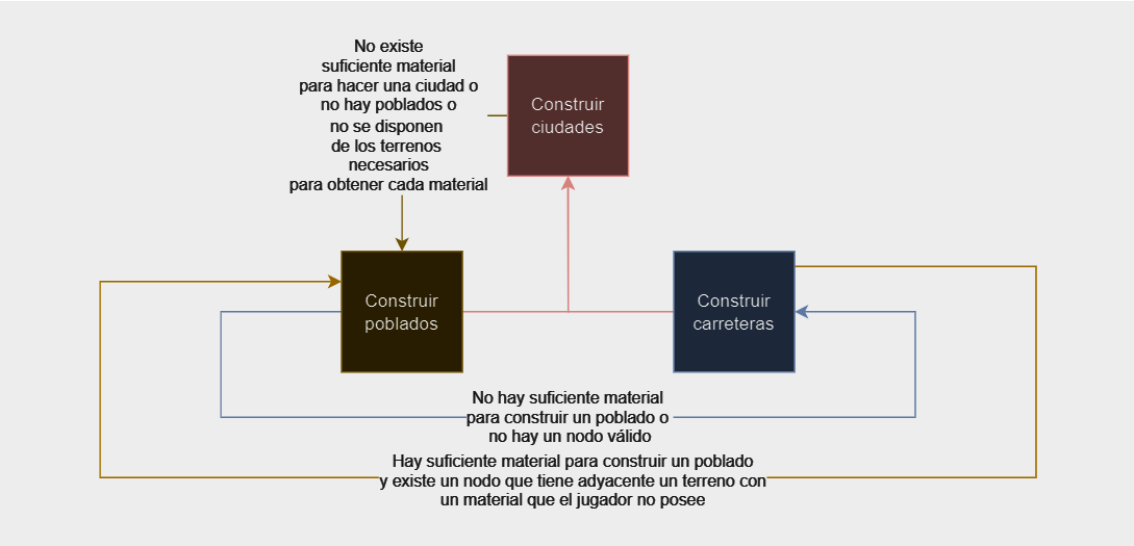

Figura 8: Esquema de estados y relaciones para FSM pacífica

La segunda estrategia, como se puede observar en la figura 8, basará su juego en tratar de construir sus carreteras hacia el exterior del mapa, no construirá cartas de desarrollo, y tratará de ganar por ciudades y poblados y carretera más larga.

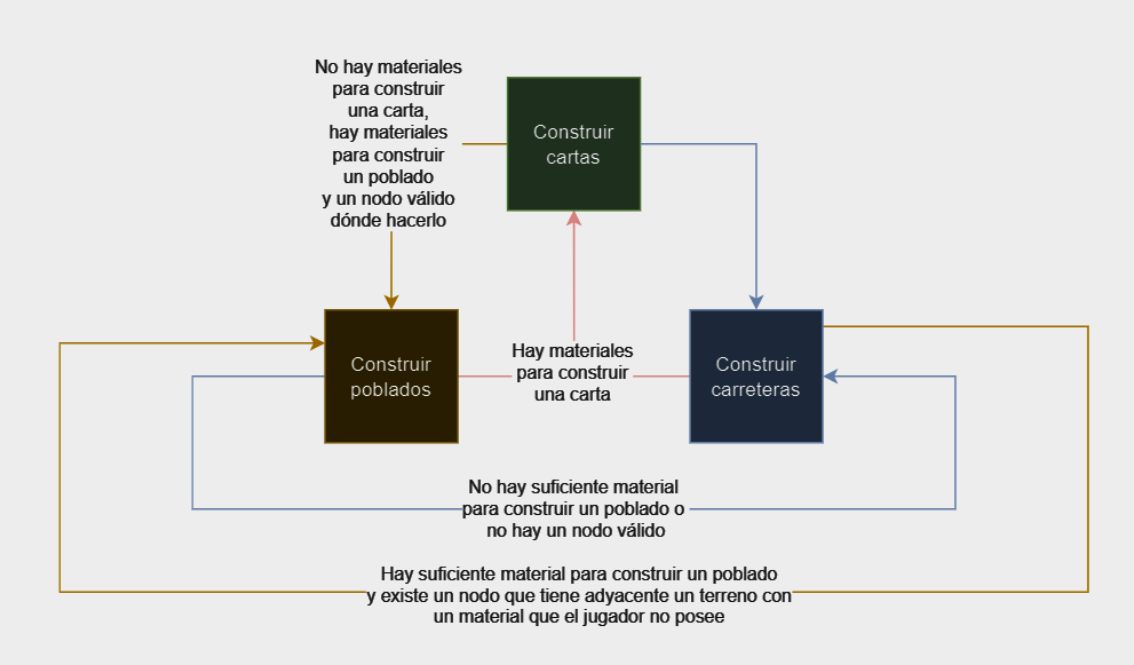

Figura 9: Esquema de estados y relaciones FSM basada en cartas

La última estrategia que se va a implementar basa su juego en construir Cartas como acción principal, y no construye ciudades para no tener que malgastar materiales en ello, como se puede observar en la figura 9. Su manera de ganar es mediante el ejército

más grande, la carretera más larga, puntos de victoria ocultos y construcción de poblados.

3.5.4 Framework de análisis de resultados

Una vez estén desarrollados los agentes, hay que definir una estrategia para evaluar su rendimiento, así como su comportamiento en distintos escenarios.

Debido al alto volumen de partidas, no se hace viable tratar los datos a nivel de archivos como preparó Adrián. Es por ello por lo que, se ha decido establecer un nuevo elemento en la estructura del PyCatan para analizar las partidas, el cual será explicado en el apartado 3.5.5.

En cuanto al análisis del modelo inteligente, se han decidido establecer distintos escenarios, cada cual encargado de comprobar y ejemplificar algunos comportamientos de los agentes.

Dentro de este modelo, también será relevante analizar como de relevante es el orden de turno dentro del juego y de las estrategias generadas. Con ello, por ejemplo, obtendremos con qué estrategia es mejor jugar si se es el jugador inicial, y si los jugadores inteligentes son capaces de identificarlas.

El Formato de salida de la aplicación ya había sido modificado al principio de este proyecto para facilitar este apartado. Sin embargo, presentaba un problema a la hora de intentar analizarlo: Era necesario otro desarrollo capaz de condensar toda la información en un único punto.

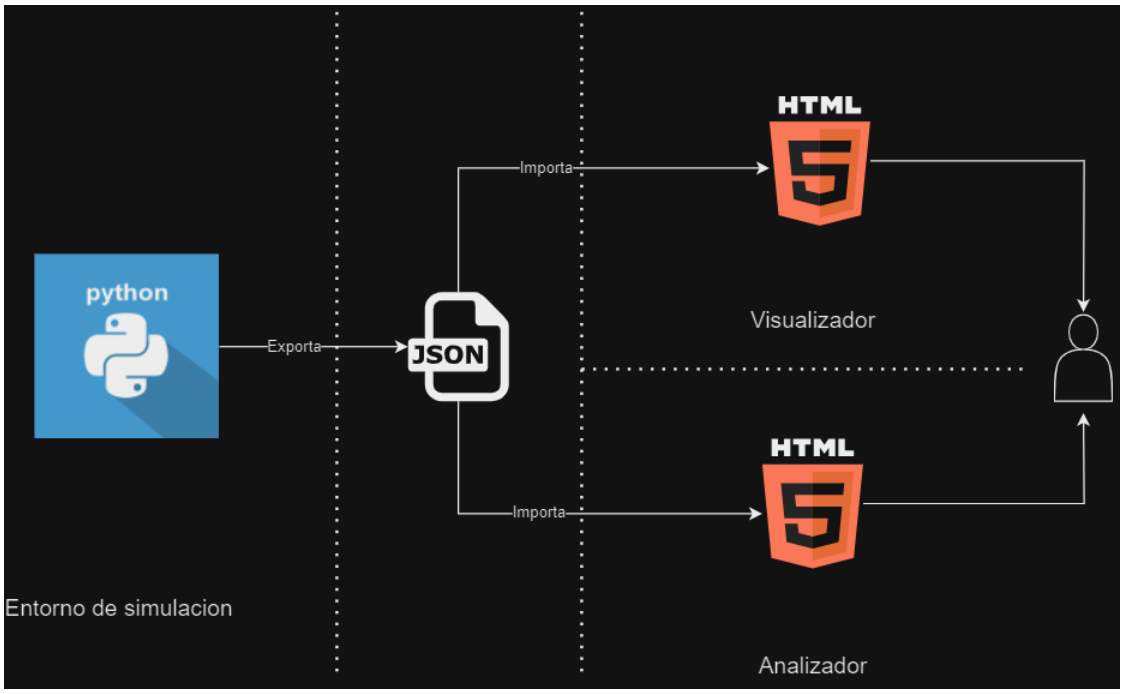

Figura 10: Nueva estructura de la aplicación PyCatan

Es por ello que se ha añadido una nueva pieza a la infraestructura del PyCatan, como se puede observar en la figura 10, llamada Analizador. Aprovechando el desarrollo del visualizador del PyCatan, se ha decidido hacer uso de la misma metodología a la hora de desarrollar una interfaz gráfica de análisis de resultados. Con ello, se podrán analizar el porcentaje de victorias, así como otras métricas de interés.

4. Desarrollo

En este apartado, se expondrán todos los cambios en el código original del PyCatan para adaptarse a los requisitos, así como el desarrollo de los agentes que se han seleccionado.

4.1 Cambios en la plataforma de simulación

Durante el año 2023 y parte del 2024, Adrián Heras estuvo refactorizando el código fuente de la aplicación, para poder otorgarle un cuerpo más técnico, así como una mayor accesibilidad para futuros desarrollos de agentes. Sin embargo, desarrollando estos mismos, se han encontrado carencias que ha habido que subsanar para poder aplicar las metodologías que se necesitaban para el desarrollo y el análisis de resultados.

Con esto, los cambios que se han realizado en el código del PyCatan son los siguientes (Figura 11):

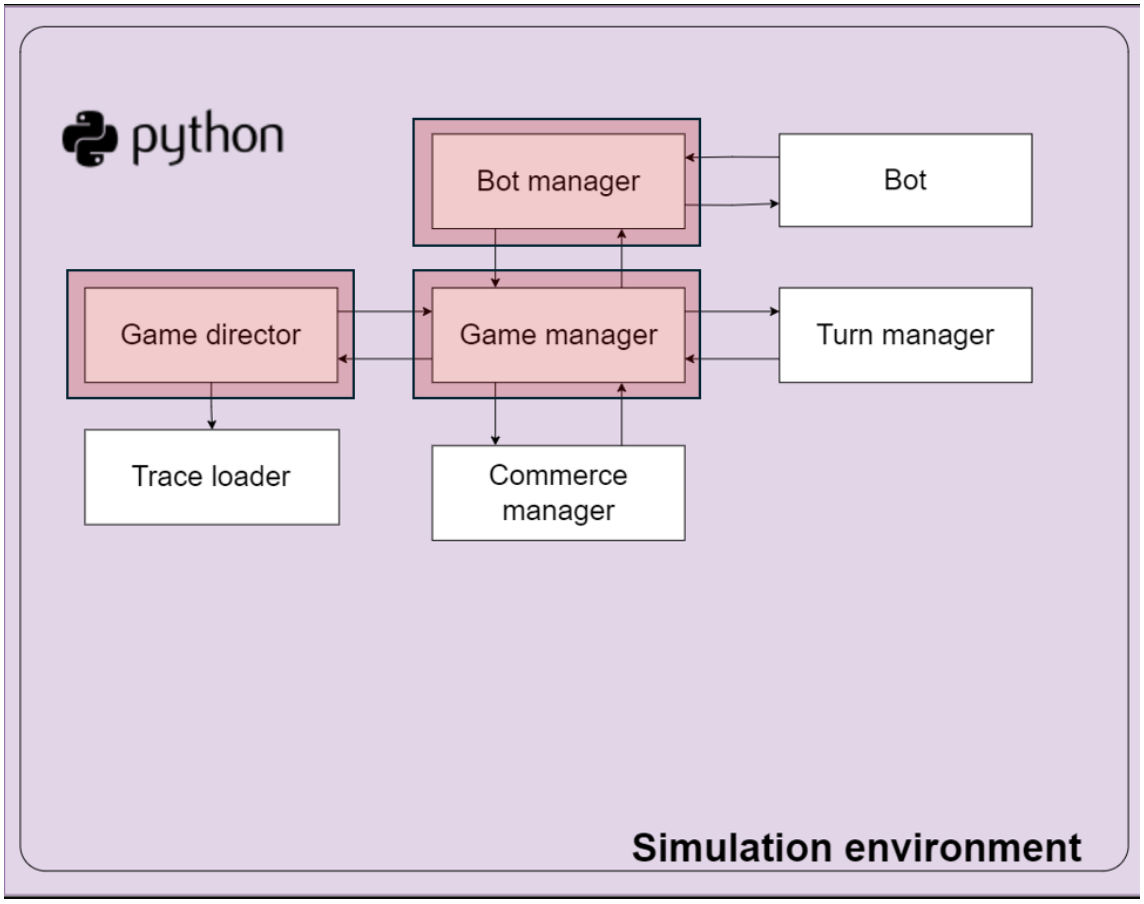

Figura 11: Clases modificadas en el entorno de simulación

- Se ha añadido un nuevo parámetro a la clase GameDirector, la encargada de generar el bucle de partidas. Hasta el desarrollo de este proyecto, cuando se inicializaba el programa, se podía elegir un bot para cada jugador, el cual se mantenía durante las n partidas solicitadas al programa. Sin embargo, para los métodos avanzados que se utilizarán posteriormente, era interesante generar escenarios dónde en cada partida los jugadores pudieran cambiar el bot que habían seleccionado. Con esta idea, se ha modificado la clase GameDirector para añadir un modo avanzado. Este modo está presente en todas las funciones que interaccionan con el comienzo y el final de partida, ya que es cuando el Bot Manager asigna Agentes a cada jugador. Con ello, se ha conseguido dividir el código original con un código nuevo para dichas funciones, puesto que realizarán tareas distintas.

- Para una mayor facilidad a la hora de poder analizar los resultados de las partidas, así como de disponer de un resumen de la misma en el archivo de traza que genera al final de la ejecución, se ha añadido a la traza un nuevo apartado, denominado “resume”, que recopila la información relevante, sobre todo de cara al análisis de la IA más avanzada, de forma más estructurada y resumida, como por ejemplo, que clase está jugando cada jugador en la partida, cuantas rondas se han jugado, y el estado actual de cada agente en el subapartado resume_bots. La estructura del nuevo apartado se muestra en la Figura 12:

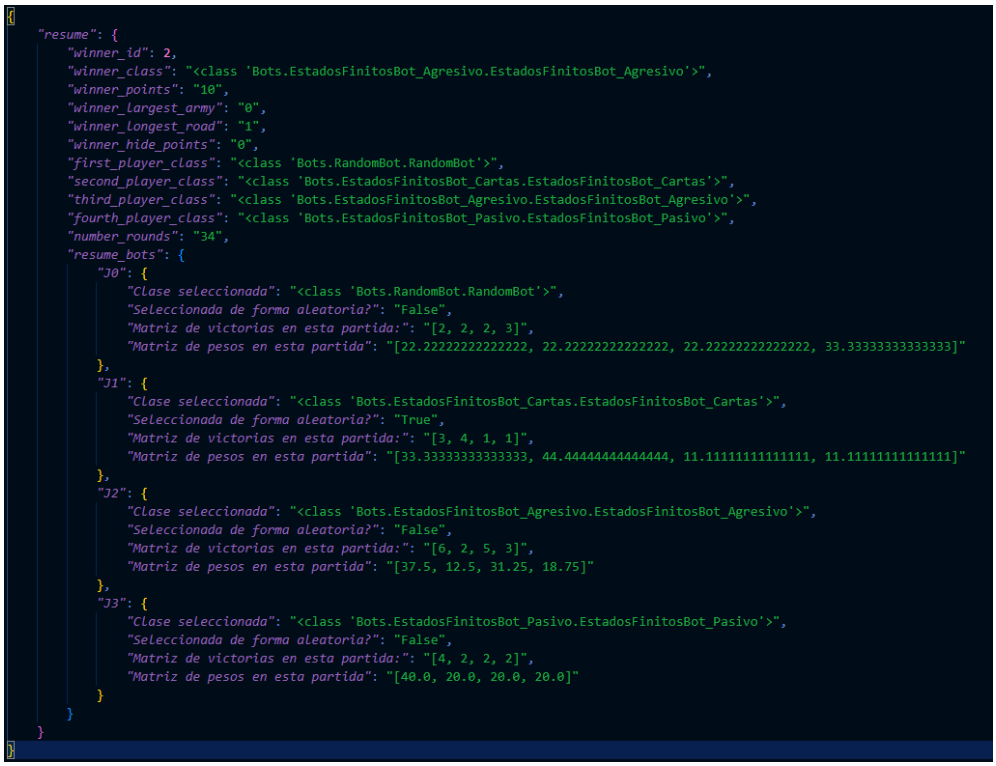

Figura 12: Apartado resume en el archivo de salida

Como se puede observar en la figura 12, el apartado resume_bots solo está disponible en el modo avanzado, ya que muestra la matriz de pesos que le ha llevado al jugador a elegir un el agente para esta partida, así como si lo ha elegido él mismo o se le ha asignado de manera aleatoria, y una matriz de victorias con cada una de las estrategias.

- A raíz de la introducción del modo avanzado, se ha modificado también el Bot Manager. Esta clase, asignaba los agentes que se le indicaban por consola a cada jugador, y no los modificaba entre partidas. Al seleccionar el nuevo modo, el primer bot que se selecciona para cada jugador se hace de manera aleatoria entre las distintas estrategias desarrolladas, pero en cada reinicio de partida, se selecciona uno nuevo de entre todas las opciones. Esta selección puede ser aleatoria, o basada en una matriz de pesos interna que posee cada jugador, que se va modificando conforma va ganando partidas. Esta modificación será explicada con mayor profundidad en el apartado de implementación de las IAs avanzadas.
- Como se ha explicado con anterioridad, se ha querido dejar fuera del alcance de este proyecto el apartado de negociación entre jugadores. Es por ello por lo que se han tenido que modificar los agentes por defecto que propuso Adrián y que se han utilizado como “pruebas” y “control” durante la realización de este

proyecto. Se ha modificado la función “on_comerce_phace” para que devuelva un None, que el GameManager interpreta como que no quiere intercambiar.

- En el código original de Adrián, no existía un límite de carreteras que pudiera construir cada jugador en una partida. Sin embargo, las reglas del catán especifican que no pueden construirse más construcciones que fichas dispone el jugador. En el caso de las carreteras, este límite es 15. En etapas tempranas del proyecto, se observó como algunas estrategias llevaban a estados dónde un jugador llenaba el mapa de carreteras, dejando a los demás jugadores “inhabilitados” a la hora de expandirse y fundar nuevos poblados.
- Al haber eliminado el comercio del juego, la recolección de materiales es totalmente aleatoria, dependiente del valor de los dados y de la colocación de los poblados de los jugadores. Es por ello que existen ciertas configuraciones dónde la partida se puede alargar más de lo esperado. Por ello, se ha decidió implementar en la clase GameDirector, en el bucle de rondas de la partida, una condición extra a la finalización de la partida. Ahora, además de comprobar si hay ganador o no cada ronda, también comprueba si la partida lleva más de 500 rondas. Si es así, la partida se detiene y se otorga la victoria al que más puntos tenga en ese momento.

4.2 Desarrollo de los agentes Heurísticos simples

Una vez definido el diseño de los nuevos agentes a desarrollar, en los siguientes apartados, se explicarán con más detalle los desarrollos que se han seguido en las distintas estrategias utilizadas.

4.2.1 Agente de Máquina de Estados Finitos, FSM

Para conseguir implementar el modelo, se ha creado una nueva clase dentro del agente, denominada change_state.

En esta clase, se comprueba mediante una cláusula match-case en qué estado se encuentra actualmente el agente, y mediante secuencias if-else, se comprueba si tiene que pasar a otro estado o mantenerse en el que se encuentra. Luego, si el estado ha cambiado, se actualiza a nivel global.

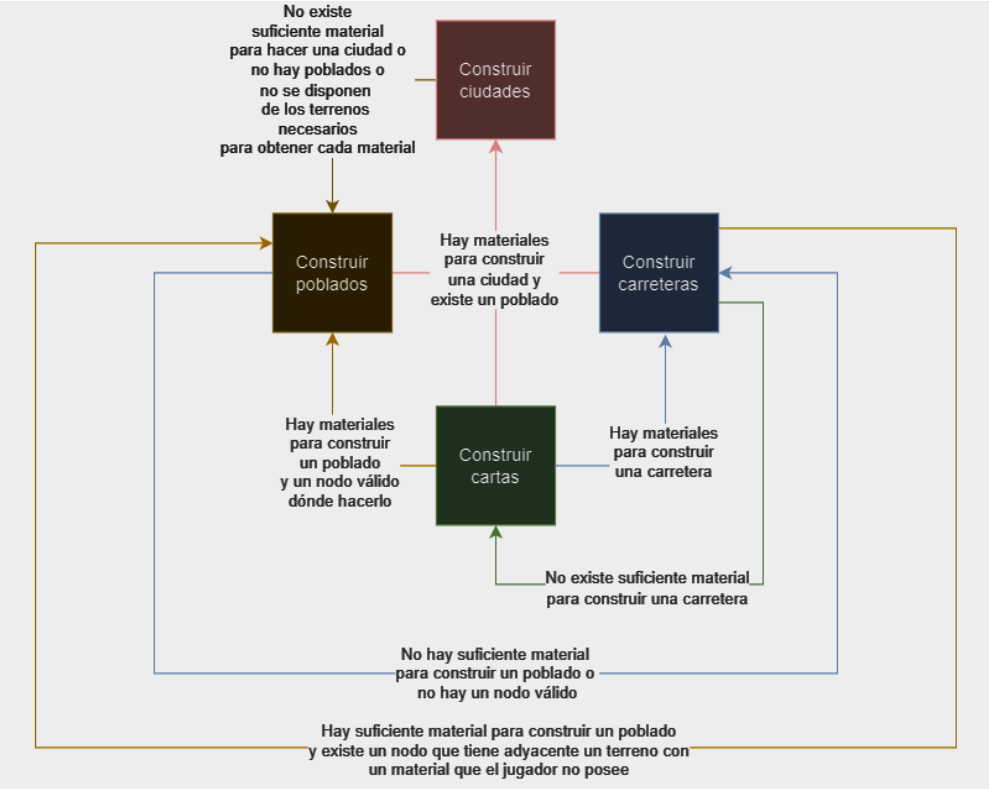

Figura 13: Esquema de la FSM implementada

En los siguientes apartados, se explicarán los distintos estados que se ven en la figura 13, así como las relaciones entre ellos y su implementación en código.

4.2.1.1 Construir Ciudades

Las ciudades son la fuente de puntos reutilizable por excelencia, ya que cada una de ellas otorgan 2 puntos al final de la partida. Para el comportamiento del agente agresivo, es lo más importante, ya que después de haber controlado el mapa, lo más fácil es construir poblados y ciudades en los nodos a los que tiene acceso.

```
match self.estado_actual:
    case EstadosConstants.CIUDAD:
            if not self.hand.resources.has_this_more_materials(BuildConstants.CITY) or self.town_number == 0 or len(
                missing_terrain) != 0:
                if (self.hand.resources.has_this_more_materials(BuildConstants.TOWN) and len(
                        possibilities) != 0) or (len(place_to_build_town) != 0 and missing_terrain != 0):
                    estado_nuevo = EstadosConstants.POBLADO
                elif self.hand.resources.has_this_more_materials(BuildConstants.ROAD) and len(
                            place_to_build_town) == 0 and number_of_roads < 15:
                        estado_nuevo = EstadosConstants.CARRETERA
                else:
                        estado_nuevo = EstadosConstants.CARTAS
```

Figura 14: Implementación de las relaciones de "Construir Ciudad"

Existen tres variables que se han añadido en esta sección del código al repertorio de variables de los agentes para obtener un mayor control sobre el cambio de los estados. Estas son:

- Missing_terrain: Número de materiales a los que el jugador no tiene acceso con los poblados que tiene actualmente construidos
- Place_to_build_town: Lista de nodos dónde el jugador puede construir un poblado que le daría acceso a algún material que no posee
- Number of roads: Número total de carreteras del jugador

Como se puede observar en la figura 14, el agente cambia de estado siempre y cuando no existan materiales suficientes para construir una ciudad, o no se tenga ningún nodo con un poblado donde construir una ciudad, o si le falta algún tipo de material. Como ya se ha explicado, los agentes que se van a desarrollar darán prioridad a tener acceso a todos los materiales disponibles en la partida.

Si alguna de estas condiciones se cumple, se comprueba si se cumple alguna de las condiciones para pasar a los otros estados:

- Si hay materiales suficientes para construir un poblado, o tiene algún sitio donde construir un poblado para conseguir un material que le falta, pasa al estado Construir Poblado. Si pasa a este estado sin tener los materiales necesarios, se conseguirá que falle en la construcción y por lo tanto "ahorre" materiales esta ronda.
- Si no le es posible construir un poblado, pero tiene materiales para construir una carretera, y no tiene ningún nodo donde colocar un poblado para conseguir algún material que le falte, y tiene menos de 15 carreteras, pasa al estado Construir Carreteras. La variable Place_to_build_town es 0 cuando el jugador dispone de acceso a todos los materiales, así que en este tipo de sentencias solo se utiliza en el juego temprano.
- Si no le es posible ninguna de las anteriores, pasa a Construir Cartas.

En cuanto a la implementación del comportamiento del agente a la hora de la fase de construcción,

4.2.1.2 Construir Poblados

Los poblados son necesarios para poder construir las ciudades, y también para poder diversificar los recursos que se pueden disponer durante la partida. Aun dando menos puntos que las ciudades, son igual de importantes, sobre todo en este proyecto, ya que es lo que da acceso a los materiales durante la partida

```
case EstadosConstants.POBLADO:
                if self.hand.resources.has_this_more_materials(
                        BuildConstants.CITY) and self.town_number != 0 and missing_terrain == 0:
                    estado_nuevo = EstadosConstants.CIUDAD
                elif (not self.hand.resources.has_this_more_materials(BuildConstants.TOWN) or len(
                        possibilities) == 0 or (len(place_to_build_town) != 0 and missing_terrain != 0)):
                    if self.hand.resources.has_this_more_materials(BuildConstants.ROAD) and len(
                            place_to_build_town) == 0 and number_of_roads < 15:
                        estado_nuevo = EstadosConstants.CARRETERA
                    else:
                        estado_nuevo = EstadosConstants.CARTAS
```

Figura 15: Implementación de las relaciones de "Construir Poblado"

Como se puede ver en la figura 15, el agente cambia de estado a Ciudad siempre y cuando tenga suficientes recursos para construirla, exista un poblado sobre el cual hacerlo,y tenga todos los tipos de terrenos accesibles. Si no cumple los requisitos para construir un poblado, trata de pasar al estado Construir carretera, similar al apartado anterior. Por último, si no puede construir nada, pasa al estado Construir cartas.

```
case EstadosConstants.POBLADO:
                self.estado_poblado += 1
                if self.hand.resources.has_this_more_materials(BuildConstants.TOWN):
                    possibilities = self.board.valid_town_nodes(self.id)
                    for node_id in possibilities:
                        for terrain_piece_id in self.board.nodes[node_id]['contacting_terrain']:
                            if self.board.terrain[terrain_piece_id]['probability'] == 4 or \
                                    self.board.terrain[terrain_piece_id]['probability'] == 5 or \
                                    self.board.terrain[terrain_piece_id]['probability'] == 6 or \
                                    self.board.terrain[terrain_piece_id]['probability'] == 8 or \
                                    self.board.terrain[terrain_piece_id]['probability'] == 9 or \
                                    self.board.terrain[terrain_piece_id]['probability'] == 10:
                                self.town_number += 1  # Añadimos un pueblo creado
                                return {'building': BuildConstants.TOWN, 'node_id': node_id}
```

Figura 16: Implementación del comportamiento de "Construir Poblado"

En este estado, como se puede observar en la figura 16, el agente construirá un poblado sobre un nodo válido, es decir, accesible mediante una carretera y que esté a al menos dos nodos de distancia de otro poblado. Además, dentro de los posibles nodos, el código lo coloca en alguno que tenga un terreno adyacente con una buena probabilidad de salir.

4.2.1.3 Construir Carreteras

Las carreteras son el método de expansión en el catán. Con ellas, el jugador consigue expandirse para acceder a construir poblados en otros nodos, así como también tener la posibilidad de bloquear el avance de los oponentes.

```
match self.estado_actual:
            case EstadosConstants.CARRETERA:
                if self.hand.resources.has_this_more_materials(
                        BuildConstants.CITY) or self.town_number == 0 and missing_terrain == 0:
                    estado_nuevo = EstadosConstants.CIUDAD
                elif (self.hand.resources.has_this_more_materials(BuildConstants.TOWN) and len(
                        possibilities) != 0) or (len(place_to_build_town) != 0 and missing_terrain != 0):
                    estado_nuevo = EstadosConstants.POBLADO
                elif not self.hand.resources.has_this_more_materials(BuildConstants.ROAD) or len(
                            place_to_build_town) != 0:
                    estado_nuevo = EstadosConstants.CARTAS
```

Figura 17: Implementación de las relaciones de "Construir Carretera"

El estado "Construir carretera", como el resto y tal y como se ve en la figura 17, cambia a ciudad siempre y cuando se cumplan todas sus condiciones. Después de intentarlo, si no lo consigue, intenta cambiar a poblado. Si no tiene las condiciones necesarias, y no dispone de materiales para construir una carretera, cambia a cartas.

```
match self.estado_actual:
            case EstadosConstants.CARRETERA:
                self.estado_carretera += 1
                if self.hand.resources.has_this_more_materials(BuildConstants.ROAD):
                    possibilities = self.board.valid_road_nodes(self.id)
                    missing_terrain = self.missing()
                    if len(missing_terrain) != 0:
                        for road_obj in possibilities:
                            av_terrain = self.board.__get_contacting_terrain__(road_obj['finishing_node'])
                            for av in av_terrain:
                                if self.board.terrain[av]['terrain_type'] in missing_terrain:
                                    return {'building': BuildConstants.ROAD,
                                            'node_id': road_obj['starting_node'],
                                            'road_to': road_obj['finishing_node']}

                    possible_roads_torwards_oponent = []

                    for road_obj in possibilities:
                        for node in  self.board.__get_adjacent_nodes__(road_obj['finishing_node']):
                            if self.board.nodes[node]['player'] != -1 and self.board.nodes[node]['player'] != self.id:
                                possible_roads_torwards_oponent.append(road_obj)
                    if len(possible_roads_torwards_oponent):
                        road_node = random.randint(0, len(possible_roads_torwards_oponent) - 1)
                        return {'building': BuildConstants.ROAD,
                                'node_id': possible_roads_torwards_oponent[road_node]['starting_node'],
                                'road_to': possible_roads_torwards_oponent[road_node]['finishing_node']}

                    will_build = random.randint(0, 2)
                    if will_build:
                        if len(possibilities):
                            road_node = random.randint(0, len(possibilities) - 1)
                            return {'building': BuildConstants.ROAD,
                                    'node_id': possibilities[road_node]['starting_node'],
                                    'road_to': possibilities[road_node]['finishing_node']}
```

Figura 18: Implementación del comportamiento de "Construir Carretera"

En este estado, si el agente no tiene acceso a algún material, el agente tratará de construir una carretera hacía un nodo que tenga adyacente un material que necesita. Si

ya posee todos los materiales, tratará de construir una carretera en algún camino que vaya hacia algún oponente, como se puede ver en la figura 18, si no, optará por carreteras exteriores, ya que son las mejores para conseguir el objetivo de “Carretera más larga”, puesto que hay menos posibilidades de encontrarse a algún oponente.

4.2.1.4 Construir Cartas

En el caso de la programación de este agente, se ha considerado “Construir cartas” como un estado inferior al resto. Sin embargo, dependiendo de la estrategia, podría ser incluso el principal, puesto que, entre las cartas, se encuentran los ejércitos (ya que sirven para el objetivo de final de partida “Mayor ejército”) y los puntos, que existen un total de 5 en el mazo. Si se consiguieran todos, con los dos poblados iniciales, ya se tendrían nueve puntos de los 10, así que es una estrategia a tener en cuenta.

```
case EstadosConstants.CARTAS:
    if self.hand.resources.has_this_more_materials(BuildConstants.CITY) and self.town_number != 0 and len(
            missing_terrain) == 0 and missing_terrain == 0:
        estado_nuevo = EstadosConstants.CIUDAD
    elif (self.hand.resources.has_this_more_materials(BuildConstants.TOWN) and len(
                possibilities) != 0) or (len(place_to_build_town) != 0 and missing_terrain != 0):
            estado_nuevo = EstadosConstants.POBLADO
    elif not self.hand.resources.has_this_more_materials(BuildConstants.ROAD) or len(
                    place_to_build_town) != 0 or number_of_roads >= 15:
                estado_nuevo = None
    else:
                estado_nuevo = EstadosConstants.CARRETERA
```

Figura 19: Implementación de las relaciones de “Construir Cartas”

El estado Cartas, al ser el último en la jerarquía de estados para esta estrategia, tratará de cambiar a cualquier otro estado en orden de Ciudad, Poblado, carretera siempre y cuando cumpla alguno de sus requisitos, tal y como se ha mostrado en la figura 19.

En este estado, el agente tratará de construir una carta de desarrollo. Si no le es posible, el método de on_build_phase le devuelve un none al GameManager, por tanto interpreta que no tiene nada más que construir esta ronda y pasa el turno.

4.2.2 Agente de Sistema basado en Reglas

En el caso del agente basado en reglas, el desarrollado elegido para implementarlas ha sido mediante árboles de if-else, ya que estos sirven para conservar la jerarquía que se quiere implementar en las reglas. Siguiendo las reglas diseñadas en etapas anteriores del proyecto, se ha conseguido lo siguiente:

- Mediante el árbol de sentencias if-else en la función on_build_phase, se consigue una lógica jerárquica, dónde el agente va a evaluar una a una las reglas

en orden descendente, hasta que una se cumpla y realice la acción correspondiente.

- Se han añadido a todas las construcciones de carreteras y de uso de cartas de desarrollo la lógica de las reglas diseñadas, con su correspondiente estrategia adaptada a las mismas.
- En el caso del desarrollo de este agente, no ha sido necesaria la creación de nuevas funciones dentro de la clase.

4.3 Desarrollo de los agentes para la estrategia avanzada

Con las 3 estrategias planteadas en el diseño, como ya se ha avanzado en apartados anteriores, se ha creado un nuevo modo dentro de la aplicación denominado modo avanzado. Con esto, se podrá controlar y hacer cambiar a los agentes de clases. La implementación a nivel de código ha sido la siguiente:

Primero, se han añadido a la clase BotManager dos variables globales nuevas, denominadas j_lista_pesos y j_lista_pesos_real.

```
j_lista_pesos = [[1, 1, 1, 1],
                 [1, 1, 1, 1],
                 [1, 1, 1, 1],
                 [1, 1, 1, 1]]


j_lista_pesos_real = [[25, 25, 25, 25],
                      [25, 25, 25, 25],
                      [25, 25, 25, 25],
                      [25, 25, 25, 25]]
```

Figura 20: Matrices de pesos para elección de estrategias

Estas sirven para llevar un recuento del número de victorias de cada jugador con cada tipo de estrategia ([‘Agresivo’,’Pasivo’,’Cartas’,’Aleatoria’]), y para calcular las probabilidades de la segunda matriz, como se puede ver en la figura 20. Se le pasa a cada jugador su sección de la matriz, para que pueda tener en cuenta los porcentajes basados en las victorias anteriores.

Esta segunda matriz se calcula mediante una función sencilla también implementada en el GameManager, denominada create_weights.

Al inicializar el BotManager, se establecen los porcentajes por defecto en cada bot, para no afectar a los resultados.

```
elif for_advanced:
    my_list = ['EstadosFinitosBot_Agresivo', 'EstadosFinitosBot_Pasivo', 'EstadosFinitosBot_Cartas',
               'RandomBot']
    primer = choices(my_list, [25,25,25,25], k=1)[0]
    segundo = choices(my_list, [25,25,25,25], k=1)[0]
    tercer = choices(my_list, [25,25,25,25], k=1)[0]
    cuarto = choices(my_list, [25,25,25,25], k=1)[0]
    self.first_bot_class = getattr(__import__('Bots.' + primer, fromlist=[primer]), primer)
    self.second_bot_class = getattr(__import__('Bots.' + segundo, fromlist=[segundo]), segundo)
    self.third_bot_class = getattr(__import__('Bots.' + tercer, fromlist=[tercer]), tercer)
    self.fourth_bot_class = getattr(__import__('Bots.' + cuarto, fromlist=[cuarto]), cuarto)
```

Figura 21: Implementación de elección de estrategias en Bot_Manager

Como se puede observar en la figura 21, se ha añadido el Randombot a las posibles estrategias, para tener un elemento de control para poder asegurar que los agentes generados ganan más veces que un jugador jugando de manera aleatoria.

Con esto, cada vez que se termina una partida, el jugador vencedor suma uno a la casilla correspondiente de su vector de victorias, y se recalculan los valores de la matriz de probabilidades reales.

4.4 Desarrollo del Analyzer

El analyzer se trata de una interfaz web capaz de leer y exponer los datos de los csvs de las partidas de una manera cómoda. Para la realización de esta interfaz, se han utilizado una estructura básica del desarrollo de front-end web. En concreto, se han utilizado estos 3 archivos para crearla:

- Index.html: Define la estructura básica de la página con un botón para cargar el archivo y una tabla para mostrar los resultados. Incluye enlaces a los archivos CSS y JavaScript.
- Styles.css: Proporciona estilos para la página, asegurando que la interfaz sea limpia. Estiliza el botón, la tabla y otros elementos para que sean visualmente agradables.
- Script.js: Almacena los scripts de la lógica de la aplicación, que son los siguientes:
    - loadData: Lee el archivo JSON seleccionado por el usuario y lo analiza.
    - calcularVictoriasPorEstrategia: Calcula el número total de victorias por cada estrategia.
    - calcularPorcentajesVictorias: Calcula el porcentaje de victorias para cada estrategia.
    - mostrarResultados: Muestra los resultados en la tabla

Figura 22: Interfaz de inicio del Analyzer

La utilización de la herramienta es muy simple. Al ejecutarla, aparece un botón con "Cargar archivo" de la figura 22, que permite cargar el JSON de la última partida realizada en esa traza, ya que con ella, se obtiene toda la información en el resumen de los agentes, en este caso, el número de victorias.

Estrategia	Total de Victorias	Porcentaje de Victorias (%)
agresiva	1563	31.17%
pasiva	2867	57.17%
cartas	566	11.29%
aleatoria	19	0.38%
Total	5000	100.00%

Figura 23: Resultados analizados por el analyzer

Una vez seleccionado el archivo, como se observa en la figura 23, la interfaz calcula el porcentaje de victorias por estrategia, mostrando una tabla en pantalla.

5. Resultados

En este apartado, se analizarán los resultados obtenidos por cada una de las aproximaciones seguidas en el último apartado, haciendo hincapié en las directrices de lo que se quiere conseguir con este análisis.

Para el análisis de los agentes básicos que se han desarrollado, se ha decidido evaluar su eficacia frente al RandomBot modificado sin comercio en un total de 5000 partidas. En los otros casos, se ha decidido evaluar también su rendimiento en 5000 partidas, menos en las los escenarios que presentaban una mayor aleatoriedad, que se han decidido evaluar 10000 partidas.

Para estos últimos, se han decidido generar distintos escenarios, los cuales son los siguientes:

- Todas las estrategias tienen la misma probabilidad (Aleatorio), *5.3.1*
- Siguen sus conocimientos un 50% de las veces (Eligen_la_mitad), *5.3.2*
- Siguen sus conocimientos un 80% de las veces (Elige_80), *5.3.3*
- Siguen sus conocimientos siempre (Elige_siempre), *5.3.4*
- Siguen sus conocimientos un 50% de las veces, pero comienzan teniendo una estrategia preferida (Preferencia), *5.3.5*

Estos escenarios serán explicados con más detalles en sus correspondientes apartados.

Con estos escenarios, se han establecido unas directrices para poder centrar el análisis de resultados, esperando los siguientes resultados:

- Número de partidas ganadas por cada estrategia en cada escenario. Aquí, se debería observar como las estrategias desarrolladas son superiores a la aleatoria.
- Determinar si es relevante el orden de turno. En este caso, cabe analizar si el orden de turno es muy relevante a la hora de ganar o no partidas.
- Determinar si los agentes encuentran alguna estrategia superior. Con ello, se obtendrá la mejor estrategia desarrollada.
- Determinar si los agentes encuentran alguna estrategia inferior. Dependiendo si tienen mucha o poca capacidad de decisión, se verá cómo van descartando dicha estrategia.

- Determinar si, mediante el escenario basado en preferencias, son capaces de cambiarlas si ganan más partidas con otras estrategias.

Como se ha explicado en apartados anteriores, existen partidas en las cuales los jugadores no pueden hacer nada más que esperar que les salgan los números de los materiales que necesitan, y si estos tienen poca probabilidad, es difícil acabar la partida.

Tipo de partida	Partidas no acabadas	Total de partidas	Porcentaje
FSM	48	5000	0,96%
Reglas	60	5000	1,20%
Aleatorio	440	10000	4,40%
Eligen_la_mitad	192	10000	1,92%
Elige_80	94	5000	1,88%
Elige_siempre	395	5000	7,90%
Preferencia	125	5000	2,50%

Tabla 1: Número de partidas con más de 500 rondas

Después de incorporar el límite de 500 rondas, como se puede observar en la tabla 1, las partidas no acabadas suponen un porcentaje relativamente bajo con respecto al total. Sin embargo, se van a dejar fuera del análisis en los siguientes puntos, para no tener ruido en los resultados.

Además, cuando se hable de las matrices de peso, cabe destacar que el orden de aparición de cada estrategia en estas matrices es el siguiente:

[“Agresivo”, “Pasivo”, “Cartas”, “Aleatorio”]

5.1 Resultado simulación FSM contra RandomBot

Tras el desarrollo del agente basado en una máquina de estados finitos, se ha realizado una simulación de 5000 partidas, siendo está la configuración de los agentes de cada jugador:

- Jugador 1: EstadosFinitosBot
- Jugador 2: RandomBot
- Jugador 3: RandomBot
- Jugador 4: RandomBot

Tras la simulación, se perseguía el objetivo de que el agente desarrollado en este proyecto ganara al menos el 70% de las partidas frente a los bots aleatorios. Esto, significaría que el agente toma mejores decisiones que si lo hiciera de manera aleatoria, ya que en un estado "normal", dónde cada agente tuviera las mismas oportunidades y comportamientos, se obtendría un 25%. Tras la simulación de las 5000 partidas, los resultados obtenidos son los siguientes:

Estados finitos		
Jugador	**Partidas ganadas**	**Porcentaje**
1: EstadosFinitosBot	4154	83%
2: RandomBot	252	5%
3: RandomBot	310	6,2%
4: RandomBot	284	5,8%

Tabla 2: Número de victorias con FSM

Como se puede observar en la tabla 2, el agente basado en la máquina de estados finitos gana un 83% de las partidas. Con ello, se obtiene el resultado esperado, un porcentaje elevado de victorias contra jugadores que juegan de manera aleatoria.

5.2 Resultado simulación Reglas contra RandomBot

De forma similar al agente basado en la máquina de estados finitos, el agente basado en reglas se ha evaluado utilizando la misma metodología. En este caso, la configuración de jugadores es la siguiente:

- Jugador 1: ReglasBot
- Jugador 2: RandomBot
- Jugador 3: RandomBot
- Jugador 4: RandomBot

Tras las 5000 partidas, los resultados han sido los siguientes:

Reglas		
Jugador	**Partidas ganadas**	**Porcentaje**
1: ReglasBot	3862	77,2%
2: RandomBot	358	7,1%
3: RandomBot	387	7,8%
4: RandomBot	393	7,9%

Tabla 3: Número de partidas con sistema basado en reglas

Tras esto, como se puede ver en la tabla 3, se obtiene un porcentaje de victorias del 77% del agente basado en reglas y, por tanto, se obtiene un resultado satisfactorio del mismo.

5.3 Partidas ganadas por escenario se simulación de pruebas

En este apartado, se analizarán escenarios que se ha propuesto para el desarrollo de una estrategia avanzada dentro de la herramienta. Para ello, se van a analizar las victorias de cada tipo de estrategia en cada uno de los escenarios, obteniendo las mejores estrategias, así como demostrando que son mejores que una estrategia aleatoria.

5.3.1 Todas las estrategias tienen la misma probabilidad

En este primer escenario, los agentes no tendrán en cuenta su matriz de victorias. Siempre tendrán que elegir aleatoriamente entre alguna de las 4 estrategias, todas con la misma probabilidad. Se ha hecho esta aproximación como grupo de control y, además, para ver cómo afecta el orden de turno de manera más notoria, ya que ningún jugador sigue una estrategia determinada.

Aleatorio		
Estrategia	**Partidas ganadas**	**Porcentaje**
Agresivo	2945	29,5%
Cartas	2463	24,6%
Pasivo	2951	29,5%
RandomBot	1201	12,0%

Tabla 4: Partidas ganadas por tipo de estrategia en "simulación aleatoria"

En esta simulación, se ha obtenido el resultado esperado. En la tabla 4 se observa como cada estrategia que no es la aleatoria tiene un porcentaje de victorias de entorno al 30%. Esto hace que, por lo general, si los jugadores no tienen preferencias y eligen de manera aleatoria la estrategia que utilizan, se distribuyen las victorias de manera normal entre las distintas estrategias. Además, también se concluye que la estrategia aleatoria es

inferior a las demás, siendo el porcentaje de victoria con estrategia de un 88%, frente a un 12% eligiendo de manera aleatoria.

5.3.2 Siguen sus conocimientos un 50% de las veces

En este segundo escenario, cuando a cada jugador se le da la opción de cambiar de bot, tiene un 50% de probabilidades de seguir su vector de probabilidades, o de usar el que hace que todo tenga la misma probabilidad. Haciendo este tipo de cambios, se puede comenzar a ver cómo van cambiando los resultados.

Elige mitad		
Estrategia	**Partidas ganadas**	**Porcentaje**
Agresivo	3341	33,4%
Cartas	2440	24,4%
Pasivo	3115	31,2%
RandomBot	709	7,1%

Tabla 5: Partidas ganadas por tipo de estrategia en simulación "eligiendo la mitad"

En la tabla 5, se observa cómo una vez los jugadores ya se les permite comenzar a decidir que estrategia quieren en función de las anteriores victorias, ganan menos con una estrategia aleatoria. Hablamos de que el 92.9% de victorias son de las estrategias con los agentes desarrollados en este proyecto. Además, ya se puede empezar a discernir que estrategias son más proclives a ganar partidas, pero conforme vayan pudiendo elegir más, se verá más claro.

5.3.3 Siguen sus conocimientos un 80% de las veces

En este caso, ya existe únicamente 1/5 de probabilidades en los cuales el bot no sigue sus conocimientos. Aquí ya se comienza a dejar el factor azaroso atrás.

Elige 80		
Estrategia	**Partidas ganadas**	**Porcentaje**
Agresivo	1709	34,2%
Cartas	1093	21,9%
Pasivo	1720	34,4%
RandomBot	286	5,7%

Tabla 6: Partidas ganadas por tipo de estrategia en simulación "eligiendo el 80%"

Similar a la anterior, y como se puede ver en la tabla 6, al elegir en un mayor número de ocasiones la estrategia que quiere seguir el jugador, disminuye aún más el número de partidas que ganan de manera aleatoria, y poco a poco se comienzan a separar las estrategias que tienen mayor éxito, como son la agresiva y la pasiva.

5.3.4 Siguen sus conocimientos siempre

Con esta estrategia, los agentes siempre tienen en cuenta su matriz de probabilidades, dando pie a que sigan la estrategia óptima, teniendo en cuenta que sea con la que más veces ha ganado.

Elige siempre		
Estrategia	**Partidas ganadas**	**Porcentaje**
Agresivo	1538	30,8%
Cartas	537	10,7%
Pasivo	2818	56,4%
RandomBot	13	0,3%

Tabla 7: Partidas ganadas por tipo de estrategia en simulación "eligiendo siempre"

En este último caso de elección, al dejar decidir siempre al jugador que estrategia desea utilizar, una vez avanza el número de partidas, comienzan a darse cuenta que la estrategia aleatoria no proporciona victorias. Es por ello, que dejan de usarla casi por completo, como se puede observar en la tabla 7, de las 5000 partidas jugadas, solo se han ganado un 0.3% de las partidas con una estrategia aleatoria. Además, y como ya se ha avanzado en los anteriores puntos, se definen dos estrategias dominantes, como son la agresiva y la pasiva. Sin embargo, la estrategia pasiva gana más de la mitad de las partidas, siendo la que mayores victorias les ha reportado en esta simulación.

5.3.5 Siguen sus conocimientos un 50%, pero con preferencia

Con este último caso, se quiere comprobar que si, por ejemplo, un jugador comienza prefiriendo jugar de manera pasiva, si ve que gana más siendo agresivo, si cambiarían sus preferencias. Se ha añadido el elemento de que puedan no seguir sus conocimientos para propiciar a que jueguen con otras estrategias.

Las preferencias que se han elegido son las siguientes:

- Jugador 0: Estrategia pasiva
- Jugador 1: Estrategia agresiva
- Jugador 2: Estrategia agresiva
- Jugador 3: Estrategia de Cartas

Aleatorio		
Estrategia	Partidas ganadas	Porcentaje
Agresivo	2232	44,6%
Cartas	852	17,0%
Pasivo	1765	35,3%
RandomBot	26	0,5%

Tabla 8: Partidas ganadas por tipo de estrategia en simulación "preferencia"

En este escenario, como se puede observar en la tabla 8, se mantiene lo ya visto en los otros con respecto a la estrategia aleatoria. Además, se refuerza la hipótesis que la estrategia basada en cartas es inferior a las dos propuestas. Por último, entre las dos que proporcionaban un mayor número de victorias, en este caso, reporta un porcentaje superior la estrategia agresiva, seguramente porque dos de los 4 jugadores comenzaban con esta estrategia como preferida.

5.4 Orden de turno

Orden de turno	Partidas ganas	Porcentaje
0	13009	38,5%
1	6224	18,4%
2	7850	23,3%
3	6671	19,8%

Tabla 9: Partidas ganadas orden de turno

Como se puede observar en la tabla 9, el orden de turno determina en gran medida el número de victorias. Como se puede observar, el jugador que comienza las partidas gana un 38,5% de las partidas, un 13,5% más si no fuera relevante el orden de turno.

5.3 Mejores estrategias

Global		
Estrategia	Partidas ganadas	Porcentaje
Agresivo	11765	34,9%
Cartas	7385	21,9%
Pasivo	12369	36,6%
RandomBot	2235	6,6%

Tabla 10: Partidas ganadas por tipo de estrategia

En los puntos anteriores, se ha visto como los agentes tenían tendencia a dos de las 3 estrategias propuestas. En la tabla 10 se ve como esta tendencia se confirma, y las estrategias agresivas y pasivas con más interesantes para ganar las partidas.

5.4 Resultados en la aproximación por preferencias

Preferencias		
Jugador	Matriz inicial	Matriz final
0	[6.0, 82.0, 6.0, 6.0]	[13.74, 82.07, 3.03, 1.14]
1	[82.0, 6.0, 6.0, 6.0]	[90.95, 4.14, 3.67, 1.22]
2	[82.0, 6.0, 6.0, 6.0]	[84.78, 11.73, 2.82, 0.65]
3	[6.0, 6.0, 82.0, 6.0]	[6.13, 13.12, 79.88, 0.86]

Tabla 11: Variación de la matriz de pesos en simulación por preferencias

Con este análisis, se pretendía observar si los agentes varían o no sus preferencias cuando comienzan con una ya predefinida. Recordar que, los números que aparecen en las listas de la tabla 11 son el porcentaje por el cual eligen cada una de las estrategias, siguiendo el siguiente orden:

["Agresivo", "Pasivo", "Cartas", "Aleatorio"]

Se ve como los agentes han continuado teniendo como preferida su estrategia original. Sin embargo, existen algunos matices a destacar en los resultados, que a lo mejor se hubieran hecho más evidentes si se ejecutaran más partidas:

- Los jugadores han bajado mucho su porcentaje de victorias con el RandomBot. Era de esperar, ya que, aunque lo elijan, es raro que ganen con él, y por tanto, tienden a utilizarlo menos.
- Los 3 primeros jugadores han aumentado sus victorias con la estrategia Agresiva. Sin embargo, el cuarto las ha mantenido. Presumiblemente, el jugador que va último en la partida no se aprovecha tan bien de las estrategias agresivas como el resto de los jugadores.

En definitiva, con el número de partidas que se han ejecutado, no se hace visible un cambio grande en la preferencia de cada jugador. Sin embargo, parece que, con un número muy elevado de partidas, se acercarían cada vez más a las estrategias vistas en los puntos anteriores.

5.5 Resultados de "siempre eligen"

Siempre eligen		
Jugador	Victorias	Matriz final
0	1806	[55.23, 40.40, 4.19, 0.16]
1	892	[36.96, 27.26, 35.11, 0.65]
2	1262	[14.94, 77.62, 6.88, 0.54]
3	946	[0.52, 91.03, 8.13, 0.31]

Tabla 12: Resultados simulación donde siempre eligen.

En la tabla 12, se puede observar cómo han acabado las partidas cada uno de los jugadores en el escenario de “siempre eligen”. Es interesante, ya que se puede observar lo siguiente:

- Los jugadores en primera y tercera posición tienen más victorias que los otros dos jugadores.
- Se puede observar que, la estrategia agresiva es mejor conforme antes comience la partida el jugador. Por tanto, es una estrategia muy dependiente del orden de turno.
- Al igual que la agresiva, la estrategia pasiva también se observa como aumenta su porcentaje de victorias conforme más tarde le llegue el turno al jugador.
- Por último, la estrategia de cartas, parece tener un mayor porcentaje de victorias si el jugador comienza segundo en la partida.

5.6 Reflexión de los resultados

Tras la extracción de resultados, se ha podido observar algunas cuestiones que permiten establecer algunas generalizaciones sobre las estrategias seguidas, las limitaciones que se han establecido al inicio del proyecto, y las técnicas aplicadas en cada uno de los agentes:

1- El orden de turno es muy importante, ya que, al comienzo de la partida, el primero siempre tiene ventaja ya que posee todos los nodos del tablero para colocar su poblado, y cuando coloca el segundo poblado, solo existen 4 en el tablero, ya que han colocado los otros 3 jugadores. Con ello, es más probable que consiga todos los recursos rápido en la partida.

2- Las estrategias implementadas (Agresiva, pasiva, y basada en cartas) son bastante equitativas y tienen probabilidades similares a la hora de ganar. Sin embargo, la estrategia basada en cartas se queda un poco por detrás, seguramente porque si alguien más en la partida construye cartas, esas cartas del mazo dejan de existir para el jugador que decide adoptar esta estrategia, y por tanto, tiene menos probabilidades de robar las cartas que necesita.

3- En general, las estrategias implementadas son mucho más eficientes que una estrategia aleatoria. Si bien es cierto, no se ha conseguido erradicar a la estrategia aleatoria, pero se ha conseguido que, cuando los jugadores tienen la capacidad de elegir, la estrategia aleatoria gane un 0.3% de las partidas, porcentaje muy alentador y que arroja los buenos resultados obtenidos.

4- También se ha podido observar que, dependiendo del orden de juego, unas estrategias funcionan mejores que otras. Al ser el jugador inicial, el jugador

agresivo va “un turno por delante”, haciendo siempre él las acciones primero, pudiendo bloquear los pasos del resto de jugadores. Sin embargo, los últimos jugadores prefieren decantarse por una estrategia pasiva, no compitiendo con los jugadores iniciales por terreno, y tratando de expandirse por zonas costeras.

5- Por último, se puede observar un descenso significativo de las victorias por agentes no aleatorios cuando se incluye el modelo avanzado. Con esto, podemos afirmar que los jugadores tienen más tendencia a ganar con estrategias no aleatorias si se ejecuta el método avanzado.

6. Conclusiones

El presente trabajo de fin de grado se ha realizado con el propósito de ampliar el trabajo realizado por Adrián Heras en el año 2023. En él, se planteaba un entorno de simulación controlado de agentes inteligentes previamente desarrollados, con el fin de otorgar a alumnos de la universidad y otros interesados un entorno en el cual probar distintas estrategias y metodologías basadas en inteligencia artificial. Con este fin de ampliarlo, se han desarrollado un total de 5 agentes, basados en una máquina de estados finitos y en sistemas basados en reglas.

Además, con el fin de aumentar el alcance del proyecto de PyCatan, se han añadido nuevos desarrollos dentro de la herramienta, así como la corrección de algunos errores fruto del alcance limitado del proyecto en su desarrollo. Entre otros, se ha añadido un nuevo elemento al sistema que permite analizar las partidas, así como un nuevo modo que permite tratar a cada jugador como un agente inteligente capaz de elegir su estrategia de juego en base a ciertos parámetros.

El objetivo final del proyecto era el diseño y desarrollo de agentes basados en heurística simple, así como la aplicación de algún modelo avanzado que permita variación en los resultados. Con los resultados obtenidos, se puede observar que el objetivo se ha cumplido.

Con la creación de los agentes basados en heurística simple, en concreto, la máquina de estados finitos, se han desarrollado agentes con estrategias distintas a la pensada inicialmente, permitiendo generar un modelo capaz de, dependiendo de los resultados en las partidas anteriores, fuera capaz de elegir qué estrategia era mejor para la siguiente partida.

6.1 Posibles mejoras

Dentro del proyecto se ha eludido por completo el carácter de negociación dentro del juego. Si bien es un aspecto muy importante dentro del juego de mesa Catán y otorga mucho más dinamismo a las partidas, aumentaba mucho el alcance de este proyecto. Es por ello que, cuando se planteo el presente trabajo, se decidió dividir esta parte en otro proyecto, ya que las estrategias a seguir dentro de la negociación pueden ser muy distintas a lo desarrollado en este proyecto. Sin embargo, sería muy interesante llegar a poder combinar ambos proyectos, consiguiendo agentes que sean capaces de aprender de las distintas estrategias en la colocación y construcción de fichas, así como

de como negocian sus contrincantes y que le interesa más negociar en cada momento de la partida.

Otro aspecto relevante que no se ha tratado con una extensa profundidad ha sido el analizador. Si bien se ha creado un programa funcional, capaz de obtener los resultados de una manera visual y rápida, existen muchas más métricas que pueden aportar información a la hora de analizar los resultados. Es por ello que sería interesante aumentar esta parte de la infraestructura, con nuevas métricas o gráficas que permitan una explotación de datos más exhaustiva e interesante.

6.2 Competencias trasversales

Dentro de este proyecto, se han desarrollado las siguientes competencias transversales adquiridas en la universidad: *Comprensión e integración de conocimientos, Aplicación y pensamiento práctico, Innovación, creatividad y emprenduría y Pensamiento crítico*.

6.3 Objetivos y metas de desarrollo sostenible

Este proyecto está estrechamente vinculado con el Objetivo de Desarrollo Sostenible número 4, que busca garantizar una "Educación de calidad", ya que, en la educación universitaria, este proyecto ofrece una solución tecnológica avanzada que promueve el aprendizaje en áreas clave como la lógica y las matemáticas. A través de esta iniciativa, se pretende no solo fortalecer las competencias técnicas de los futuros estudiantes que utilicen estos agentes como base, sino también fomentar el desarrollo del pensamiento crítico y la resolución de problemas. Se puede encontrar más información al respecto en el anexo 1.

Referencias

Adeniyi, A. E., Brahma, B., Adebiyi, M. O., Awotunde, J. B., Jimoh, R. G., Olasinde, E., & Bandyopadhyay, A. (2024). Development of Two Dimension (2D) Game Engine with Finite State Machine (FSM) Based Artificial Intelligence (AI) Subsystem. *Procedia Computer Science*, *235*, 2996-3006.

Candra, A., Budiman, M. A., & Pohan, R. I. (2021, June). Application of a-star algorithm on pathfinding game. In *Journal of Physics: Conference Series* (Vol. 1898, No. 1, p. 012047). IOP Publishing.

Cao, L. (2020). AI in finance: A review. *Available at SSRN 3647625*.

Galdames, I. S. (2023). Inteligencia artificial en Medicina Humana. *International Journal of Medical and Surgical Sciences*, *10*(1), 1-4.

Heras Reche, A. (2023). *PyCatan: Desarrollo de un entorno de simulación de Settlers of Catan para la implementación de bots inteligentes* (Doctoral dissertation, Universitat Politècnica de València).

Muggleton, S. (2014). Alan Turing and the development of Artificial Intelligence. *AI communications*, *27*(1), 3-10.

Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., & Lucas, S. M. (2019). General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, *11*(3), 195-214.

Shannon, C. E. (1950). XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), 256-275.

Sharma, S., & Sharma, V. How AI Transforms Play: The Evolution of Artificial Intelligence in Video Games. *Media and AI: Navigating*, 151.

Recursos en línea:

Coppelia Robotics. (n.d.). https://www.coppeliarobotics.com/

NetLogo. (n.d.). https://ccl.northwestern.edu/netlogo/

Repast. (n.d.). https://repast.github.io/

Wikipedia. (n.d.). *Los colonos de Catán*. https://es.wikipedia.org/wiki/Los_colonos_de_Cat%C3%A1n (Consultado el 12 de julio de 2023).

Palanca, J. (n.d.). PyGOMAS. GitHub. https://github.com/javipalanca/pygomas

ANEXO 1 – Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**x**
ODS 2. **Hambre cero.**				**x**
ODS 3. **Salud y bienestar.**				**x**
ODS 4. **Educación de calidad.**	**x**			
ODS 5. **Igualdad de género.**				**x**
ODS 6. **Agua limpia y saneamiento.**				**x**
ODS 7. **Energía asequible y no contaminante.**				**x**
ODS 8. **Trabajo decente y crecimiento económico.**				**x**
ODS 9. **Industria, innovación e infraestructuras.**				**x**
ODS 10. **Reducción de las desigualdades**.				**x**
ODS 11. **Ciudades y comunidades sostenibles**.				**x**
ODS 12. **Producción y consumo responsables.**				**x**
ODS 13. **Acción por el clima.**				**x**
ODS 14. **Vida submarina.**				**x**
ODS 15. **Vida de ecosistemas terrestres.**				**x**
ODS 16. **Paz, justicia e instituciones sólidas.**				**x**
ODS 17. **Alianzas para lograr objetivos.**				**x**

Este proyecto se alinea de manera significativa con el Objetivo de Desarrollo Sostenible (ODS) número 4, que persigue la meta de garantizar una "Educación de calidad". El ODS 4 es crucial en la promoción de oportunidades educativas inclusivas y equitativas, así como en el fomento del aprendizaje a lo largo de toda la vida para todos. En el contexto de la educación superior, este proyecto se erige como una herramienta tecnológica avanzada que tiene el potencial de transformar la enseñanza y el aprendizaje en disciplinas fundamentales como la lógica y las matemáticas. Mediante la implementación de esta solución innovadora, se busca no solo fortalecer las competencias técnicas de los estudiantes, sino también contribuir al desarrollo de habilidades esenciales para el siglo XXI, como el pensamiento crítico y la capacidad de resolver problemas complejos.

La relevancia de este proyecto en relación con el ODS 4 radica en su capacidad para abordar algunos de los desafíos persistentes en la educación superior, particularmente aquellos relacionados con l Este proyecto se alinea de manera significativa con el Objetivo de Desarrollo Sostenible (ODS) número 4, que persigue la meta de garantizar una "Educación de calidad". El ODS 4 es crucial en la promoción de oportunidades educativas inclusivas y equitativas, así como en el fomento del aprendizaje a lo largo de toda la vida para todos. En el contexto de la educación superior, este proyecto se erige como una herramienta tecnológica avanzada que tiene el potencial de transformar la enseñanza y el aprendizaje en disciplinas fundamentales como la lógica y las matemáticas. Mediante la implementación de esta solución innovadora, se busca no solo fortalecer las competencias técnicas de los estudiantes, sino también contribuir al desarrollo de habilidades esenciales para el siglo XXI, como el pensamiento crítico y la capacidad de resolver problemas complejos.

La relevancia de este proyecto en relación con el ODS 4 radica en su capacidad para abordar algunos de los desafíos persistentes en la educación superior, particularmente aquellos relacionados con la enseñanza de materias que son fundamentales para la formación de profesionales en un mundo cada vez más digitalizado. Las competencias en lógica y matemáticas son esenciales no solo para carreras específicas, como la ingeniería o las ciencias exactas, sino también para un amplio espectro de disciplinas que requieren un pensamiento estructurado y la capacidad de analizar y resolver problemas de manera efectiva. En este sentido, el proyecto proporciona una plataforma que facilita un aprendizaje más profundo y personalizado, adaptándose a las necesidades individuales de los estudiantes y permitiendo una comprensión más sólida de los conceptos.

Además, este proyecto se inserta en una tendencia global que busca integrar la tecnología de manera efectiva en los procesos educativos. Las herramientas tecnológicas, cuando se utilizan de manera adecuada, tienen el potencial de enriquecer la experiencia de aprendizaje, haciéndola más interactiva, accesible y acorde con las demandas del entorno contemporáneo. Este proyecto, en particular, se destaca por su enfoque en áreas que tradicionalmente han sido percibidas como difíciles por los estudiantes, ofreciendo recursos que pueden hacer más accesibles y comprensibles conceptos abstractos, como los que se encuentran en la lógica y las matemáticas.

Asimismo, al fomentar el desarrollo de competencias en pensamiento crítico y resolución de problemas, el proyecto se alinea con otro aspecto fundamental del ODS 4, que es preparar a los estudiantes no solo para superar los desafíos académicos, sino también para enfrentar los retos del mundo real. El pensamiento crítico es una habilidad transversal que permite a los individuos analizar información de manera rigurosa, cuestionar suposiciones y tomar decisiones informadas. Por otro lado, la capacidad de resolver problemas es crucial en prácticamente todas las esferas de la vida, desde el ámbito profesional hasta el personal.