Introducción

Universal Paperclips es un juego de simulación incremental en el que el jugador produce y vende clips de papel, comenzando con acciones manuales simples y progresando hacia la automatización completa de la producción mediante algoritmos y sistemas de optimización. A medida que aumenta la eficiencia, el juego introduce la gestión de recursos, la inversión en investigación tecnológica y la expansión hacia objetivos de mayor escala, mostrando cómo un agente con un objetivo aparentemente trivial puede generar efectos complejos y de gran alcance.

El desafío principal es maximizar la eficiencia en la producción, ventas, precios y automatización, lo que lo convierte en un entorno ideal para probar técnicas de aprendizaje por refuerzo (Reinforcement Learning, RL).

El juego se relaciona directamente con el concepto de "paperclip maximizer" en inteligencia artificial, propuesto por Nick Bostrom. Este concepto describe un agente de IA hipotético que tiene como único objetivo maximizar la cantidad de clips de papel. A pesar de que su meta es trivial, un agente suficientemente avanzado podría desplegar estrategias altamente eficaces y potencialmente destructivas para alcanzar su objetivo, demostrando cómo un objetivo mal alineado con los valores humanos puede conducir a resultados indeseables. Universal Paperclips sirve como una ilustración lúdica de este principio, mostrando de manera interactiva cómo la optimización sin restricciones puede escalar a consecuencias imprevistas.

El objetivo de nuestro trabajo es entrenar un modelo de Machine Learning que sea capaz de jugar y ganar el juego. Una vez logrado esto, como objetivo secundario, se propone que el modelo sea capaz de resolver el juego en tiempos cada vez menores.

Identificamos las siguientes dificultades que se presentan a la hora de intentar resolver este problema con técnicas de Machine Learning:

- Las estrategias óptimas no son triviales: las decisiones iniciales (por ejemplo, cuándo invertir en automatización o subir precios) afectan drásticamente el rendimiento futuro.
- Una estrategia óptima requiere sacrificar ciertas métricas positivas a corto plazo para obtener un mejor rendimiento en el futuro. Por ejemplo, ahorrar recursos para poder obtener mejoras en la producción.
- El juego consta de tres etapas claramente definidas. Las mecánicas de cada etapa difieren bastante entre sí, así como las estrategias más óptimas en cada una. Un modelo de Machine Learning podría aprender a desempeñarse muy bien en una etapa y luego tener dificultades para adaptarse a las demás.
- El juego presenta durante toda una partida, alrededor de 70 posibles acciones únicas que el jugador puede tomar. Una representación del entorno podría llegar a tener cerca de 200 variables. Esta cantidad de datos podría ser problemática para algunos tipos de algoritmos.
- El juego simula el paso del tiempo, por lo que la técnica utilizada deberá ser capaz de tener

esto en cuenta, y como objetivo secundario, minimizar el tiempo total necesario para ganar.

- Algunas variables de estado del juego tienen rangos de variación muy grandes (e.g. la cantidad de clips producidos varía desde 0 a 3x10^55). Esta variación es problemática para algunas técnicas de Machine Learning, por lo que deberán normalizarse apropiadamente muchas variables.

Marco Teórico

Aprendizaje por Refuerzo (Reinforcement Learning – RL)

El **aprendizaje por refuerzo** (RL, por sus siglas en inglés) es un paradigma del aprendizaje automático en el cual un **agente** aprende a tomar decisiones mediante la **interacción continua con un entorno**.
A diferencia del aprendizaje supervisado, donde el modelo aprende a partir de ejemplos con respuestas correctas, en RL el agente **descubre estrategias óptimas** a través de la **experiencia directa**, buscando **maximizar una recompensa acumulada** a lo largo del tiempo.

El proceso de aprendizaje se basa en la retroalimentación: el agente ejecuta una acción, el entorno responde con un nuevo estado y una recompensa, y el agente ajusta su política de decisión en función de los resultados obtenidos.

Componentes Fundamentales del Aprendizaje por Refuerzo

El marco del RL está compuesto por los siguientes elementos esenciales:

- **Agente:** entidad que aprende y toma decisiones con el objetivo de maximizar su recompensa acumulada.
- **Entorno:** sistema con el cual interactúa el agente; determina los resultados de las acciones y define la dinámica del problema.
- **Acciones (A):** conjunto de decisiones posibles que el agente puede ejecutar en cada estado.
- **Estados (S):** representación de la situación actual del entorno percibida por el agente.
- **Recompensa (R):** señal numérica que evalúa el resultado inmediato de una acción.
- **Política (π):** estrategia o función que define la probabilidad de seleccionar una acción dada un estado.
- **Función de valor (V o Q):** mide la utilidad esperada de un estado o acción bajo una política determinada, y guía el proceso de optimización.

En conjunto, estos componentes permiten formalizar el problema como un **proceso de decisión de**

Markov (MDP), donde el agente busca la política óptima que maximice la recompensa esperada.

Algoritmo Seleccionado: PPO (Proximal Policy Optimization)

El algoritmo elegido para el entrenamiento del agente es **PPO (Proximal Policy Optimization)**, una técnica moderna de aprendizaje por refuerzo basada en políticas, desarrollada por OpenAI. PPO pertenece a la familia de los métodos **actor-crítico**, en los cuales dos redes neuronales trabajan de manera conjunta: el *actor* propone las acciones que el agente debe ejecutar según la política actual, mientras que el *crítico* evalúa el valor esperado de dichas acciones con el fin de mejorar la política en las siguientes iteraciones. Esta estructura permite equilibrar la exploración y la explotación, garantizando un aprendizaje más estable y eficiente que los métodos tradicionales.

La principal característica de PPO es su **estabilidad durante el entrenamiento**, lograda mediante la introducción de una función objetivo denominada "proximal", que restringe la magnitud de las actualizaciones de la política. Este mecanismo de *clipping* evita cambios abruptos en los parámetros del modelo, reduciendo el riesgo de que el agente olvide comportamientos previamente aprendidos o que su política se desestabilice. Esta propiedad resulta particularmente útil en entornos complejos o con recompensas ruidosas, donde los ajustes de gran magnitud pueden conducir a resultados inconsistentes.

La elección de PPO se justifica frente a otras alternativas, como **DQN (Deep Q-Network)**, debido a la naturaleza del entorno a simular. El juego *Universal Paperclips* presenta un espacio de estados amplio y casi continuo, en el que intervienen múltiples variables interdependientes (número de clips, recursos, mejoras, inversión, entre otras). En este tipo de entornos, los métodos tabulares clásicos, como Q-learning, se vuelven imprácticos, ya que requerirían almacenar valores para cada posible combinación de estados y acciones, lo que implica un crecimiento exponencial y un uso de memoria inabordable. En contraste, PPO utiliza redes neuronales para aproximar tanto la política del agente como la función de valor, lo que le permite generalizar hacia estados no vistos y manejar espacios de acción grandes o continuos.

Otra ventaja relevante de PPO es su capacidad para manejar **recompensas diferidas**. En *Universal Paperclips*, muchas decisiones implican sacrificar beneficios inmediatos —como invertir recursos en mejoras o automatización— para obtener una mayor producción a largo plazo. PPO, al basarse en el cálculo de ventajas entre el retorno esperado y el valor estimado del estado, permite que el agente aprenda a equilibrar decisiones de corto y largo plazo, favoreciendo estrategias que maximicen la producción acumulada y la eficiencia general del proceso. De este modo, el agente puede desarrollar comportamientos planificados y sostenibles, en lugar de limitarse a maximizar recompensas instantáneas.

En cuanto al diseño de la función de recompensa, se propone que esta refleje tanto el progreso inmediato como los objetivos finales del juego. La **recompensa principal** debe asociarse con la cantidad total de clips producidos o con la finalización exitosa del juego, mientras que **recompensas intermedias** pueden otorgarse por incrementos en la tasa de producción o la adquisición de mejoras relevantes. Asimismo, **penalizaciones leves** por tiempo transcurrido o decisiones ineficientes pueden ayudar a que el agente aprenda a priorizar inversiones inteligentes sobre la simple fabricación continua, incentivando la planificación a largo plazo. Este equilibrio entre recompensas y penalizaciones resulta fundamental para guiar la exploración del agente sin

sesgarlo hacia comportamientos de corto alcance.

Por todo lo anterior, **PPO se considera la opción más robusta y flexible** para la experimentación inicial. Su capacidad para aprender políticas generalizables, manejar recompensas complejas y mantener la estabilidad del entrenamiento lo convierte en un método adecuado para entornos de simulación económica o de gestión como *Universal Paperclips*. Aunque no garantiza una solución óptima en términos matemáticos, ofrece un equilibrio muy efectivo entre rendimiento, estabilidad y capacidad de generalización, y constituye una base sólida para el desarrollo y evaluación de agentes de aprendizaje por refuerzo en este tipo de contextos. No obstante, se mantiene abierta la posibilidad de explorar otros algoritmos en fases posteriores del proyecto, en función de los resultados experimentales obtenidos.

Métricas de evaluación

Con el fin de evaluar el desempeño del agente y la calidad del proceso de entrenamiento, se propone un conjunto de **métricas iniciales y opcionales**.
Estas métricas no son definitivas y podrán **modificarse o ampliarse a lo largo del desarrollo del proyecto**, conforme se obtengan resultados experimentales y se identifiquen nuevas variables relevantes.

1. Métricas de Desempeño Básico

- **Recompensa acumulada total:** cantidad total de clips producidos o ganancias económicas obtenidas en un episodio.
- **Eficiencia temporal:** cantidad de clips generados por segundo o por paso de simulación.
- **Eficiencia económica:** relación entre clips producidos y unidades de inversión o costo total.
- **Tasa de mejora:** cantidad de pasos necesarios para alcanzar determinados hitos de producción (por ejemplo, 1.000.000 de clips).

2. Estabilidad y Consistencia del Comportamiento

- **Estabilidad de la política:** variación de las decisiones del agente entre episodios o ventanas de entrenamiento.
- **Volatilidad de la producción:** desviación estándar de la cantidad de clips producidos por unidad de tiempo.
- **Suavidad de las decisiones:** frecuencia de cambios drásticos en las acciones seleccionadas.
- **Persistencia de estrategias:** duración promedio de secuencias consecutivas de la misma acción.

3. Eficiencia del Aprendizaje

- **Sample efficiency:** relación entre la recompensa alcanzada y la cantidad de pasos de entrenamiento utilizados.
- **Tiempo a convergencia:** número de episodios necesarios hasta que la recompensa media se estabilice.
- **Regret acumulado:** diferencia acumulada entre el retorno óptimo esperado y el retorno efectivamente obtenido.

4. Métricas de Diagnóstico del Entrenamiento (Propias de RL)

- **Entropía de la política:** mide el equilibrio entre exploración y explotación del agente.

5. Uso de Recursos y Restricciones

- **Utilización del inventario:** porcentaje de tiempo en el que el agente mantiene niveles adecuados de alambre o recursos.
- **Costos operativos:** gasto total por unidad de producción sostenida.
- **Cumplimiento de restricciones:** número de violaciones de condiciones límite (por ejemplo, stock negativo o fondos insuficientes).

6. Robustez y Generalización

- **Resiliencia al ruido:** degradación del rendimiento frente a perturbaciones o incertidumbre en las observaciones.
- **Capacidad de transferencia:** habilidad del modelo para adaptarse a pequeñas variaciones en las reglas del entorno.

7. Fiabilidad Estadística

- **Promedio y desviación estándar:** cálculo de media y dispersión sobre múltiples ejecuciones (diferentes semillas).
- **Reproducibilidad:** variación de resultados entre corridas con la misma configuración experimental.

2. Herramientas y frameworks

Node.js:

Se empleará **Node.js** para replicar la lógica base del juego *Universal Paperclips*, utilizando el código original disponible en decisionproblem.com/paperclips/main.js y la versión modificada alojada en GitHub – jgmize/paperclips.

El objetivo es simular el entorno del juego localmente, definiendo los **estados**, **acciones** y **recompensas**, sin necesidad de conectarse al juego original ni de implementar una interfaz gráfica. De este modo, se obtiene un entorno controlado que permite el entrenamiento y prueba del agente de inteligencia artificial.

Gymnasium:

Se considera el uso de la librería **Gymnasium** (sucesora de OpenAI Gym) para definir formalmente el entorno de aprendizaje por refuerzo, lo que facilitará la comunicación entre la simulación del juego y los algoritmos de entrenamiento.

Esta integración permitiría modelar la interacción agente–entorno mediante las funciones estándar reset() y step(), compatibles con múltiples frameworks de *Reinforcement Learning*.

Entrenamiento del Agente:
Existen diversas alternativas en evaluación para la fase de entrenamiento, que deberán analizarse con mayor profundidad antes de seleccionar la más adecuada. Entre ellas se destacan:

1. **Integración Node.js – Python:**
   Ejecutar la simulación del juego como un servidor local en Node.js y utilizar **Stable-Baselines3** (Python) para entrenar el modelo PPO (Proximal Policy Optimization). La comunicación se realizaría mediante solicitudes HTTP o WebSocket, enviando las acciones del agente y recibiendo los estados y recompensas del entorno.

2. **Implementación completa en JavaScript:**
   Desarrollar tanto la simulación como el entrenamiento dentro del ecosistema JavaScript, utilizando librerías como **TensorFlow.js**, **tfjs-rl** o **reinforce-js**.
   Esta opción permitiría mantener todo el flujo en un mismo entorno tecnológico, aunque con menor madurez y documentación respecto a las soluciones disponibles en Python.

3. **Entrenamiento distribuido con RLlib:**
   Explorar el uso de **RLlib** (basado en Ray) para realizar entrenamientos más complejos o distribuidos. En este caso, podría integrarse con Node.js mediante un *wrapper* o capa de comunicación que permita conectar el entorno simulado con el motor de aprendizaje Python.

Opciones de Comunicación:
Para la conexión entre el entorno simulado (Node.js) y el agente de entrenamiento (Python o JS) se evaluarán principalmente dos alternativas:

- **Express:** Framework web minimalista que permitiría definir endpoints REST (/reset, /step,

etc.) para intercambiar información entre el juego y la IA de forma sencilla y estructurada.

- **Socket.io:** Librería orientada a la comunicación en tiempo real, adecuada para escenarios donde se requiera transmisión continua de datos (por ejemplo, visualizar en vivo la ejecución del agente).

Ambas opciones presentan ventajas distintas y se analizarán en función del desempeño, facilidad de integración y necesidades del proyecto.

Visualización y Monitoreo:

Para el seguimiento del proceso de entrenamiento se podría utilizar **TensorBoard**, herramienta que permite visualizar métricas de rendimiento, recompensas acumuladas, pérdidas de política y otros indicadores relevantes durante la evolución del agente.

Ejecución opcional en entorno gráfico:

De manera experimental, se evaluará la posibilidad de integrar herramientas como **Playwright** o **Selenium** para cargar el juego en un navegador y permitir que el agente interactúe directamente con la interfaz web original.

Sin embargo, esta opción no constituye el objetivo principal del proyecto, ya que implica mayores requerimientos de procesamiento, tiempos de ejecución más prolongados y una complejidad técnica superior. Solo se considerará en etapas avanzadas si los recursos y el cronograma lo permiten.