UNIVERSIDAD DE
LA LAGUNA

PROYECTO JAVIVI

Informe del proyecto

09 DE FEBRERO DE 2022

Enlace Github:
https://github.com/alu0101233598/JAVIVI-SI122

Realizado por
VIREN S. DHANWANI DHANWANI
JAVIER CORREA MARICHAL
GABRIEL GARCÍA JAUBERT

ÍNDICE

Enlace al Acta de Constitución

JAVIVI

Propuesta

Descripción

El proyecto consta de diversos escenarios en los que uno o varios agentes inteligentes deberán superar alguna prueba por medio de aprendizaje automático. Dichos escenarios dispondrán de obstáculos, posiciones iniciales aleatorias y una meta, dando importancia a la colaboración entre agentes.

Proyectos similares

- Coche autónomo: Una idea para el proyecto fue crear un agente que actuara como coche autónomo, y que aprendiera a aparcar por sí sólo. Tras esta propuesta se encuentra un concepto parecido al trabajo escogido, ya que también debemos desarrollar un agente que aprenda a desenvolverse en entornos diferentes.
- CoinRun: otro proyecto que encontramos interesante fue un generador de escenarios aleatorios, donde un agente aprendía a cumplir un objetivo, en este caso recoger una moneda, como si de un videojuego se tratase.

Recursos a usar

- Unity: Para la representación gráfica de nuestro entorno, así como la de los agentes, utilizaremos Unity, un potente motor de videojuegos que nos permite crear escenarios y diseñar agentes con facilidad.
- ML-Agents: Para implementar aprendizaje automático, utilizaremos el paquete ML-Agents, que proporciona un framework para crear, entrenar, y probar agentes haciendo uso de Machine-Learning. Este paquete está construido sobre Tensorflow y PyTorch.
- Cuda Developer Toolkit: Debido al alto rendimiento necesario para entrenar a los agentes en un tiempo razonable, el uso de Cuda Developer Toolkit se vuelve imprescindible. Este software permite a Unity utilizar los núcleos de las tarjetas gráficas como principal

unidad de cómputo, y no la CPU, de esta manera aceleramos el proceso de entrenamiento.

- Google Drive: Los archivos de Unity requieren más almacenamiento de lo que un repositorio de GitHub corriente es capaz de ofrecer, por lo que los archivos los compartimos entre el equipo con Google Drive.
- Google Docs: Necesitaremos una herramienta que permita redactar el informe de manera colaborativa, y Google Docs permite esto.

Tecnologías de IA

Enlace a la presentación realizada:

https://docs.google.com/presentation/d/1oS-BRN2t7dm30xy61yEUySpCrC86H3IHdLkLNna9hEM/edit?usp=sharing

Para la realización de este trabajo, se ha elegido utilizar el conjunto de herramientas de Aprendizaje Automático para agentes en Unity (ML-Agents Toolkit). Este proyecto de carácter Open Source incluye herramientas para la creación de "cerebros de agentes" a través de diversas técnicas y mecanismos de aprendizaje a través del Machine Learning. La tecnología más prominente ofrecida por este set de desarrollo incluye los siguientes modos de aprendizaje:

- Deep Reinforcement Learning. El Aprendizaje por Refuerzo o Reinforcement Learning considera el problema de un agente computacional que aprende a tomar decisiones por ensayo y error. El Aprendizaje por Refuerzo Profundo (ARP) incorpora el aprendizaje profundo a la solución, lo que permite a los agentes tomar decisiones a partir de datos de entrada no estructurados sin necesidad de ingeniería manual del espacio de estados. Los algoritmos de ARP son capaces de tomar entradas muy grandes (por ejemplo, cada píxel representado en la pantalla de un videojuego) y decidir qué acciones realizar para optimizar un objetivo (por ejemplo, maximizar la puntuación del juego).

ML-Agents provee dos implementaciones de algoritmos de aprendizaje por refuerzo:

- *Proximal Policy Optimization (PPO)*. Es el algoritmo utilizado por defecto. Es un método que ha probado ser más estable y de un mayor propósito general que muchos otros algoritmos en este campo. Desarrollado por

OpenAI, ha sido ampliamente utilizado en las investigaciones realizadas por la compañía.

- *Soft Actor-Critic (SAC)*. A diferencia de la PPO, la SAC es *off-policy*, lo que significa que puede aprender de las experiencias recogidas en cualquier momento del pasado. A medida que se recogen las experiencias, se colocan en un búfer de repetición de experiencias y se extraen aleatoriamente durante el entrenamiento. Esto hace que el SAC sea significativamente más eficiente en cuanto a las muestras, necesitando a menudo de 5 a 10 veces menos muestras para aprender la misma tarea que el PPO. Sin embargo, SAC tiende a requerir más actualizaciones del modelo. SAC es una buena opción para entornos más pesados o lentos (alrededor de 0,1 segundos por paso o más). SAC es también un algoritmo de "máxima entropía", y permite la exploración de forma intrínseca.

- Imitation Learning. En muchos escenarios, y especialmente en el caso de los videojuegos, es posible que se requiera que el agente aprenda a realizar una tarea compleja. Si bien es siempre posible que el agente sea capaz, por prueba y error, de deducir la tarea que tiene que llevar a cabo, este proceso puede ser muy costoso en cuanto a tiempo y computación se refiere.

  A menudo es más intuitivo demostrar simplemente el comportamiento que queremos que realice un agente, en lugar de intentar que aprenda mediante métodos de ensayo y error. Por ejemplo, en lugar de entrenar indirectamente a un agente con la ayuda de una función de recompensa, podemos darle ejemplos del mundo real de observaciones del juego y acciones de un controlador del juego para guiar el comportamiento del agente. El aprendizaje por imitación utiliza pares de observaciones y acciones de una demostración para aprender una política.

  El aprendizaje por imitación puede utilizarse solo o junto con el aprendizaje por refuerzo. Si se utiliza solo, puede proporcionar un mecanismo para aprender un tipo específico de comportamiento (es decir, un estilo específico de resolver la tarea). Si se utiliza junto con el aprendizaje por refuerzo, puede reducir drásticamente el tiempo que el agente tarda en resolver el entorno. Esto puede ser especialmente pronunciado en entornos de recompensa escasa.

- *Environment-specific*. ML-Agents provee una funcionalidad adicional denominada *Self-Play*, la cual permite entrenar modelos simétricos

(como podría ser un juego de fútbol o tenis, donde ambos equipos se rigen por las mismas reglas) y asimétricos (como el escondite, donde existen dos bandos) en un entorno específico. Con el *self-play*, un agente aprende en los juegos adversarios compitiendo contra versiones fijas y pasadas de su oponente (que podría ser él mismo como en los juegos simétricos) para proporcionar un entorno de aprendizaje más estable y estacionario.

ML-Agents proporciona la funcionalidad para entrenar comportamientos cooperativos, es decir, grupos de agentes que trabajan hacia un objetivo común, donde el éxito del individuo está vinculado al éxito de todo el grupo. En este escenario, los agentes suelen recibir recompensas como grupo. Por ejemplo, si un equipo de agentes gana un partido contra el equipo contrario, todos son recompensados, incluso los agentes que no han contribuido directamente a la victoria. Esto dificulta el aprendizaje de lo que hay que hacer como individuo: puedes obtener una victoria por no hacer nada, y una derrota por hacerlo lo mejor posible.

En ML-Agents, proporcionamos MA-POCA (MultiAgent POsthumous Credit Assignment), que es un entrenador multiagente que entrena a un crítico centralizado, una red neuronal que actúa como "entrenador" de todo un grupo de agentes. A continuación, puede dar recompensas al equipo en su conjunto, y los agentes aprenderán la mejor manera de contribuir a conseguir esa recompensa. También, se pueden dar recompensas a los agentes de forma individual, y el equipo trabajará conjuntamente para ayudar al individuo a alcanzar esos objetivos. Durante un episodio, los agentes pueden ser añadidos o eliminados del grupo, como cuando los agentes aparecen o mueren en una partida. Si los agentes son eliminados a mitad del episodio (por ejemplo, si los compañeros de equipo mueren o son retirados del juego), seguirán sabiendo si sus acciones contribuyeron a que el equipo ganara más tarde, lo que permite a los agentes realizar acciones beneficiosas para el grupo incluso si provocan que el individuo sea retirado del juego (es decir, el autosacrificio). MA-POCA también puede combinarse con el *Self-Play* para formar equipos de agentes que jueguen entre sí.

Desarrollo

Lista de tareas y cronograma

Un día laboral equivale a 2 horas de trabajo

1. Investigación {5 días - Viren, Gabriel, Javier}: Investigar las tecnologías y librerías necesarias para la ejecución del proyecto.

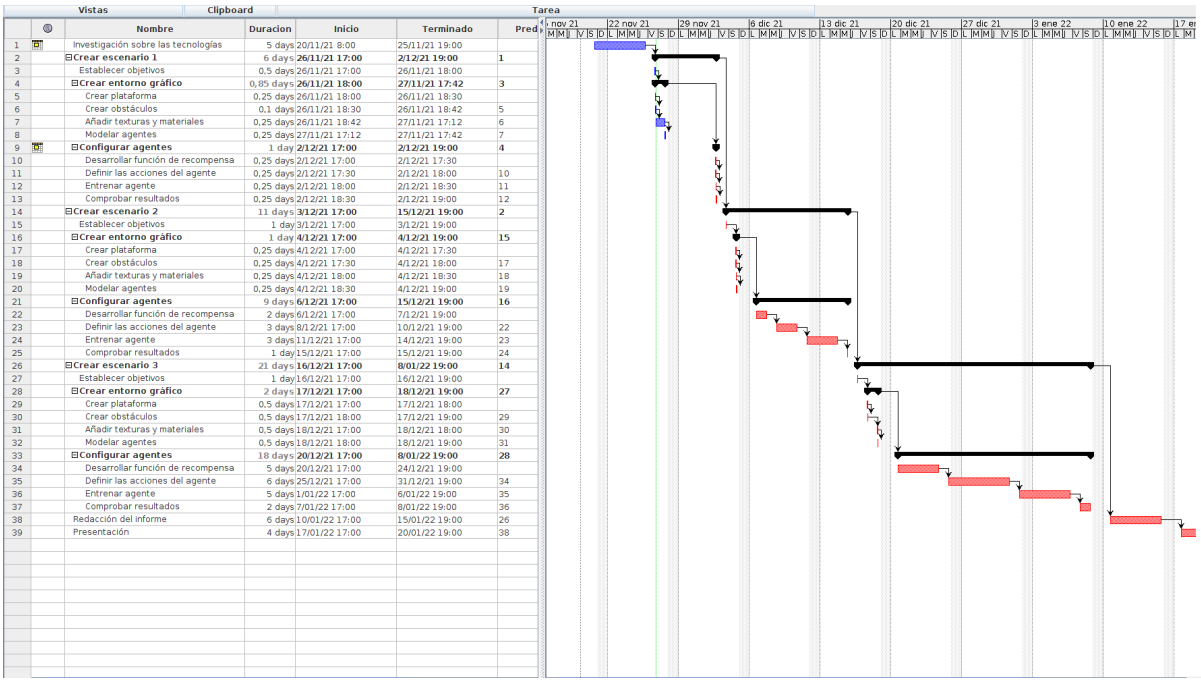
2. Crear escenario 1 {6 días - Viren, Gabriel, Javier, Ordenador de desarrollo}: Crear entorno gráfico donde realizar una simulación sencilla en la que un agente debe alcanzar un objetivo. La creación de cada escenario involucra una fase de diseño y otra de configuración. En esta última fase, se desarrollan las funciones de recompensa, se definen las acciones del agente, el cual será entrenado y, finalmente, comprobamos los resultados obtenidos.

3. Crear escenario 2 {11 días - Viren, Gabriel, Javier, Ordenador de desarrollo}: Escenario más complejo donde el agente debe, además de llegar al objetivo, evitar obstáculos.

4. Crear escenario 3 {21 días - Viren, Gabriel, Javier, Ordenador de desarrollo}: Último escenario, en el cual el agente colabora con otro para conseguir un objetivo.

5. Redacción del informe {6 días - Viren, Gabriel, Javier}:: Escribir la Memoria del Proyecto, indicando los resultados obtenidos.

6. Presentación {4 días - Viren, Gabriel, Javier, Ordenador de desarrollo}: Elaborar una presentación donde exponemos los resultados del proyecto.



	Nombre	Duracion	Inicio	Terminado	Pred
1	Investigación sobre las tecnologías	5 days	20/11/21 8:00	25/11/21 19:00	
2	**Crear escenario 1**	6 days	**26/11/21 17:00**	**2/12/21 19:00**	**1**
3	Establecer objetivos	0,5 days	26/11/21 17:00	26/11/21 18:00	
4	**Crear entorno gráfico**	0,85 days	**26/11/21 18:00**	**27/11/21 17:42**	**3**
5	Crear plataforma	0,25 days	26/11/21 18:00	26/11/21 18:30	
6	Crear obstáculos	0,1 days	26/11/21 18:30	26/11/21 18:42	5
7	Añadir texturas y materiales	0,25 days	26/11/21 18:42	27/11/21 17:12	6
8	Modelar agentes	0,25 days	27/11/21 17:12	27/11/21 17:42	7
9	**Configurar agentes**	1 day	**2/12/21 17:00**	**2/12/21 19:00**	**4**
10	Desarrollar función de recompensa	0,25 days	2/12/21 17:00	2/12/21 17:30	
11	Definir las acciones del agente	0,25 days	2/12/21 17:30	2/12/21 18:00	10
12	Entrenar agente	0,25 days	2/12/21 18:00	2/12/21 18:30	11
13	Comprobar resultados	0,25 days	2/12/21 18:30	2/12/21 19:00	12
14	**Crear escenario 2**	11 days	**3/12/21 17:00**	**15/12/21 19:00**	**2**
15	Establecer objetivos	1 day	3/12/21 17:00	3/12/21 19:00	
16	**Crear entorno gráfico**	1 day	**4/12/21 17:00**	**4/12/21 19:00**	**15**
17	Crear plataforma	0,25 days	4/12/21 17:00	4/12/21 17:30	
18	Crear obstáculos	0,25 days	4/12/21 17:30	4/12/21 18:00	17
19	Añadir texturas y materiales	0,25 days	4/12/21 18:00	4/12/21 18:30	18
20	Modelar agentes	0,25 days	4/12/21 18:30	4/12/21 19:00	19
21	**Configurar agentes**	9 days	**6/12/21 17:00**	**15/12/21 19:00**	**16**
22	Desarrollar función de recompensa	2 days	6/12/21 17:00	7/12/21 19:00	
23	Definir las acciones del agente	3 days	8/12/21 17:00	10/12/21 19:00	22
24	Entrenar agente	3 days	11/12/21 17:00	14/12/21 19:00	23
25	Comprobar resultados	1 day	15/12/21 17:00	15/12/21 19:00	24
26	**Crear escenario 3**	21 days	**16/12/21 17:00**	**8/01/22 19:00**	**14**
27	Establecer objetivos	1 day	16/12/21 17:00	16/12/21 19:00	
28	**Crear entorno gráfico**	2 days	**17/12/21 17:00**	**18/12/21 19:00**	**27**
29	Crear plataforma	0,5 days	17/12/21 17:00	17/12/21 18:00	
30	Crear obstáculos	0,5 days	17/12/21 18:00	17/12/21 19:00	29
31	Añadir texturas y materiales	0,5 days	18/12/21 17:00	18/12/21 18:00	30
32	Modelar agentes	0,5 days	18/12/21 18:00	18/12/21 19:00	31
33	**Configurar agentes**	18 days	**20/12/21 17:00**	**8/01/22 19:00**	**28**
34	Desarrollar función de recompensa	5 days	20/12/21 17:00	24/12/21 19:00	
35	Definir las acciones del agente	6 days	25/12/21 17:00	31/12/21 19:00	34
36	Entrenar agente	5 days	1/01/22 17:00	6/01/22 19:00	35
37	Comprobar resultados	2 days	7/01/22 17:00	8/01/22 19:00	36
38	Redacción del informe	6 days	10/01/22 17:00	15/01/22 19:00	26
39	Presentación	4 days	17/01/22 17:00	20/01/22 19:00	38

Histórico

A fecha 2021-12-10 hemos finalizado las tareas de investigación y avanzado la creación del escenario 1. La fase de investigación resultó satisfactoria y también involucró la instalación de librerías importantes como ML-Agents en Unity o Pytorch, entre otras.

Además, creamos un escenario sencillo donde un agente es capaz de llegar a un objetivo satisfactoriamente tras entrenarlo, aunque todavía no hemos podido calcular los resultados obtenidos.

A fecha 2022-01-21, hemos continuado nuestro proyecto de acuerdo a la planificación prevista, diseñando y configurando el segundo escenario de juego. Este escenario consiste en un nivel del juego Flappy Bird, donde el jugador ha de intentar mejorar su puntuación navegando el escenario y evitando chocar contra las tuberías.

Puesto que la programación de este videojuego es una tarea compleja que no se había visto reflejada en nuestro cronograma en una primera instancia, se decidió encontrar un proyecto de Unity donde el Flappy Bird ya hubiese sido desarrollado. Nuestro trabajo entonces ha consistido en comprender esta implementación y realizar las modificaciones necesarias para poder integrar el motor de ML-Agents de Unity: se ha añadido un techo al nivel, configurado un sistema de puntos de control para el uso de sensores integrados en el jugador, retocado el sistema de puntuación y de generación de niveles y, por último, se ha comenzado con el desarrollo de las funciones recompensas para el agente. Para la

finalización de este escenario, faltaría el entrenamiento del agente a través de un sistema de aprendizaje curricular y por imitación.

A fecha 2022-02-08 hemos finalizado el proyecto, habiendo completado el Flappy Bird y desarrollado un aero-jóquey donde los agentes deben colaborar para marcar en la portería enemiga, además de ser un juego competitivo donde ambos equipos quieren siempre conseguir la mayor cantidad de función de recompensa.

Problemas encontrados

- Problemas de incompatibilidad de versiones entre librerías.
- Documentación escasa sobre la librería ML-Agents.
- Restricciones de hardware para entrenar al agente.
- Debido a la dificultad de los escenarios, hemos tenido que utilizar herramientas del paquete ML-Agents que han ocasionado algunos problemas. Un ejemplo de esto fue aprender cómo utilizar “ray perception” para que nuestro agente pudiera detectar objetos en su entorno. Su implementación fue más complicada de lo esperado lo que causó una demora respecto al cronograma del proyecto a día 2022-01-21.

Requisitos y Objetivos alcanzados

1. Programa básico que consiga que un agente se mueva en línea recta hacia su objetivo.

2. Programa que sea capaz de superar un entorno calificado como complejo.

3. Programa donde el agente interactúa con otro agente de manera sencilla.

4. Memoria del proyecto.

A fecha 2022-01-21, los primeros requisitos ya se han cumplido. Estos requisitos son: la creación del programa básico, en el que no sólo el agente viaja en línea recta hacia su objetivo, sino que siendo situado el agente y el objetivo final en localizaciones completamente aleatorias, el agente es capaz de alcanzarlo sin tocar muros ni caerse de la plataforma.

Por otra parte, también hemos conseguido superar un entorno complejo, siendo éste el Flappy Bird ya mencionado y las pruebas han resultado satisfactorias.

En cuanto al tercer requisito se cumplió con el aero-jóquey, incluso se mejoró haciendo que se enfrenten a otros dos agentes que tienen el mismo objetivo. Por último, la memoria se redactó satisfactoriamente.

Descripción del funcionamiento

Nuestro proyecto busca la exploración de los conceptos teóricos de la asignatura aplicados dentro de entornos simulados cuyo único fin es la demostración de los mismos de una forma práctica. Como tal, nuestro proyecto se centra principalmente en la investigación y en la obtención de resultados de interés; por lo que no existe una forma predeterminada en la que utilizar los escenarios desarrollados para el mismo. Algunos de los casos de uso del mismo son los siguientes:

- Uso de los cerebros entrenados. En cada uno de los escenarios realizados, participa al menos un agente que ha debido de ser entrenado de acuerdo a la función de recompensa propuesta para dicho supuesto. El cerebro de este agente puede ser posteriormente exportado para su uso en otros proyectos similares, ahorrando al desarrollador la labor de entrenamiento, que puede ser muy intensiva en cuanto a recursos necesarios se refiere.
- Uso con fines educativos. Puesto que los escenarios se encuentran ya codificados y preparados para su uso, un posible uso de nuestro proyecto es utilizar estas escenas para demostrar conceptos básicos de Machine Learning. La interfaz gráfica proporcionada por el panel de control de Tensor Board permite observar el comportamiento y aprendizaje del agente de forma intuitiva.
- Detección de fallos en un videojuego. Como desarrollador, no se puede probar en profundidad un juego para ver si el objetivo puede ser alcanzado de una manera que no era la prevista, por lo que se puede entrenar a un agente y ver si puede alcanzar la meta de la manera intencionada o por el contrario existe algún fallo de diseño o de programación que permite al jugador alcanzar el objetivo antes, ahorrando tiempo y recursos en la fase de pruebas.

- Creación de Inteligencia Artificial en videojuegos. En un juego colaborativo, se podría querer practicar con una IA para mejorar y ver cómo interactuar ante distintas situaciones.

Conclusiones

En este proyecto hemos explorado el uso de las redes neuronales para el entrenamiento de agentes aplicados a videojuegos. Esta aproximación a la hora de desarrollar inteligencias artificiales en aplicaciones de entretenimiento es cada vez más popular, puesto que estas cada vez permiten un mayor grado de libertad al jugador en cuanto a posibles acciones a realizar se refiere. La IA tradicional falla en proporcionar una respuesta adecuada a este paradigma, puesto que un mayor espacio de posibles acciones por parte del jugador se traduce en una mayor cantidad de estados que los desarrolladores deben de prever y programar con antelación; a menudo convirtiéndose en una tarea muy costosa o imposible de completar.

El enfoque que ha sido desarrollado durante la elaboración de este proyecto, con el uso de redes neuronales y aprendizaje profundo como método de entrenamiento, ha demostrado ser una potente herramienta a la hora de dotar de un mayor conocimiento a la IA de un determinado videojuego. La capacidad de adaptación de los agentes entrenados permite la implementación de mecánicas de juego complejas, como es la búsqueda de caminos en entornos destructivos o la confección de planes dinámicos para llegar a un cierto objetivo en circunstancias cambiantes. Por estas razones, es común ver el uso de librerías como *ML-Agents* más frecuentemente en el desarrollo de videojuegos.

Si bien un sistema de inteligencia artificial basada en agentes es robusto y, en algunos casos, más fácil de desarrollar que un sistema de IA tradicional; una de las desventajas más evidentes encontradas durante el desarrollo de este proyecto ha sido la necesidad de poseer un hardware con un suficiente poder de cómputo como para poder realizar el proceso de entrenamiento en un plazo razonable de tiempo. El número de observaciones que necesita realizar el agente puede verse aumentado rápidamente, incidiendo directamente en el tiempo de entrenamiento necesario; y si bien existen técnicas como el aprendizaje curricular o el

aprendizaje por imitación para agilizar el proceso, estos pueden ser costosos de configurar y mantener. Debido a las características del entrenamiento, no es hasta en un estado avanzado de este proceso que se puede determinar si el agente persigue o no la política propuesta; en caso contrario, cualquier cambio en la función de recompensa puede implicar la confección de un nuevo modelo, con todos los costes que ello implica.

Bibliografía

- Code Monkey - Making Flappy Bird in Unity 2019. (2019). Unity Code Monkey. https://unitycodemonkey.com/video.php?v=b5Wpni9KPik

- An Introduction to Proximal Policy Optimization (PPO) in Deep Reinforcement Learning. (2019, 4 junio). YouTube. https://www.youtube.com/watch?v=vQ_ifavFBkI&t=479s&ab_channel=Udacity-DeepRL

- Actor Critic Algorithms. (2017, 16 diciembre). YouTube. https://www.youtube.com/watch?v=w_3mmm0P0j8&t=301s&ab_channel=SirajRaval

- U. (2017). GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit. GitHub - MLAgents. https://github.com/Unity-Technologies/ml-agents

- Lowe, R. (2017, 7 junio). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv.Org. https://arxiv.org/abs/1706.02275

- Jakob N. Foerster. (2017, diciembre). Counterfactual Multi-Agent Policy Gradients. University of Oxford, United Kingdom. https://arxiv.org/pdf/1705.08926.pdf