REINFORCEMENT LEARNING APLICAT A VIDEOJOCS

Martí Capel Ruiz

2 de juliol de 2024

Resum– El Treball de Fi de Grau se centrarà en l'aplicació del Reinforcement Learning (RL) a jocs utilitzant la biblioteca Gymnasium de Python. L'objectiu principal és comprendre els fonaments del RL i aplicar aquests coneixements en diversos entorns de joc, des de jocs més simples, com el cart pole, fins a jocs més complexos, incloent jocs multijugador. Inicialment, es realitza un estudi profund de la teoria del RL. A continuació, es procedeix amb l'exploració i el domini de la biblioteca Gymnasium, una eina essencial per a la simulació i l'entrenament d'agents de RL en entorns controlats. El projecte es desenvolupa en fases: primer, l'implementació d'un algorisme tabular per un joc simple; després, la creació d'un agent més avançat per a un joc complex de un jugador; i finalment, l'adaptació d'aquest agent per competir en un entorn multiagent utilitzant la biblioteca PettingZoo.

Paraules clau– Reinforcement Learning, Gymnasium, PettingZoo, Python, Deep Reinforcement Learning, Agent de RL, Videojocs, Multiplayer, Avaluació d'Agents, Machine Learning

Abstract– This Final Degree Project will focus on applying Reinforcement Learning (RL) to games using the Python library Gymnasium. The main objective is to understand the fundamentals of RL and apply this knowledge in various game environments, from simpler games like cart pole to more complex ones, including multiplayer games. Initially, a deep study of RL theory is conducted. Then, the exploration and mastery of the Gymnasium library, an essential tool for the simulation and training of RL agents in controlled environments, are undertaken. The project is developed in phases: first, the implementation of a tabular algorithm for a simple game; then, the creation of a more advanced agent for a complex single-player game; and finally, the adaptation of this agent to compete in a multi-agent environment using the PettingZoo library.

Keywords– Reinforcement Learning, Gymnasium, PettingZoo, Python, Deep Reinforcement Learning, RL Agent, Video Games, Multiplayer, Agent Evaluation, Machine Learning

1 INTRODUCCIÓ - CONTEXT DEL TREBALL

El Reinforcement learning (RL) es basa en la creació d'agents capaços de navegar per entorns complexos. A diferència d'altres tècniques de machine learning (ML), aquest no necessita exemples de resultats correctes i incorrectes, sinó que aprèn mentre navega per l'entorn a partir d'unes recompenses preestablertes. Aquest paradigma del ML ens obre portes per a la creació de nou coneixement, per exemple, el model AlphaZero [**?**] creat per DeepMind. AlphaZero era capaç de jugar als escacs, el go o, fins i tot, el Starcraft. Posteriorment, aquest model va ser modificat per crear Alpha Tensor [8] el qual va trobar algorismes més eficients per multiplicar matrius dels que teníem en el moment. En aquest sentit, els videojocs són un entorn perfecte per fer proves, comparar i millorar els algoritmes utilitzats.

• E-mail de contacte: 1573350@uab.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Jordi Casas (Ciències de la Computació)
• Curs 2023/24

1.1 Objectius

L'objectiu principal és comprendre els fonaments del RL i aplicar aquests coneixements en diversos entorns de joc, des de jocs més simples, com el cart pole, fins a jocs més complexos, incloent alguns de l'Atari. Finalment aplicarem aquests models per jugar a jocs multiagent.

- **Comprendre els fonaments del RL:** Per començar he d'aprendre com funciona el RL. Per fer-ho utilitzaré principalment dos llibres: el primer és RL an

Introduction, de Richard S. Sutton. El segon és Deep Reinforcement Learning Hands-On, de Maxim Lapan. Ambdós llibres s'han de llegir i també faré alguns exercicis que proposen per assegurar que entenc cada cosa que expliquen.

- **Aprendre sobre Gymnasium i PettingZoo:** El següent pas és entendre les biblioteques que estaré utilitzant. La primera és Gymnasium, una biblioteca de Python open source creada per Farama, que ofereix una sèrie d'entorns llestos per ser utilitzats en problemes de RL. La segona és PettingZoo, una biblioteca derivada de Gymnasium però que permet utilitzar entorns multiagent. Aquesta serà utilitzada per al torneig.
- **Implementar models tabulars:** Els primers models seran tabulars, utilitzats per jugar a un joc simple amb pocs estats. Per programar-los utilitzaré Gymnasium. Els dos tipus de models que faré són: Monte Carlo with Weighted Importance Sampling i TD Q-Learning, amb una sola Q i amb doble Q.
- **Implementar models DRL:** Una vegada implementats els models simples, implementaré models basats en Deep Learning per poder jugar a un joc més complex. Els models a implementar són els següents: Cross-Entropy Method, Deep Q-Networks, Policy Gradient, Advantage Actor-Critic.
- **Modificar els models per ser multiagent:** Quan els models estiguin llestos per jugar al joc d'un jugador, hauran de ser modificats per poder jugar en un entorn multi agent, preparant-los per al següent i últim objectiu.
- **Realitzar proves dels models en entorns multiagent nous:** Una vegada tingui els models preparats, procediré a fer un torneig on els models competeixin entre ells per determinar quin és el que té un millor rendiment a l'hora de jugar competitivament.

2 Metodologia

Per dur a terme el projecte, he decidit aplicar una metodologia semblant a Scrum per gestionar el meu treball. Començo creant una llista de tasques prioritzada i, a continuació, divideixo el meu temps en períodes de treball definits anomenats sprints i d'una duració de 2 setmanes. Selecciono les tasques per a cada sprint i reviso el meu progrés diàriament mitjançant breus reunions de Scrum amb mi mateix. Al final de cada sprint, reviso el meu treball i reflexiono sobre els meus objectius i el que he après, ajustant les meves estratègies i planificació segons sigui necessari. Aquest procés iteratiu de treball em permet gestionar el meu projecte de manera eficaç, millorant constantment el meu rendiment i adaptant-me als canvis i reptes que puguin sorgir.

3 Introducció al Reinforcement Learning

En aquesta secció explicaré els conceptes bàsics que són comuns a tots els problemes de RL.

En tot problema de RL tenim un entorn que és el lloc on es desenvolupen les accions, també tenim un agent, que és l'encarregat de navegar per l'entorn a través d'accions. Navegar l'entorn otorgarà recompenses que podran ser utilitzades per l'agent per valorar les accions que pren.

3.1 Conceptes bàsics

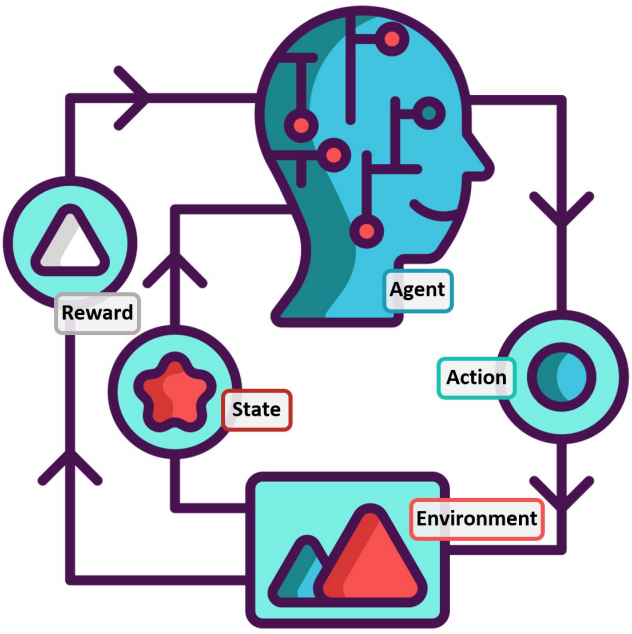

Fig. 1: Diagrama de l'agent i l'entorn

Per entendre com funcionen els models, primer cal entendre alguns conceptes bàsics mostrats en la figura 1:

- Estat o Observació (s): És la descripció d'un moment en concret dins de l'entorn. Hi ha un nombre finit d'estats. Aquests poden ser tant una casella dins d'un tauler com la posició de cada partícula dins d'un model meteorològic.
- Acció (a): Per a cada estat, l'agent tindrà una sèrie d'accions entre les quals podrà escollir. Aquestes decisions influenciaran l'entorn per moure l'agent al següent estat.
- Recompensa (r): Després de cada acció, l'agent obtindrà una recompensa. Aquestes recompenses seran utilitzades per l'agent per aprendre.

4 Estat de l'art

El reinforcement learning és un camp en constant evolució, en el qual sorgeixen noves tècniques contínuament. En aquest apartat, faré un petit resum, des de les tècniques més bàsiques fins a les més modernes.

4.1 Mètodes tabulars

Els mètodes tabulars es basen en la creació de taules (o matrius) que s'aniran actualitzant durant l'entrenament per donar un valor a cada acció en cada estat. Aquests valors representen l'esperança de recompensa futura que s'espera

obtenir a partir de l'execució de cada acció en cada estat. Això permet que, una vegada les taules estan llestes, l'agent sigui capaç d'escollir la millor acció en cada moment, comparant els valors de la taula entre si.

4.1.1 Monte Carlo

Els mètodes de Monte Carlo es basen en l'aprenentatge a partir de la simulació d'episodis complets. Al final de cada episodi, es calculen els valors de les accions preses en funció de les recompenses acumulades. Això és especialment útil en entorns episòdics, on hi ha un inici i un final clar per a cada episodi.

La idea bàsica és actualitzar la funció de valor $Q(s, a)$ mitjançant el retorn (la suma de les recompenses futures ponderades per un factor de descompte γ) observat en un episodi complet. La fórmula de l'actualització de Monte Carlo és:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(G_t - Q(s, a)\right)$$

on G_t és el retorn observat des del temps t i α és la taxa d'aprenentatge.

Aquests mètodes poden ser on-policy, on l'agent aprèn de les accions preses per la seva pròpia política, o off-policy, on l'agent aprèn de les accions preses per una altra política.

4.1.2 Temporal Difference (TD)

En contraposició als mètodes de Monte Carlo, els mètodes de Temporal Difference (TD) actualitzen els valors de les accions després de cada pas en lloc d'esperar fins al final de l'episodi. Això permet un aprenentatge més ràpid i és aplicable en entorns no episòdics.

El mètode TD més bàsic és TD(0), que actualitza la funció de valor $Q(s, a)$ utilitzant l'equació de Bellman:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma Q(s', a') - Q(s, a)\right)$$

on r és la recompensa immediata obtinguda després de prendre l'acció a en l'estat s, i s' i a' són el nou estat i l'acció seleccionada a partir de s'.

Un altre mètode TD comú és el SARSA (State-Action-Reward-State-Action), que, a diferencia de TD(0) , és un mètode estictament on-policy. El nom SARSA prové de la seqüència de quíntuples (s, a, r, s', a') utilitzades per l'actualització.

El Q-learning és un mètode TD off-policy que busca trobar la política òptima independentment de la política seguida per l'agent. Utilitza l'equació:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$$

on l'actualització es basa en la recompensa immediata r i el valor màxim de Q per al nou estat s'.

Els mètodes tabulars tenen limitacions quan es treballa amb entorns amb molts estats, ja que simular suficients episodis per cobrir tots els estats pot ser costós. Si un estat no s'ha simulat mai, el valor en la taula no serà correcte, la qual cosa pot portar l'agent a prendre decisions equivocades. Tot i això, aquests mètodes continuen sent rellevants en determinats contextos, especialment quan es treballa amb entorns petits o quan es necessita interpretabilitat en les solucions. Malgrat que els avanços en RL sovint es centren en tècniques més avançades com el Deep Reinforcement Learning, no es pot ignorar la importància dels mètodes tabulars en el camp.

4.2 Mètodes aproximats (Deep learning)

Els mètodes aproximats són utilitzats per gestionar la complexitat d'entorns amb molts estats o accions. En lloc d'utilitzar taules per emmagatzemar els valors dels estats i accions, els mètodes aproximats fan servir funcions aproximadores, com ara xarxes neuronals, per estimar els valors dels estats i accions. Aquestes xarxes són capaces d'aprendre representacions complexes i no lineals dels valors a partir de les dades d'experiència.

Hi ha dues tasques principals que les xarxes neuronals poden aprendre:

- **Funció de valor:** Una xarxa neuronal pot ser entrenada per predir el valor esperat d'un estat o d'una acció. Això permet als agents prendre decisions basades en les prediccions del valor esperat de les diferents opcions disponibles.
- **Política:** Una altra opció és entrenar una xarxa neuronal per predir directament les accions òptimes en un estat determinat. Això es coneix com a aprenentatge de política directa i permet als agents prendre decisions sense necessitat de calcular explícitament els valors dels estats.

4.2.1 Cross-Entropy Method

El mètode de Cross-Entropy és una tècnica d'optimització que es basa en la selecció i actualització de mostres d'accions segons la seva rendibilitat. Funciona generant una població d'accions, seleccionant les millors segons una mesura de rendiment i actualitzant la distribució de les accions basant-se en aquestes mostres.

Els passos bàsics del CEM són:

1. Generar una població d'accions segons una distribució inicial.
2. Executar les accions i avaluar el rendiment.
3. Seleccionar les millors accions basades en el rendiment.
4. Actualitzar la distribució de les accions segons les millors mostres.
5. Repetir els passos fins a la convergència.

4.2.2 Deep Q-Networks (DQN)

Deep Q-Networks (DQN) és una extensió de l'algoritme Q-learning que utilitza xarxes neuronals per aproximar la funció de valor Q. Això permet gestionar entorns amb espais d'estat continus o molt grans.

L'algoritme DQN utilitza una xarxa neuronal per estimar els valors Q i un mecanisme de repetició d'experiències (*experience replay*) per trencar la correlació entre les seqüències d'experiències, emmagatzemant-les en una

memòria de reproducció i entrenant la xarxa amb mostres aleatòries d'aquesta memòria.

L'actualització dels valors Q en DQN es basa en l'equació:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

4.2.3 Policy Gradient

Els mètodes de Policy Gradient estimen directament la política òptima sense necessitat de construir una funció de valor Q. Això es fa ajustant els paràmetres de la política per maximitzar una funció objectiu que reflecteix el rendiment esperat.

El gradient de la política es calcula utilitzant la següent fórmula:

$$\nabla J(\theta) = E_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right] \tag{1}$$

Aquest mètode és especialment útil en problemes amb espais d'accions continus.

4.2.4 Advantage Actor-Critic (A2C)

A2C és una millora dels mètodes de Policy Gradient que combina els avantatges dels mètodes d'actor-critic. Utilitza dos models: l'actor, que aprèn la política, i el critic, que estima el valor de l'estat.

L'actor actualitza la política seguint el gradient:

$$\nabla J(\theta) = E_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s,a) \right] \tag{2}$$

On $A^\pi(s,a)$ és la funció avantatge, definida com:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{3}$$

El critic actualitza els valors seguint l'objectiu de mínim quadrat. La funció de pèrdua per al critic és la següent:

$$L(\phi) = E_{(s_t, r_t, s_{t+1}) \sim \pi_\theta} \left[(r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t;\phi))^2 \right] \tag{4}$$

On ϕ són els paràmetres del critic.

Aquest procés assegura que el critic aprèn a estimar correctament el valor dels estats, que després és utilitzat per l'actor per actualitzar la política.

4.2.5 Proximal Policy Optimization (PPO)

PPO és una millora dels mètodes de Policy Gradient que utilitza una tècnica de clipping per assegurar que les actualitzacions dels paràmetres de la política no siguin massa grans, mantenint així l'estabilitat de l'aprenentatge.

L'objectiu de PPO és maximitzar la següent funció:

$$L^{CLIP}(\theta) = E_t \Big[\min \Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \Big) \Big] \tag{5}$$

On $r_t(\theta)$ és el ratio entre les probabilitats de la nova i la vella política, $\hat{A}_t$ és l'estimació avantatge, i ϵ és un hiperparàmetre que determina el límit de clipping.

Aquest enfocament permet un entrenament més robust i estable comparat amb els mètodes tradicionals de Policy Gradient.

5 Frozen Lake amb mètodes tabulars

Frozen Lake és un entorn de Reinforcement Learning proporcionat per la biblioteca Gymnasium de Python. Aquest és un entorn senzill que utilitzarem per aplicar dos mètodes tabulars (Q-Learning i Monte Carlo) i comparar-los entre ells

5.1 Descripció de l'entorn

Frozen Lake consisteix en una graella on l'agent ha de moure's des d'una posició inicial fins a una posició objectiu. La graella està composta de diferents tipus de cel·les:

- **S (Start):** La cel·la d'inici on es troba l'agent al començament de cada episodi.
- **F (Frozen):** Cel·les segures per on l'agent es pot moure.
- **H (Hole):** Cel·les amb forats on l'agent cau i l'episodi acaba.
- **G (Goal):** La cel·la objectiu que l'agent ha d'assolir.

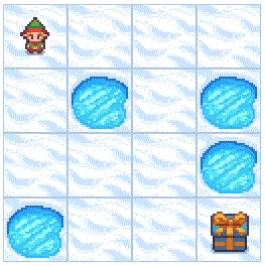

Fig. 2: Estat inicial Frozen Lake 4x4

L'objectiu de l'agent és trobar un camí segur des de la cel·la d'inici fins a la cel·la objectiu sense caure en els forats. L'agent pot realitzar quatre accions possibles: amunt, avall, esquerra i dreta. Depenent de si l'entorn és relliscós (*slippery*) o no, l'agent pot desplaçar-se de manera precisa o relliscar cap a una direcció diferent a la desitjada.

5.2 Resultats

Fig. 3: Comportament òptim en cada estat

Tant el model Q-Learning com el model Monte Carlo han estat capaços d'arribar al comportament òptim per a aquest joc en menys de 10000 passos. També s'observa com els dos entrenaments han estat quasi instantanis, amb un temps de menys de 1 segón. Aquests resultats demostren que els dos models són completament capaços de resoldre entorns simples amb un nombre petit d'estats.

6 MÈTODES DE DEEP LEARNING

Aquests mètodes no depenen d'una taula que guardi els valors dels estats i les accions, sinó que utilitzen xarxes neuronals, per aproximar aquests valors. Això permet als agents ser capaços de generalitzar i poder actuar de forma correcta en estats que no s'hagin visitat amb anterioritat

6.1 Cart Pole

Cart pole ens servirà com a entorn de proves pels agents, al ser un entorn simple ens permet fer un millor seguiment per si algun dels models falla i, així, trobar els errors més facilment

L'entorn consisteix en un eix horitzontal en el que es mou un vagó amb un segment vertical unit per un dels extrems al vagó. L'objectiu de l'agent serà el de mantenir el segment de forma vertical, compensant la caiguda amb moviments laterals

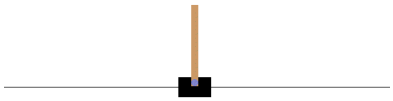

Fig. 4: Estat inicial Cart Pole

L'agent reb com a entrada un vector amb 4 nombres que indiquen la posició en la recta, l'angle que es forma entre el segment i la vertical, la velocitat del vagó, i la velocitat angular del segment. Amb aquesta informació l'agent ha d'escollir entre moure el vagó a la dreta o a l'esquerra.

6.2 Pong

El segon entorn utilitzat serà el Pong, un joc de l'Atari en el que hi ha una pilota movent-se pel mapa i els 2 jugadors han d'intentar moure la seva barra (cadascuna posicionada en un extrem horitzontal de la pantalla) verticalment per tal d'impedir que la pilota els sobrepassi i, així, enviar-la cap a l'oponent.

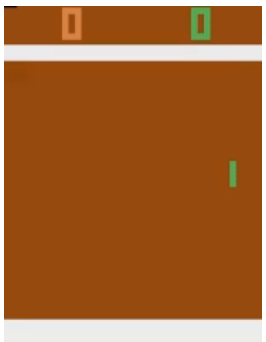

Fig. 5: Estat inicial Pong

Aquest entorn ofereix la possibilitat d'entrenar agents en espais més complexes que el cart pole i serà el que utilitzarem per comparar el rendiment dels models.

6.2.1 Espai d'estats

Cada estat serà un frame de la pantalla durant la partida, aquest frame és una imatge amb 3 canals de color (red, green, blue) de 210x160

6.2.2 Espai d'accions

Per interactuar amb l'entorn, l'agent haurà d'escollir entre 6 possibles accions derivades dels comandaments de l'Atari

- **Noop:** Per no fer res i passar al següent estat
- **Fire:** Per iniciar la partida
- **Right:** Per moure la teva barra cap amunt
- **Left:** Per moure la teva barra cap abaix
- **Rightfire:** En aquest joc no té utilitat
- **Leftfire:** En aquest joc no té utilitat

En el cas del Pong, les accions *Rightfire* i *Leftfire* no tenen cap utilitat i seran equivalents a l'acció *Noop*, i l'acció *Fire* només ens serveix per iniciar la partida, una vegada iniciada també serà equivalent a l'acció *Noop*

6.2.3 Recompensa

Cada vegada que la pilota toca sobrepassa la pala i toca el costat del nostre agent (dreta) l'agent obté una recompensa de -1, en canvi, quan la pilota toca el costat rival, l'agent obté una recompensa de 1. Durant un episodi es juguen 21 partides, això vol dir que la recompensa total màxima és de 21 i la mínima de -21.

6.2.4 Wrappers

Per tal de facilitar l'aprenentatge del nostre agent, afegirem una sèrie de wrappers al nostre entorn. Un wrapper és una capa addicional que es pot aplicar a un entorn per modificar el seu comportament sense canviar el codi del propi entorn.

- **MaxAndSkipEnv:** Aquest wrapper fa que l'entorn realitzi l'acció seleccionada n vegades seguida i només retorni l'última frame, això s'utilitza per augmentar la diferència entre frames, ja que sinó aquesta és molt petita en la majoria dels casos.
- **FireResetEnv:** La funció d'aquest wrapper és fer que al reiniciar l'entorn es faci automàticament l'acció FIRE i així permetre que l'agent actui en una partida ja iniciada. L'agent podria aprendre a fer aquesta acció però augmentaria bastant el temps d'entrenament
- **WarpFrame:** Aquest wrapper canvia la mida i els canals de les observacions que se li passen a l'agent. En concret passa les imatges de RGB a Grayscale i les fa de mida 84x84
- **ImageToPyTorch:** La funció d'aquest wrapper és canviar la organització de la imatge de [Height, Width, Channels] a [Channels, Height, Width] ja que és el format que utilitza la xarxa neuronal

- **BufferWrapper:** Aquest wrapper fa que l'observació que se li passa a l'agent no sigui una única imatge sinó les últimes N imatges. Això dóna una idea al model de magnituds com la velocitat de la pilota o la velocitat de les pales

- **ScaledFloatFrame:** Finalment tenim un wrapper que canvia els valors dels pixels de Int (0-256) a Float (0-1). Això es fa perquè la xarxa neuronal funciona millor quan les dades estan normalitzades.

6.3 Mètodes d'avaluació

Per avaluar i comparar els models farem servir dues mètriques:

- **N steps** Els steps que ha de donar l'agent per arribar a 19 de recompensa mitjana (en els últims 100 episodis) en el joc del Pong.

- **Steps/s** El número mig de steps que es fan per cada segon

Les dues mètriques es poden combinar per calcular el temps total que ha tardat en arribar als 19 de recompensa mitjana.

6.4 Deep-Q Networks

El primer model que utilitzarem és el Deep-Q Network (DQN). Aquest és una millora del model tabular Temporal Difference. Es basa en tenir una xarxa neuronal que doni com a output els valors Q de les accions que es poden prendre un estat donat com a input.

6.4.1 Exploració

Per permetre una bona exploració de l'entorn, utilitzarem el paràmetre ϵ. Cada vegada que hem de fer una acció, aquesta serà aleatòria amb probabilitat ϵ. Quan l'entrenament comença, $\epsilon = 1$ (fent que cada elecció sigui aleatòria) i aquest valor anirà disminuint mentre l'entrenament avança, fins a un mínim tal que $0,05 > \epsilon > 0$.

6.4.2 Trencar la correlació

Un dels principals desafiaments és gestionar la correlació entre les experiències per millorar l'eficiència de l'aprenentatge. Això és important perquè les experiències correlacionades poden fer que els entrenaments siguin menys eficients i inestables. Per fer-ho s'utilitzen dues tècniques:

- **Replay buffer** Aquesta tècnica consisteix a emmagatzemar les experiències en una memòria de replay buffer. Durant l'entrenament, es prenen mostres aleatòries d'aquest buffer en lloc d'utilitzar la seqüència d'experiències com es van generar.

- **Target Network** En lloc d'actualitzar els valors Q directament a partir de la xarxa principal, s'utilitza una segona xarxa (target network) que es sincronitza amb la xarxa principal cada N steps.

6.4.3 Cicle d'entrenament

El cicle d'entrenament és el següent:

- **Actualitzar variables** S'actualitzen les variables ϵ o Target Network si és necessari.

- **Escollir una acció** S'escull una acció a partir de l'estat actual amb probabilitat ϵ de que sigui aleatòria i probabilitat $1 - \epsilon$ de que sigui la que té un major valor de Q(s, a) segons la xarxa.

- **Realitzar l'acció** Aquesta acció es realitza i s'emmagatzema la tupla d'experiència (estat, acció, recompensa, estat següent) en el buffer

- **Calcular loss i actualitzar xarxa** Si el replay buffer té suficients experiències, s'agafen unes quantes aleatòriament, es calcula la loss i s'actualitza la xarxa.

6.4.4 Xarxa neuronal

La xarxa neuronal utilitzada constarà de dues parts. La primera part seran 3 capes convolucionals per analitzar la imatge d'entrada de forma que es redueixin el nombre de features que passaran a la segona part.

- **ConvLayer0:** 32 filtres de 8x8 amb un stride de 4

- **ConvLayer1:** 64 filtres de 4x4 amb un stride de 2

- **ConvLayer2:** 64 filtres de 3x3 amb un stride de 1

Una vegada ha actuat la part convolucional, les 3136 features restants passen a dues capes fully connected, aquestes reduiran el nombre de features fins a 6, de forma que corresponguin als valors Q de cada acció que es pot prendre en l'estat actual.

- **HiddenFullyConnected:** 512 neurones

- **OutputLayer:** 6 neurones de sortida

Totes les capes estan separades per la funció d'activació ReLu.

A l'inici de l'entrenament utilitzarem un valor ϵ entre 0 i 1 per fer que l'acció escollida sigui aleatòria amb probabilitat ϵ, aquest valor disminuirà amb el temps. Igual que en el model TD això afavoreix que el nostre agent explori l'entorn a l'inici per fer-se una idea de l'espai d'estats.

6.4.5 Resultats

Aquest mètode, tot i que és simple, ha demostrat uns bons resultats. Un 96% dels entrenaments han arribat a 19 de reward mitjana. Aquest 4% restat és degut a una exploració insuficient. També hem provat de no tallar l'entrenament als 19 i deixar que s'arribi a 21 de reward (cosa que significaria que el model juga sense cap error) i totes les proves han arribat però el temps per passar de 19 a 21 és el mateix que per arribar de -21 a 19, per tant, les proves les hem seguit fent amb 19 de límit.

Amb una fase d'exploració de 100.000 steps ha estat suficient perquè els models puguin aprendre les nocions bàsiques i, després, seguir millorant. Tot i això, hi ha hagut alguns entrenaments en els que no ha estat suficient.

També hem fet una exploració de dos hiperparàmetres clau:

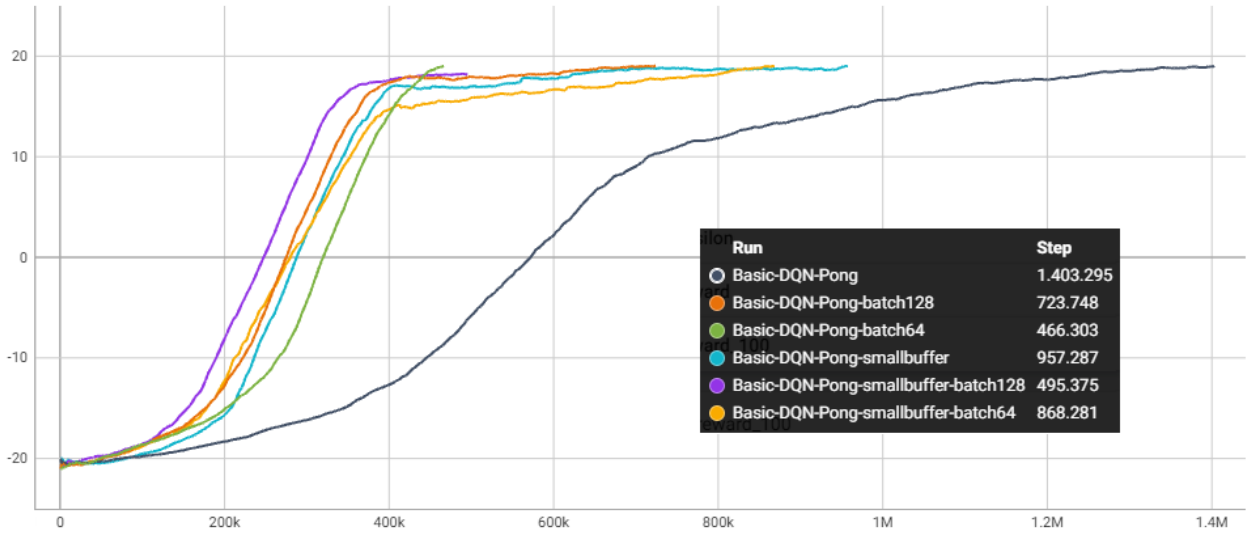

Fig. 6: Evolució del reward mig

- **El buffer:** Aquest paràmetre és molt important ja que un buffer petit no permet trencar la correlació entre les experiències i un buffer massa gran fa que s'agafin mostres de les quals han passat molts steps i, també, augmenta l'espai en memòria del buffer. Un buffer de 10^3 experiències, ha estat massa petit i els models no han pogut evolucionar i un buffer de 10^6 ha estat massa gran i ha ralentitzat molt l'execució del programa en la màquina.
- **El batch:** El batch és la quantitat d'experiències que s'agafen del buffer per entrenar cada step. Aquest paràmetre permet al model fer ús de la paral·lelització al entrenar. Sempre que es pugui, serà millor utilitzar un batch gran, ja que l'entrenament serà més ràpid, tot i que un batch massa gran pot ralentitzar l'entrenament si la màquina no té suficient potència de càlcul.

Buffer	Batch	Steps	Steps/s	Temps en h
10^4	32	957.287	73	3,6
	64	868.281	59	4
	128	495.375	32	4,3
10^5	32	1.403.295	65	6
	64	466.303	46	2,8
	128	868.281	29	6,9

TAULA 1: COMPARACIÓ RESULTATS EN ELS ENTRENAMENTS DQN

Com es pot observar, el buffer més petit ha permès uns entrenaments més ràpids, tant pel nombre de steps com pel temps que es tarda en fer cada un.

6.5 Millores a les DQN

Una vegada tenim el model DQN funcionant, li podem afegir diferents canvis per tal d'intentar accelerar el procés d'aprenentatge. Algunes d'aquestes millores són canvis subtils en la forma de calcular el valor de Q i d'altres són canvis en la xarxa o en la forma d'agafar les experiències del buffer.

6.5.1 N-Steps DQN

Aquesta idea s'utilitza des de fa temps, va ser introduïda per Richard Sutton ([3]) el 1988. Per entendre-la hem de fer una ullada a la equació de Bellman utilitzada per calcular el valor de Q.

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) \tag{6}$$

Com es pot veure, aquesta equació és recursiva, per tant, la podem expressar de la següent forma:

$$Q(s_t, a_t) = r_t + \gamma \max_a [r_{a,t+1} + \gamma \max_{a'} Q(s_{t+2}, a')] \tag{7}$$

Tenint en compte que l'acció en temps t+1 ha estat escollida de forma òptima, podem ometre $\max_a$ per obtenir el següent:

$$Q(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2}, a') \tag{8}$$

Així, podem fer unroll de l'equació les vegades que volguem per fer un aprenentatge "acelerat". El problema que tenim és que estem suposant que l'elecció de l'acció en temps t+1 és òptima i, no sempre és així, especialment a l'inici quan la nostra xarxa està aprenent, llavors, com més steps estem fent, més error acumulem en el valor de Q. S'ha de trobar el punt mig entre un valor massa gran (amb molt error acumulat) i un valor massa petit (que ralentitza el procés de traspàs d'informació entre estats).

Per aplicar aquesta tècnica al nostre model, hem de fer que les experiències emmagatzemades continguin N transicions entre estats i, així, calcular el valor Q que ha d'aprendre la xarxa utilitzant la nova fórmula.

6.5.2 Double DQN

La següent idea va ser proposada en el paper *Deep Reinforcement Learning with Double Q-learning* [4]. Els autors van demostrar que el model DQN té tendència a sobreestimar els valors de Q. Per resoldre el problema van proposar modificar l'equació per calcular el valor de Q:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, \arg\max_a Q(s_{t+1}, a)) \quad (9)$$

Al fer servir la xarxa principal per escollir les accions i la xarxa secundària només per avaluar-les, es soluciona aquest problema de sobreestimació.

És un canvi simple però suposa una millora considerable

6.5.3 Noisy Network

Fins ara, per mantenir l'exploració utilitzàvem el valor d'epsilon, però, això suposava fer tuning de més hiperparàmetres per tal de tenir una bona exploració. El 2018 es va publicar el paper *Noisy Networks for Exploration* [5] que proposava la idea de fer servir soroll en els pesos de les capes fully-connected i ajustar els paràmetres d'aquest soroll durant l'entrenament.

6.5.4 Prioritized Buffer

Unaltra idea proposada pels investigadors de Deep Mind en l'article *Prioritized Experience Replay* [6] és la de prioritzar les experiències del buffer que tinguin una loss major i, per tant, que modifiquin més la xarxa neuronal. Aquest mètode augmenta la velocitat de convergència al fer que variïn més els pesos de la xarxa.

6.5.5 Resultats

Tots els experiments han estat fets amb una mida del buffer de 10^5 experiències i un batch de 32.

Buffer	Steps	Steps/s	Temps en h
Bàsic	1.403.295	65	6
2-Steps	1.061.122	60	4,9
Double	904.930	61	4,1
Noisy	531.996	53	2,8
Prioritzat	835.182	46	5

TAULA 2: COMPARACIÓ RESULTATS DE LE DQN

Com es pot observar en la taula, totes les millores han aportat una disminució del nombre de steps necessaris per l'entrenament, mentre que la velocitat no ha disminuït molt. Hi ha hagut una disminució de tots els temps d'entrenament.

En aquest cas, el model *noisy* ha demostrat ser el millor, això podria ser degut a que els nostres paràmetres d'exploració no són òptims, per tant, el mètode noisy networks ajuda a millorar la fase d'entrenament inicial i, permet seguir explorant una vegada ha acabat aquesta.

6.6 Policy Gradients

Fins ara els nostres models ens servien per calcular els valors de Q(s, a) de cada estat i, després, aplicàvem una política per escollir quina acció agafar (normalment aquesta política és la d'escollir la acció amb un valor major)

La idea dels mètodes de Policy Gradients (PG) és saltarse aquest pas i que sigui el model el que esculli quines accions prendre. Per fer-ho, el model calcularà les probabilitats de que les accions siguin escollides i, després, es farà una selecció aleatòria basada en aquestes probabilitats

Un avantatge d'aquests models és la de ser utilitzats en entorns amb un espai d'accions molt gran, ja que no s'ha de fer la operació de buscar el valor màxim d'entre totes les Q(s, a).

6.6.1 Exploració

Un altre gran avantatge de utilitzar les probabilitats és que ja no necessitem un valor ϵ per explorar, ja que de tant en tant s'escollirà una acció que no és la que té un major valor (cosa que no passa amb els DQN). Pot acabar passant que amb el temps una acció tingui una probabilitat molt alta, per evitar això calculem la entropia (una distribució equiprobable té una entropia alta i una distribució poc equiprobable té una entropia baixa) i la restem en la funció de loss, per motivar al nostre agent a no estar massa segur de quina és la acció correcta.

6.6.2 Trencar la correlació

Tal i com fèiem en els DQN, hem de trencar la correlació entre les mostres que li mostrem al model per entrenar, per fer-ho teníem un buffer molt gran amb el que escollíem les mostres aleatòriament cada entrenament. Això ja no ho podem fer ja que al ser un mètode on-policy, fer servir mostres generades amb una altra política no ens aportaria informació sobre com millorar l'actual.

Per solucionar aquest problema utilitzarem vàries instàncies de l'entorn de forma paral·lela, així, cada step el farem en un entorn diferent i, per tant, no hi haurà correlació entre les mostres.

6.6.3 Advantage Actor-Critic

Advantage Actor-Critic (A2C) és una millora dels models de Policy Gradients. Aquesta tècnica es basa en fer que la xarxa neuronal calculi dos components:

- **Actor:** El component actor s'encarrega d'actualitzar la política del model (és a dir, la distribució de probabilitats sobre les accions a prendre).
- **Crític:** El component crític estima el valor de l'acció (advantage function), que mesura quant millor o pitjor és una acció comparada amb la mitjana de les accions possibles en aquest estat.

El crític dona retroalimentació a l'actor sobre la qualitat de les accions preses, permetent així una millor actualització de la política.

6.6.4 Resultats

En el cas dels models de Policy Gradients, els resultats no són tant prometedors com en els models DQN. El model PG bàsic, no ha estat capaç de convergir i sobrepassar dels -19 de recompensa mitjana.

Per altra banda, el model A2C sí que ha estat capaç de jugar al Pong de forma satisfactòria. Ha arribat als 19 de recompensa mitjana en 10.710.471 frames, a una velocitat mitjana 273 frames/s això significa que l'entrenament ha durat 10,9 hores.

En aquest cas, aquest número tan elevat de frames comparat amb el DQN pot ser degut a que en el model DQN,

cada frame es genera una nova experiència i s'entrena el model amb un batch de les experiències anteriors, en canvi, en el model A2C, cada frame s'enmagatzema una experiència i, després de ser utilitzades, aquestes experiències no es tornen a utilitzar, això fa que aquests models aprofitin més cada pas donat pel model.

7 ENTORNS MULTIAGENT

En aquest apartat es descriu el procés d'adaptació dels models de Reinforcement Learning (RL) per treballar en entorns multiagent, utilitzant la biblioteca PettingZoo. L'objectiu és explorar la interacció entre diferents agents en un entorn comú

7.1 Biblioteca Petting Zoo

PettingZoo és una biblioteca de Python derivada de Gymnasium que permet la simulació d'entorns multiagent. Aquesta biblioteca proporciona una interfície comuna per a diversos tipus de jocs i entorns on múltiples agents poden interactuar de manera competitiva o cooperativa. La biblioteca permet interactuar amb els entorns de dues maneres:

- **Agent Environment Cycle (AEC):** Aquesta manera d'interacció segueix un cicle on els agents actuen de manera seqüencial dins l'entorn. Cada agent observa l'estat actual, pren una acció, i després l'entorn actualitza el seu estat abans de passar al següent agent. Aquesta metodologia és útil per simular escenaris on les accions d'un agent depenen directament de les accions dels altres agents, com en jocs de taula o altres entorns seqüencials.

- **Parallel Environment:** En aquest mode, tots els agents interactuen amb l'entorn de manera simultània, prenent accions en paral·lel. Això és especialment útil per a escenaris on les decisions dels agents poden ser preses independentment o en temps real, com en simulacions de trànsit o entorns de robòtica col·laborativa.

7.2 Knights Archers and Zombies

Knights Archers and Zombies (KAZ) és un dels entorns proporcionats per la biblioteca PettingZoo, dissenyat específicament per a la investigació en sistemes multiagent. En aquest entorn, els agents prenen el paper de cavallers o arquers, cadascun amb les seves pròpies habilitats i objectius.

7.2.1 Descripció de l'entorn

Com es pot observar en la figura, a KAZ, els cavallers (color gris) i els arquers (color vermell) col·laboren per defensar-se contra les onades de zombis (color verd) que intenten arribar al seu territori (la part inferior de la pantalla). Els cavallers estan equipats amb espases per combatre en distàncies curtes, mentre que els arquers utilitzen arcs per atacar a distància. Els zombis, per altra banda, intenten apropar-se als humans per atacar-los i destruir les seves defenses.

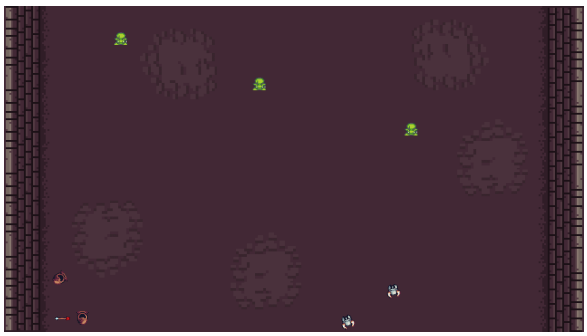

Fig. 7: Exemple de pantalla KAZ

7.2.2 Metodologia d'aprenentatge

Per entrenar els agents en aquest entorn, s'han utilitzat els algorismes que eren capaços de jugar a l'entorn Pong

- **Deep Q-Learning (DQN):** La versió més completa de DQN, que inclou totes les millores mencionades en l'apartat del Pong.

- **Advantage Actor-Critic:** La mateixa versió de A2C utilitzada en el joc del Pong

Els dos models han estat modificats per acceptar els inputs de l'entorn KAZ i per poder entrenar més d'un agent a la vegada, creant una xarxa neuronal per cada agent entrenat.

7.2.3 Resultats experimentals

Tot i els esforços invertits en l'entrenament dels agents en l'entorn KAZ, no s'ha aconseguit la convergència en cap dels models ni els agents, aquests no han estat capaços d'aprendre les dinàmiques del joc i els zombis guanyen sempre, sent ignorats pels cavallers i els arquers.

- **Cavallers:** Els cavallers entenen que si hi ha un zombi aprop han d'atacar-lo però no són capaços de perseguir els que entren en la seva línia de visió.

- **Arquers:** Els arquers disparen fletxes aleatòriament en la direcció de la que venen els zombis, aquesta ha estat l'estratègia que els hi ha proporcionat més recompensa durant l'entrenament.

Anàlisi de les dificultats Diversos factors van contribuir a la dificultat dels models per aprendre a jugar de manera efectiva en l'entorn KAZ:

1. **Complexitat de l'entorn:** L'entorn KAZ presenta un alt nivell de complexitat, amb molts estats possibles i interaccions entre agents. Això fa que sigui difícil per als models de RL explorar l'espai d'estats de manera exhaustiva i identificar les accions òptimes.

2. **Interaccions entre agents:** La necessitat de coordinació entre múltiples agents humans (cavallers i arquers) ha resultat ser un repte significatiu. Els algorismes de RL utilitzats no són suficients per a fomentar una col·laboració eficaç entre els agents.

3. **Recompenses difícils:** La funció de recompensa en l'entorn KAZ pot ser difícil de descompondre en passos intermedis útils per l'aprenentatge dels agents. Això pot portar a una retroalimentació insuficient per als agents, limitant la seva capacitat per aprendre de les seves accions.

8 Conclusions

La implementació de models de Reinforcement Learning en entorns multiagent ha demostrat ser una eina poderosa per a l'estudi de la interacció entre agents.

Els experiments en l'entorn Pong ha permès visualitzar l'aprenentatge de l'agent en un entorn complex com pot ser un joc de l'Atari. Els models actuals han demostrat ser suficients per jugar a aquests jocs, tot i que s'ha vist que tenen les seves limitacions. També s'ha pogut observar com els models DQN són superiors als models Policy Gradient per a entorns amb espais d'accions molt limitats. Els models DQN aprofiten més les accions que fa l'agent al ser capaços de guardar-les en un buffer per ser utilitzades en varis entrenaments, per tant, són més eficients en entorns on el cost dels samples és alt.

Els experiments en l'entorn KAZ han revelat les limitacions dels algorismes de RL actuals per a entorns multiagent més complexos. Tot i la falta de progrés en l'aprenentatge efectiu dels agents, aquests resultats subratllen la necessitat de desenvolupar nous mètodes i estratègies per abordar els reptes associats amb la complexitat dels entorns multiagent. Això podria incloure l'ús de tècniques avançades com l'aprenentatge profund col·laboratiu, el transfer learning entre agents, o la integració de models híbrids que combinin RL amb altres enfocaments d'intel·ligència artificial.

Agraïments

M'agradaria donar les gràcies al meu tutor per guiar-me i ajudar-me tant en el procés d'aprenentatge com en el de implementació. També donar les gràcies a familiars i amics, sense els quals hauria sigut més difícil poder arribar fins aquí.

Referències

[1] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction* The MIT Press 2020 Cambridge, Massachusetts London, England. Second Edition 978-0262193986

[2] Maxim Lapan. *Deep Reinforcement Learning Hands-On*. Packt Publishing 2018 Birmingham - Mumbai First Edition 978-1788834247

[3] Richard S. Sutton *Learning to Predict by the Methods of Temporal Differences*, Machine Learning 1988.

[4] Hado van Hasselt, Arthur Guez, David Silver. *Deep Reinforcement Learning with Double Q-learning*, Proceedings of the AAAI Conference on Artificial Intelligence 2015.

[5] Fortunato and others. *Noisy Networks for Exploration*, International Conference on Learning Representations (ICLR) 2018.

[6] Tom Schaul and others. *Prioritized Experience Replay*, International Conference on Learning Representations (ICLR) 2015.

[7] Ziyu Wang and others. *Dueling Network Architectures for Deep Reinforcement Learning*, International Conference on Machine Learning (ICML) 2015.

[8] DeepMind. *Discovering novel algorithms with AlphaTensor*, Nature 2022.

[9] Terry, J K and Black, Benjamin and Hari, Ananth *SuperSuit: Simple Microwrappers for Reinforcement Learning Environments*, 2020.

APÈNDIX

Descripció	Data Inici	Duració (dies)	Data final
Comprendre els fonaments del RL	11/03/2024	39	19/04/2024
Llegir llibre RL an Introduction	11/03/2024	7	18/03/2024
Llegir llibre DRL Hands-On	30/03/2024	20	19/04/2024
Aprendre sobre Gymnasium i PettingZoo	18/03/2024	50	07/05/2024
Llegir documentació Gymnasium	18/03/2024	1	19/03/2024
Llegir documentació PettingZoo	03/05/2024	4	07/05/2024
Implementar un model tabular	19/03/2024	11	30/03/2024
Escollir un entorn	19/03/2024	1	20/03/2024
Programar el model tabular	20/03/2024	5	25/03/2024
Implementar el model a Gymnasium	25/03/2024	3	28/03/2024
Probar el model	28/03/2024	2	30/03/2024
Implementar un model DRL	19/04/2024	14	03/05/2024
Escollir un entorn	19/04/2024	1	20/04/2024
Programar el model DRL	20/04/2024	7	27/04/2024
Implementar el model a Gymnasium	27/04/2024	4	01/05/2024
Probar el model	01/05/2024	2	03/05/2024
Modificar el model per ser multiagent	07/05/2024	17	24/05/2024
Crear un entorn multiagent	07/05/2024	10	17/05/2024
Fer que el agent funcioni a PettingZoo	17/05/2024	7	24/05/2024
Entrenar els models multiagent	24/05/2024	15	08/06/2024
Entrenar els models amb un agent	24/05/2024	3	27/05/2024
Entrenar els models amb més agents	27/05/2024	6	02/06/2024
Gravar videos de les partides	02/06/2024	1	03/06/2024
Analitzar resultats	03/06/2024	2	05/06/2024
Descriure conclusions	05/06/2024	3	08/06/2024
Escriure Informe final	08/06/2024	7	15/06/2024
		Data limit	16/06/2024

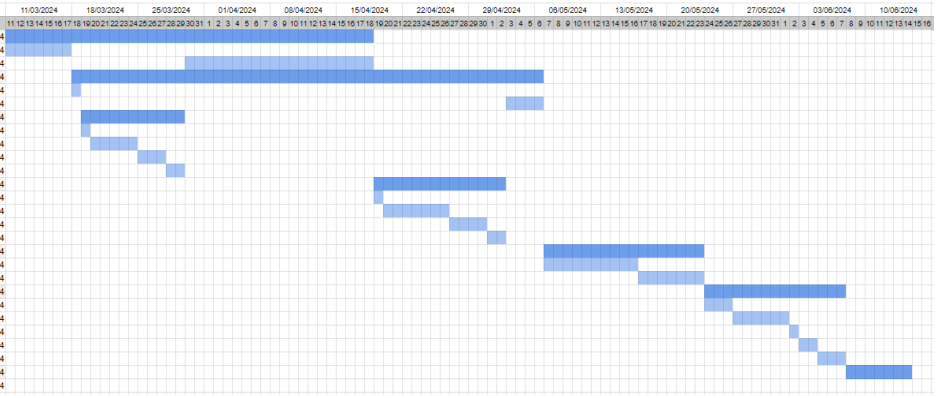

Fig. 8: Planificació del TFG