
This is the **published version** of the bachelor thesis:

Fernández Ortiz, Adrià. *Aprenentatge per reforç aplicat a videojocs clàssics i anàlisi de les dades d'entrenament mitjançant una pipeline de dades*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334369>

under the terms of the  license.

Aprenentatge per Reforç aplicat a videojocs clàssics i anàlisi del procés d'entrenament mitjançant una pipeline de dades

Adrià Fernández Ortiz

Resum—Aquest treball presenta el disseny i desenvolupament d'agents intel·ligents capaços de jugar als videojocs clàssics Snake i 2048 mitjançant tècniques d'aprenentatge per reforç. Per a cada joc s'han implementat dos enfocaments diferents: un agent basat en Q-Learning i un agent basat en Deep Q-Networks (DQN), que utilitza xarxes neuronals per aproximar la funció de valor. Paral·lelament, s'ha desenvolupat un sistema de registre de dades per emmagatzemar mètriques com les recompenses acumulades, les puntuacions obtingudes i l'evolució dels agents durant l'entrenament. Els resultats obtinguts permeten comparar quantitativament ambdós enfocaments i mostren que els agents DQN aconsegueixen un rendiment superior, especialment en entorns amb espais d'estats molt grans, com és el cas del joc 2048.

Paraules clau—Aprenentatge per reforç, Q-Learning, Deep Q-Networks, Snake, 2048, anàlisi de dades, pipeline de dades.

Abstract—This Project presents the design and development of intelligent agents capable of playing the classic video games Snake and 2048 using Reinforcement Learning techniques. Two different approaches were implemented for each game: a Q-Learning-based agent and a Deep Q-Network (DQN) agent, which uses artificial neural networks to approximate the action-value function. In addition, a data logging system was developed to record relevant training metrics such as accumulated rewards, game scores, and the agents' performance throughout the learning process. The collected data enabled a quantitative comparison of both approaches, showing that DQN agents achieve superior performance, particularly in environments with very large state spaces, such as the 2048 game.

Index Terms—Reinforcement Learning, Q-Learning, Deep Q-Networks, Snake, 2048, Data Analytics, Data Pipeline.

1 INTRODUCCIÓ - CONTEXT DEL TREBALL

APRENENTATGE per reforç (Reinforcement Learning, RL) és una branca de l'aprenentatge automàtic que estudia com un agent pot aprendre a prendre decisions mitjançant la interacció amb un entorn, amb l'objectiu de maximitzar una recompensa acumulada. A diferència de l'aprenentatge supervisat, en el RL l'agent no disposa de respostes correctes explícites, sinó que ha d'explorar l'espai d'estats i accions per descobrir estratègies òptimes.

En els darrers anys, el RL ha adquirit una gran rellevància gràcies a èxits destacats en videojocs i entorns complexos, especialment amb la introducció de tècniques com Deep Q-Networks (DQN). Els videojocs clàssics constitueixen un entorn ideal per experimentar amb aquests algorismes, ja que ofereixen regles definides, espais d'estat controlables i mètriques de rendiment clares.

Aquest TFG se centra en l'aplicació de tècniques d'aprenentatge per reforç als jocs Snake i 2048. Aquests dos entorns presenten característiques diferents: Snake té un espai d'estats seqüencial amb dependència temporal clara, mentre que 2048 combina estratègia a curt i llarg termini amb un component estocàstic.

A més del desenvolupament dels agents, el treball posa especial èmfasi en l'anàlisi del procés d'entrenament mitjançant una pipeline de dades que permeti registrar, emmagatzemar i visualitzar les mètriques generades durant l'aprenentatge. Aquest enfocament aporta una dimensió analítica que va més enllà de l'obtenció de bons resultats, centrant-se també en la comprensió del comportament dels agents.

L'objectiu general del treball és dissenyar, implementar i analitzar agents de RL aplicats a videojocs clàssics, integrant una metodologia experimental rigorosa basada en l'anàlisi de dades.

- E-mail de contacte: afernandezblanxart@gmail.com
- Menció realitzada: Computació
- Treball tutoritzat per: David Acuña Torras (Àrea de Ciències de la Computació i Intel·ligència Artificial)
- Curs 2025/26

2 OBJECTIUS DEL TFG

2.1 Objectiu general

Desenvolupar agents basats en l'aprenentatge per reforç

capaços de jugar als jocs Snake i 2048, i analitzar el seu procés d'entrenament mitjançant una pipeline de dades específica.

2.2 Objectius específics

1. Dissenyar i implementar versions programables dels jocs Snake i 2048 en Python.
2. Modelar formalment els entorns com a processos de decisió de Markov (MDP).
3. Implementar un agent basat en Q-Learning tabular per als jocs Snake i 2048.
4. Analitzar les limitacions dels mètodes tabulars en espais d'estats grans.
5. Implementar agents basats en funcions d'aproximació mitjançant xarxes neuronals (DQN) per tal de superar les limitacions dels mètodes tabulars
6. Desenvolupar una pipeline de dades per registrar:
 - Recompenses acumulades
 - Puntuacions
 - Nombre d'episodis
 - Evolució de la política
7. Crear eines de visualització per analitzar l'evolució temporal de l'aprenentatge.
8. Comparar el rendiment dels diferents enfocaments implementats.

L'abast del treball inclou el desenvolupament complet del sistema i l'anàlisi experimental, però no contempla la creació d'una interfície gràfica avançada ni l'optimització per a producció.

3 METODOLOGIA

La metodologia del treball es basa en quatre fases principals:

3.1 Modelització de l'entorn

Els jocs es definiran formalment com a MDP, especificant:

- Espai d'estats
- Espai d'accions
- Funció de recompensa
- Condicions de finalització

Es prestarà especial atenció a la definició de la funció de recompensa, ja que aquesta condiciona el comportament emergent de l'agent.

3.2 Implementació dels algorismes de RL

Tot i que inicialment es va plantejar l'ús de mètodes tabulars com a enfocament principal actualitzant la Q-Table segons l'equació:

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

Durant el desenvolupament s'han observat limitacions

importantes en entorns amb espais d'estats elevats.

Com a conseqüència, s'ha decidit incorporar una aproximació basada en Deep Reinforcement Learning [4] mitjançant Deep Q-Networks (DQN), amb l'objectiu de millorar la capacitat de generalització dels agents.

3.3 Disseny de la pipeline de dades

Es desenvoluparà un sistema de registre que capturi automàticament:

- Episodi
- Recompensa total
- Longitud de l'episodi
- Puntuació final
- Paràmetres d'entrenament

Les dades es guardaran en formats estructurats (CSV) per facilitar-ne l'anàlisi posterior.

3.4 Anàlisi i visualització

S'utilitzaran eines com Matplotlib per generar:

- Corbes d'aprenentatge
- Mitjanes mòbils
- Comparatives entre experiments

L'avaluació es basarà en mètriques quantitatives i en la interpretació del comportament emergent.

4 DESENVOLUPAMENT DEL TREBALL

4.1 Implementació dels entorns de joc

4.1.1 Joc Snake

El joc Snake constitueix un dels entorns utilitzats en aquest treball per a l'aplicació de tècniques d'aprenentatge per reforç. Es tracta d'un entorn discret, determinista i amb una dinàmica seqüencial clara, fet que el converteix en un cas adequat per a la validació inicial d'algorismes com Q-Learning i Deep Q-Networks.

Descripció general de l'entorn

L'objectiu del joc és controlar una serp que es desplaça per una graella bidimensional amb l'objectiu de consumir aliments que apareixen en posicions aleatòries. Cada vegada que la serp menja, augmenta la seva longitud. El joc finalitza quan la serp col·lisiona amb els límits de l'entorn o amb el seu propi cos.

L'entorn s'ha definit com una graella de dimensions fixes, on cada cel·la representa una possible posició discreta de la serp o del menjar.

Representació de l'entorn

La configuració del joc es defineix en el fitxer `config.py`, on

s'estableixen els paràmetres principals:

- Dimensions de la graella (amplada i alçada)
- Mida de cada cel·la
- Colors utilitzats per a la representació gràfica
- Velocitat d'actualització del joc (FPS)

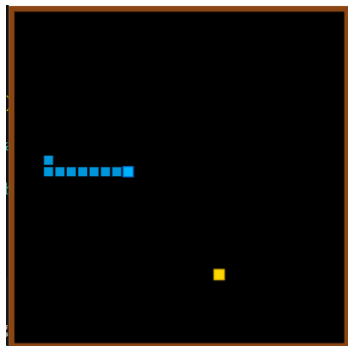


Figura 1. Interfície gràfica del joc Snake implementat

Modelització de la serp

La serp es modelitza mitjançant una estructura de dades de tipus *deque*, que permet insercions i eliminacions eficients als extrems. El cap de la serp es manté sempre a l'inici de la seqüència.

Cada element del cos de la serp es representa com una tupla (x, y) corresponent a una posició dins la graella.

El moviment de la serp es basa en una direcció actual, representada com un vector discret:

- $(1, 0)$: dreta
- $(-1, 0)$: esquerra
- $(0, 1)$: avall
- $(0, -1)$: amunt

Per evitar comportaments no vàlids, s'han implementat restriccions en el canvi de direcció:

- No es permet invertir la direcció (gir de 180 graus)
- Es limita a un únic canvi de direcció per iteració

El moviment es realitza afegint un nou cap en la direcció actual i eliminant l'últim segment del cos, excepte en el cas que la serp hagi de créixer.

Gestió del creixement

El creixement de la serp es controla mitjançant un flag (*growing*). Quan la serp consumeix un aliment, aquest flag s'activa, i en la següent iteració no s'elimina l'últim segment del cos, provocant un increment de longitud.

Generació i gestió del menjar

El menjar es representa com una posició aleatòria dins la

graella. La seva generació es realitza mitjançant valors enters dins els límits definits.

Per evitar inconsistències, s'ha implementat un mecanisme que garanteix que el menjar no aparegui en una posició ocupada per la serp. En cas contrari, es genera una nova posició fins que aquesta és vàlida.

Detecció de col·lisions

El sistema contempla dos tipus de col·lisions que determinen la finalització de l'episodi:

- Col·lisió amb les parets: quan el cap de la serp surt dels límits de la graella
- Col·lisió amb el propi cos: quan el cap coincideix amb qualsevol altra part del cos

Aquestes condicions permeten definir de manera clara els estats terminals de l'entorn.

Bucle principal del joc

El control del joc es realitza mitjançant un bucle principal implementat amb la llibreria *pygame*, que s'encarrega de:

1. Processar els esdeveniments d'entrada (teclat)
2. Actualitzar l'estat del joc (moviment de la serp, detecció de col·lisions, interacció amb el menjar)
3. Renderitzar l'entorn gràfic

La separació entre lògica de joc i representació visual facilita la futura integració amb agents d'aprenentatge per reforç, ja que permet substituir el control manual per decisions automàtiques sense modificar la dinàmica de l'entorn.

4.1.2 Joc 2048

El joc 2048 constitueix el segon entorn utilitzat en aquest treball i presenta una complexitat superior respecte a Snake, tant pel que fa a la representació de l'estat com a la dinàmica del joc. Es tracta d'un entorn discret, estocàstic i amb una component estratègica significativa, fet que el converteix en un cas d'estudi adequat per analitzar les limitacions dels mètodes tabulars i la necessitat d'utilitzar aproximacions basades en funcions.

Descripció general de l'entorn

L'objectiu del joc és combinar fitxes numèriques dins una graella per tal d'obtenir valors cada vegada més elevats, idealment arribant a la fitxa 2048. El joc es desenvolupa en una graella quadrada de mida fixa, on cada cel·la pot contenir un valor numèric o estar buida.

A cada moviment, totes les fitxes es desplacen en la direcció indicada (amunt, avall, esquerra o dreta). Quan dues fitxes amb el mateix valor entren en contacte, es

fusionen en una sola amb valor doble. Després de cada moviment vàlid, es genera una nova fitxa (2 o 4) en una posició aleatòria buida.

El joc finalitza quan no queden moviments possibles.

Representació de l'entorn

L'estat del joc es representa mitjançant una matriu bidimensional (*grid*) de dimensions fixes, on cada element correspon al valor d'una cel·la. Les cel·les buides es representen amb el valor 0.

Aquesta representació és més complexa que en el cas de Snake, ja que l'espai d'estats creix exponencialment amb el nombre de possibles combinacions de valors en la graella.

		2	4
		2	4
	2	16	4
8	16	32	4

Figura 2. Interfície gràfica del joc 2048 implementat

Modelització del tauler

La classe principal que gestiona la lògica del joc és *Board*, que encapsula l'estat de la graella i les operacions que es poden realitzar sobre aquesta.

En la inicialització, el tauler es crea buit i es generen dues fitxes inicials en posicions aleatòries. Aquesta inicialització introdueix ja una component estocàstica en l'entorn.

Dinàmica dels moviments

El moviment de les fitxes es descompon en tres operacions principals aplicades a cada fila o columna:

1. **Compressió:** desplaçament de totes les fitxes cap a una direcció eliminant els espais buits intermedis
2. **Fusió:** combinació de fitxes adjacents amb el mateix valor
3. **Nova compressió:** reorganització després de la fusió

El moviment cap a l'esquerra es defineix com a operació base (*move_left*), mentre que la resta de moviments es construeixen mitjançant rotacions del tauler.

Ús de rotacions per simplificar la lògica

Per evitar duplicació de codi, els moviments en altres direccions (dreta, amunt i avall) es defineixen a partir de rotacions successives del tauler. Aquesta estratègia permet reutilitzar la lògica del moviment cap a l'esquerra per a tots els casos.

Generació de noves fitxes

Després de cada moviment vàlid, es genera una nova fitxa en una posició aleatòria buida. El valor de la fitxa és 2 amb una probabilitat del 90% i 4 amb una probabilitat del 10%.

Aquest mecanisme introdueix una component estocàstica significativa en l'entorn, fet que dificulta l'aprenentatge i diferencia aquest joc d'altres entorns més deterministes com Snake.

Detecció de finalització

El joc finalitza quan no existeixen moviments possibles. Aquesta condició es verifica mitjançant:

- Absència de cel·les buides
- Inexistència de fitxes adjacents amb el mateix valor (tant horitzontalment com verticalment)

Bucle principal del joc

El control del joc es realitza mitjançant un bucle principal que:

1. Processa les entrades de l'usuari
2. Actualitza l'estat del tauler segons l'acció seleccionada
3. Renderitza la nova configuració del joc
4. Comprova la condició de finalització

Aquest esquema és coherent amb la formulació d'un entorn d'aprenentatge per reforç, on cada iteració correspon a una transició d'estat.

4.2 Implementació dels agents d'aprenentatge per reforç

4.2.1 Agents basats en Q-Learning

El Q-Learning és un algoritme d'aprenentatge per reforç basat en valors que permet a un agent aprendre una política òptima sense disposar d'un model de l'entorn. Aquest mètode es basa en l'estimació iterativa de la funció de valor acció-estat, representada mitjançant una Q-Table.

L'actualització dels valors de la Q-Table es realitza segons:

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

on:

- α és la taxa d'aprenentatge
- γ és el factor de descompte
- r és la recompensa immediata
- s' és el nou estat

4.2.1.1 Agent Snake

Representació de l'estat

Un dels aspectes més rellevants en la implementació de l'agent per al joc Snake és la definició de l'estat. Enlloc d'utilitzar la posició completa de la serp (que generaria un espai d'estats molt gran), s'ha optat per una representació reduïda basada en característiques rellevants de l'entorn.

L'estat es defineix com una tupla de valors binaris que inclou:

- Perill immediat en tres direccions (endavant, esquerra, dreta)
- Direcció actual de la serp (codificada en format one-hot)
- Posició relativa del menjar respecte al cap de la serp

```
def get_state(self):
    ...
    return (
        danger_straight,
        danger_left,
        danger_right,
        dir_up,
        dir_down,
        dir_left,
        dir_right,
        food_up,
        food_down,
        food_left,
        food_right
    )
```

Figura 3. Representació de l'estat utilitzat per l'agent de Snake.

Aquesta representació permet reduir significativament la dimensionalitat de l'espai d'estats, facilitant l'aprenentatge amb mètodes tabulars.

Espai d'accions

L'agent disposa de quatre accions possibles corresponents als moviments bàsics:

- Amunt
- Dreta
- Avall
- Esquerra

Aquestes accions es mapegen a valors enters per facilitar la seva gestió dins de la Q-Table.

Funció de recompensa

La funció de recompensa està dissenyada per incentivar

comportaments desitjables i penalitzar accions perjudicials:

- +10 per consumir menjar
- -10 en cas de col·lisió (estat terminal)
- Petita recompensa positiva (+0.2) si la serp s'acosta al menjar
- Penalització (-0.2) si s'allunya del menjar
- Penalització addicional (-0.01) per cada pas

Procés d'entrenament

L'entrenament es realitza mitjançant la interacció contínua entre l'agent i l'entorn, utilitzant el mètode *step* per obtenir transicions d'estat.

Una característica destacable de la implementació és l'ús d'un enfocament multi-agent, en el qual diversos agents s'entrenen en paral·lel en diferents instàncies de l'entorn. Aquest enfocament permet:

- Augmentar la velocitat d'exploració
- Diversificar les experiències recollides
- Millorar la qualitat de la Q-Table global

Després de cada generació, es combina el coneixement dels diferents agents en un únic model global, seleccionant els millors valors observats per a cada estat-acció.

A més, es guarda periòdicament el millor model obtingut, permetent la persistència i reutilització del coneixement adquirit.

Estratègia d'exploració

L'agent utilitza una política ϵ -greedy per equilibrar exploració i explotació.

Amb probabilitat ϵ , l'agent selecciona una acció aleatòria i amb probabilitat $1-\epsilon$, selecciona l'acció amb major valor Q estimat:

$$a = \operatorname{argmax}_Q(s, a)$$

Inicialment, el valor d' ϵ és elevat per afavorir l'exploració de l'entorn. Progressivament, aquest valor disminueix segons:

$$\epsilon = \max(\epsilon_{\min}, \epsilon * \text{decay})$$

Aquesta estratègia permet que l'agent explori moltes situacions durant les primeres fases de l'entrenament i posteriorment consolidi les estratègies apreses.

```
def act(self, state):
    if random.random() < self.epsilon:
        return random.randint(0, 3)
    return torch.argmax(self.policy_net(state)).item()
```

Figura 4. Estratègia ϵ -greedy per a la selecció d'accions durant l'entrenament del model.

4.2.1.2 Agent 2048

Representació de l'estat

A diferència del joc Snake, el joc 2048 presenta un espai d'estats considerablement més gran, fet que dificulta l'aplicació directa de mètodes tabulars. En aquest cas, l'estat es representa com una tupla de 16 valors corresponents a les cel·les del taulell (4x4).

Per tal de reduir la magnitud dels valors i facilitar l'aprenentatge, s'ha aplicat una transformació logarítmica basada en potències de 2. Concretament, cada valor del taulell es transforma segons:

- $0 \rightarrow 0$ (cel·la buida)
- $2 \rightarrow 1$
- $4 \rightarrow 2$
- $8 \rightarrow 3$
- ...

```
if value == 0:
    state_tuple.append(0)
else:
    power = int(math.log2(value))
    state_tuple.append(min(power, 15))
```

Figura 5. Transformació del taulell del joc 2048 en una representació discreta basada en potències de 2.

A més, els valors es limiten a un màxim per evitar creixements descontrolats. Aquesta representació permet mantenir una codificació més compacta de l'estat, tot i que l'espai resultant continua essent molt elevat.

Espai d'accions

L'agent disposa de quatre accions possibles:

- Amunt
- Dreta
- Avall
- Esquerra

No obstant això, no totes les accions són sempre vàlides. Per aquest motiu, s'ha implementat un mecanisme per determinar les accions possibles en cada estat, simulant moviments sobre una còpia del tauler i comprovant si produeixen canvis.

Funció de recompensa

El sistema de recompenses és un dels elements més importants del procés d'entrenament, ja que determina quins comportaments seran reforçats.

En aquest joc es va definir el següent esquema:

- recompensa positiva per fusionar tiles
- petita penalització per moviments sense fusió

- penalització elevada per moviments invàlids
- penalització final en perdre la partida

La recompensa per fusionar tiles es calcula de manera logarítmica mitjançant:

$$\text{reward} = \log_2(\text{score_gain})$$

Aquest enfocament evita que els valors de recompensa creixin excessivament quan apareixen tiles molt grans.

Procés d'entrenament

De manera similar al cas de Snake, l'entrenament es realitza utilitzant múltiples agents en paral·lel, cadascun interactuant amb una instància independent de l'entorn.

Al final de cada generació, es fusiona el coneixement adquirit pels diferents agents en una única Q-Table global, seleccionant els millors valors observats per a cada estat-acció. Aquest mecanisme permet accelerar el procés d'aprenentatge i aprofitar la diversitat d'experiències.

Estratègia d'exploració

L'agent utilitza una política ϵ -greedy amb decaïment progressiu del paràmetre ϵ , similar al cas de Snake. No obstant això, a causa de la gran variabilitat de l'entorn, es requereix un període d'exploració més prolongat per cobrir una part significativa de l'espai d'estats.

4.2.2 Agents basats en Deep Q-Networks

Tot i que els mètodes basats en Q-Learning tabular permeten resoldre problemes senzills amb espais d'estats reduïts, presenten limitacions importants quan s'apliquen a entorns més complexos.

Per tal de superar aquestes limitacions, s'ha optat per utilitzar Deep Q-Networks (DQN), una extensió del Q-Learning que empra xarxes neuronals per aproximar la funció de valor Q. Aquest enfocament permet treballar amb espais d'estats de gran dimensionalitat i millorar la capacitat de generalització de l'agent [3].

En aquest apartat es descriu la implementació dels agents basats en DQN per als jocs Snake i 2048, així com les principals decisions de disseny adoptades.

4.2.2.1 Agent Snake

A més de l'agent basat en Q-Learning, també es va desenvolupar un segon agent utilitzant la tècnica Deep Q-Network (DQN). Aquest enfocament substitueix la Q-Table per una xarxa neuronal capaç d'aproximar la funció de valor Q, fet que permet treballar amb representacions de l'estat més riques i millorar la capacitat de generalització davant situacions no observades durant l'entrenament.

En aquesta implementació es mantenen els mateixos

elements definits per a l'agent Q-Learning pel que fa a la representació de les accions, la política d'exploració ϵ -greedy i el sistema de recompenses. Per aquest motiu, aquests aspectes no es tornen a descriure en aquest apartat.

Arquitectura general del sistema DQN

A diferència de l'enfocament basat en Q-Table, on els valors Q s'emmagatzemen explícitament en una taula, el model DQN utilitza una xarxa neuronal per aproximar la funció de valor $Q(s,a)$.

Aquesta funció estima la recompensa futura esperada d'executar una acció concreta a en un estat determinat s .

L'arquitectura implementada segueix l'estructura clàssica de Deep Q-Learning i està formada pels següents components:

- Xarxa neuronal principal (*policy network*).
- Xarxa objectiu (*target network*).
- Replay buffer.
- Sistema d'exploració ϵ -greedy.

Representació de l'estat

L'estat utilitzat és una extensió del cas tabular i està format per 13 variables:

- Perills immediats (recte, dreta, esquerra)
- Direcció actual (codificació one-hot)
- Posició relativa del menjar
- Distància normalitzada en els eixos X i Y

A diferència del model amb Q-Table, s'introdueixen variables contínues (distàncies), fet que permet a la xarxa neuronal captar millor la relació espacial dins del joc.

Arquitectura de la xarxa neuronal

El model utilitzat és una xarxa neuronal totalment connectada (*fully connected*). Aquesta xarxa es pot entendre com una funció matemàtica que transforma un conjunt d'entrades (estat del joc) en un conjunt de sortides (valor de cada acció).

Concretament, la xarxa està formada per:

- Una capa d'entrada amb 13 valors (l'estat)
- Dues capes ocultes de 128 neurones cadascuna
- Una capa de sortida amb 4 neurones (una per acció possible)

```
self.net = nn.Sequential(
    nn.Linear(input_size, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, output_size)
)
```

Figura 6. Arquitectura de la xarxa neuronal utilitzada en el model DQN per aproximar la funció Q.

Cada neurona realitza una combinació lineal de les seves entrades:

$$z = w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n + b_z$$

on:

- x_i són les entrades del model (les variables que representen l'estat del joc).
- w_i són els pesos sinàptics, que determinen la importància de cada entrada en el càlcul de la neurona.
- b és el biaix (bias), un valor addicional que permet ajustar la sortida de la neurona independentment de les entrades.

Internament, les neurones de les capes ocultes aprenen patrons específics durant l'entrenament. Algunes poden especialitzar-se en detectar situacions de perill, mentre que altres poden aprendre configuracions favorables per apropar-se al menjar o evitar col·lisions.

Experience Replay

Durant l'entrenament, les experiències de l'agent (estat, acció, recompensa, següent estat) s'emmagatzemen en una memòria anomenada *replay buffer* [2].

En el codi, això es correspon amb la classe *ReplayBuffer*, que guarda les experiències mitjançant el mètode *push()*.

```
def push(self, state, action, reward, next_state, done):
    self.buffer.append((state, action, reward, next_state, done))
```

Figura 7. Emmagatzematge i mostreig d'experiències mitjançant el mecanisme de *Replay Buffer*.

Posteriorment, en lloc d'entrenar amb l'última experiència, es selecciona un conjunt aleatori (batch) utilitzant *sample()*. Aquest procés s'utilitza en el mètode *train()* de l'agent.

Aquesta tècnica és important perquè:

- Evita que el model aprengui de dades correlacionades
- Millora l'estabilitat de l'aprenentatge
- Permet reutilitzar experiències passades

Xarxa objectiu (Target Network)

Un dels principals problemes del Deep Q-Learning és la inestabilitat de l'entrenament causada pel fet que la

mateixa xarxa neuronal s'utilitza simultàniament per predir els valors Q actuals i calcular els valors objectiu de Bellman.

Això provoca que els objectius canviïn contínuament durant l'aprenentatge, generant oscil·lacions i possibles problemes de divergència. Per reduir aquesta inestabilitat, s'utilitza una segona xarxa neuronal anomenada Target Network.

Aquesta xarxa manté una còpia congelada dels pesos de la xarxa principal durant diversos passos d'entrenament. L'objectiu Q es calcula segons l'equació de Bellman:

$$Q_{\text{target}} = r + \gamma * \max_a Q_{\text{target}}(s', a')$$

on:

- r és la recompensa immediata
- γ és el factor de descompte
- s' és el següent estat

La target network s'actualitza periòdicament copiant els pesos de la policy network. Aquest mecanisme evita oscil·lacions excessives i problemes de divergència durant l'entrenament.

```
def update_target(self):
    self.target_net.load_state_dict(self.policy_net.state_dict())
```

Figura 8. Actualització periòdica de la xarxa objectiu per estabilitzar l'aprenentatge.

Procés d'entrenament

El sistema utilitza múltiples entorns executats en paral·lel, però un únic agent global que aprèn de totes les experiències generades.

El flux general és el següent:

1. L'agent selecciona una acció (*act()*)
2. L'entorn retorna el nou estat i la recompensa
3. L'experiència es guarda al *replay buffer* (*remember()*)
4. Es mostreja un batch aleatori
5. La xarxa s'actualitza (*train()*)

A més, cada cert nombre de passos es sincronitza la *target network*, tal com es veu en el bucle principal d'entrenament.

4.2.2.2 Agent 2048

Per superar les limitacions observades amb les Q-Tables en el joc 2048, es va implementar també un agent basat en Deep Q-Networks (DQN). Aquest enfocament permet aproximar la funció de valor mitjançant una xarxa neuronal, evitant així la necessitat d'emmagatzemar explícitament totes les combinacions possibles d'estats i accions.

En el cas de 2048, aquesta diferència és especialment important, ja que el nombre de configuracions possibles

del tauler és extremadament elevat. Mentre que amb Q-Learning tradicional l'espai d'estats creix de manera descontrolada, l'ús d'una xarxa neuronal permet generalitzar patrons i prendre decisions sobre estats no observats prèviament durant l'entrenament.

Pel que fa al procés d'aprenentatge, aquest agent utilitza els mateixos mecanismes descrits anteriorment per al DQN del joc Snake: una política d'exploració ϵ -greedy, un buffer d'experiències (*Experience Replay*), una xarxa objectiu (*Target Network*) per estabilitzar l'actualització dels valors Q, i la mateixa representació de l'estat i sistema de recompenses definit per al joc 2048 implementat mitjançant Q-Learning.

Arquitectura general del sistema

L'agent DQN desenvolupat per al joc 2048 segueix la mateixa arquitectura general implementada en el cas del joc Snake:

- policy network
- target network
- replay buffer
- exploració ϵ -greedy

Arquitectura de la xarxa neuronal

La xarxa neuronal és l'element principal de l'agent DQN. La seva funció és rebre l'estat actual del taulell i calcular quin valor Q té cadascuna de les quatre accions possibles.

La implementació utilitza una xarxa *feed-forward* completament connectada:

```
def __init__(self, input_size, output_size):
    super().__init__()
    self.net = nn.Sequential(
        nn.Linear(input_size, 256),
        nn.ReLU(),
        nn.Linear(256, 256),
        nn.ReLU(),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Linear(128, output_size)
    )
```

Figura 9. Arquitectura de la xarxa neuronal utilitzada per l'agent DQN del joc 2048.

L'entrada de la xarxa està formada pels 16 valors del taulell. A partir d'aquesta informació, la xarxa processa els patrons interns mitjançant diverses capes ocultes.

Cada neurona realitza una combinació matemàtica dels valors d'entrada. Internament:

1. Rep diversos valors.
2. Multiplica cada valor per un pes intern.
3. Suma tots els resultats.
4. Aplica una funció d'activació.

L'objectiu és detectar patrons útils dins del taulell.

Per exemple, algunes neurones poden acabar especialitzant-se en detectar:

- Fitxes grans agrupades.
- Files ordenades.
- Zones del taulell amb espais lliures.
- Possibles combinacions futures.

Durant l'entrenament, els pesos de les neurones es van modificant automàticament per aprendre quins patrons condueixen a puntuacions més elevades.

Entre les capes lineals s'utilitza la funció d'activació ReLU (*Rectified Linear Unit*). Aquesta funció introdueix no linealitat dins la xarxa neuronal, permetent que el model aprengui patrons complexos. Sense funcions d'activació, tota la xarxa es comportaria com una única operació lineal simple.

La capa de sortida retorna els valors Q estimats per a les quatre accions possibles:

$$Q(s,up), Q(s,right), Q(s,down), Q(s,left)$$

4.3 Visualització de resultats i gràfiques d'entrenament

Per analitzar l'evolució dels diferents agents entrenats, es va implementar un sistema de visualització de dades basat en gràfiques generades a partir dels registres d'entrenament guardats en fitxers CSV.

Durant cada episodi d'entrenament es registren valors com:

- Episodi actual.
- Score obtingut.
- Valor d'epsilon.

Posteriorment, aquestes dades es carreguen i es representen gràficament mitjançant la llibreria Matplotlib.

Per facilitar la interpretació dels resultats, es va aplicar una mitjana mòbil de 50 episodis. Aquest suavitzat redueix les oscil·lacions produïdes per l'exploració aleatòria pròpia del reinforcement learning i permet observar millor la tendència general de l'aprenentatge.

Les gràfiques generades mostren principalment:

- Evolució del score al llarg dels episodis.
- Velocitat d'aprenentatge dels agents.
- Estabilitat del model.
- Comparativa entre agents Q-Table i DQN.

Aquestes comparatives permeten observar visualment les diferències de rendiment entre enfocaments tabulars i xarxes neuronals, tant en el joc Snake com en el 2048.

5 PRESENTACIÓ I DISCUSSIÓ DE RESULTATS

Per avaluar el rendiment dels agents desenvolupats s'han comparat les dues aproximacions implementades en aquest treball: Q-Learning mitjançant Q-Table i Deep Q-Network (DQN). L'avaluació s'ha realitzat als jocs Snake i 2048 analitzant l'evolució de la puntuació obtinguda al llarg dels episodis d'entrenament.

5.1 Resultats del joc Snake

La Figura 10 mostra l'evolució dels agents entrenats per al joc Snake. Durant les primeres fases de l'entrenament tots dos models presenten un comportament similar, ja que predomina l'exploració de l'entorn.

A partir d'aproximadament l'episodi 4000, l'agent DQN comença a superar clarament el model basat en Q-Table, estabilitzant-se al voltant dels 55-60 punts davant dels aproximadament 25 punts assolits per l'agent tabular.

Aquesta diferència s'explica principalment per la capacitat del DQN de generalitzar entre estats mitjançant la xarxa neuronal. A més, el model DQN utilitza una representació de l'estat més completa, mentre que la Q-Table requereix reduir el nombre d'estats per evitar una explosió combinatòria.

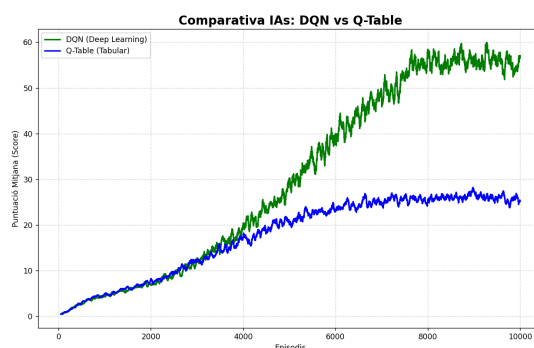


Figura 10 Comparativa de l'evolució dels agents DQN i Q-Table al joc Snake.

5.2 Resultats del joc 2048

La Figura 11 mostra els resultats obtinguts al joc 2048. En aquest entorn la diferència entre ambdós models és encara més significativa.

L'agent Q-Table es manté pràcticament estable al voltant dels 330-350 punts durant tot l'entrenament, mentre que l'agent DQN continua millorant progressivament fins a assolir puntuacions properes als 900-1000 punts.

Aquest resultat confirma que les aproximacions tabulars presenten importants limitacions quan l'espai d'estats és molt gran. En canvi, el DQN és capaç d'aprendre patrons generals i desenvolupar estratègies més eficients, obtenint un rendiment molt superior.

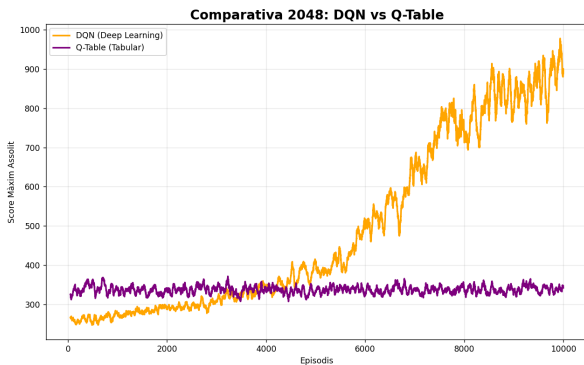


Figura 11. Comparativa de l'evolució dels agents DQN i Q-Table al joc 2048.

6 CONCLUSIONS

Aquest Treball Final de Grau ha permès desenvolupar i comparar dues aproximacions d'aprenentatge per reforç aplicades als videojocs Snake i 2048: Q-Learning mitjançant Q-Table i Deep Q-Networks.

Els resultats obtinguts mostren que els models basats en DQN ofereixen un millor rendiment en ambdós entorns, especialment en el joc 2048, on la gran dimensió de l'espai d'estats limita considerablement l'eficàcia de les Q-Tables. Això confirma que les xarxes neuronals constitueixen una solució més escalable per a problemes de major complexitat.

A nivell personal, aquest projecte m'ha permès aprofundir en els principals conceptes de l'aprenentatge per reforç, comprendre les diferències entre els mètodes tabulars i els basats en xarxes neuronals, així com adquirir experiència en el disseny d'entorns d'entrenament, la definició de funcions de recompensa i l'anàlisi dels resultats obtinguts.

Com a treball futur, es podria estudiar la incorporació d'algorismes més avançats, com Double DQN, Dueling DQN o Prioritized Experience Replay, així com ampliar el projecte a altres videojocs o entorns de major complexitat.

ÚS DE LA IA GENERATIVA

Durant el desenvolupament del treball s'utilitzaran eines d'IA generativa com a suport en tasques de redacció, revisió lingüística i aclariment conceptual. En cap cas la IA substituirà el desenvolupament tècnic ni el disseny experimental del projecte. Tot el codi implementat serà desenvolupat i validat per l'estudiant.

L'ús d'aquestes eines es limitarà a:

- Millora de redacció acadèmica
- Suggestiments estructurals
- Resolució de dubtes puntuals

AGRAÏMENTS

La realització d'aquest Treball Final de Grau ha representat el tancament d'una etapa important de la meua formació com a enginyer informàtic. Per aquest motiu, m'agradaria dedicar unes línies a totes aquelles persones que, d'una manera o altra, han contribuït a fer possible aquest projecte.

En primer lloc, vull agrair al meu tutor, David, el seguiment realitzat durant el desenvolupament del treball. Les seves observacions, correccions i recomanacions han estat de gran ajuda per orientar el projecte i millorar tant els aspectes tècnics com la qualitat de la memòria.

També voldria donar les gràcies als meus companys i amics, Eduard i Luis Adrian, per l'interès que han mostrat pel projecte des del primer moment. Els seus comentaris i suggeriments aportats durant les diferents fases de desenvolupament, han permès detectar possibles millores i contribuir a obtenir un resultat més complet.

Per últim, vull expressar el meu més profund agraïment a la meua família. El seu suport constant, la confiança dipositada en mi i els ànims rebuts al llarg de tota la carrera han estat essencials per arribar fins aquí. Aquest treball també és, en part, el resultat del seu esforç i del seu recolzament durant tots aquests anys.

BIBLIOGRAFIA

- [1] Towers, J. et al. (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments. Farama Foundation. Aquesta documentació recull l'estàndard de la indústria per a la creació d'entorns de simulació en IA. S'ha consultat per dissenyar i programar les versions interactives de Snake i 2048, garantint que la pipeline de dades rebí correctament els vectors d'estat, accions i recompenses.
- [2] PyTorch Contributors. (2026). Reinforcement Learning (DQN) Tutorial. PyTorch Documentation. Aquest tutorial tècnic detalla la implementació cap a final d'un agent Deep Q-Network utilitzant PyTorch. S'ha consultat per entendre l'arquitectura pràctica del codi, específicament com dissenyar i connectar el *Replay Buffer* i la *Target Network* per estabilitzar l'aprenentatge del nostre agent.
- [3] Mnih, V. et al. (2015). *Human-level control through deep reinforcement learning*. Nature. Aquest article introdueix el Deep Q-Network (DQN), combinant Q-Learning amb xarxes neuronals profundes. S'ha consultat per entendre les limitacions dels mètodes tabulars en espais d'estats grans i com es poden superar mitjançant aproximació de funció, especialment rellevant per al joc 2048.
- [4] Simonini, T., & Hugging Face Team. (2023). Deep Reinforcement Learning Course. Hugging Face. Aquest curs proporciona una introducció teoricopràctica completa als algorismes d'aprenentatge per reforç. S'ha consultat per comprendre els fonaments conceptuals de la transició des del Q-Learning mètode tabular fins als mètodes basats en Deep RL, servint de guia per estructurar el procés d'entrenament inicial.