Aplicació de tècniques d'aprenentatge computacional per la creació d'agents jugadors de Sushi Go.

Jose Montufo Rosal
Grau d'Enginyeria Informàtica
Àrea d'Intel·ligència Artificial

Nom Consultor/a: Joan M. Nuñez Do Rio
Professor responsable de l'assignatura: Carles Ventura Royo

Juny 2020

FITXA DEL TREBALL FINAL

Títol del treball:	*Aplicació de tècniques d'aprenentatge computacional per la creació d'agents jugadors de Sushi Go.*
Nom de l'autor:	*Jose Montufo Rosal*
Nom del consultor/a:	*Joan M. Nuñez Do Rio*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament:	*06/2020*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Intel·ligència Artificial*
Idioma del treball:	*Català*
Paraules clau	*Reinforcement Learning, AI Bot, Game*

Resum del Treball (màxim 250 paraules):

L'aplicació de tècniques d'aprenentatge per reforç als jocs de taula ha estat l'objecte en els darrers anys de multitud de projectes entre la comunitat científica especialitzada. Les mecàniques i les regles dels jocs de taula acostumen a formar un entorn idoni per ser utilitzats com a banc de proves de les eines que proporciona l'àrea de l'aprenentatge per reforç.

Aquest projecte va néixer amb la finalitat d'utilitzar el joc de cartes Sushi Go com a base per a la creació de diversos agents intel·ligents capaços d'aprendre una estratègia que els hi permeti resultar competitius a un humà. Els objectius del projecte són la comparació del rendiment que proporcionen diverses tècniques d'aprenentatge per reforç, estudiar l'estratègia òptima que utilitzen, i crear una UI que permeti els usuaris enfrontar-se als agents.

Per aconseguir aquesta finalitat, s'ha modificat una implementació preexistent per construir un entorn estàndard d'OpenAI Gym per a Sushi Go. Posteriorment, s'ha utilitzat l'entorn per aplicar els diferents algoritmes d'aprenentatge en la creació dels agents. Finalment, s'ha realitzat la comparació entre els agents per determinar els algoritmes més òptims, i s'ha descrit l'estratègia que segueixen els agents amb millor rendiment.

Al final del projecte, l'autor s'ha enfrontat en una sèrie de partides amb el millor agent, sent capaç de guanyar gairebé la totalitat. Aquest fet no fa més que indicar que els agents encara tenen molt marge de millora, sigui aplicant nous algoritmes, o ampliant l'espai d'estats que utilitzen per obtenir informació de l'entorn.

Abstract (in English, 250 words or less):

The application of reinforcement learning techniques to board games has been the subject in recent years of many projects among the specialized scientific community. The mechanics and rules of board games tend to form an ideal environment to be used as a test bed for the tools provided by the area of reinforcement learning.

This project was born in order to use the Sushi Go card game as a basis for creating various intelligent agents capable of learning a strategy that would allow them to be competitive against a human. The goals of the project are to compare performance provided by various reinforcement learning techniques, study the optimal strategy they use, and create a UI that allows users to confront agents.

To achieve this goal, a pre-existing implementation has been modified to build a standard OpenAI Gym environment for Sushi Go. Subsequently, the environment has been used to apply the different learning algorithms in the creation of the agents. Finally, the comparison between the agents to determine the most optimal algorithms was performed, and the strategy followed by the best performing agents was described.

At the end of the project, the author has challenged in a series of games to the best agent, being able to win almost all. This fact only indicates that agents still have much room for improvement, either by applying new algorithms, or by expanding the state space they use to obtain information from the environment.

Agraïments

Tanta gent a la qual agrair…

Però tots ells em permetran que
aquest treball estigui dedicat en
exclusiva a una única persona.

Mama, tenias razón como siempre.
“Lo conseguirás”.

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

L'aplicació de tècniques d'aprenentatge computacional al món dels jocs, siguin videojocs o jocs de taula, ha estat l'objecte en els darrers anys de multitud de projectes i investigacions entre la comunitat científica especialitzada en sistemes d'intel·ligència artificial.

L'aprenentatge computacional (ML pel seu nom en anglès, Machine Learning), com a branca dins de l'àrea de la intel·ligència artificial, s'encarrega de proporcionar tècniques i mètodes per la creació de sistemes que tinguin la capacitat de determinar el seu comportament a partir de l'experiència. I quina és la necessitat d'aquests sistemes? Per quin motiu el programador no defineixen directament el comportament del sistema? Perquè no sempre és possible. A vegades, el programador no pot anticipar totes les situacions a les quals es trobarà el sistema, o no es poden predir totes les variacions de l'entorn al qual s'adreça el sistema creat. També ens trobem amb tasques que el programador pot realitzar sense problema, però que la seva programació resulta humanament impossible. Qualsevol programador serà capaç de llegir la lletra manuscrita, però cap d'ells serà capaç de programar un sistema de lectura sense utilitzar mètodes d'aprenentatge automàtic.[6]

Les aplicacions del ML són tantes que seria impossible enumerar-les totes. Sistemes de reconeixement de veu, reconeixement facial, classificació d'imatges, classificació de seqüències d'ADN, vehicles autònoms, robòtica, anàlisi de comportament pel consum, anàlisi de productivitat, diagnòstic mèdic, etcètera ...

Tot i que les primeres referències a la disciplina van aparèixer a la dècada dels 50, no va ser fins als 90 que es va produir l'explosió definitiva a causa de la capacitat més gran de càlcul dels sistemes, l'accés a grans quantitats de dades, i a l'aparició de l'aprenentatge profund amb les xarxes neuronals. El creixement a la darrera dècada ha estat tan gran, i les capacitats dels agents han augmentat tan ràpidament, que fins i tot una organització d'investigació en el camp de la intel·ligència artificial, OpenAI [8], va ser creada amb l'objectiu de promoure i desenvolupar la disciplina, però a la vegada investigar l'amenaça que suposa per a la humanitat la possibilitat d'arribar a generar intel·ligències artificials de nivell humà.

Una de les primeres aportacions d'OpenAI va ser la publicació d'un entorn de treball específic per a l'estandardització dels entorns per a projectes de l'àreade l'aprenentatge per reforç, l'OpenAI Gym. La interfície proporcionada per OpenAI Gym facilita la documentació, publicació, desenvolupament, comparació i reutilització de projectes de RL.

L’aprenentatge per reforç (RL pel seu nom en anglès, Reinforcement Learning) és una de les tres grans àrees dins del ML, juntament amb l’aprenentatge supervisat i l’aprenentatge no supervisat. La diferència entre les àrees del ML radica en la informació de la qual es disposa sobre les dades que s’utilitzen per generar el coneixement. Els sistemes supervisats i els no supervisats disposen de totes les dades a l’inici del procés d’aprenentatge, i per tant el procés no té cap influència sobre les dades obtingudes. Als sistemes de RL, en canvi, les dades li arriben al llarg de tot el procés d'aprenentatge, i la interacció entre el sistema que està aprenent i l’entorn sobre el qual funciona influeix directament en quines dades rep i quina informació obté el sistema per poder aprendre.

Els mètodes de RL basen la seva capacitat d'aprendre a interactuar amb l’entorn en la prova i error. Cadascuna de les decisions que prengui el sistema suposarà l'obtenció d’un premi o d’un càstig, depenent de l’acció realitzada. Inicialment, l'agent no tindrà cap coneixement sobre quina estratègia seguir per tal d'obtenir la màxima quantitat de premis possible. Per tant, les decisions preses en el moment de realitzar una acció es basaran en l'atzar. Aquestes decisions preses, però, tindran com a resultat un nombre de premis més o menys alt al llarg de la interacció amb l’entorn. Les millors recompenses reforçaran les accions preses, mentre que les pitjors les penalitzaran. D'aquesta manera el sistema, amb el pas del temps, coneixerà les accions que proporcionen un major reforç, i cada cop basarà més les seves decisions en aquest coneixement i menys en l'atzar. En finalitzar el procés d'aprenentatge, disposarem d'un sistema que utilitzarà el coneixement obtingut per triar aquelles accions que li proporcionen un millor resultat al llarg del temps.

Els jocs de taula són entorns que s'adapten molt bé als sistemes de RL. Hi ha un element, el jugador, que ha d'adquirir un coneixement sobre el funcionament del joc. I no només això, sinó que ha de ser capaç de trobar la millor estratègia que el porti a aconseguir un objectiu concret: aconseguir una puntuació màxima, guanyar la partida a un adversari, o complir uns objectius determinats. Al llarg de la partida el jugador realitzarà una sèrie de jugades que li proporcionaran una recompensa en funció d'en quina mesura li apropa a aconseguir l'objectiu final.

Com a demostració de la utilitat de les tècniques de RL quan es relacionen amb el món dels jocs de taula, trobem l'èxit del projecte AlphaGo [9] (i posteriorment de l'AlphaGo Zero), que va ser capaç de crear un agent capaç de guanyar els millors jugadors professionals de Go. Aquesta fita va ser notícia a gran part dels informatius de tot el món. Per aconseguir-ho, es va utilitzar una combinació de diverses tècniques de RL que, utilitzades conjuntament, han permès crear un jugador artificial capaç de triar l'estratègia més òptima per aconseguir l'objectiu de guanyar partides de Go. Actualment, les estratègies utilitzades per l'Alpha Go Zero durant les seves partides són analitzades a escoles de jugadors de Go per tal de ser aplicades per jugadors humans.

Sushi Go [4] és un joc de cartes del tipus "filler", és a dir, un joc a on no es requereix molt temps de preparació de les partides, aquestes no solen ser de llarga durada, i les regles són senzilles i fàcils de recordar. No obstant això, tot i la seva senzillesa, les regles han de proporcionar al joc d'una profunditat suficient perquè arribar a l'objectiu final suposi un repte per als participants, de forma que l'atzar no cobri més importància que l'estratègia o l'habilitat dels jugadors.

L'objectiu d'aquest projecte consisteix a aplicar els mètodes proporcionats per la disciplina de l'aprenentatge per reforç al joc Sushi Go. Per poder fer-ho, abans haurem de disposar d'una implementació estable del joc, adaptat a la interfície d'OpenAI Gym. D'aquesta manera, l'entorn estarà disponible per qualsevol que vulgui realitzar la seva pròpia versió dels agents, ja que serà totalment reutilitzable.

Els agents generats, a més a més de poder ser utilitzats per suposar un repte per a jugadors humans, tindran dues funcions addicionals: d'una banda, pretenem que la comparació dels agents generats ens permeti treure conclusions sobre el rendiment dels mètodes de RL utilitzat. D'altra banda, s'analitzarà l'estratègia utilitzada pels agents que tinguin un millor comportament amb l'objectiu d'aprendre d'aquestes estratègies per aplicar-les a partides entre humans.

Tot i que en aquest projecte ens hem enfocat en el món dels jocs, les tècniques de RL tenen també aplicacions en el món de la medicina [10], la biologia [11] o la química [12].

1.2 Objectius del Treball

1.2.1 Objectius generals

Ens marquem els següents objectius generals pel nostre projecte:

- Utilització dels coneixements obtinguts al llarg del grau per tal d'aplicar-los en el context del projecte triat.
- Adquisició de nous coneixements, especialment en l'àmbit de la intel·ligència artificial i de l'aprenentatge computacional, més enllà dels obtinguts a les assignatures prèvies del grau. Més concretament, formació en els mètodes propis de l'àrea de l'aprenentatge per reforç.
- Aplicació i consolidació els coneixements adquirits de gestió de projectes de tipus predictiu (o tradicional).
- Generació d'una interfície que permeti a un usuari jugar al Sushi Go i enfrontar-se a agents intel·ligents entrenats mitjançant tècniques d'aprenentatge reforçat.
- Extracció de conclusions que puguin ser aplicades a altres jocs, o a altres processos de la vida real.

1.2.1 Objectius específics

A l'inici del projecte, per tal de determinar els objectius específics, s'ha realitzat un estudi inicial sobre les tècniques pròpies de l'aprenentatge per reforç, per tal de determinar les més comunes. Aquesta formació inicial ens va portar a determinar els següents objectius específics per al projecte:

- Preparació de l'entorn que permeti la realització del projecte.
- Estudi dels altres projectes ja existents, per tal d'arribar a un enteniment de què fan i com ho fan.
- Adaptació i millora d'una de les implementacions del joc disponibles. Reestructuració seguint el paradigma de la orientació a objectes.
- Implementació d'una estructura de gestió de les accions, i una estructura de gestió dels estats de l'entorn, el més independents posible de la resta de la implementació, que permeti que la seva gestió i modificació sigui transparent a la resta del sistema creat.
- Creació d'una UI que utilitzi la implementació realitzada per permetre a un usuari jugar al joc.
- Realització de les adaptacions necessàries per poder registrar la implementació del joc com a entorn d'OpenAI gym.
- Aplicació de diferents tècniques d'aprenentatge per reforç per generar agents jugadors de Sushi Go.
    - Agent basat en l'algoritme Q-Learning.
    - Agent basat en l'algoritme Deep Q-Learning.
    - Agent basat en Policy Gradients.
    - Agent basat en mètodes de Monte Carlo.
- Per cadascun dels agents:
    - Realitzar una tasca d'investigació i aprenentatge de l'algoritme.
    - Definir l'estat o estats a utilitzar.
    - Implementar l'agent.
    - Aplicar les millores a l'algoritme disponibles.
    - Determinar els paràmetres que donen un millor resultat.
- Comparativa del rendiment dels agents generats amb diferents tècniques entre ells. Fer que juguin partides entre ells i anàlisi de qui té un percentatge més gran de victòries.
- Extracció d'informació rellevant de l'estratègia utilitzada pels agents que donen un millor rendiment, per obtenir conclusions.

1.3 Enfocament i mètode seguit

El projecte consta de tres fases ben diferenciades, cadascuna de les quals requereix la finalització de l'anterior per tal de ser realitzada. La metodologia a utilitzar pel que fa al desenvolupament general del projecte serà un procés seqüencial clàssic en cascada. No obstant això, al llarg del desenvolupament del projecte ens trobarem en situacions a les quals sigui necessari realitzar modificacions al treball de les fases prèvies. Les tres fases que es duran a terme són les següents:

1. Implementació de l'entorn del joc

Per tal de poder realitzar tasques d'aprenentatge que ens permetin generar agents intel·ligents que utilitzin estratègies òptimes en el joc del Sushi Go, el primer que necessitem és una implementació de les regles del joc, que ens permeti simular partides de forma automàtica. Per fer-ho, partirem d'una de les implementacions existents [2], i la modificarem i ampliarem per a adaptar-la a les peculiaritats del nostre projecte. S'estructurarà el codi en classes i mòduls separats, aplicant la metodologia de l'orientació a objectes. També es realitzaran canvis a les regles del joc implementat al projecte existent, ja que està basada en l'ampliació del joc i no en les regles de la versió original, que és la que utilitzarem en el nostre projecte. A més a més, es realitzaran les adaptacions necessàries perquè la implementació segueixi l'estructura estàndard d'OpenAI Gym, i es generaran estructures independents per a determinar els estats i les accions del joc que ens permetran generar de forma flexible diferents versions dels agents a les fases posteriors. Finalment, en aquesta fase desenvoluparem la UI que permetrà realitzar partides entre un jugador humà i els agents que es generaran a les fases posteriors.

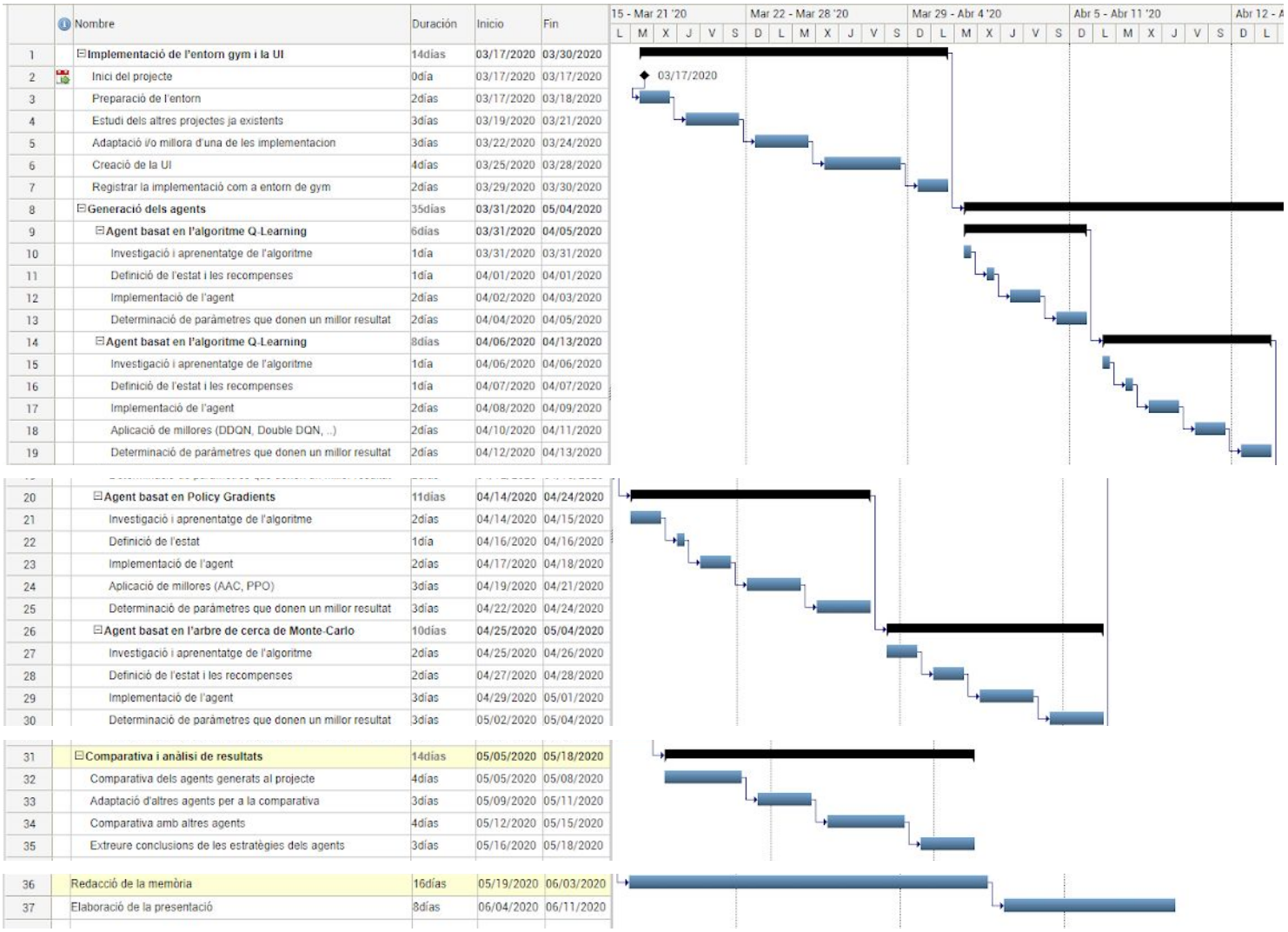
2. Generació d'agents

A aquesta segona fase, s'estudiaran diversos mètodes d'aprenentatge per reforç, i s'utilitzarà l'entorn generat a la fase 1 per tal de crear els agents que es comportin de forma òptima per tal de maximitzar la puntuació obtinguda al joc. El rendiment dels agents creats a partir d'un mateix algoritme està directament determinat tant pels estats utilitzats per a la seva construcció, i pels hiperparàmetres que s'utilitzen. Per a un mateix algoritme, és possible realitzar múltiples versions diferents amb resultats molt diversos. Per tal de determinar la millor combinació d'estats i hiperparàmetres possible per a cadascun dels algoritmes d'aprenentatge per reforç quan els apliquem al nostre entorn, es compararà el rendiment obtingut amb cada combinació i es triarà la versió que obtingui un resultat més òptim.

3. Comparativa i anàlisi de resultats

A la darrera fase es realitzarà una comparativa entre els agents generats pels diferents algoritmes per tal de determinar quin aconsegueix un comportament més òptim, tant en nombre de victòries com en puntuació mitja. A més a més, s'obtindran estadístiques de les accions triades pels millors agents per tal d'extreure les característiques pròpies de les estratègies més òptimes.

1.4 Planificació inicial del Treball



	Nombre	Duración	Inicio	Fin
1	**Implementació de l'entorn gym i la UI**	14días	03/17/2020	03/30/2020
2	Inici del projecte	0día	03/17/2020	03/17/2020
3	Preparació de l'entorn	2días	03/17/2020	03/18/2020
4	Estudi dels altres projectes ja existents	3días	03/19/2020	03/21/2020
5	Adaptació i/o millora d'una de les implementacion	3días	03/22/2020	03/24/2020
6	Creació de la UI	4días	03/25/2020	03/28/2020
7	Registrar la implementació com a entorn de gym	2días	03/29/2020	03/30/2020
8	**Generació dels agents**	35días	03/31/2020	05/04/2020
9	**Agent basat en l'algoritme Q-Learning**	6días	03/31/2020	04/05/2020
10	Investigació i aprenentatge de l'algoritme	1día	03/31/2020	03/31/2020
11	Definició de l'estat i les recompenses	1día	04/01/2020	04/01/2020
12	Implementació de l'agent	2días	04/02/2020	04/03/2020
13	Determinació de paràmetres que donen un millor resultat	2días	04/04/2020	04/05/2020
14	**Agent basat en l'algoritme Q-Learning**	8días	04/06/2020	04/13/2020
15	Investigació i aprenentatge de l'algoritme	1día	04/06/2020	04/06/2020
16	Definició de l'estat i les recompenses	1día	04/07/2020	04/07/2020
17	Implementació de l'agent	2días	04/08/2020	04/09/2020
18	Aplicació de millores (DDQN, Double DQN, ...)	2días	04/10/2020	04/11/2020
19	Determinació de paràmetres que donen un millor resultat	2días	04/12/2020	04/13/2020
20	**Agent basat en Policy Gradients**	11días	04/14/2020	04/24/2020
21	Investigació i aprenentatge de l'algoritme	2días	04/14/2020	04/15/2020
22	Definició de l'estat	1día	04/16/2020	04/16/2020
23	Implementació de l'agent	2días	04/17/2020	04/18/2020
24	Aplicació de millores (AAC, PPO)	3días	04/19/2020	04/21/2020
25	Determinació de paràmetres que donen un millor resultat	3días	04/22/2020	04/24/2020
26	**Agent basat en l'arbre de cerca de Monte-Carlo**	10días	04/25/2020	05/04/2020
27	Investigació i aprenentatge de l'algoritme	2días	04/25/2020	04/26/2020
28	Definició de l'estat i les recompenses	2días	04/27/2020	04/28/2020
29	Implementació de l'agent	3días	04/29/2020	05/01/2020
30	Determinació de paràmetres que donen un millor resultat	3días	05/02/2020	05/04/2020
31	**Comparativa i anàlisi de resultats**	14días	05/05/2020	05/18/2020
32	Comparativa dels agents generats al projecte	4días	05/05/2020	05/08/2020
33	Adaptació d'altres agents per a la comparativa	3días	05/09/2020	05/11/2020
34	Comparativa amb altres agents	4días	05/12/2020	05/15/2020
35	Extreure conclusions de les estratègies dels agents	3días	05/16/2020	05/18/2020
36	Redacció de la memòria	16días	05/19/2020	06/03/2020
37	Elaboració de la presentació	8días	06/04/2020	06/11/2020

Figura 1.1 Diagrama de Gantt de la planificació inicial

Fita 1: Implementació de l'entorn OpenAI gym i la UI (2 setmanes)

Setmana 1:

- Preparació de l'entorn que permeti la realització del projecte.
- Estudi dels altres projectes ja existents, per tal d'arribar a un enteniment complet de què fan i com ho fan.
- Adaptació i/o millora d'una de les implementacions del joc disponibles, o en cas de no ser possible realització una de nova.

Setmana 2:

- Creació d'una UI que utilitzi la implementació realitzada per jugar de forma visual al joc.
- Adaptacions per poder registrar la implementació del joc com a entorn de gym.

Fita 2: Generació d'agents (5 setmanes)

Setmanes 3 i 4:

- Agent basat en l'algoritme Q-Learning.
    - Investigació i aprenentatge de l'algoritme
    - Definició de l'estat i les recompenses
    - Implementació de l'agent
    - Determinació de paràmetres que donen un millor resultat
- Agent basat en l'algoritme Deep Q-Learning
    - Investigació i aprenentatge de l'algoritme
    - Definició de l'estat i les recompenses
    - Implementació de l'agent
    - Aplicació de millores (DDQN, Double DQN, ..)
    - Determinació de paràmetres que donen un millor resultat

Setmanes 5, 6 i 7:

- Agent basat en Policy Gradients
    - Investigació i aprenentatge de l'algoritme
    - Definició de l'estat i les recompenses
    - Implementació de l'agent
    - Aplicació de millores (AAC, PPO)
    - Determinació de paràmetres que donen un millor resultat
- Agent basat en l'arbre de cerca de Monte-Carlo
    - Investigació i aprenentatge de l'algoritme
    - Definició de l'estat i les recompenses
    - Implementació de l'agent
    - Determinació de paràmetres que donen un millor resultat

Fita 3: Comparativa i anàlisi de resultats (2 setmanes)

Setmana 8:

- Comparativa del rendiment dels agents generats amb diferents tècniques entre ells. Fer que juguin partides entre ells i anàlisi de qui té un percentatge més gran de victòries.
- Adaptació dels agents creats per altres projectes per tal que puguin ser comparats amb els generats per al projecte.

Setmana 9:

- Comparativa del rendiment dels agents generats amb els ja existents creats per altres projectes prèviament.
- Extracció d'informació rellevant de l'estratègia utilitzada pels agents que donen un millor rendiment, per obtenir conclusions.

1.5 Breu sumari de productes obtinguts

A continuació, ens disposem a detallar els productes que s'han obtingut al llarg del projecte:

- Implementació del sistema que ens ha de permetre realitzar les partides de Sushi Go, ja sigui tant de forma automàtica per permetre l'entrenament dels agents, com partides entre un usuari i els agents que anem creant.
- Agents creats, alguns creats de forma manual, i altres basats en Q-Learning, en Deep Q-Learning, i en mètodes de Monte Carlo.
- Informe amb les conclusions obtingudes de comparar els diferents agents entre si.
- Informació extreta de les estratègies utilitzades pels agents amb un millor rendiment.
- Aquesta memòria del projecte.
- Presentació virtual del projecte, a on s'exposen els seus aspectes principals per tal d'introduir el seu contingut al tribunal.

1.6 Breu descripció dels altres capítols de la memòria

El procés de desenvolupament del projecte es descriu al llarg dels 7 capítols que formen aquesta memòria.

Després d'aquest capítol que serveix com a introducció a la resta del projecte, el segon capítol proveeix al lector amb una introducció a l'aprenentatge per reforç. Per fer-ho, es comença situant l'àrea dins del món de la intel·ligència artificial i de l'aprenentatge computacional. També s'introdueixen els conceptes generals que serveixen com a base de tot projecte d'aquesta àrea, i alguns dels algoritmes que s'utilitzaran a la resta del projecte per generar els agents de Sushi Go. Finalment, es presenta també l'entorn OpenAI Gym que permet estandarditzar els entorns utilitzats en projectes d'aprenentatge per reforç.

Al tercer capítol es fa una descripció completa de les regles de Sushi Go. En aquest capítol també es descriuen les diferents versions existents del joc, i finalment es fa un

estudi dels projectes d'aprenentatge per reforç que han utilitzat prèviament Sushi Go com a entorn per als agents creats.

El quart capítol descriu la metodologia seguida per, en primera instància, adaptar una de les implementacions trobades a un projecte preexistent per crear un entorn i adaptar-ho per ser registrat com a entorn Gym. Posteriorment, es descriu el procés d'implementació dels algoritmes que han de permetre la construcció dels agents.

Al cinquè capítol, es realitzen un gran nombre de partides entre els agents generats per tal de comparar el rendiment que proporciona cadascun d'ells, i determinar quin és l'algoritme, i en quines condicions, que ha creat l'agent més competitiu. També s'analitza l'estratègia utilitzada pels agents amb millor resultats per definir l'estratègia que dóna millor resultat en el joc.

Als darrers capítols es redacten les conclusions obtingudes durant la realització del projecte, i la bibliografia utilitzada per la seva creació.

2. Introducció a l'aprenentatge per reforç

El paradigma de l'aprenentatge per reforç és una àrea de l'aprenentatge computacional, que és una branca del camp de la intel·ligència artificial.

2.1 Conceptes previs

Intel·ligència Artificial

La intel·ligència artificial (IA) és l'estudi i el disseny de sistemes que mostren un comportament que es pot considerar intel·ligent. Un agent es considera intel·ligent quan és capaç d'interactuar amb l'entorn, i ho fa de tal forma que els seus actes són apropiats a les circumstàncies i li permeten aconseguir una determinada fita. Altres aspectes que denoten un comportament intel·ligent són la capacitat d'aprendre de l'experiència, i/o l'adaptabilitat a canvis en l'entorn i les fites. Es poden determinar dos objectius principals de la IA: un objectiu científic de comprensió dels aspectes que determinen que un sistema és intel·ligent, i un objectiu d'enginyeria consistent en la creació de mètodes que permetin la creació d'elements intel·ligents [5]

Aprenentatge computacional

L'aprenentatge computacional (AC) [6] és la branca dins de l'àrea de la IA centrat en l'estudi de mètodes i tècniques que permetin generar sistemes que milloren el seu rendiment a partir de les observacions prèvies realitzades a l'entorn. Aquestes tècniques són necessàries quan el dissenyador de l'agent intel·ligent no pot anticipar quin és el comportament òptim per maximitzar el rendiment a l'entorn al qual es troba l'agent. Els motius pels quals el dissenyador pot no disposar d'aquest coneixement són:

- Quan no es poden anticipar totes les situacions a les quals es pot trobar l'agent a l'entorn. Un robot aspirador ha d'aprendre el mapa de la casa que està netejant. El dissenyador del robot no pot conèixer tots els mapes de totes les cases a on s'utilitzaran els robots.
- Quan no es poden anticipar tots els canvis que es poden produir en l'entorn. Un agent que inverteix al mercat borsari s'ha d'adaptar a les fluctuacions de l'entorn. Aquestes fluctuacions són impredictibles pel dissenyador.
- Quan el dissenyador pot realitzar la tasca, però no pot explicar com realitzar la tasca. El dissenyador pot reconèixer les cares dels seus coneguts, però és molt difícil explicar com distingir les cares.

Els sistemes d'AC, per poder aprendre, precisen una sèrie d'observacions sobre l'entorn. Sobre aquestes observacions, l'entorn ens pot proporcionar un feedback, o no. Aquest feedback és un coneixement sobre l'observació provinent d'un element fiable,

un "supervisor". Per exemple, per una observació que sigui una fotografia, el feedback corresponent podria ser "a la fotografia hi apareix un gos". O per exemple, un agent que juga al tres en ratlla, per a una determinada jugada rebrà el feedback de si ha guanyat la partida, ha perdut, o si la partida encara no ha finalitzat. Existeixen tres tipus de feedback que determinen les tres grans àrees de l'AC:

- A l'aprenentatge no-supervisat, es disposa d'observacions de l'entorn, però no de cap feedback. L'objectiu de l'aprenentatge no-supervisat és trobar patrons a les observacions, per poder determinar relacions entre elles i agrupar-les. Un agent que estigui aprenent a conduir podria distingir entre dies de molt tràfic i dies de poc tràfic sense necessitar que un supervisor indiqui que hi ha dies amb més tràfic i dies amb menys.
- A l'aprenentatge supervisat, es disposa de partida d'un conjunt d'observacions amb el seu feedback corresponent. Si es consideren les observacions les dades d'entrada del sistema, i el feedback la sortida, l'agent ha de ser capaç d'estimar una funció que mapeji els inputs amb els outputs corresponents. A partir d'aquesta funció, l'agent podrà generar l'output corresponent a qualsevol input sense necessitat del feedback. Per exemple, si es proporciona a un agent una sèrie d'observacions que són fotos de gats o fotos de gossos, i per cada fotografia es proporciona la informació de quin animal hi apareix, l'agent ha de ser capaç de determinar una funció que a partir d'una foto, categoritza l'animal que hi apareix com a un gat o un gos, sense necessitat de feedback.
- A l'aprenentatge per reforç, inicialment no es disposa ni d'observacions, ni de feedback. Les observacions i el feedback s'obtenen al mateix temps que l'agent aprèn de l'entorn. Per poder obtenir observacions i feedbacks dels quals aprendre, l'agent ha d'interactuar amb l'entorn. El feedback que proporciona l'entorn representa una recompensa o penalització per l'acció realitzada. A partir dels feedbacks obtinguts, l'agent ha de ser capaç d'aprendre a prendre decisions que li permetin maximitzar les recompenses obtingudes.

2.2 Conceptes generals

L'aprenentatge per reforç consisteix a aprendre quines accions s'han de realitzar en una determinada situació amb l'objectiu de maximitzar una recompensa obtinguda. Per tal d'arribar a aquest objectiu, les investigacions en aquesta àrea proveeixen d'un conjunt de tècniques i mètodes que permeten fer un càlcul del valor que s'espera obtenir en realitzar una determinada acció a una determinada situació. D'aquesta manera, un agent que obtingui aquesta informació, serà capaç de triar, entre totes les accions possibles, aquella que tingui un valor futur esperat més gran.

La base del funcionament dels agents creats amb tècniques d'aprenentatge per reforç és la interacció de l'agent amb l'entorn. Fruit de cadascuna d'aquestes interaccions els agents obtindran o bé una recompensa, o bé un càstig. Les interaccions que produeixin una recompensa seran prioritzades per l'agent, mentre que les que resulten en un càstig seran penalitzades.

Elements de l’aprenentatge per reforç

Agent: Un agent és un sistema que realitza accions. Un agent és, per exemple, un jugador d’un joc de cartes, un jugador de tres en ratlla, un robot aspirador que neteja el terra d’una habitació, o un cotxe automàtic.

Entorn: L'entorn és el món amb el qual interactuen els agents,i a on realitzen les seves accions. Els entorns reben les accions dels agents que hi interactuen, i proporcionen la recompensa o càstig corresponents.

Un aspecte clau dels entorns és si són episòdics o continus. Un entorn episòdic és aquell a on es pot determinar un instant inicial i un instant final de la tasca que realitza un agent a l'entorn. Per tant, el nombre d'accions que es realitzen en un episodi, i per tant el nombre de recompenses obtingudes, és un nombre finit. Un joc de cartes és un entorn episòdic: la tasca a realitzar comença a l'inici de la partida, i finalitza quan la partida acaba. Es poden continuar jugant més partides, però la puntuació a maximitzar ve determinada per l'episodi. Quan comença una nova partida, la puntuació torna a ser zero.

Un entorn continu és aquell a on l'instant inicial és conegut, però el procés podria durar potencialment fins a l'infinit. Per exemple, un agent de comerç borsari automàtic treballa sobre un entorn continu: es vol obtenir la recompensa màxima acumulada al llarg del temps, de forma indefinida, sense determinar en quin punt es vol finalitzar el procés.

Espai d’accions: L’espai d’accions defineix el conjunt d’accions que un agent pot realitzar a l’entorn en una determinada situació. Un cotxe automàtic pot accelerar, reduir marxa, girar, o realitzar qualsevol altra de les accions que el cotxe pugui realitzar. Un jugador de tres en ratlla té tantes accions possibles com caselles lliuren es troben al tauler.

Espai d'estats: L'estat és la percepció que té l'agent de la situació concreta en la qual es troba a l'entorn en un determinat instant. L'espai d'estats, per la seva banda, és el conjunt total d'estats que pot percebre l'agent. La dimensió de l'espai d'estats dependrà de la quantitat d'informació que l'agent obtingui de l'entorn. Per exemple, un robot aspirador pot tenir un únic sensor que detecta si hi ha una paret a menys d'un metre de distància. Aquest seria un estat amb només dos valors possibles, hi ha paret o no hi ha paret. Òbviament, aquesta informació és molt limitada. D'altra banda, si el robot disposa de 20 sensors al llarg de la seva circumferència que són capaços d'obtenir la distància a la qual es troben a la paret, la percepció de l'agent és molt més completa, i per tant l'espai d'estats és molt més gran. En aquest cas, la informació que té l'agent sobre l'entorn també és molt més gran.

En entorns episòdics, l’espai d’estats conté sempre un estat especial anomenat estat final, que determina que l’episodi ha finalitzat.

Policy: La policy és l'estratègia que segueix un agent per determinar quina acció triar en un determinat estat. És una funció que, per cada possible estat de l'entorn a l'espai d'estats, retorna quina acció ha de realitzar l'agent. Les policies també poden ser estocàstiques, i en comptes de determinar una única acció, retornaran una distribució de la probabilitat de realitzar cadascuna de les accions possibles dins de l'espai d'accions.

Recompensa: La recompensa és el valor numèric obtingut de l'entorn després que l'agent realitzi una determinada acció en un determinat estat. L'objectiu dels agents és maximitzar la suma de les recompenses a llarg termini. A vegades, el valor de la recompensa que s'obté d'una acció és fix, mentre que en altres entorns el valor pot ser determinat per un procés estocàstic en funció de l'acció i de l'estat.

Funció valor: El valor d'un estat determina l'estimació de la suma de recompenses que es preveuen obtenir de l'entorn quan es troba en aquell estat i quan se segueix una determinada policy. Quan un agent decideix quina acció triar, no només influeix en la recompensa que obtindrà de forma immediata, sinó que aquesta acció també determina l'estat en el qual quedarà l'entorn després de realitzar l'acció. Per aquest motiu, el benefici de realitzar una acció no ve només determinat per la recompensa immediata, sinó que també pel valor de l'estat al qual s'arriba després de l'acció. Per posar un exemple, en una partida d'escacs, un jugador té l'opció d'eliminar un peó rival exposant el seu rei, o pot enrocar el rei i protegir-lo. Si elimina el peó, obtindrà una recompensa directa, per l'estat en el qual es trobarà tindrà un valor molt baix, perquè el rei correrà perill i les seves opcions de victòria final es reduiran. En canvi, l'opció de l'enrocament no proporciona un benefici directe perquè no elimina cap fitxa rival, però en canvi el següent estat de la partida tindrà un valor superior perquè les probabilitats de victòria seran més grans.

La funció valor també pot aplicar a un parell estat-acció: en aquest cas, la funció valor representa la recompensa esperada d'aplicar l'acció en l'estat, més totes les recompenses esperades fins al final de l'episodi.

Experiment: En un procés d'aprenentatge per reforç, es consideren experiments cadascun dels episodis que realitza un agent a l'entorn durant el procés d'entrenament. Els experiments serveixen a l'agent per adaptar la seva policy per tal de maximitzar el valor total obtingut. El procés d'aprenentatge per reforç en entorns episòdics es basa en la repetició d'un nombre determinat d'experiments.

Experiència: Una experiència és el coneixement obtingut de l'entorn a cadascun dels passos realitzats en un experiment. Una experiència està formada per l'estat en el qual es troba l'entorn abans del pas realitzat, per l'acció triada, per la recompensa obtinguda, i pel nou estat en el qual es troba l'entorn després de realitzar l'acció.

Formalització dels problemes d'aprenentatge per reforç: els processos de decisió finits de Markov

Els processos de decisió finits de Markov (MDP) són una formalització clàssica de la realització d'una seqüència d'accions, que determinen no només la recompensa obtinguda, sinó que també els estats subseqüents. Els MDP poden ser utilitzats per formalitzar qualsevol problema que s'hagi de resoldre mitjançant aprenentatge per reforçament.

Per començar amb l'aplicació dels MDP als sistemes d'aprenentatge per reforçament, es defineix la interfície agent-entorn amb els elements que hem definit prèviament de la següent manera:

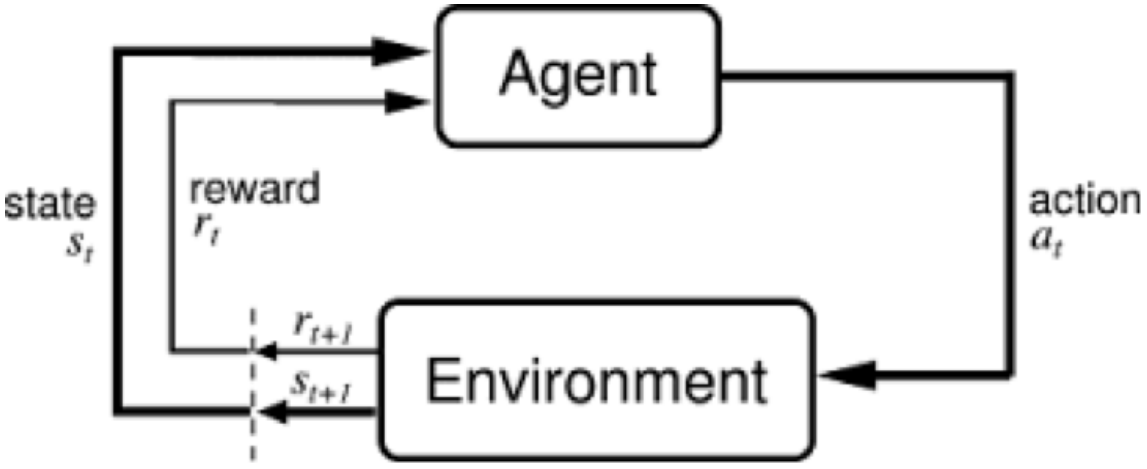


Figura 2.1 Interacció agent-entorn en processos de decisió de Markov. [7] Pàgina 38

La interfície defineix un agent, que en un instant *t* es troba en un estat s_t, i realitza una acció a_t. L'entorn, quan l'agent realitza aquesta acció, reacciona canviant a l'estat s_{t+1}, i proporciona una recompensa r_{t+1}. L'agent rep aquest nou estat i recompensa, i utilitzarà aquestes dades per determinar la següent acció a realitzar. Per tant, si es defineix l'estat inicial S_0, es pot determinar que la interacció en el temps entre l'agent i l'entorn genera una seqüència o trajectòria de la forma:

$$S0,\ A0,\ R1,\ S1,\ A1,\ R2,\ S2,\ A2,\ R3,\ldots \qquad \textit{(2.1) [7]}$$

Als MDP, els conjunts d'estats possibles, d'accions i de recompenses tenen un nombre finit d'elements. Els MDP també defineixen que les variables aleatòries R_t i S_t segueixen una distribució de probabilitat que depèn exclusivament de l'estat i l'acció anteriors.

$$p(s',\ r \mid s,\ a) = Pr\{S_t = s',\ R_t = r \mid S_{t-1} = s,\ A_{t-1} = a\} \qquad \textit{(2.2) [7]}$$

A partir d'aquesta distribució, es pot computar qualsevol altra probabilitat que sigui necessari calcular sobre l'entorn. Per exemple, es pot obtenir la fórmula per obtenir la probabilitat d'arribar a un estat *s'* partint des d'un estat *s* i realitzant una acció *a*.

$$p(s' \mid s, a) = Pr\{ S_t = s' \mid S_{t-1} = s, A_{t-1} = a \} = \sum_{r \in R} p(s', r \mid s, a) \qquad \textit{(2.3) [7]}$$

La fórmula també permet calcular la recompensa esperada a partir de realitzar una acció *a* en un estat *s*.

$$r(s, a) = E[R_t \mid S_{t-1} = s, A_{t-1} = a \} = \sum_{r \in R} r \sum_{s' \varepsilon S} p(s', r \mid s, a) \qquad \textit{(2.4) [7]}$$

En resum, els MDP defineixen la interfície de la interacció que es produeix entre els agents i els entorns a tot sistema d'aprenentatge per reforç, i que hi ha d'haver una distribució de probabilitat que, a partir de cada estat *s* i cada acció *a*, determini la probabilitat d'obtenir una recompensa *r* i arribar a un estat *s'*.

El guany total esperat

L'objectiu d'un sistema d'aprenentatge per reforç és maximitzar la suma de totes les recompenses obtingudes a partir de cert moment *t*. Per tant, per a un episodi que comença a l'instant *t* i finalitza a *T*, el valor a maximitzar és el guany esperat G_t.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \qquad \textit{(2.5) [7]}$$

Aquesta fórmula només és vàlida per a entorns episòdics, a on existeix un instant T. En el cas dels entorns continus, com que T és infinit, el valor de G_t segurament també ho és. Per tant, s'ha d'afegir a la fòrmula el concepte de descompte, i definir la fórmula de guany descomptat esperat com:

$$G_t = R_{t+1} + \gamma * R_{t+2} + \gamma^2 * R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad \textit{(2.6) [7]}$$

a on γ, el factor de descompte, té un valor entre 0 i 1. Amb aquest valor s'aconsegueix donar més importància a les recompenses més immediates, i proporcionalment cada cop menys a les recompenses que s'esperen més lluny en el temps. D'aquesta manera s'aconsegueix que el guany esperat en entorns continus ja no és infinit. Als entorns episòdics, utilitzant la fórmula *(2.6)* com a retorn a maximitzar també s'aconsegueix donar més importància a les recompenses més properes en el temps i que es poden considerar més segures, i menys a les recompenses que estan lluny en el temps i per tant són més arriscades o menys probables d'aconseguir.

La funció valor

S'havia definit la funció valor d'un estat com l'estimació de la suma de recompenses que es preveuen obtenir de l'entorn a partir de l'estat seguint una policy determinada. A partir de la fórmula del guany descomptat esperat *(2.6)*, es pot definir la fórmula de la funció valor d'un estat *s* quan se segueix una policy π de la forma:

$$v_\pi(s) \;=\; E_\pi[G_t \mid S_t \;=\; s] \;=\; E_\pi[\sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1} \mid S_t \;=\; s\,] \qquad \textit{(2.7) [7]}$$

és a dir, el valor esperat d'un estat s seguint una policy π és l'esperança del guany descomptat partir d'aquell estat, quan es trien les accions que determina la policy. La fórmula per a la funció valor d'un parell estat-acció és:

$$q_\pi(s,\; a) \;=\; E_\pi[G_t \mid S_t = s,\; A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1} \mid S_t = s,\; A_t = a] \qquad \textit{(2.8) [7]}$$

Es pot veure com, utilitzant la fórmula *(2.8)*, es poden estimar els valors dels estats a partir de la repetició d'experiències: mantenint la policy π i guardant, per cada estat *s*, la mitjana dels guanys obtinguts a partir de l'estat, aquestes mitjanes del guany acaben convergint en el valor dels estats per a la policy π. Aquesta fórmula suposa la base per als algoritmes de tipus Monte Carlo.

A partir de la fórmula del valor de l'estat *(2.7)*, se'n pot derivar una de nova de gran importància, la fórmula de Bellman, que permet el càlcul del valor d'un estat de forma recursiva a partir dels valors dels possibles estats successors:

$$v_\pi(s) \;= \sum_a \pi(a|s) \sum_{s',\, r} p(s',\; r \mid s,\; a)\; [r \;+\; \gamma v_\pi(s')] \qquad \textit{(2.9) [7]}$$

La fórmula de Bellman per a parells estat-acció seria la següent:

$$q_\pi(s,\; a) \;= \sum_{s',\, r} p(s',\; r \mid s,\; a)\; [r \;+\; \gamma v_\pi(s')] \qquad \textit{(2.10) [7]}$$

Si les fórmules *(2.7)* i *(2.8)* suposen la base dels algoritmes tipus Monte Carlo, les fórmules *(2.9)* i *(2.10)* ho són per als algoritmes d'aprenentatge per diferència temporal (dels que es tracten a aquest projecte, en formen part el de Q-Learning i el de Deep Q-Learning).

Finalment, es defineix la policy òptima com aquella que, per a qualsevol estat *s*, proporciona un valor d'estat màxim entre totes les policies possibles. Per tant, trobar la policy òptima és l'objectiu de qualsevol sistema d'aprenentatge per reforç. A partir de la fórmula de Bellman, definim la fórmula de la policy òptima com:

$$v_*(s) \;= max_a \sum_{s',\, r} p(s',\; r \mid s,\; a)\; [r \;+\; \gamma v_\pi(s')] \qquad \textit{(2.11) [7]}$$

La relació entre l'exploració i l'explotació

A l'apartat anterior s'ha determinat que la funció valor, tant d'un estat com d'un parell estat-acció, es pot estimar a partir de la repetició d'experiments. No obstant això, l'objectiu dels mètodes d'aprenentatge per reforç no és calcular aquests valors, sinó obtenir la policy òptima que maximitza la funció valor per cada estat. Per aquest motiu, partint d'una policy amb un comportament que inicialment serà aleatori, s'haurà de seguir una estratègia d'actualització per la policy que faci que el seu comportament s'apropi amb el pas dels experiments al de la policy òptima.

Quan comença l'entrenament d'un agent d'aprenentatge per reforç, s'inicialitza amb una policy que no té cap coneixement previ, i que per tant triarà les accions a realitzar de forma aleatòria. Les experiències inicials generades amb l'agent a l'entorn han de servir per realitzar les primeres actualitzacions de la policy de l'agent a partir de les recompenses obtingudes. Una vegada s'han realitzat les primeres actualitzacions de la policy, l'agent disposarà d'un coneixement inicial de quins parells estat-acció proporcionen un valor més gran i quins un de menor. Aquest coneixement s'ha d'aprofitar per reforçar l'aprenentatge de les experiències que han proporcionat un major benefici, ja que tenen més probabilitats de ser finalment utilitzades per la policy. D'altra banda, però, no es poden deixar de banda definitivament la resta d'accions possibles. El fet que un parell estat-acció no doni un bon resultat en una experiència inicial, no vol dir que en experiències posteriors no pugui obtenir un valor mitjà més gran, i d'aquesta manera superar una altra acció que semblava més profitosa. Per aquest motiu, tot i que s'ha de reforçar la selecció d'accions que fins al moment donen el valor màxim, s'ha de donar oportunitat a la resta d'accions per actualitzar el seu valor estimat.

Per tant, es defineixen dos mètodes de selecció de l'acció durant el procés d'entrenament d'un agent:

- Quan es tria l'acció que la policy actual estima com la que té un valor més gran, s'està fent una tasca d'**explotació**. Les tasques d'explotació reforcen el coneixement sobre aquelles accions que han funcionat millor en el passat en l'estat actual de l'entorn, i que per tant tenen més probabilitats de ser l'acció òptima per a l'estat.
- Quan es tria una acció que actualment no té el valor més gran per la policy, s'està realitzant una tasca d'**exploració**. Aquestes tasques permeten canviar el funcionament de la policy i, possiblement, determinar que el valor de l'acció explorada supera aquell de l'acció que es considerava anteriorment.

Durant el procés d'entrenament és important combinar de forma correcta les tasques d'exploració i d'explotació. Si només es realitzen tasques d'explotació, el sistema mai no trobarà una acció alternativa a la que inicialment semblava òptima. En canvi, si només es realitzen tasques d'exploració, els experiments es reparteixen equitativament entre totes les accions, les accions òptimes no estaran reforçades amb suficients experiències. Per tant, un bon balanç entre l'exploració i l'explotació és bàsic per a la convergència de la policy cap a la policy òptima.

Inicialment, el sistema no disposa de cap informació, i per tant a l'inici de l'entrenament gairebé totes les tasques que s'han de realitzar són d'exploració. A mesura que la policy comença a tenir coneixement del valor dels parells estat-acció, s'ha de començar a reduir progressivament el ràtio de tasques d'exploració, i a poc a poc augmentar el de tasques d'explotació. A mesura que el sistema convergeix cap a la policy òptima, les tasques d'exploració s'han de reduir al mínim, i incrementar les d'explotació per tal de reforçar les accions òptimes amb el màxim nombre d'experiències.

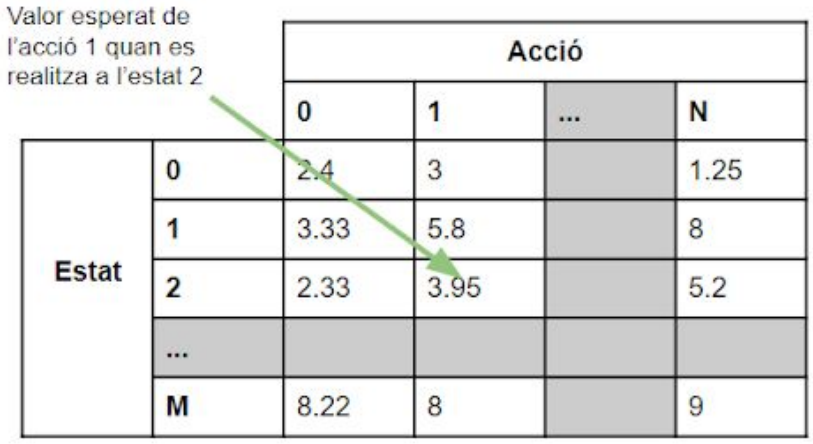
Per tal de gestionar el ràtio d'exploració/explotació que s'aplica en un determinat moment del procés d'entrenament (a partir d'aquest moment es referència aquest ràtio com a èpsilon ε), els algoritmes d'aprenentatge per reforç l'inicialitzen a un valor màxim (generalment té valor 1, és a dir, totes les tasques són d'exploració). També es

defineix una funció d'actualització de l'èpsilon que va reduint el seu valor amb el nombre d'experiències realitzades, fins al punt en el qual l'èpsilon arriba a un valor mínim a on la gran majoria de les tasques són d'explotació.

2.3 Algoritmes

2.3.1 Q-Learning

Els algoritmes de Q-Learning basen el seu funcionament en la construcció d'una taula, anomenada q-table, a on disposen de tots els valors esperats de cadascun dels parells estat-acció de l'entorn. Per tant, una de les dimensions de la taula és la mida de l'espai d'accions, i l'altra de les dimensions és la mida de l'espai d'estats que s'està utilitzant.



		Acció			
		0	1	...	N
Estat	0	2.4	3		1.25
	1	3.33	5.8		8
	2	2.33	3.95		5.2
	...				
	M	8.22	8		9

Figura 2.2 Estructura de la q-table

Per poder ser assignats a una de les files de la q-table, els estats han de ser representats per un identificador sencer únic per a cada estat de l'espai, i que es trobi entre 0 i M. El mateix aplica per a les accions de l'espai d'accions: a cadascuna se li ha d'assignar un valor entre 0 i N, que defineix la columna a on es trobaran els valors de realitzar l'acció des de cadascun dels estats.

La q-table representa la policy de l'agent que s'està construint. Per cada estat, determinar l'acció que tria l'agent consisteix només a cercar, d'entre els valors a la fila de la q-table pertanyent a l'estat, aquella columna amb un valor màxim. L'acció que pertany a la columna amb valor màxim, serà l'acció que l'agent ha de prendre en aquell estat, perquè és l'acció per la qual s'ha calculat un valor màxim en aquell estat.

Per aconseguir que l'agent apliqui la policy òptima, el procés d'entrenament de l'algoritme Q-Learning ha d'aconseguir que els valors de la q-table convergeixin cap als valors reals de la policy òptima per a tots els parells state-acció.

Mitjançant tècniques de programació dinàmica, és possible generar un algoritme que, alternant tasques d'avaluació de la policy amb tasques de millora de la policy, aconsegueix convergir en la taula que representa la policy òptima per a l'entorn. Aquests algoritmes requereixen recórrer diverses vegades el conjunt complet d'estats, i en la majoria de casos el cost computacional és prohibitiu.

Per tant, és necessària una fórmula que, en comptes de fer recorreguts sobre el conjunt d'estats, avaluï i millori la policy continguda a la q-table actualitzant els valors de les cel·les durant el procés d'entrenament del sistema, utilitzant la informació que ens proporciona l'entorn en forma de recompenses i nous estats obtinguts de realitzar una acció en un determinat estat. Partint de la fórmula de Bellman *(2.10)* es deriva el següent mètode d'actualització dels valors dels parells estat-acció.

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_t + \gamma * max_a\, Q(s_{t+1}, a) - Q(s_t, a_t)) \qquad \textit{(2.12) [7]}$$

a on:

- $Q(s_t,\ a_t)$ és l'antic valor de la cel·la a la q-table per a l'estat *s* i l'acció *a*
- $max_a\, Q(s_{t+1},\ a)$ és el valor de l'estat nou al qual s'ha arribat amb l'acció realitzada
- $r_t + \gamma * max_a\, Q(s_{t+1},\ a)$ és el nou valor estimat per l'acció que s'ha realitzat, ja que és la suma de la recompensa obtinguda per l'acció, més el valor del nou estat al qual s'ha arribat
- $r_t + \gamma * max_a\, Q(s_{t+1},\ a) - Q(s_t,\ a_t)$ és la diferència temporal entre el valor que s'ha obtingut, i el que s'esperava obtenir
- α és el ràtio d'aprenentatge. Aquest ràtio determina la velocitat a la qual la diferència temporal obtinguda amb cadascun dels experiments s'incorpora al valor antic de la cel·la.

A partir d'aquesta fórmula, es defineix l'algoritme de q-learning de la següent forma:

```
q_table = Inicialitzar_valors_a_q_table()
epsilon = EPSILON_MAXIMA

mentre (nombre_experiments_realizats < total_experiments_a_realitzar)
{

  s = env.obtenir_estat_inicial()
  mentre (no env.episodi_ha_finalitzat())
  {
    exploració = determinar_exploracio_o_explotacio(epsilon)

    si (exploració)
      a = env.obtenir_acció_aleatoria()
    sino // explotació
      a = argmax_a (q_table[s])

    r, s' = env.realitzar_accio(a)

    si (env.episodi_ha_finalitzat())

   // Si l'episodi ha finalitzat, el següent estat és l'estat final, i
   // per tant el seu valor és 0, no esperem obtenir més recompenses

      q_table[s][a] += ràtio_aprenentatge * (reward - q_table[s][a])
```

```
    sino

      q_table[s][a] += ràtio_aprenentatge * (reward + descompte *
                       max(q_table(s')) - q_table[s][a])

    epsilon = reduïr_valor(epsilon)

    s = s' // Actualitzem l'estat amb el nou estat per a la següent
           // iteració
  }
}
```

Aplicant aquest algoritme, si es realitzen un nombre d'experiments suficients, la q-table acabarà convergint i els valors de les cel·les es mantindrà estable encara que es realitzin nous experiments. En aquest punt, s'haurà obtingut una policy màxima, que o bé pot ser la policy òptima, o bé pot haver resultat en un màxim local. Una opció o l'altra depenen en gran manera de la configuració del ràtio d'aprenentatge, del valor de descompte, i també de la gestió del valor d'èpsilon al llarg de l'entrenament. Per tant, per tal de trobar la combinació d'aquests paràmetres que produeixi la policy més propera a l'òptima, és necessari generar diferents agents amb diferents combinacions de paràmetres i comparar els seus resultats obtinguts per determinar quin obté un màxim valor de les seves accions.

2.3.1.1 Expected SARSA

L'algoritme Expected Sarsa té un funcionament semblant al de Q-Learning, amb la diferència de la forma de calcular el valor esperat per al nou estat. En comptes de considerar el valor del nou estat com el màxim de tots els valors que es poden aconseguir realitzant les diferents accions, es calcula com la mitjana ponderada del valor de totes les accions possibles des del següent estat, en funció de la probabilitat de prendre cada acció. La següent fórmula determina el valor actualitzat d'una acció *a* realitzada en un estat *s* en funció de l'experiència realitzada:

$$Q^{new}(s_t,) \leftarrow Q(s_t, a_t) + \alpha * (r_t + \gamma * \sum_a \pi(a \mid s_{t+1}) * Q(s_{t+1}, a) - Q(s_t, a_t)) \qquad \textit{(2.13) [7]}$$

a on $\pi(a \mid s_{t+1})$ és la probabilitat de triar l'acció *a* quan s'utilitza la policy estocàstica π a l'estat s_{t+1}.

L'algoritme Q-Learning té molts avantatges: és simple, fàcil de comprendre i d'implementar, i si es troben els paràmetres adequats i amb la quantitat d'entrenament suficient, es pot obtenir la policy òptima. En contra, té una limitació molt important: la mida màxima de la q-table. Les dimensions de la q-table són determinades directament per la mida de l'espai d'estats, i de l'espai d'accions. L'espai d'accions habitualment té una mida limitada, però l'espai d'estats pot arribar a contenir una quantitat immensa d'estats, depenent de la seva definició. L'emmagatzemament en memòria i els temps d'accés de la q-table limiten la mida de la q-table, i per tant limiten també la mida de l'espai d'estats que es pot utilitzar.

Per comprendre millor la limitació que suposa la dimensió de la q-table, s'utilitzarà un exemple: un robot aspirador que tingui 20 sensors a la seva circumferència, i que cadascun calcula la distància a l'obstacle més proper. Cada sensor arriba a una distància màxima de 10 metres, i tenen una sensibilitat d'un centímetre. Per tant, cada sensor pot tenir 1000 valors possibles. Com que l'estat el formen la combinació de tots 20 sensors, el conjunt de l'espai d'estats el formarien 1000^{20} estats diferents. Encara que l'espai d'accions el formin solament dues accions, una taula de $2*1000^{20}$ cel·les no és gestionable de cap manera.

2.3.2 Deep Q-Learning

L'algoritme Deep Q-Learning soluciona la limitació de l'algoritme Q-Learning en el que respecta a l'espai d'estats. Aquest algoritme, en lloc de guardar el valor de cadascun dels parells estat-acció, és capaç d'aproximar aquests valors. Per fer-ho, els algoritmes de Deep Q-Learning canvien la q-table per una xarxa neuronal: en comptes de tenir una taula que ens diu el valor exacte d'un parell estat-acció, tindrem una xarxa neuronal que ens estima aquest valor.

La xarxa neuronal que estimarà els valors de les diferents accions tindrà com a entrada un estat de l'entorn. Aquest estat, en comptes d'estar representat per un identificador sencer únic com era necessari a l'algoritme de Q-Learning, estarà representat per un vector d'atributs. Per tant, a l'exemple del robot aspirador, els valor d'entrada de la xarxa neuronal serà el vector que conté la mesura obtinguda per cadascun dels vint sensors dels quals disposa. La sortida de la xarxa tindrà la mida del nombre d'accions possibles: per cada acció possible, retornarà l'estimació del valor de l'acció a partir de l'estat indicat a l'entrada de la xarxa.

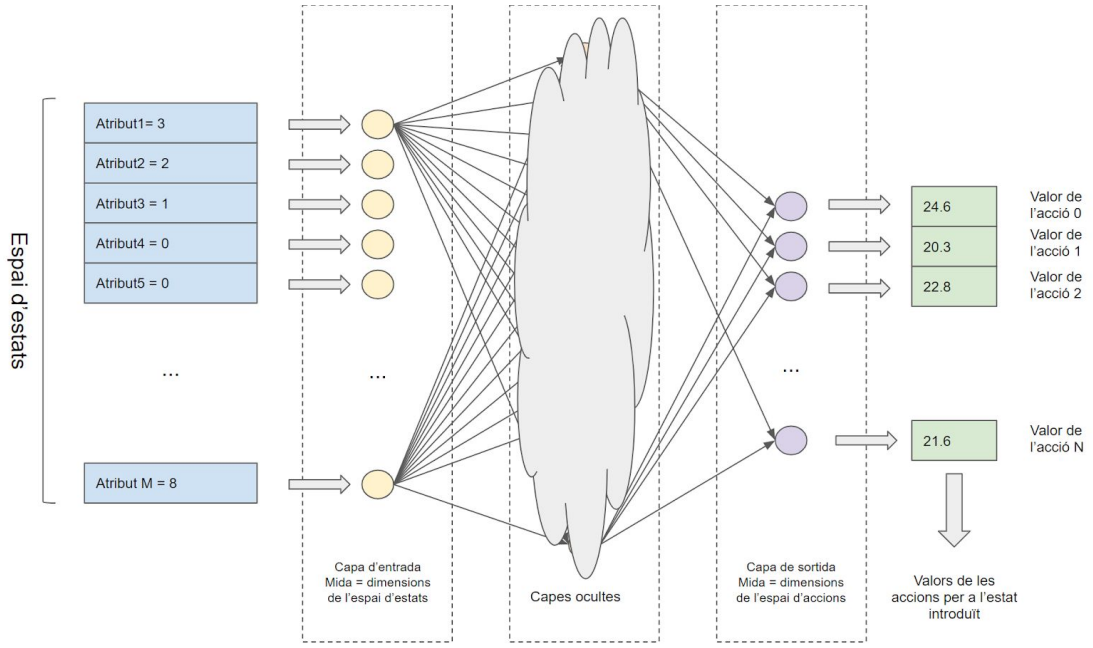


Figura 2.3 Entrada i sortida de la xarxa neuronal en algoritme Deep Q-Network

L'algoritme d'entrenament amb Deep Q-Learning, en principi, és molt similar al de Q-Learning: es manté un valor d'èpsilon que gestiona la divisió de tasques exploratòries o d'explotació, es realitza l'acció triada a l'entorn, s'obté una recompensa i un nou estat, i amb aquestes dades s'actualitza l'agent mitjançant la fórmula de Bellman *(2.10)*. En aquest cas, però, el funcionament intrínsec de la xarxa neuronal permet simplificar la fórmula de la forma:

$$Q^{new}(s_t,\ a_t) \leftarrow r_t\ +\ \gamma\ *\ max_a\ Q(s_{t+1},\ a\ ) \qquad \textit{(2.14) [13]}$$

La simplificació és possible pel fet que el factor d'aprenentatge ha desaparegut de la fórmula, i això succeeix perquè el propi optimitzador de la xarxa neuronal disposa del seu propi factor d'aprenentatge, que aplica quan s'entrena la xarxa amb el nou valor. Per tant, l'algoritme bàsic de Deep Q-Learning seria el següent:

```
q_network = Inicialitzar_q_network()
epsilon = EPSILON_MAXIMA

mentre (nombre_experiments_realizats < total_experiments_a_realitzar)
{

  s = env.obtenir_estat_inicial()
  mentre (no env.episodi_ha_finalitzat())
  {
    exploració = determinar_exploracio_o_explotacio(epsilon)

    si (exploració)
      a = env.obtenir_acció_aleatoria()
    sino // explotació
      q_valors =q_network.estimar(s) // Retorna la estimació que fa la
                                     // xarxa neuronal dels valors per
                                     // a cadascuna de les accions
                                     // possibles
      a = argmax_a (q_valors)

    r, s' = env.realitzar_accio(a)

    si (env.episodi_ha_finalitzat())

   // Si l'episodi ha finalitzat, el següent estat és l'estat final, i
   // per tant el seu valor és 0, no esperem obtenir més recompenses

      nou_valor = reward

    sino
      q_valors_nou_estat = q_network.estimar(s')
      nou_valor = reward + descompte*max(q_valors_nou_estat)

    q_network.entrenar(s, a, nou_valor)

    epsilon = reduïr_valor(epsilon)

    s = s' // Actualitzem l'estat amb el nou estat per a la següent
```

```
            // iteració
  }
}
```

Aquesta versió simplificada de l'algoritme pot tenir diversos problemes per convergir correctament en una policy òptima. El primer dels problemes ve provocat pel fet que, amb aquest algoritme, la xarxa s'actualitza de forma seqüencial en el mateix ordre en el qual es produeixen les experiències. S'utilitzarà un exemple per explicar la problemàtica que això suposa. Es construeix un algoritme de Deep Q-Learning per a l'entorn d'un joc de taula, i es realitza l'entrenament contra dos rivals diferents, que segueixen estratègies oposades al joc. El sistema entrena realitzant una partida contra el primer dels rivals, i aprèn de totes les accions preses la millor policy per contrarestar l'estratègia d'aquest jugador. Posteriorment, juga contra l'altre rival, i com que juga d'una forma molt diferent, el sistema adapta la seva estratègia a aquest nou jugador, i "oblida" com guanyar a l'anterior. L'objectiu del nostre sistema és aconseguir una policy capaç de jugar igual de bé contra el primer rival que contra el segon, però el fet d'adaptar la xarxa neuronal de forma seqüencial produeix un biaix en el seu funcionament cap a l'últim jugador contra el qual ha entrenat.

Per tal d'evitar el problema descrit, s'utilitza la memòria d'experiències. En comptes d'entrenar la xarxa a cada pas amb la darrera experiència que s'ha realitzat, aquesta és guarda en una memòria. La memòria funciona com una cua, les experiències més noves reemplacen les més antigues quan es troba plena. Cada cert nombre de passos realitzats, s'entrena la xarxa amb una mostra aleatòria de totes les experiències que es troben a la memòria. D'aquesta manera, s'aconsegueix desacoblar temporalment l'entrenament de la xarxa amb la generació d'experiments, i per tant s'aconsegueix que desaparegui el biaix per l'ordre de les experiències.

Un altre avantatge de tenir una memòria d'experiències és que permet aprofitar la capacitat de les xarxes neuronals de ser entrenades per paquets. En comptes d'utilitzar únicament una experiència aleatòria de la memòria per entrenar la xarxa, s'obté tot un paquet de longitud definida, i s'utilitzen totes aquestes experiències per entrenar la xarxa d'un sol cop. A més a més, la memòria d'experiències i l'actualització de la xarxa per paquets permet la utilització d'una mateixa experiència en més d'un entrenament diferent de la xarxa mentre estigui disponible a la memòria, abans de ser reemplaçada per experiències més noves.

El segon dels problemes que presenta l'algoritme original de Deep Q-Learning, és el fet que per entrenar la xarxa, l'algoritme utilitza els valors de les accions en el següent estat que s'han estimat amb la mateixa xarxa neuronal. Aquest fet provoca que, quan s'actualitza la xarxa amb el nou valor objectiu, a la vegada també s'actualitzen els valors amb el qual s'ha calculat aquest valor objectiu. Aquest fet provoca que a la mateixa vegada que apropem la nostra xarxa al valor esperat, l'objectiu al qual s'intenta apropar s'allunya d'aquest nou valor, i aquest fenomen provoca una gran oscil·lació a l'actualització de la xarxa que introdueix una gran inestabilitat. Per tal de solucionar aquest problema, s'afegeix al sistema una segona xarxa neuronal: la xarxa de valors objectiu. Aquesta xarxa és una còpia de la xarxa de valors (o xarxa principal), però que no s'actualitza a cada cop que ho fa la xarxa principal. La xarxa de valors objectiu servirà per estimar els valors de les accions del següent estat, i com que la xarxa que

s'actualitza és la principal, aquests valors es mantindran fixos temporalment. Tot i que la xarxa de valors objectiu no s'actualitza de forma habitual, els seus valors han de ser una còpia aproximada de la xarxa principal. Per aconseguir-ho, cada cert nombre d'actualitzacions, la xarxa d'objectius clona el pes dels seus nodes a partir dels de la xarxa principal.

Una vegada realitzades les modificacions per solucionar les problemàtiques trobades, l'algoritme de Deep Q-Learning funciona de la següent manera:

```
q_network = Inicialitzar_q_network()
target_network = copiar(q_network)
memoria_d_experiencies= Inicialitzar_memoria()
epsilon = EPSILON_MAXIMA

mentre (nombre_experiments_realizats < total_experiments_a_realitzar)
{

  s = env.obtenir_estat_inicial()
  mentre (no env.episodi_ha_finalitzat())
  {
    exploració = determinar_exploracio_o_explotacio(epsilon)

    si (exploració)
      a = env.obtenir_acció_aleatoria()
    sino // explotació
    {
      q_valors =q_network.estimar(s) // Retorna la estimació que fa la
                                     // xarxa neuronal dels valors per
                                     // a cadascuna de les accions
                                     // possibles
      a = argmax_a (q_valors)
    }
    r, s' = env.realitzar_accio(a)

    final = env.episodi_ha_finalitzat()

    memoria_d_experiencies.afegir_nou_experiment(s, a, r, s', final)

    paquet_d_experiencies = memoria_d_experiencies.obtenir_aleatori()

    per cada exp a paquet_d_experiencies:
    {
      si (experiencia.final)
        exp.nou_valor = reward
      sino
        q_val_nou_estat = target_network.estimar(s')
        exp.nou_valor = reward + descompte * max(q_val_nou_estat)
    }

    q_network.entrenar(paquet_d_experiencies)

    epsilon = reduïr_valor(epsilon)
    si (s_ha_d_actualitzar_target_network)
      target_network.copiar_de(q_network)
```

```
    s = s' // Actualitzem l'estat amb el nou estat per a la següent
           // iteració
  }
}
```

2.2.2.1 Double Deep Q-Learning

L'algoritme Double Deep Q-Learning és una teòrica millora sobre el funcionament de l'algoritme Deep Q-Learning. Aquesta millora serveix per resoldre un problema ocasionat per la xarxa d'objectius. A la darrera versió de l'algoritme Deep Q-Learning, es tria com a valor del nou estat el de l'acció que s'estimava amb un valor més alt a la xarxa d'objectius. Però la xarxa d'objectius només s'actualitza cada cert temps, i és possible que en un cert punt aquesta acció ja no sigui la que té un valor més alt a la xarxa principal, que és la que té els valors actualitzats. Per tant, s'està sobreestimant el valor d'algunes accions. Per aquest motiu, el Double Deep Q-Learning realitza una modificació que consisteix a determinar l'acció del següent estat a considerar a partir de la seva pròpia estimació, però calcular el nou valor amb el qual actualitzar la xarxa a partir del valor de la xarxa d'objectius. La part modificada respecte a l'algoritme Deep Q-Learning seria:

```
si (experiencia.final)
  experiencia.nou_valor = reward
sino
  q_valors_nou_estat = q_network.estimar(s')
  q_valors_objectiu = target_network.estimar(s')
  seguent_accio = argmax(q_valors_nou_estat)
         experiencia.nou_valor     =     reward     +     descompte     *
q_valors_obj[seguent_accio]
```

D'aquesta manera, s'aconsegueix que l'acció a considerar en el següent estat s'adapti a les actualitzacions que s'han realitzat a les darreres iteracions, a la vegada que els valors objectiu es mantenen fixes i no afecten l'estabilitat de la xarxa com succeeix quan només hi ha una xarxa neuronal.

2.3.3 Mètode Monte Carlo

Els mètodes d'aprenentatge introduïts fins ara utilitzen una estratègia d'aprenentatge per diferència temporal. Aquests mètodes actualitzen el seu coneixement a cada pas que realitza l'agent, quan encara no ha finalitzat l'episodi (excepte si és l'últim pas abans d'arribar a l'estat final). Per tant, en el moment d'actualitzar, coneixen la recompensa immediata obtinguda per la darrera acció, però no el valor total que s'ha aconseguit a la resta de l'episodi. Per aquest motiu, aquells mètodes depenen de l'estimació del valor de l'estat posterior al que s'arriba després de realitzar l'acció, per tal de determinar el valor de l'estat en el qual es trobava.

Els mètodes basats en Monte Carlo, en canvi, no actualitzen el seu coneixement a cada pas que es realitza. Aquests algoritmes completen un episodi sencer, guardant a

cada pas les dades de l'experiència per, quan s'arriba a l'estat final, poder calcular les recompenses que s'han acumulat després de realitzar cadascuna de les accions. Si es repeteixen els experiments múltiples cops, en repetides ocasions l'entorn es trobarà en un mateix estat i l'agent haurà realitzat la mateixa acció. La mitjana dels valors obtinguts per totes aquestes experiències que comparteixen estat i acció realitzada hauria de convergir en el valor esperat per al parell estat-acció.

L'algoritme de Monte Carlo que s'implementarà utilitza un diccionari per emmagatzemar el coneixement que s'anirà adquirint per cadascun dels estats que s'hagin trobat al llarg de l'entrenament. Les claus utilitzades pel diccionari serà l'identificador únic que identifica cada estat. Els valors seran nodes que representen el coneixement que es té sobre l'estat. Aquests nodes, per la seva banda, contenen un diccionari a on la clau estarà formada per una acció que s'ha pres a l'estat en qüestió, i el valor conté el coneixement obtingut del parell estat-acció als experiments anteriors. Inicialment, el diccionari d'estats es troba buit, i per tant els diccionaris d'accions no existeixen. El node pertanyent a un estat no es crea fins que no es realitza una experiència en aquell estat. De la mateixa manera, el node per a una acció en un determinat estat tampoc no es crea fins que no es produeix una experiència amb la combinació estat-acció corresponent.

Figura 2.4 Estructura de dades per mètode de tipus Monte Carlo

En el cas de definicions d'estats de baixa dimensionalitat, el volum de dades emmagatzemats per aquest algoritme serà equivalent al de la q-table: amb poques iteracions és probable que tots els estats possibles tinguin creat el seu node, i que per a cada node s'hagin explorat totes les accions possibles. En canvi, l'estructura de nodes que s'utilitza en aquest projecte per realitzar l'algoritme de tipus Monte Carlo és més tolerant a definicions d'estats de dimensionalitat més alta. Això és degut al fet que aquells nodes d'estats que no són possibles, o que són molt poc probables, mai seran creats. En canvi, a la q-table, es troben representats tots els estats possibles, independentment de si succeeixen al procés d'entrenament o no.

L'algoritme realitza tres tasques per a cadascun dels experiments que es produeixen:

- **Expansió**: Quan es troba amb un node estat a on no s'han experimentat prèviament totes les accions possibles es realitza una tasca d'expansió, triant aleatòriament una de les accions per les quals encara no es té cap informació.

- **Selecció**: Cada cop que l'episodi es troba en un estat a on ja es té una estimació del valor per totes les accions possibles, es tria l'acció a realitzar. Per fer-ho, s'utilitza la següent fórmula per determinar quina és l'acció més prometedora [14]:

$$\frac{w_i}{n_i} + c\sqrt{\frac{ln\, N_i}{n_i}} \qquad \textit{(2.15) [14]}$$

  a on w_i és el valor de l'acció, n_i el nombre de vegades que s'ha triat prèviament aquesta acció, i N_i el nombre de cops que s'ha experimentat prèviament l'estat, independentment de l'acció triada. *c* és el paràmetre d'exploració: a més valor de c, més valor tindran les accions menys explorades, i per tant s'explorarà de forma més habitual.
- **Retropropagació**: Una vegada un experiment ha finalitzat, i es disposa de totes les experiències, es realitza la tasca d'actualització dels nodes per tots els estats que hem trobat a l'experiment. Per fer-ho, es comença per l'estat final, a on el valor del node és la recompensa obtinguda per l'acció triada. Per tant, s'actualitza el node pertanyent aquesta darrera acció, calculant la mitjana amb tots els valors obtinguts en episodis anteriors. Després, l'algoritme va retrocedint pels estats que ha visitat de l'últim cap al principi, sumant a cada pas la recompensa obtinguda. Per afegir el nou valor obtingut a la mitjana, seguim la següent fórmula recursiva que no requereix tots els valors obtinguts anteriorment, sinó que calcula la nova mitjana a partir del nou valor, la mitjana anterior, i el nombre d'experiències anteriors:

$$Q_{n+1} = Q_n + \frac{1}{n}[V_n - Q_n] \qquad \textit{(2.16) [7]}$$

  El factor 1/n és el que determina que totes les experiències tinguin exactament el mateix pes en la determinació del nou valor, independentment si ha succeït a l'inici de l'entrenament o al final. Si se substitueix per un valor fix, aquest factor farà que el pes dels darrers valors obtinguts sigui més gran que el dels primers. Aquest factor d'adaptació permet que, quan l'entorn canvia amb el temps, la policy s'adapti més ràpidament a aquests canvis.

2.4 Entorns de desenvolupament (OpenAI Gym)

OpenAI és una companyia dedicada a la recerca i el desenvolupament de projectes d'Intel·ligència Artificial. Va ser creada l'any 2015, quan Elon Musk i Sam Altman, juntament amb altres inversors van anunciar la seva formació com a organització sense ànim de lucre. OpenAI va néixer amb l'objectiu de promoure la investigació i la producció de sistemes d'intel·ligència artificial amigable, que proveeixin d'un benefici per a la humanitat en el seu conjunt, sense la limitació d'obtenir un benefici econòmic per a l'organització. Una altra de les motivacions originals d'OpenAI és la investigació i l'estudi de la seguretat dels projectes d'intel·ligència artificial. Una de les grans preocupacions dels fundadors d'OpenAI és que si la intel·ligència artificial avançada avancés prou com per arribar al nivell humà, podria arribar al punt de suposar un risc

per a tota la humanitat. Quan s'arribi a aquest punt, OpenAI es postula com a una institució que prioritzarà el bé general per sobre dels interessos particulars.

El 2016, OpenAI publica Gym, una eina per al desenvolupament i la comparativa de projectes d'aprenentatge per reforç. L'objectiu de Gym és estandarditzar la definició dels entorns, proporcionant una interfície unificada de forma que tots els entorns registrats disposen de les mateixes estructures i els mateixos mètodes per comunicar-se amb els agents. Aquesta estandardització facilita el procés de documentació, publicació, desenvolupament, comparació i reutilització de projectes de RL.

Tots els entorns publicats a Gym han d'implementar la interfície *Env*. Aquesta interfície encapsula l'entorn obligant a què proporcioni una sèrie de mètodes i d'atributs comuns per a tots els entorns registrats.

Els atributs que tot entorn que implementa la interfície Env ha de tenir són un espai d'accions, un espai d'estats (a Gym es diu espai d'observacions), i un rang de recompenses possibles. Per a la definició dels espais, Gym proporciona una interfície Space i classes que la implementen que permeten definir les estructures més habituals dels espais d'accions i d'estats. Per una banda, la classe *Discrete*(n) defineix un espai que només disposa d'un valor discret de 0 a n. L'altra classe de tipus Space que es defineix és *Box*, que permet la definició d'estats multidimensionals (un vector d'atributs).

Pel que fa als mètodes de tot entorn *Env*, trobem els següents:

- *reset*(): Inicialitza l'entorn al seu estat inicial. Retorna l'observació (estat) inicial.
- *step*(action): Realitza una acció a l'entorn. Retorna l'observació obtinguda després de realitzar l'acció, la recompensa, i si l'episodi ha finalitzat o no.
- *render*(mode='human'): Renderitza l'estat de l'entorn de forma visual.

OpenAI Gym disposa d'un ampli catàleg d'entorns publicats clàssics dins el món de l'aprenentatge per reforç, com per exemple jocs d'Atari, o simulacions de mans robòtiques.

Entre els avantatges que proporciona l'adaptació d'un entorn propi a Gym, trobem els següents:

- Es facilita la documentació i la comprensió dels agents que utilitzen l'entorn. Qualsevol que conegui la interfície de Gym podrà entendre sense dificultats el procés de comunicació entre un agent i l'entorn.
- Permet el desenvolupament de nous agents per al mateix entorn, i possibilita la comparació dels resultats obtinguts pels diferents agents.
- Facilita la reutilització del codi dels agents per a altres entorns registrats a Gym, ja que seguiran la mateixa estructura de comunicació.

3. Sushi Go: el joc

Sushi Go [4] és un joc de cartes ideat per en Phil Walker-Harding i distribuït pel grup Devir. El joc està dissenyat per ser jugat entre 2 i 5 jugadors, i cada partida dura aproximadament uns 15 minuts.

Una partida de Sushi Go representa un àpat a un restaurant japonès a on se serveixen els plats més típics de la cuina d'aquell país: makis, nigiris, sashimi, etcètera. Els jugadors són els comensals, i les cartes que formen el joc representen els plats que, com si es trobessin en una cinta transportadora de menjar, van rotant de jugador en jugador, cadascun dels quals haurà de triar el plat que millor li convingui i deixar la resta de cartes "a la cinta" pels següents jugadors. El jugador que aconsegueixi les millors combinacions de plats és el guanyador.

3.1 Regles

Cada partida de Sushi Go consta de 3 rondes. Cada ronda comença amb el repartiment d'un nombre determinat de cartes a cada jugador. El nombre de cartes a repartir, entre 7 i 10, ve definit per la següent taula:

Jugadors	Cartes
2	10
3	9
4	8
5	7

Taula 3.1. Sushi Go: Cartes per jugador

A cada torn, tots els jugadors han de triar una de les cartes de la mà, i posar-la davant seu cap per avall. Quan tots els jugadors han triat la seva carta, la mostren tots a la vegada, i l'afegeixen a la seva taula amb la resta de cartes triades prèviament a la ronda. Una vegada s'han mostrat totes les cartes, els jugadors entreguen la resta de cartes de la seva mà al jugador de la seva esquerra, i reben les del jugador de la seva dreta.

Quan acaba una ronda, cada jugador rebrà una quantitat de punts en funció de les cartes que tingui a la seva taula. Cadascun dels plats (cartes) puntua d'una forma diferent:

Nigiri: Els nigiris donen un, dos o tres punts en funció del seu tipus. Un nigiri de truita suma un punt, un nigiri de salmó dona dos punts, i un de calamar dona tres punts.

Wasabi: Una carta de wasabi per si mateixa no dóna punts, però el següent nigiri que es posi a la taula després que s'hagi posat un wasabi, triplica el seu valor. Per tant, si disposem d'un wasabi lliure a la taula, i posem un nigiri de truita, en comptes d'un punt ens proporcionarà tres punts. Un nigiri de salmó sobre un wasabi en dona sis punts, i un de calamar ens sumaria nou punts.

Gyoza: Les gyoces ens proporcionen més punts per cada gyoza que ja teníem a la taula. Si a una ronda només tenim una gyoza a la taula, obtindrem un únic punt. Dos gyoces proporcionen tres punts, tres gyoces proporcionen sis, quatre gyoces deu i finalment cinc o més gyoces proporcionen un total de 15 punts.

Tempura: Les cartes de tempura sempre puntuen en parells. Una única carta de tempura no dóna punts, però dues tempures a la taula del jugador li proporcionen cinc punts. És possible tenir més d'un parell de tempures a la mateixa ronda, i cada parell de cartes proporcionarà cinc punts.

Sashimi: Els sashimis puntuen de forma similar a la tempura, però en comptes d'en parells, puntuen en grups de tres cartes. Un o dos sashimis no donen punts, però tres cartes de sashimi a la taula suposen deu punts per al jugador a la ronda.

Maki: Les cartes de maki poden proporcionar un, dos o tres rotllos. Quan acaba la ronda, tots els jugadors han de sumar el nombre de rotllos que han acumulat. El jugador amb més rotllos suma sis punts, i el segon n'obtindrà tres.

Palets: La carta de palets no dóna punts. Quan un jugador té la carta de palets a la seva taula, té l'opció d'utilitzar-la fins al final de la ronda. Quan un jugador utilitza la carta de palets en un torn, pot posar dues de les cartes de la seva mà a la taula en comptes de només una com és habitual. Aquest jugador ha de posar la carta de palets a la seva mà abans de passar-la al següent jugador per començar el següent torn.

Pudding: Les cartes de postres, al contrari de tota la resta, no puntuen al final de cada ronda, sinó que es mantenen fins al final de la tercera. Quan acaba la tercera ronda i s'han contat les puntuacions obtingudes per la resta de cartes, cada jugador suma el nombre de puddings que ha acumulat al llarg de la partida. El jugador que tingui més puddings sumarà sis punts a la seva puntuació final. En canvi, el jugador amb menys puddings en restarà sis.

NIGIRI	**WASABI**	**GYOZA**	**TEMPURA**
De calamar: **3** De salmó: **2** De truita: **1**	Següent nigiri **x3**	1 2 3 4 5 **1 3 6 10 15**	Parella: **5** Només un: **0**
SASHIMI	**MAKI**	**PALETS**	**PUDDING**
Trio: **10** Només un o dos: **0**	Al finalitzar la ronda El que més: **6** El segon: **3**	S'utilitzen per jugar dues cartes en lloc d'una	Al final de la partida El que més: **6** El que menys: **-6**

Taula 3.2 Sushi Go: Resum de funcionament de les cartes

Al final de cada ronda, una vegada s'han sumat les puntuacions de cada jugador, totes les cartes de les taules dels jugadors es retiren, a excepció dels puddings que es mantenen fins al final de la partida.

Quan finalitza la darrera ronda, i s'han sumat tots els punts a tots els jugadors incloent els dels puddings, el jugador amb més puntuació és el guanyador. En cas d'empat a punts entre dos a més jugadors, guanya el que ha acumulat més puddings. En cas d'empatar també en nombre de puddings, es produeix un empat final.

3.2 Versions

A més a més de la versió original del joc, existeix una ampliació anomenada "Sushi Go Party". Aquesta expansió introdueix una gran quantitat de tipus de carta noves amb les seves regles i mètodes de puntuació particulars. A més a més, incrementa el nombre de jugadors possibles fins a vuit.

A l'ampliació no es juga amb totes les cartes disponibles, sinó que abans de la partida es tria el "menú" que s'utilitzarà, és a dir, es trien els vuit tipus de carta amb les quals es jugarà la partida. Per tant, es forma la baralla amb les cartes que pertanyen als tipus seleccionats, i s'aparten la resta. Algunes regles del joc original, així com les puntuacions i particularitats d'alguns dels tipus de cartes que hi pertanyen, s'han modificat a l'ampliació per adaptar-se millor a les seves característiques de la versió.

3.3 Projectes d'aprenentatge per reforç preexistents

Abans de començar amb la fase d'implementació del projecte, s'ha fet recerca d'altres projectes que hagin utilitzat Sushi Go com a entorn per realitzar els agents de RL.

An OpenAi-like environment for the sushi go card game [1]:

- La implementació del joc no és completa. No han afegit totes les regles (l'acció de la carta "Palets" no està inclosa)
- Implementa un entorn que imita els entorns d'*OpenAI gym*, però no de forma completa, i per tant no es pot registrar com a entorn Gym.
- Té alguns esbossos a on es fan permutacions de la prioritat de cada tipus de carta, per determinar quina dóna un resultat millor.
- A la carpeta "learning" s'han aplicat diversos algoritmes de RL basats en normes estocàstiques.

Monte Carlo for Sushi Go (to be used with reinforcement learning) [2] :

- La implementació del joc és completa i funcional, però no segueix les regles del joc original sinó les de l'ampliació "Sushi Go Party".
- Les accions i els estats no segueixen una estructura d'un entorn *OpenAI gym* ni similar. Com a conseqüència és molt difícil determinar el funcionament dels diferents algoritmes que implementa.
- Implementació d'un algoritme de cerca Monte-Carlo
    - Utilitza un factor d'adaptació fixat, en lloc de la mitjana del valor obtingut en prendre una acció en un estat determinat.
    - Utilitza una q-table per emmagatzemar els valors dels parells estat-acció.
- Implementació d'un algoritme basat en "Expected Sarsa"
    - En aquest cas realitza una implementació a mig camí entre l'algoritme Expected Sarsa i l'algoritme de Q-Learning. El valor d'actualització es calcula en part com la mitjana de totes les accions, i en part com el de l'acció que proporciona valor màxim. La proporció de cadascun dels valors la determina un paràmetre èpsilon.
- Implementació d'un algoritme basat en Deep Q-Learning
    - Proporciona el codi de la classe amb la qual es generen els agents, però no es troba disponible el codi que utilitza la classe per generar els agents.
    - Utilitza totes les millores i variants que utilitzarem a la nostra versió: memòria d'experiències, xarxa de valors objectius, i Double Deep Q-Learning

Making Tasty Sushi using Reinforcement Learning and Genetic Algorithms [3]:

> En aquest cas, trobem la memòria del projecte, però no disposem del codi. Han utilitzat un joc lleugerament canviat, substituint algunes cartes per unes altres (provinents de l'expansió del joc, "Sushi Go Party"). A aquest projecte no es va aconseguir que els algoritmes de RL milloressin significativament el comportament de l'agent, però mitjançant algoritmes genètics sí que van obtenir millors resultats.

Una vegada revisats els projectes preexistents, podem descartar la possibilitat de comparar directament els resultats obtinguts dels altres projectes amb el nostre, fent jugar els agents creats pel nostre projecte amb els trobats en els altres, a causa de les modificacions que s'hauran de realitzar tant en la implementació de l'entorn com en l'estructura dels espais d'accions utilitzats.

4. Metodología de la implementació

4.1 Punt de partida

Amb la revisió dels projectes preexistents, es pot concloure que cap disposa d'una implementació que s'adapti completament als requeriments i objectius plantejats en aquest projecte, ja que o bé és incompleta, o bé aplica unes normes lleugerament diferents de les del joc amb el qual s'ha decidit realitzar el projecte. No obstant això, la implementació trobada al projecte "Monte Carlo for Sushi Go (to be used with reinforcement learning)"[2] servirà com a punt de partida per a realitzar la versió pròpia. A continuació s'indiquen les modificacions que s'han de desenvolupar:

- Modificacions necessàries per adaptar la implementació a les normes del joc “Sushi Go” en comptes de la versió “Sushi Go Party”
- Creació d’una estructura de les classes en fitxers separats per millorar la seva mantenibilitat.
- Creació d’un element “Game” que s’encarregui del control de la partida, que faciliti l’obtenció de l’estat, i que gestioni les accions que realitzen els jugadors.
- Creació d’un element “Action” que permeti la gestió centralitzada de les accions que es poden realitzar a cada torn del joc.
- Aplicació del paradigma d’orientació a objectes a la implementació, millorant la seva llegibilitat i adaptabilitat. Aquest punt és primordial de cara a aplicar canvis en el futur, o que pugui ser reutilitzable en altres projectes.
- Generació d’un entorn OpenAI Gym per a Sushi Go, que implementi la interficie *Env*, i que d’aquesta manera faciliti la reusabilitat tan pròpia com per part de qualsevol que vulgui generar els seus propis projectes de RL per a aquest joc.

4.2 Implementació de l’entorn Sushi Go

Durant la realització del projecte s’han pres les següents consideracions que afecten directament a la implementació:

- S’han definit dos modes de joc. Un mode anomenat “SingleGame”, a on cada torn el jugador determina només l’acció que realitza un dels jugadors, el jugador “humà”. Les accions dels altres jugadors vindran determinades pels agents que es proporcionaran com a paràmetre. Al mode “MultiplayerGame”, en canvi, a cada torn es determinen externament les accions de tots els jugadors que hi participen. Aquest mode permetrà controlar manualment per part de l’usuari totes les jugades de tots els jugadors.

- Quan s’utilitza una carta Nigiri o Maki, en cap cas pot donar un millor resultat utilitzar una carta de menys valor que una altra disponible a la mà. No hi ha cap estratègia a la qual pugui suposar un avantatge utilitzar un Nigiri de 2 punts,

quan a la mà es disposa d'un altre Nigiri de 3 punts. Per tant la implementació haurà de permetre utilitzar aquestes cartes de dos modes diferents: de forma que automàticament es trobi a la mà la carta de més valor per a un determinat tipus quan el jugador és un agent, o de forma que es respecti la carta triada amb el seu valor per quan el jugador estigui controlat per un humà.

- Tot i que s'han realitzat les modificacions per afegir les regles del joc original, s'ha mantingut l'opció de jugar amb les regles de la versió "Sushi Go Party". Les regles a utilitzar es poden determinar com a un paràmetre inicial del joc.

A continuació es descriuen tots els elements del joc que formen part de la implementació:

Carta: Elements que representen cadascuna de les cartes del joc. Estan definits de forma jeràrquica, de forma que aplicant els conceptes de polimorfisme i sobrecàrrega les instàncies de cadascuna de les cartes tenen el comportament d'acord amb les regles que li pertoquen. Cada tipus carta s'encarrega de calcular la puntuació que aporta a un jugador en el moment de ser utilitzada. Per exemple, cadascun dels tipus de Nigiri té fixat un valor d'1, 2 o 3 per la puntuació que aporta, però si està relacionat a una carta Wasabi, triplica la puntuació. Els Palets no proporcionen punts de forma directa, i per tant aporten sempre un valor de 0. La carta de Tempura aporta un valor de 5 punts, però només quan la quantitat de tempures que hi ha a la taula és un nombre parell. D'aquesta manera, la primera tempura aporta 0 punts, la segona 5, la tercera 0, la quarta 5, etcètera...

Mà: Aquest element representa el conjunt de cartes a la mà de cadascun dels jugadors. Permet obtenir de la mà una carta d'un determinat tipus, en funció de l'acció triada pel jugador. Aquest mètode té la peculiaritat que tria la carta de valor més alt del tipus sol·licitat. Això permet que si busquem una carta del tipus Nigiri o Maki, esculli de la mà aquella que doni un valor més gran. Però a la vegada, si es busca específicament un Maki de valor 1, buscarà aquesta carta en concret, ignorant les de més valor.

Taula: Element que representa les cartes a la taula del jugador, és a dir, les cartes que s'han utilitzat prèviament a la ronda. A més de mantenir el conjunt de cartes a la taula, s'encarrega de buidar la taula entre rondes, de retornar la carta de Palets quan ha estat utilitzada, o d'afegir les cartes utilitzades al conjunt, enllaçant en cas necessari les cartes de tipus Wasabi amb el darrer Nigiri.

Jugador: Aquest element serveix per representar als jugadors de la partida. Els jugadors s'identifiquen per un número, i tenen una mà per a la ronda actual, i una taula. Mantenen també la puntuació acumulada pel jugador, i els punts guanyats en el darrer torn (la recompensa). Entre les tasques d'aquest mòdul trobem la de crear la mà inicial a cada inici de ronda, la de passar la mà actual al següent jugador, la d'obtenir la mà del jugador anterior, o la de jugar un torn i obtenir el guany corresponent a la jugada realitzada:

```
def play_a_turn(self, first_card_type, second_card_type = None):

        assert first_card_type is not None

        chopsticks_action = second_card_type is not None

        original_score = self.get_current_score()

        self.__play_a_card(first_card_type)

        if chopsticks_action:

            self.__play_a_card(second_card_type)
            self.__return_chopsticks_to_hand()

        turn_reward = self.get_current_score() - original_score
        self.__set_reward(turn_reward)

        self.__hand_cards_to_next_player()
```

Baralla: Aquest element representa la baralla de cartes. En funció del mode de joc (original o "Party") es generen les cartes que s'utilitzaran a la partida. Aquest mòdul també s'encarrega de repartir de forma aleatòria les cartes als jugadors. Per mantenir l'opció de la versió "Party", es mantenen els mètodes d'afegir postres entre rondes, i de retornar les cartes utilitzades a la baralla, que no s'executen en el joc original.

Joc: Element que defineix el comportament de la partida. Es defineix el funcionament dels dos modes de joc: "SingleGame" i "MultiplayerGame". Una partida conté els jugadors que hi pertanyen, i la baralla que s'utilitza. Aquest mòdul també controla el flux correcte de la partida: s'inicialitzen els jugadors repartint les cartes a l'inici, es gestionen les tres rondes, i es calculen les puntuacions al final de cada ronda. També ofereix els mètodes perquè els jugadors puguin interactuar amb la partida, indicant l'acció a realitzar a cada torn. El següent codi mostra la implementació d'un torn en el mode "SingleGame":

```
def play_cards(self, cards):

  self.get_player(0).play_a_turn(cards[0], cards[1])

  for agent_index, agent in enumerate(self.__get_agents()):

    player_number = agent_index + 1
    player = self.get_player(player_number)

    legal_actions = self.get_legal_actions_of_player(player_number)
    action = agent.choose_action(legal_actions)

    cards = action.get_pair_of_cards()
    self.play_player_cards(player_number, cards)
```

```
    self.finish_turn()

    return self.get_player(0).get_last_action_reward()
```

Diagrama d'entitats de la implementació de Sushi Go

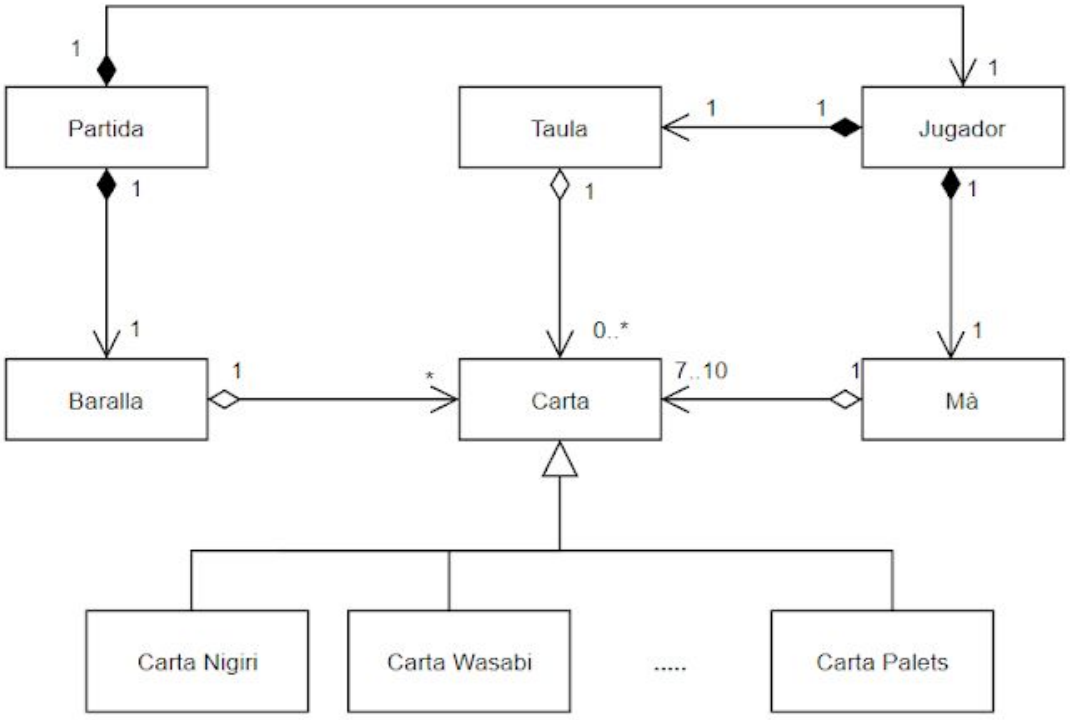


Figura 4.1 Diagrama d'entitats de la implementació de Sushi Go

4.3 Implementació dels mòduls de suport per a l'aprenentatge per reforç

Els mòduls que es descriuen en aquesta secció són aquells que serveixen per representar conceptes propis de l'aprenentatge per reforç, aplicats a la implementació realitzada per Sushi Go.

4.3.1 Espai d'accions

Aquest mòdul conté els elements que permeten gestionar per part dels agents automàtics i dels processos d'aprenentatge, les accions que pot realitzar un jugador en el torn d'una partida de Sushi Go. El conjunt d'accions que es poden realitzar determinen l'espai d'accions de l'entorn.

Les accions tenen dues representacions diferents:

- Una acció pot ser representada per un tipus de carta (o dos en cas que sigui una acció d'utilitzar palets). Aquesta representació és la utilitzada per la implementació de Sushi Go.
- Una acció pot ser representada per un identificador numèric. Els identificadors han de ser únics per cadascuna de les accions, i han de ser consecutius.

En cas de les accions de palets, l'ordre de les cartes no té cap efecte tret d'un cas específic, quan es juga a la vegada una carta Wasabi i una carta Nigiri. Com que el Wasabi sempre es lliga amb el següent Nigiri que utilitza el jugador, no té el mateix efecte utilitzar el Nigiri, i després el Wasabi, que romandrà lliure per al següent Nigiri que s'utilitzi a torns posteriors, que posar primer el Wasabi, i a continuació el Nigiri, que quedarà associat al Wasabi que s'acaba d'utilitzar.

Per tant, tenim tres tipus d'accions possibles: bàsiques, accions dobles de palets, i una acció doble especial de jugar un Wasabi abans que un Nigiri. Definim tot l'espai d'accions disponibles a la següent taula:

#	Carta 1	Carta 2	#	Carta 1	Carta 2	#	Carta 1	Carta 2
0	Palets	-	13	Nigiri	Gyoza	26	Sashimi	Sashimi
1	Nigiri	-	14	Nigiri	Puding	27	Sashimi	Tempura
2	Wasabi	-	15	Wasabi	Wasabi	28	Sashimi	Gyoza
3	Maki	-	16	Wasabi	Maki	29	Sashimi	Puding
4	Sashimi	-	17	Wasabi	Sashimi	30	Tempura	Tempura
5	Tempura	-	18	Wasabi	Tempura	31	Tempura	Gyoza
6	Gyoza	-	19	Wasabi	Gyoza	32	Tempura	Puding
7	Puding	-	20	Wasabi	Puding	33	Gyoza	Gyoza
8	Nigiri	Nigiri	21	Maki	Maki	34	Gyoza	Puding
9	Nigiri	Wasabi	22	Maki	Sashimi	35	Puding	Puding
10	Nigiri	Maki	23	Maki	Tempura	36	Wasabi	Nigiri
11	Nigiri	Sashimi	24	Maki	Gyoza			
12	Nigiri	Tempura	25	Maki	Puding			

Taula 4.2 Espai d'accions de Sushi Go

Per tal de reduir el nombre d'accions possibles, s'han pres les següents decisions:

- No es defineixen les accions dobles de posar uns palets i una altra carta. Utilitzar uns palets per posar uns altres palets i una altra carta seria equivalent a l'acció de posar l'altra carta sense utilitzar els palets a la taula.

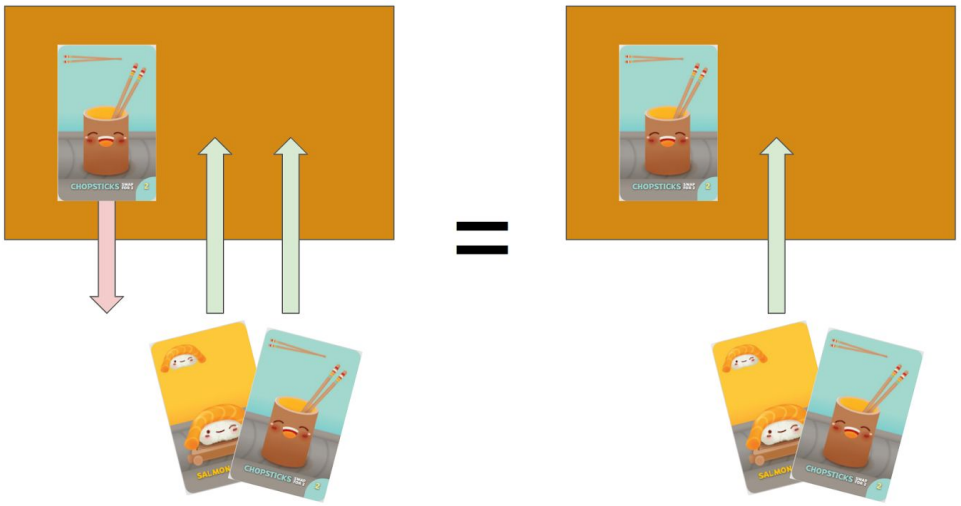


Figura 4.3 Equivalència de jugades dobles de Palets

- No es distingeixen els tres tipus de Nigiri, ni els tres tipus de carta de Maki. Una acció que implica posar una carta de Nigiri, implica utilitzar el Nigiri que hi hagi a la mà de més gran valor. En cap cas pot donar millor resultat utilitzar un Nigiri que no sigui el de més valor de la mà, i el mateix aplica a les cartes Maki.

Aquest mòdul permet obtenir el número que pertany a un parell de cartes, i també realitzar el procés invers. Una altra de les seves funcionalitats és que a partir de les cartes a la mà d'un jugador, determina el conjunt d'accions que aquest jugador pot realitzar, i per tant obtenir el llistat d'accions legals del jugador al torn en el qual es troba la partida.

4.3.2 Espais d'estats

Si s'analitza l'entorn creat per a realitzar partides de Sushi Go, es troba que des del punt de vista d'un jugador es pot definir una gran quantitat d'espais d'estat diferents, depenent de la precisió de l'observació que necessiti l'algoritme d'aprenentatge que utilitza aquell espai. Els espais d'estats més senzills poden disposar només del moment en el qual es troba la partida. Aquests espais disposaran només de dos atributs, el torn, i la ronda.

L'espai d'estats es pot ampliar afegint informació de quines cartes disposa el jugador a la seva mà. Com que les accions que impliquen l'ús de Nigiri i de Maki no tenen en compte el valor, la informació de quin és el valor que del Nigiri i del Maki màxim que hi ha a la mà és una informació rellevant per tal de prendre la decisió de quina acció triar. Per tant, l'estat de la mà tindrà dos atributs, el valor màxim de Maki, i el valor màxim de Nigiri.

L'estat també pot proporcionar la informació de les cartes que té el jugador a la seva taula. Evidentment, el fet de tenir cap, un o dos Sashimi a la taula pot ser determinant per decidir si la millor jugada és utilitzar un altre Sashimi o una carta diferent. En canvi, la quantitat de Nigiri utilitzats fins al moment no aporta cap informació de cara a determinar el valor que es pot obtenir a la resta de la partida. L'estat de la taula es pot representar per set atributs, un per cadascun dels tipus de carta a excepció dels Nigiri.

Un jugador, a part del contingut de la seva taula, també pot conèixer el de la taula del jugador de la seva dreta, a qui passa les cartes, o el del jugador del qual les rep, o el de tots els jugadors. Cadascuna de les taules dels rivals es pot representar per 7 atributs addicionals.

Fins i tot, es podria afegir a l'espai d'estats del jugador quines cartes ha tingut a la mà prèviament, i per tant saber quines cartes rebrà de nou si s'eliminen les que utilitzin els seus rivals abans que la mà torni a ser la seva.

La dimensió de l'estat no ve donada només pel nombre d'atributs acumulats per tenir la informació de la partida, de la mà, i de les taules dels jugadors. Per cada atribut, també es pot obtenir una informació més o menys acurada. Per exemple, per a l'atribut que determina el nombre de makis que té un jugador a la taula, es pot representar amb el nombre total de makis sense cap filtre (que pot arribar a tenir fins a 28 valors possibles), es pot fixar un valor màxim (valors de 0 a 8, on 8 representa que hi ha 8 makis o més) o tenir només dos valors que siguin "menys de cert valor" i "més de cert valor".

Els estats poden anar des dels més limitats en quantitat d'informació, amb una dimensionalitat petita, fins a estats amb una dimensionalitat molt alta que donen una informació completa i acurada. Alguns algoritmes obtindran millors resultats d'aquests espais que proporcionen molta informació. Altres, en canvi, no seran capaços de gestionar espais amb una dimensionalitat tan alta, o se'n veuran greument perjudicats per aquesta dimensionalitat, i per tant necessitaran versions més limitades de l'espai d'estats. L'objectiu del mòdul d'estats és facilitar la creació i gestió de tants espais d'estats com siguin necessaris durant la implementació dels algoritmes d'aprenentatge que s'utilitzen al projecte.

Els elements pertanyents a aquest mòdul extreuen de l'element "Jugador" de la implementació els atributs que tenen definits. Una vegada obtinguda la informació de la partida, l'estat es pot representar en qualsevol de les dues formes que poden ser necessàries per als algoritmes de RL que s'implementen en aquest projecte:

- Els algoritmes Q-Learning i el de mètode Monte Carlo necessiten obtenir l'estat com a identificador numèric. Els identificadors han de ser únics per cadascun dels estats, i han de ser consecutius.
- Els algoritmes de Deep Q-Learning necessiten obtenir l'estat com un vector amb els valors dels atributs que el formen. Aquesta representació és la que s'utilitza també als entorns OpenAI Gym.

El mòdul de gestió d'espais d'estat implementat disposa de mètodes per realitzar transformacions dels estats de qualsevol representació a qualsevol altra segons sigui

necessari. El procés per obtenir l'identificador a partir dels atributs que formen l'estat és el següent:

```
accumulated_value = 0
base = 1

accumulated_value += attribute1 * base
base *= attribute1_possible_values

accumulated_value += attribute2 * base
base *= attribute1_possible_values

...

accumulated_value += attributeN * base

return accumulated_value
```

mentre que el procés contrari, la transformació de l'identificador als atributs de l'estat, s'ha implementat de la següent manera:

```
attribute1 = state_id % attribute1_possible_values
state_id = state_id / attribute1_possible_values

attribute2 = state_id % attribute2_possible_values
state_id = state_id / attribute2_possible_values

...

attributeN-1 = state_id % attributeN-1_possible_values
state_id = state_id / attributeN-1_possible_values

attributeN = state_id
```

Aquest sistema de gestió dels espais d'estats permet definir de forma senzilla el contingut de l'estat, facilitant la creació de tantes versions diferents com siguin necessàries per a realitzar els agents.

Els estats més complexos poden ser generats de forma jeràrquica combinant altres estats més senzills entre ells. D'aquesta manera, un espai que representi l'estat d'un jugador pot estar format per un espai que representi l'estat de la partida, un espai que representa l'estat de la seva mà, i un espai que representa l'estat de la taula. L'espai que conté als altres tres estats generarà l'identificador dels estats propis a partir dels identificadors generats pels tres espais que conté.

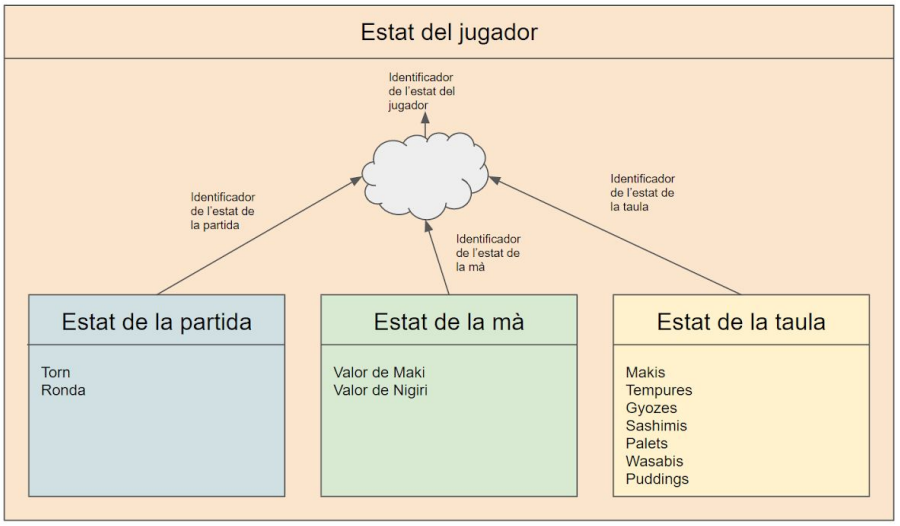


Figura 4.4 Combinació d'espais d'estats

4.3.3 Agents

Aquest mòdul serveix com a interfície per als agents que es construeixen al llarg d'aquest projecte. Tots els agents estan relacionats amb l'element "Jugador" que estan encarregats de controlar, i tots han de provenir d'un mètode que tria una acció entre les accions possibles, i que per tant determina quina és la policy de l'agent. Els agents també proporcionen els mètodes necessaris per permetre que es pugui continuar aprenent a partir del resultat obtingut per les accions realitzades.

La policy dels agents pot ser generada de forma automàtica mitjançant els algoritmes de RL, o de forma manual programant directament el comportament de l'agent. Abans de començar la generació d'agents mitjançant RL, es creen un conjunt d'agents de forma manual que s'utilitzaran de rivals per als entrenaments inicials dels agents de generació automàtica.

Inicialment, es crea l'agent més simple possible, anomenat RandomAgent, que tria entre totes les accions possibles una a l'atzar. Posteriorment, s'han creat diversos agents programats manualment per seguir unes polítiques molt simples, com per exemple prioritzar un tipus de carta per sobre la resta, o al contrari, evitar al màxim possible jugar un determinat tipus de carta.

Cada implementació dels diferents agents tindrà associat un espai d'estats que determina els atributs de la partida que rep l'agent per tal de prendre la decisió de quina acció realitzar. Per als agents que es generen automàticament és indispensable que s'utilitzi exactament el mateix espai d'estats que el que s'ha utilitzat per crear-ho. D'aquesta manera, s'assegura que la representació de tots els estats és la mateixa al procés d'aprenentatge, i a la utilització d'aquest aprenentatge adquirit.

L'espai d'estats associat a l'agent obtindrà del jugador que controla l'agent els atributs que precisa per determinar l'estat actual. Una vegada disposa de l'estat, aquest es pot

transformar a la representació d'estat que sigui necessària per aplicar la policy de l'agent, en funció de l'algoritme de RL que s'ha utilitzat per a la seva construcció.

4.4 Adaptació a l'entorn OpenAI Gym

Per poder registrar un paquet com a entorn d'OpenAI Gym, s'han realitzat els següents passos:

- Crear l'estructura de directoris i fitxers que requereix l'entorn per poder ser registrat.
- Crear la classe de l'entorn com a subclasse de gym.Env
- Definir l'atribut action_space, que conté les accions possibles a cada pas. En el cas de l'entorn que s'està creant, aquestes accions possibles són sempre un subconjunt molt petit del total d'accions. Per tant, s'ha definit un espai dinàmic per tal de contenir aquest atribut.
- Definir també l'atribut observation_space necessari per als entorns OpenAI Gym, que defineix els estats possibles de l'entorn. Mitjançant les classes del mòdul d'estats es pot construir directament l'observació de l'estat de la partida.
- Implementar els mètodes propis dels entorns d'OpenAI Gym:
    - step, a on en el nostre cas es juga un torn del joc, determinat per l'acció obtinguda per paràmetre. Retorna el benefici obtingut, la nova observació i si ha finalitzat la partida o no.
    - reset, a on s'inicia una nova partida del joc.
    - render, a on es mostra per pantalla l'estat actual del joc.

4.5 Desenvolupament dels agents

Abans de desenvolupar els agents, s'han definit una sèrie de criteris a mantenir en la construcció de cadascun d'ells. Mantenir aquests criteris és primordial per poder realitzar la comparativa del rendiment dels agents realitzats a les mateixes condicions.

- Tots els experiments es realitzen sobre les regles originals de Sushi Go, i no les de l'expansió Sushi Go Party. Tot i que l'entorn permet totes dues possibilitats, es fixen les regles a les de la versió original del joc.
- El nombre de jugadors a tots els experiments és de dos jugadors.
- El valor a maximitzar de la policy dels agents que es construeixen és la puntuació total obtinguda a cada experiment. L'objectiu dels agents és obtenir la puntuació màxima possible, independentment de si s'aconsegueix la victòria o no. Per tant, les accions triades sempre tenen l'objectiu d'afavorir la puntuació que aconsegueix l'agent al final de la partida, encara que l'acció triada permeti al rival aconseguir encara més punts.

4.5.1 Agents rivals per a l'entrenament

L'estratègia que segueix l'agent rival tindrà un pes determinant a la policy que aprendrà l'agent generat. Un agent entrenat solament contra un rival que utilitza una estratègia

fixada, aprendrà una estratègia òptima per obtenir el màxim resultat contra aquest rival, però no serà flexible quan s'enfronti a altres agents amb estratègies diverses.

Per aquest motiu s'ha de seguir una estratègia de generació d'agents per fases. Inicialment, no es disposa de cap agent entrenat de forma automàtica. Per tant, els primers agents s'hauran d'entrenar enfrontant-los amb l'agent RandomAgent, i amb la resta d'agents creats de forma manual. Posteriorment, quan es disposa de les primeres versions d'agents automàtics, es poden utilitzar al seu torn com a rivals, suposadament més competitius que els manuals, per a versions posteriors dels agents. D'aquesta manera s'aconsegueixen agents que s'adapten millor a les estratègies rivals. Per tant, s'estableix el següent procediment per fases de creació d'agents, que s'aplica a cadascun dels algoritmes de RL utilitzats al projecte:

- Fase 1: Els agents s'enfronten a l'agent RandomAgent a tots els experiments.
- Fase 2: Per cada experiment, es tria aleatòriament un dels agents generats manualment.
- Fase 3: Per cada experiment, es tria aleatòriament qualsevol dels agents generats a les fases 1 o 2.
- Fase 4: Per cada experiment, es tria aleatòriament qualsevol dels agents manuals, o qualsevol dels agents generats a les fases 1, 2 o 3.

4.5.2 Constructor mitjançant algoritme Q-Learning

Definició dels estats

Un dels factors fonamentals que han de permetre l'algoritme de Q-Learning obtenir una policy capaç d'obtenir un rendiment òptim és la dimensió de l'espai d'estat. Al capítol d'introducció d'aquest algoritme (2.2.1) s'explica com la mida de la q-table limita la capacitat de l'espai d'estats a utilitzar.

Per a la realització dels agents i la comparació entre el resultat obtingut amb l'espai d'estats més reduït possible, i un espai amb una quantitat d'informació de la partida més acceptable, són definits els següents espais:

- GameState: Aquest estat proveeix únicament informació del torn i de la ronda a la qual ens trobem a la partida. Per tant, disposa d'únicament 30 estats diferents.

GameState	
Atribut	Valors
Torn	1-10
Ronda	1-3

Taula 4.5 Atributs de l'espai GameState

- PlayerSimplState: Aquest estat, a més de determinar el torn i ronda en què es troba la partida, també conté informació molt simplificada de les cartes que té el jugador a la seva taula. En aquest cas, el nombre d'estats diferents en els quals es pot trobar la partida és de 8.640.

PlayerSimplState

GameState	
Atribut	Valors
Torn	1-10
Ronda	1-3

TableSimplState	
Atribut	Valors
Tempura	0-1
Sashimi	0-2
Gyoza	0-2
Maki	0-1
Pudding	0-1
Palets	0-1
Wasabi	0-1

Taula 4.6 Atributs de l'espai PlayerSimplState

Per poder accedir als valors de la q-table, és necessari que els estats siguin un identificador numèric únic, que serveixi per indexar la q-table. Per obtenir aquest identificador, s'utilitza la transformació d'estat a nombre definida a l'espai d'estats, a la secció 4.3.2.

Detalls de la implementació

Per realitzar la implementació de l'algoritme Q-Learning, es parteix de la base del codi definit al capítol 2.2.1 que introdueix el seu funcionament. Per realitzar els experiments, s'interactua directament amb l'entorn OpenAI generat específicament per realitzar partides de Sushi Go.

Es gestiona l'èpsilon (el ràtio d'exploració) determinant com a hiperparàmetres el seu valor màxim, el valor mínim, i la velocitat a la qual es decrementa el seu valor. El decrement es realitza a una velocitat exponencial.

Una particularitat de l'entorn de Sushi Go és que a cada torn generalment només seran legals una petita porció de les accions de l'espai d'accions. Per aquest motiu, per calcular el valor del següent estat, s'ha d'obtenir el valor més gran dels parells estat-acció que siguin legals després de l'acció realitzada, en comptes del valor màxim de l'estat entre totes les accions.

```
q_table = np.zeros((self.state_size, self.action_size))

for episode in range(self.total_episodes):

    state = self.env.reset()
    done = False
    episode_rewards = 0

    while not done:

     state_object = self.env.state_type.build_by_observation(state)
     state_number = state_object.get_state_number()

     exp_exp_tradeoff = random.uniform(0, 1)

     legal_actions = self.env.action_space.available_actions

     if exp_exp_tradeoff > epsilon:    #

       # explotació, cerca a la q-table l'acció legal que té un valor
       # màxim per a l'estat

       self.find_legal_action_w_max_value(state_number, legal_actions)
     else:
      # exploració, tria una acció possible aleatòria
       action = self.env.action_space.sample()

     new_state, reward, done, info = self.env.step(action)

     new_state_object = env.state_type.build_by_observation(new_state)
     new_state_number = new_state_object.get_state_number()
     new_legal_actions = self.env.action_space.available_actions

     if not done:
       # Obté el valor de les noves accions possibles al nou estat
       new_legal_acts_q_values = obtain_val_in_new_stat(new_legal_acts)

       # Aplica l'actualització amb la fòrmula de Bellman. Com a valor
       # del nou estat, considera el valor màxim entre les noves
       # accions possibles.

       self.qtable[state_number, action] =
         self.qtable[state_number, action] +
         self.learning_rate *
           (reward +
            self.discount * np.max(new_legal_acts_q_values) -
            self.qtable[state_number, action])
      else:
        # A l'estat final, el valor del següent estat és 0.

        self.qtable[state_number, action] =
          self.qtable[state_number, action] +
          self.learning_rate * (reward - self.qtable[state_number, action])
```

```
    episode_rewards += reward
    state = new_state
    epsilon = self.min_epsilon +
                (self.max_epsilon - self.min_epsilon) *
                 np.exp(-self.decay_rate * episode)
```

Tot i que un altre dels paràmetres de l'algoritme és el nombre total d'experiments a realitzar, s'ha afegit un sistema de "checkpoints" que guarda l'estat de la q-table cada 1000 experiments. D'aquesta manera, en cas que per qualsevol motiu l'entrenament es vegi aturat abans de finalitzar el total d'experiments, només es perd el coneixement obtingut des del darrer "checkpoint".

```
if episode % 1000 == 0 and episode > 0:

  episodes_batch_id = int(episode / 1000)
  batch_filename = self.filename + "-" + str(episodes_batch_id)

  self.save_qtable(batch_filename)
```

Les q-tables tant dels punts intermedis, com la taula final, es guarden físicament al disc dur. D'aquesta manera, la taula es pot carregar posteriorment pel seu ús per part de l'agent corresponent. A més, en cas que es determini que és necessari continuar amb l'entrenament durant un cert nombre d'experiments més, és possible carregar la taula des del constructor i d'aquesta manera aprofitar el seu contingut per partir des del coneixement de la taula anterior, i d'aquesta manera no haver de començar l'entrenament des de l'inici.

```
if previous_table_filename is not None:

  Q_input = open(previous_table_filename, 'rb')
  self.qtable = pickle.load(Q_input)
  Q_input.close()
```

Determinació dels millors paràmetres

Per tal de trobar la millor versió possible d'un agent que funcioni a partir de l'algoritme de Q-Learning implementat, es poden definir fins a sis hiperparàmetres diferents: la definició d'estat, el ràtio d'aprenentatge, el descompte (o gamma), i els tres paràmetres que determinen l'èpsilon (el ràtio d'exploració). Per realitzar la cerca de la resta de paràmetres, s'han fixat els valors dels paràmetres d'exploració, i només s'han modificat els altres tres.

S'ha començat fent una primera exploració dels resultats obtinguts pels agents generats per una combinació d'una de les definicions d'estats, de dos valors diferents de ràtio d'aprenentatge, i dos valors diferents de descompte. El concepte considerat per determinar la qualitat d'un agent és la millor puntuació mitjana per partida

obtinguda en un interval de 1000 experiments durant l'entrenament. A la taula 4.7 trobem aquests resultats.

ràtio d'aprenentatge	descompte	Puntuació mitjana per experiment amb la definició d'estat GameState	Puntuació mitjana per experiment amb la definició d'estat PlayerSimplState
0.7	0.7	47.83	52.11
0.7	1	47.72	53.36
1	0.7	47.4	50.7
1	1	47.43	50.49

Taula 4.7 Q-Learning. Determinació dels millors paràmetres. Resultats 1

Es pot observar com la diferència dels resultats entre un espai d'estats i l'altre són molt grans, i per tant descartem a partir d'aquest moment el GameState. Pel que fa al ràtio d'aprenentatge, sembla evident que un valor de 0.7 funciona millor que el valor 1, i per tant s'analitzen altres valors propers a 0.7. Pel que fa al descompte, no s'han observat una tendència clara en quin dóna un millor resultat, i per tant es continua l'anàlisi modificant només els valors del ràtio d'aprenentatge:

		descompte	
		0.7	1
ràtio d'aprenentatge	0.5	52.64	53.99
	0.7	52.11	53.36
	0.8	51.28	52.49

Taula 4.8 Q-Learning. Determinació dels millors paràmetres. Resultats 2

En aquest cas sí que s'observa que el valor 1 per al descompte proporciona una millor puntuació mitjana que no pas el de 0.7. També es veu que els valors de ràtio d'aprenentatge més baixos obtenen puntuacions més altes. Per tant, es continua provant diferents valors del ràtio.

ràtio d'aprenentatge	Definició d'estat PlayerSimplState, descompte = 1
0.1	52.63
0.3	54.21
0.4	54.13
0.5	53.99
0.6	53.74
0.7	53.36
0.8	52.49

Taula 4.9 Q-Learning. Determinació dels millors paràmetres. Resultats 3

Es determina que els paràmetres que han donat millor resultat són l'espai d'estats PlayerSimplState, un ràtio d'aprenentatge de 0.3, un descompte d'1.

Implementació de l'agent

Una vegada s'ha generat la q-table que defineix la policy que ha de seguir l'agent, s'implementa un agent que consulta els valors dels parells estat-acció a la taula per determinar l'acció a realitzar a cada pas.

```
def choose_action(self, legal_actions):

  state = self.state_type.build_by_player(self.get_player())
  state_number = state.get_state_number()

  legal_acts_q_values = []
  for action in legal_actions:

    action_num = action.get_action_number()
    legal_acts_q_values .append(self.q_table[state_number,action_num])

  self.last_action_taken = legal_actions[np.argmax(legal_acts_q_values)]

  return self.last_action_taken
```

El mètode per actualitzar la q-table en funció del resultat obtingut per l'acció triada és bàsicament el mateix que s'utilitza durant el procés d'entrenament de l'agent per a cada experiència.

4.5.3 Constructor mitjançant algoritme de tipus Monte Carlo

Definició dels estats

A la secció 2.2.3 s'explica com el mètode utilitzat per guardar els nodes de l'algoritme de tipus Monte Carlo proporcionen una tolerància millor a espais d'estats d'una dimensionalitat més alta que els algoritmes de Q-Learning. Per aquest motiu, es defineixen dos espais d'estat més que proporcionen més informació que els espais definits per a la implementació de l'algoritme Q-Learning

- PlayerSimpleStateV2: Aquest estat, a més de la informació del PlayerSimplState, afegeix el valor màxim de nigiri i de maki que disposem a la mà.

PlayerSimplStateV2

GameState	
Atribut	Valors
Torn	1-10
Ronda	1-3

TableSimplState	
Atribut	Valors
Tempura	0-1
Sashimi	0-2
Gyoza	0-2
Maki	0-1
Pudding	0-1
Palets	0-1
Wasabi	0-1

HandState	
Atribut	Valors
Valor de Maki	0-3
Valor de Nigiri	0-3

Taula 4.10 Atributs de l'espai PlayerSimplStateV2

- PlayerMidStateV2: Aquest estat és similar a l'anterior, però la informació de la taula és més precisa. Per exemple, si a l'estat simple només tenim 3 valors pel nombre de gyoces del que disposem (de 0 a 2, de 3 a 4, o 5 o més), en aquest cas tenim els 6 valors possibles, que ens determinen exactament el nombre de gyoces que hi ha a la taula.

PlayerMidStateV2

GameState	
Atribut	Valors
Torn	1-10
Ronda	1-3

TableState	
Atribut	Valors
Tempura	0-1
Sashimi	0-2
Gyoza	0-5
Maki	0-9
Pudding	0-4
Palets	0-1
Wasabi	0-1

Taula 4.11 Atributs de l'espai PlayeMidStateV2

Detalls de la implementació

Com en el cas de l'algoritme basat en Q-Learning, realitzem la implementació de l'algoritme tipus Monte Carlo a partir de la seva definició al capítol 2.2.3 que introdueix el seu funcionament.

En aquest cas, la gestió del ràtio d'exploració es gestiona a partir del nombre d'experiències que s'han realitzat per a cada acció en un determinat estat. La fase d'exploració determina que sempre que hi hagi accions que mai s'han experimentat a l'estat, es realitzarà una tasca d'exploració d'una d'aquestes accions. Si s'ha explorat totes les accions dins de l'estat, es determina quina d'elles realitzar mitjançant la fórmula (2.15).

Pel que fa al ràtio d'adaptació, per defecte es determina que s'aplica el valor de la fórmula original i per tant totes les experiències tindran el mateix pes. Si s'estableix un valor fixat per aquest ràtio d'adaptació, llavors l'entrenament donarà més pes a les darreres experiències sobre les primeres.

El procediment de guardar coneixement intermedi, el de guardar el diccionari al finalitzar el nombre d'experiments determinat inicialment, i el de carregar el diccionari, és igual al de la implementació de l'algoritme Q-Learning.

```
for episode in range(self.total_episodes):

  state = self.env.reset()
  done = False
  episode_rewards = 0

  episode_log = deque()

  while not done:
    state_object = state_type.build_by_observation(state)
    state_number = state_object.get_state_number()

    state_node = [dict(), 0]

    if state_number not in self.mc_tree:
      self.mc_tree[state_number] = state_node
    else:
      state_node = self.mc_tree[state_number]

    state_action_nodes = state_node[0]
    state_n = state_node[1]

    legal_actions = self.env.action_space.available_actions

    actions_epsilon = []
    actions_values = []

    for legal_action in legal_actions:

      if legal_action in state_action_nodes:

        action_node = state_action_nodes[legal_action]
        action_value = action_node[0]
        action_n = action_node[1]

        a = np.math.log(state_n)
        b = a / action_n
        c = np.math.sqrt(b)

        exploràtion_rate = self.epsilon * c

        # Calcula el ràtio d'exploració amb la fórmula 2.15, i l'afegeix
        # valor de l'acció per realitzar la selecció
        actions_epsilon.append(action_value + exploràtion_rate)

        actions_values.append(action_value)

      else:
        # Si l'acció no ha estat explorada, li assignem epsilon màxim per
        # que es realitzi la expansió
        actions_epsilon.append(MAX_INT)
        actions_values.append(-1*MAX_INT)

        best_epsilons = np.argwhere(actions_epsilon ==
```

```
                                np.amax(actions_epsilon))

      chosen_epsilon = random.choice(best_epsilons)
      action = legal_actions[chosen_epsilon]

    if action not in state_action_nodes:
      state_action_nodes[action] = [0,0]

      new_state, reward, done, info = self.env.step(action)

      episode_rewards += reward
      state = new_state

      if done == True:
        # Quan l'experiència ha finalitzat, retropropagació

        accumulated_reward = 0

        while len(episode_log) > 0:

          game_turn = episode_log.pop()

          state = game_turn[0]
          action = game_turn[1]
          reward = game_turn[2]

          accumulated_reward += reward

          state_node = self.mc_tree[state]
          action_node = state_node[0][action]

          state_node[1] += 1
          action_node[1] += 1

          if self.learning_rate == None:
            alpha = 1 / action_node[1]
          else:
            alpha = self.learning_rate

          action_node[0] += (alpha * (accumulated_reward -
                                      action_node[0]))
```

Determinació dels millors paràmetres

Per determinar els millors paràmetres per aquest algoritme, primer de tot s'obté la definició de l'estat que es comporta millor amb la resta de paràmetres per defecte. El valor d'èpsilon per defecte és 2.1. Una vegada s'ha determinat l'estat que aconsegueix una millor puntuació a les partides, posteriorment es prova amb diferents valors d'èpsilon per determinar quin utilitzar finalment. El valor del ràtio d'adaptació no es

modifica, ja que els agents rivals utilitzats durant l'entrenament no modifiquen el seu comportament amb el temps, i per tant totes les iteracions han de tenir el mateix pes.

Es comença realitzant un agent per a cadascun dels estats a analitzar, i es troba la màxima puntuació mitjana per partida:

	GameState	PlayerSimplState	PlayerSimpleStateV2	PlayerMidStateV2
epsilon = 2.1	56.56	57.02	54.09	55.84

Taula 4.12 Monte Carlo. Determinació dels millors paràmetres. Resultats 1

Dels quatre estats que hem definit, el que es comporta millor és el PlayerSimpleState. Es comprova si cap altre valor d'èpsilon millora l'agent:

epsilon	PlayerSimplState
1	55.2
1.5	55.87
2.1	57.02
2.5	57.13

Taula 4.13 Monte Carlo. Determinació dels millors paràmetres. Resultats 2

S'observa que l'agent amb èpsilon 2.5 es comporta lleugerament millor, i per tant s'utilitza per a la versió definitiva de l'agent implementat amb l'algoritme de tipus Monte Carlo.

Implementació de l'agent

Una vegada es disposa del diccionari amb la policy, s'implementa un agent que, per triar una acció, consulta els valors dels nodes de les accions possibles per a l'estat actual, i selecciona l'acció que té un valor més gran. Si no troba el node de l'estat, triarà una acció aleatòria entre les accions possibles. Si l'estat no disposa d'un valor per a totes les accions, només considera les accions per a les quals disposa d'informació.

El mètode per actualitzar la q-table en funció del resultat obtingut per l'acció triada és el mateix que el procés de retropropagació que es realitza al final de cada experiment. A diferència de l'agent construït per Q-Learning, aquests agents no poden actualitzar els valors a cada pas. Per aquest motiu, l'actualització no es realitza fins que finalitza la partida.

4.5.4 Constructor mitjançant algoritme Deep Q-Learning

Definició dels estats

Un dels avantatges principals de la utilització de l'aprenentatge profund per predir el valor que ens proporciona una acció en un determinat estat és que permet definir un espai d'estats d'una dimensionalitat molt més alta que els algoritmes que hem utilitzat fins ara. L'espai d'estat CompleteState conté dades de la mà del jugador, del torn i ronda en la qual ens trobem i del nombre exacte de cartes de cada tipus que tenim en la taula. Aquestes dades les obté mitjançant l'espai d'estats PlayerFullState. A més a més, proveeix de la informació de les cartes que té cadascun de la resta de jugadors a la seva taula, en un llistat d'espais TableFullState. Per tal d'assegurar que la posició del jugador no afecta l'ordre dels atributs de l'estat en un determinat moment de la partida, els TableFullState s'afegeixen en ordre partint del jugador de la dreta del qual s'està construint l'estat, i continuant amb la resta de jugadors fins a acabar amb el que es troba a l'esquerra del jugador.

PlayerFullState

GameState	
Atribut	Valors
Torn	1-10
Ronda	1-3

TableFullState	
Atribut	Valors
Tempura	0-1
Sashimi	0-2
Gyoza	0-5
Maki	0-29
Pudding	0-9
Palets	0-5
Wasabi	0-5

HandState	
Atribut	Valors
Valor de Maki	0-3
Valor de Nigiri	0-3

Taula 4.14 Atributs de l'espai PlayerFullState

CompleteState

PlayerFullState	**TableFullState**	**TableFullState**	…
	Següent a la dreta del jugador	Dues posicions a la dreta del jugador	

Taula 4.15 Atributs de l'espai CompleteState

Estandardització de les dades

Als algoritmes implementats fins al moment, s'utilitzava un identificador numèric que definia l'estat, i servia per determinar la línia de la q-table en el cas de l'algoritme basat en Q-Learning, o per obtenir el node pertanyent a l'estat al diccionari de nodes a l'algoritme tipus Monte Carlo. En canvi, en el cas dels algoritmes de Deep Q-Learning, com a entrada de les xarxes neuronals que determinaran el valor esperat per a cada acció, es proporcionen directament a la capa d'entrada de la xarxa el llistat d'atributs que formen l'estat.

Si s'analitzen els diferents atributs que formen part de l'estat CompleteState, es pot veure que l'escala pot variar molt d'un atribut a un altre. Mentre que, per exemple, el nombre de tempures lliures en un determinat moment a la taula d'un jugador pot tenir només dos valors possibles 0 o 1, el nombre total de maki rolls pot anar des de 0 fins a 29. Si s'utilitzen aquests atributs directament sense un preprocessament previ, la diferència d'escala dels atributs produiria que tinguessin un pes més gran aquells que tenen un valor mitjà major. Per tal d'igualar el pes de tots els atributs que formen l'estat, s'ha de realitzar un procés d'estandardització. Un altre avantatge del procés d'estandardització de les dades és que genera valors d'entrada tant positius com negatius. Aquest fet facilita el procés de convergència de la xarxa neuronal.

El procés d'estandardització parteix del conjunt de dades d'entrenament, o almenys d'un conjunt significatiu. En el cas de l'aprenentatge per reforç, les dades es van generant en el procés de repetició d'experiments, i per tant inicialment no existeixen encara les dades necessàries. Per disposar d'un conjunt de dades inicial, es comença el procés només obtenint i guardant les experiències realitzades als primers experiments, fins a arribar un nombre determinat. A partir dels estats obtinguts a les primeres experiències es poden generar els paràmetres necessaris per a l'estandardització de cada atribut. Una vegada els paràmetres estan disponibles, s'han d'aplicar tant als estats de les experiències inicials com a la resta d'estats que es trobin en el futur durant el procés de creació, i posteriorment en l'ús de l'agent.

Una altra problemàtica específica d'aquest projecte pel que fa a l'estandardització dels estats és que, per gran part de les dades, la distribució no segueix una forma normal, sinó una distribució de Poisson. Això fa que hi hagi una variància molt petita en valors menors de la mitjana, peró molt gran en valors més grans. Si s'estandarditzen aquestes dades sense realitzar cap transformació prèvia, les diferències entre els valors menors de la mitjana serien molt petites, mentre que per als valors més grans serien encara més grans. Aquesta distribució afecta molt negativament a la capacitat de predicció de la xarxa neuronal. Per exemple, la diferència entre tenir un Maki o quatre Makis a la taula d'un jugador seria mínima i per tant pràcticament indistingible, quan en realitat pot suposar una diferència important de punts al final de la ronda. En canvi, la diferència entre tenir 21 o 22 Makis seria molt important, quan en realitat aquesta diferència no aporta cap informació, ja que en ambdós casos la puntuació màxima per Makis està assegurada. Per tal de modificar la distribució d'aquests atributs, abans d'estandarditzar-los s'aplica una transformació de Box-Cox [15].

Com que els paràmetres d'estandardització seran necessaris per a l'agent en el moment de determinar l'acció a realitzar, en finalitzar el procés d'entrenament s'hauran de guardar, juntament amb la xarxa neuronal que determina la policy de l'agent, perquè l'agent les pugui carregar quan sigui inicialitzat.

Estructura de la xarxa neuronal

Per tal de determinar l'estructura de la xarxa neuronal a utilitzar, s'ha seguit un procés iteratiu de cerca per trobar el nombre de capes ocultes necessàries per aconseguir que el sistema convergeixi. S'ha començat amb una única capa oculta i valors baixos per al nombre de nodes de cada capa, però no s'ha aconseguit la convergència del sistema i que el valor de la pèrdua s'anés reduint amb les diferents iteracions de l'entrenament fins que no es va utilitzar una xarxa amb 256 nodes a la capa oculta.

```
def __define_model(self):

  model = Sequential()

  model.add(Dense(256, input_dim=self.__state_size, activation='relu'))
  model.add(Dense(self.__action_size, activation='linear'))
  model.compile(loss='mse', optimizer='adam')

  return model
```

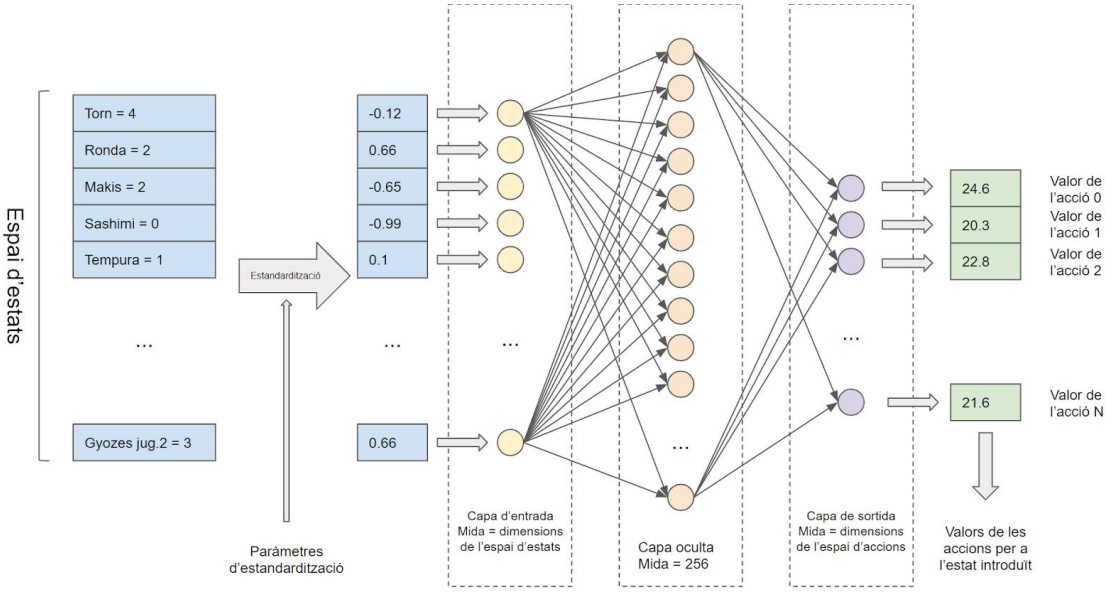


Figura 4.16 Estructura de la xarxa neuronal utilitzada a la implementació de Deep Q-Learning

Memòria d'experiments

Tal com s'indica al capítol introductori dels algoritmes de Deep Q-Learning (2.2.2), aquests utilitzen una memòria d'experiències en comptes de processar el resultat obtingut per cadascuna d'elles de forma directa. Per la implementació realitzada per aquest projecte, la memòria té una capacitat màxima de 1024 experiments. D'altra banda, cada cop que s'actualitza la xarxa neuronal amb les experiències guardades a la memòria, es realitza l'actualització de tot un paquet d'experiències. La mida del paquet d'experiències a la implementació és de 256.

S'ha afegit una funcionalitat addicional a la memòria d'experiments per a la implementació del projecte. A la secció d'estandarditzación s'estableix que és necessari disposar d'una mostra inicial d'experiments que permetin determinar els paràmetres necessaris per realitzar l'estandarditzación de la resta del procés. L'estructura de la memòria permet acumular aquests experiments inicials fins al punt de tenir la memòria plena, Arribat aquest punt, es pot realitzar la primera estandarditzación i començar amb el procés d'entrenament a partir d'aquest instant amb els paràmetres d'estandarditzación obtinguts i les experiències acumulades.

Xarxa neuronal de valors objectiu

Es determina que la xarxa neuronal de valors objectiu s'actualitza a la implementació cada 100 experiments.

Determinació dels millors paràmetres

Dels quatre paràmetres que determinen el comportament de l'algoritme, es decideix fixar-ne els tres que determinen l'epsilon, i realitzar una comparació de tres valors diferents de l'altre paràmetre, el descompte. Pel que fa a la definició de l'estat a utilitzar, s'utilitzarà només el CompleteState.

Es realitza un agent per a cadascun dels valors de descompte a analitzar, i trobem la puntuació mitjana de les partides màxima a la taula 4.17

	descompte= 0.3	descompte = 0.7	descompte = 1
epsilon = 2.1	50.36	53.42	58.32

Taula 4.17 Deep Q-Learning. Determinació dels millors paràmetres. Resultats

Es comprova que en aquest cas un descompte més gran ha donat un millor resultat. Per tant, la resta d'agents que realitzarem amb aquest algoritme els farem amb un valor de descompte igual a 1.

Implementació de l'agent

La implementació de l'agent comença amb la càrrega, en el procés d'inicialització, de la xarxa neuronal que conté la policy que s'ha generat mitjançant l'algoritme Deep

Q-Learning. A més a més, també s'han de carregar els paràmetres d'estandarditzación dels atributs dels estats.

Per determinar l'acció que realitza l'agent en un estat, el procés comença estandarditzant els atributs de l'estat actual. Posteriorment, s'estima a la xarxa neuronal el valor de totes les accions. De totes les accions que són possibles, es tria aquella per a la qual la xarxa ha estimat un valor més gran.

L'aprenentatge durant el procés d'utilització de l'agent, tot i que és possible i està implementada, és força ineficient, ja que en comptes d'actualitzar la xarxa amb un paquet de valors objectiu, realitza una actualització a cada acció realitzada.

4.5.5 Implementació de la millora Double Deep Q-Learning

Per a la realització de la millora Double Deep Q-Learning, es parteix de la implementació del constructor d'agents de tipus Deep Q-Learing, ja que el procediment és molt similar. En aquest cas, es modifica el mètode que actualitza la xarxa neuronal per tal que la millor acció en el següent estat es determini a partir de la xarxa de valors objectiu, però el valor de prendre aquesta acció es consideri de la xarxa principal, tal com es determina en la definició de la millora de l'algoritme (2.2.2.1).

Determinació dels millors paràmetres

Per tal de comparar l'algoritme Deep Q-Learning amb la seva possible millora Double Deep Q-Learning en les mateixes condicions, s'ha decidit utilitzar els mateixos paràmetres per realitzar els dos agents.

Implementació de l'agent

La implementació dels agents per a la millora Double Deep Q-Learning és exactament la mateixa que per als agents Q-Learning. Per tant, s'utilitza la mateixa implementació per als agents de tots dos algoritmes.

5. Comparativa d'agents i anàlisi de comportament

5.1 Comparativa de resultats de partides agent vs. agent

Tal com s'explica a l'apartat 4.5.1, s'utilitza cada algoritme d'aprenentatge per reforç per generar quatre agents diferents. Cadascun d'aquests quatre agents es diferencia de la resta pels rivals contra els quals s'ha enfrontat en els experiments que han servit per ser entrenats. En funció d'aquests rivals, els agents pertanyen a una fase entre la 1 i la 4. Per tant, com que tenim quatre algoritmes diferents, i quatre fases, obtenim els agents llistats a la taula 5.1. A aquesta mateixa taula definim els acrònims amb els quals ens referirem a partir d'aquest moment a cadascun dels agents generats.

Acrònim	Agent	Acrònim	Agent
QL1	QLearning Fase : 1	DQL1	Deep Q-Learning Fase : 1
QL2	QLearning Fase : 2	DQL2	Deep Q-Learning Fase : 2
QL3	QLearning Fase : 3	DQL3	Deep Q-Learning Fase : 3
QL4	QLearning Fase : 4	DQL4	Deep Q-Learning Fase : 4
MC1	Monte Carlo Fase : 1	DDQL1	Double Deep Q-Learning Fase : 1
MC2	Monte Carlo Fase : 2	DDQL2	Double Deep Q-Learning Fase : 2
MC3	Monte Carlo Fase : 3	DDQL3	Double Deep Q-Learning Fase : 3
MC4	Monte Carlo Fase : 4	DDQL4	Double Deep Q-Learning Fase : 4

Taula 5.1 Agents generats per la comparativa

Es realitza la simulació de 2000 partides per cada parell dels agents generats. Per fer la comparació, primer s'analitzen els resultats per cadascuna de les fases, i posteriorment una comparació general de tots els agents. Tot i que el projecte s'ha

centrat en la maximització del nombre de punts, i en cap moment s'ha tingut en compte la victòria o derrota com a factor de recompensa final, també es comparen els percentatges de victòries obtinguts a les partides, a més a més de la mitjana total de puntuació obtinguda. Considerarem que un algoritme és clarament millor que un altre pel que fa a victòries quan el seu percentatge supera el 55%. Quan el percentatge de victòries entre dos agents es troba entre el 45 i el 55 per cent, llavors considerarem que els seus resultats són prou similars per poder considerar que un dels agents està clarament per sobre de l'altre.

	QL1		MC1		DQL1		DDQL1	
	Win%	Rew.	Win%	Rew.	Win%	Rew.	Win%	Rew.
QL 1	50,88	51,17	49,25	51,24	72,63	54,86	67,45	55,36
MC 1	51,00	51,62	50,50	50,02	69,73	55,60	66,75	54,87
DQL 1	28,18	49,54	31,40	50,53	50,45	52,85	44,55	53,47
DDQL 1	33,03	51,33	33,65	51,26	56,05	54,94	50,25	54,83
QL 2	43,25	50,28	51,50	51,79	68,58	55,22	62,05	55,29
MC 2	43,63	50,07	42,50	50,08	63,85	54,06	60,85	54,21
DQL 2	32,85	51,24	34,95	51,94	56,30	54,54	53,80	55,25
DDQL 2	37,13	51,98	39,05	52,37	54,60	54,92	52,85	55,24
QL 3	44,75	50,66	48,13	51,99	65,20	54,52	62,15	54,79
MC 3	41,13	51,61	44,25	51,56	62,23	55,59	59,05	54,69
DQL 3	35,58	51,37	34,55	51,98	56,00	55,51	48,75	55,57
DDQL 3	31,00	51,63	34,75	52,25	58,40	54,96	47,95	55,01
QL 4	55,00	50,36	58,13	52,16	73,05	55,00	68,63	54,84
MC 4	49,75	52,29	53,38	51,38	66,48	54,95	64,15	54,49
DQL 4	35,75	52,09	34,08	52,01	54,65	54,93	50,15	55,32
DDQL 4	38,45	52,57	42,75	52,87	58,20	55,55	54,05	55,16
Mitjana	40,71	51,24	42,68	51,59	61,65	54,87	57,09	54,90

Taula 5.2 Comparativa dels agents de fase 1

A aquesta primera taula comparativa que, com a recordatori, compara els algoritmes creats mitjançant l'entrenament contra l'agent RandomAgent, ja s'observa clarament que els algoritmes basats en Q-Learning i en el mètode Monte Carlo tenen un rendiment molt més baix, tant en puntuació com en nombre de victòries, que aquells que utilitzen aprenentatge profund. Tant el QL1 com el MC1 només han estat capaços

de vèncer a un dels agents, el QL4. Si es comparen entre ells, l'algoritme MC1 obté un resultat lleugerament superior al QL1 quant a puntuació, i percentatge de victòries. Pel que fa als agents DQL1 i DDQL1, aconsegueixen un rendiment molt bo tant en victòries, com en puntuació, concepte en el qual obtenen una mitjana gairebé idèntica. Pel que fa al nombre de victòries, sí que destaca especialment l'algoritme DQL1, que aconsegueix guanyar clarament a la pràctica totalitat d'agents, a excepció dels DDQL2 i DQL4, amb els quals es queda a unes dècimes de poder considerar el seu domini superior.

	QL2		MC2		DQL2		DDQL2	
	Win%	Rew.	Win%	Rew.	Win%	Rew.	Win%	Rew.
QL 1	58,25	51,81	56,88	51,49	67,65	55,40	63,18	54,93
MC 1	49,25	51,65	58,13	52,09	65,55	55,82	61,45	54,88
DQL 1	32,35	50,98	36,80	50,56	44,10	52,81	45,85	53,58
DDQL 1	38,48	52,43	39,88	52,04	46,75	54,74	47,35	54,74
QL 2	50,88	50,79	57,63	52,42	54,35	54,55	60,45	55,01
MC 2	42,75	50,67	50,50	50,15	60,20	55,08	56,78	54,40
DQL 2	46,13	53,19	40,60	52,49	50,10	54,07	49,90	54,80
DDQL 2	40,00	52,81	43,58	53,12	50,40	55,13	50,10	54,58
QL 3	46,38	50,73	52,63	51,70	62,05	55,16	61,80	55,06
MC 3	45,25	52,81	52,75	52,75	58,68	55,03	58,48	55,14
DQL 3	37,83	51,78	39,40	52,02	47,05	55,28	50,20	55,58
DDQL 3	38,30	52,39	36,90	52,40	50,45	54,87	49,45	54,67
QL 4	54,00	49,93	63,13	51,92	66,90	55,15	62,95	54,20
MC 4	50,38	52,13	59,13	51,91	66,18	55,49	61,88	54,55
DQL 4	40,38	53,12	37,55	52,31	47,40	55,19	48,55	54,72
DDQL 4	40,83	53,06	45,93	52,92	56,25	55,58	54,75	55,44
Mitjana	44,46	51,89	48,21	52,02	55,88	54,96	55,19	54,76

Taula 5.3 Comparativa dels agents de fase 2

En aquesta segona taula es comparen els agents entrenats realitzant partides contra altres agents creats manualment que utilitzen una estratègia bàsica. Els agents QL2 i MC2 demostren una millora d'aproximadament mig punt en la mitjana de puntuacions obtingudes, que es tradueix en una pujada significativa en el nombre de partides guanyades, tot i que encara no arriben al 50% de victòries. De tota manera, la

diversitat d'estratègies dels seus rivals durant l'entrenament els ha permès donar un millor rendiment quan els hem enfrontat amb altres agents generats automàticament. Pel que fa als agents d'aprenentatge profund, en canvi, aquesta diversitat de contraris no els ha permès jugar millor que les seves versions de fase 1. El DQL2 només ha augmentat la seva puntuació mitjana molt lleugerament respecte al DQL1, mentre que el DDQL2 ha perdut algunes dècimes respecte al DDQL1. El descens del rendiment és més evident si observem el percentatge de victòries: el DQL2 passa de guanyar el 61.65% de les partides, al 55.88%, tot i haver augmentat lleugerament la puntuació. El DDQL2 redueix en menor mida el nombre de victòries, tot i la baixada de puntuació.

	QL3		MC3		DQL3		DDQL3	
	Win%	Rew.	Win%	Rew.	Win%	Rew.	Win%	Rew.
QL 1	56,25	52,33	59,50	53,86	64,88	54,64	69,53	55,73
MC 1	52,38	52,63	56,75	52,89	66,43	55,93	65,85	55,72
DQL 1	35,55	51,09	38,93	52,33	44,55	53,74	42,35	52,84
DDQL 1	38,78	52,02	41,50	52,69	52,05	55,72	52,90	55,28
QL 2	54,38	52,10	55,75	53,99	62,70	54,52	62,50	54,79
MC 2	48,00	51,32	48,88	52,21	60,95	54,40	63,70	55,20
DQL 2	38,55	52,43	42,65	53,29	53,20	55,64	50,00	54,82
DDQL 2	38,83	52,65	42,70	52,99	50,65	55,41	51,10	54,65
QL 3	50,38	50,58	55,25	54,04	63,45	54,59	58,20	54,92
MC 3	45,75	53,18	50,50	51,17	61,85	56,00	56,33	54,48
DQL 3	37,30	51,74	38,85	52,86	50,65	54,72	52,05	55,13
DDQL 3	42,40	52,94	43,83	52,82	49,20	55,15	50,55	53,31
QL 4	62,38	51,63	62,00	52,26	65,00	54,19	63,23	54,04
MC 4	52,13	52,42	52,25	52,65	66,83	55,80	63,95	55,15
DQL 4	37,45	52,36	38,88	53,02	46,00	54,94	49,75	54,80
DDQL 4	46,43	53,45	45,15	53,22	58,85	56,42	55,45	54,99
Mitjana	46,06	52,18	48,33	52,89	57,33	55,11	56,71	54,74

Taula 5.4 Comparativa dels agents de fase 3

A la taula 5.4 s'han comparat amb la resta els agents que s'han generat enfrontant-se amb els agents de les fases 1 i 2. Observant la puntuació mitjana de les partides, es detecta un lleuger increment general respecte als resultats obtinguts pels agents de les fases 1 i 2. Aquest increment es produeix sobretot a l'algoritme MC3. Pel que fa al

nombre de victòries, també es produeix un petit increment respecte als agents de la fase 2. Cal destacar, tanmateix, que en aquest aspecte els algoritmes DQL3 i DDQL3 no arriben als percentatges obtinguts per les seves versions de fase 1, sobretot per l'agent DQL1 que es manté com el més fiable en nombre de victòries, tot i tenir una menor puntuació mitjana.

	QL4		MC4		DQL4		DDQL4	
	Win%	Rew.	Win%	Rew.	Win%	Rew.	Win%	Rew.
QL 1	45,75	48,80	50,88	52,37	64,93	55,43	62,05	55,12
MC 1	43,00	50,28	48,00	50,94	66,38	55,88	58,00	54,78
DQL 1	28,28	49,38	34,05	51,02	45,80	53,85	42,35	53,82
DDQL 1	31,73	50,45	37,13	51,27	50,50	55,43	46,25	54,20
QL 2	47,25	49,63	50,50	51,69	60,10	55,48	59,55	54,95
MC 2	37,25	49,00	41,63	50,04	62,78	55,04	54,53	53,79
DQL 2	33,53	51,23	34,58	51,59	52,85	55,90	44,15	54,34
DDQL 2	37,48	51,48	38,58	51,94	51,70	55,49	45,65	54,52
QL 3	39,00	49,12	49,00	51,99	62,90	55,11	54,50	54,25
MC 3	39,38	49,59	48,25	51,79	61,53	55,77	55,98	54,58
DQL 3	35,55	50,76	33,70	52,35	54,90	56,01	41,90	54,38
DDQL 3	37,10	51,22	36,75	52,52	50,70	55,39	45,15	54,12
QL 4	50,38	47,37	61,25	52,46	68,25	55,31	67,88	55,42
MC 4	39,63	49,92	50,63	50,85	66,53	55,53	58,93	54,38
DQL 4	32,03	51,12	34,00	51,73	50,05	55,34	41,85	54,25
DDQL 4	32,48	51,23	41,43	52,10	58,85	55,99	50,50	54,64
Mitjana	38,11	50,03	43,15	51,66	58,05	55,43	51,83	54,47

Taula 5.5 Comparativa dels agents de fase 4

Finalment, a la taula 5.5 s'han comparat amb la resta els agents que s'han generat enfrontant-se amb tots els agents creats anteriorment, sigui de forma manual, sigui de forma automàtica a les fases 1, 2 i 3. Per tant, són els que, en teoria, haurien de tenir una estratègia més versàtil, i per tant obtenir un millor resultat general. Comprovem que, tret per l'agent DQL4, això no ha estat així. Els agents QL4, MC4 i DDQL4 han obtingut pitjor puntuació mitjana que les seves versions anteriors. En canvi, l'agent DQL4 sí que ha augmentat la seva puntuació, sent l'agent amb millor puntuació mitjana de tota la comparativa. No obstant això, no ha servit perquè el seu percentatge de

victòries superi al de l'algoritme DQL1, tot i que la seva puntuació mitjana és superior en més de mig punt.

A continuació es realitza la classificació general de tots els agents, tant per nombre de victòries com per mitjana de puntuació per partida.

Classificació per percentatge de victòries	
Agent	Percentatge
DQL1	61,65
DQL4	58,05
DQL3	57,33
DDQL1	57,09
DDQL3	56,71
DQL2	55,88
DDQL2	55,19
DDQL4	51,83
MC3	48,33
MC2	48,21
QL3	46,06
QL2	44,46
MC4	43,15
MC1	42,68
QL1	40,71
QL4	38,11

Classificació per puntuació mitjana	
Agent	Puntuació mitjana
DQL4	55,43
DQL3	55,11
DQL2	54,96
DDQL1	54,90
DQL1	54,87
DDQL2	54,76
DDQL3	54,74
DDQL4	54,47
MC3	52,89
QL3	52,18
MC2	52,02
QL2	51,89
MC4	51,66
MC1	51,59
QL1	51,24
QL4	50,03

Taula 5.6 Classificació per percentatge de victòries
Taula 5.7 Classificació per puntuació mitjana

En aquestes taules es poden observar més clarament algunes de les conclusions comentades prèviament. La més clara de totes és que els algoritmes basats en l'aprenentatge profund obtenen millors resultats que els altres dos.

Comparant només els agents DQL i DDQL entre ells, sense tenir en compte la fase, determinem que els agents DQL superen als agents DDQL. La teoria indicava que els agents DDQL haurien d'implicar una millora en l'aprenentatge sobre els DQL, però els resultats ens han indicat el contrari. Els motius d'aquest resultat inesperat poden ser diversos, com per exemple el fet que no s'han determinat els hiperparàmetres que

donen millor resultat específicament per la construcció dels agents DDQL. Si s'hagués realitzat un estudi per determinar els millors valors per al descompte, i sobretot pel nombre d'experiències a realitzar entre les actualitzacions de la xarxa de valors objectiu, és probable que haguessin aconseguit millors resultats dels agents DDQL, que possiblement superarien la puntuació mitjana de la resta d'agents.

Entre els algoritmes MC i QL, els que millor resultats obté tant en puntuació com en percentatge són els agents MC. La possibilitat d'utilitzar un espai d'estats més ampli que els algoritmes de QL ha estat un factor determinant.

Per comparar els resultats per fases, es fa la distinció entre els algoritmes que utilitzen aprenentatge profund, i els que no. Amb els agents QL i MC, la fase amb més puntuació mitjana i percentatge de victòries és la 3, seguida per la 2.

Pel que fa als agents DQL, si es considera la puntuació mitjana, el seu rendiment ha augmentat de forma gradual a cadascuna de les fases realitzades. No obstant això, l'agent DQL que ha funcionat millor en percentatge de victòries és precisament el que té una puntuació mitjana més baixa, el DQL1.

Finalment, respecte a la puntuació obtinguda als agents DDQL, curiosament s'ha produït la progressió contrària: l'agent creat a la fase 1 ha aconseguit la màxima puntuació entre els agents DDQL, i el seu rendiment ha baixat unes dècimes de punt a cada fase.

5.2 Anàlisi de comportament dels agents

Per tal de disposar de les dades necessàries per realitzar l'anàlisi de comportament dels agents, al llarg de totes les partides s'ha recopilat un dietari amb totes les accions que ha realitzat cada agent, i totes les accions que tenia disponibles. D'aquesta manera, es pot determinar l'ús que s'ha fet de cadascuna de les accions en funció dels torns en els quals era possible realitzar-la.

La taula 5.8 indica, per cadascun dels agents creats, el percentatge de cops que cadascuna de les accions ha estat utilitzada quan es trobava disponible. Les accions dobles de palets a les quals s'han utilitzat dues cartes s'han tingut en compte en el càlcul del percentatge. La taula presenta els agents en l'ordre de la puntuació mitjana obtinguda a la comparativa.

Agent	Palets	Nigiri	Wasabi	Maki	Sashimi	Tempura	Gyoza	Puding
DQL 4	31.40	41.43	63.98	23.82	27.08	34.70	38.33	21.65
DQL 3	18.34	42.48	52.67	26.91	41.78	46.19	17.16	23.68
DQL 2	27.81	43.50	67.64	26.47	28.63	31.91	32.60	16.73
DDQL 1	28.03	43.39	68.53	25.24	30.19	27.55	41.86	22.37
DQL 1	29.89	39.83	62.24	27.92	31.06	39.97	26.79	17.86
DDQL 2	16.95	43.38	49.66	26.90	29.04	29.35	43.12	22.67
DDQL 3	21.57	37.60	50.48	28.04	20.00	34.91	48.28	25.74
DDQL 4	45.61	33.13	53.12	23.23	31.58	24.98	50.29	20.57
MC 3	17.69	39.02	47.98	28.25	24.42	31.33	51.51	22.06
QL 3	13.38	52.68	46.98	30.40	38.03	37.98	22.33	16.29
MC 2	19.82	48.09	53.00	24.27	46.56	36.97	27.46	18.95
QL 2	14.96	56.18	49.15	27.61	34.03	50.32	19.12	16.39
MC 4	20.26	44.41	61.27	26.36	36.11	30.76	39.07	18.14
MC 1	19.35	37.58	54.27	25.63	43.15	41.62	33.37	18.34
QL 1	16.13	50.41	43.41	28.19	44.04	41.78	23.41	14.62
QL 4	14.35	53.17	21.22	28.62	33.21	56.68	25.00	16.01

Taula 5.8 Percentatge d'ús de cada tipus de carta en els agents

Analitzant la taula [taula], es poden extreure les següents conclusions:

- La carta Wasabi és amb diferència la carta més utilitzada per part dels agents que han tingut una millor puntuació. És la carta que pot arribar a proporcionar la puntuació més alta per una sola carta de tot el joc, fins a 6 punts, si es combina amb un Nigiri de 3 punts.
- La segona carta més utilitzada per tots els agents són els Nigiri. Són les cartes que proporcionen una recompensa més segura, encara que altres combinacions de cartes puguin arribar a donar una puntuació per carta més gran, la recompensa dels Nigiri és immediata. També la combinació que realitza amb el Wasabi li afegeix un valor addicional a aquestes cartes, i per tant més probabilitat per ser utilitzades.
- Entre les cartes menys utilitzades, trobem els Maki i els Puding, especialment aquests últims. Són cartes que no reben un resultat immediat, sinó que no proporcionen punts fins al final de la ronda, o de la partida. Per tant, és una recompensa molt més incerta que la de les cartes més utilitzades. De tota manera, hi ha un altre factor que ha estat molt decisiu per determinar el poc ús d'aquestes cartes, i és la decisió de què el factor a maximitzar en l'entrenament dels agents ha estat la puntuació final de l'agent, sense tenir en compte la victòria o no. El valor d'utilitzar les cartes de tipus Maki i Puding no ve donat solament per la puntuació que proporcionen, sinó també per evitar que els rivals obtinguin aquests punts que proporcionen. Com que la puntuació dels rivals no s'ha considerat de cap manera en l'entrenament dels agents, aquestes cartes han perdut part del seu valor real de cara a obtenir la victòria a la partida.
- Els palets també estan molt infrautilitzats pels agents. Aquesta dada es pot atribuir a la dificultat que probablement han tingut durant l'entrenament per determinar el valor de les accions dobles. Aquestes accions es troben disponibles en només una petita proporció de les experiències realitzades. Això fa que la seva exploració sigui molt menys freqüent, i per tant probablement en la major part dels casos no ha convergit al seu valor real. Com que el valor d'utilitzar la carta Palets ve donat directament per la possibilitat posterior d'utilitzar una acció doble, el fet de no disposar del valor òptim d'aquestes accions perjudica les opcions d'utilitzar aquesta carta.

A continuació s'analitzen amb més detall el funcionament de dos dels millors agents. D'una banda s'estudia, per cada torn, el percentatge d'utilització de cada acció que ha tingut l'agent que ha obtingut una millor puntuació mitjana. A continuació, es realitza el mateix estudi per a l'agent que ha obtingut el nombre més gran de victòries, tot i tenir una puntuació mitjana menor.

Torn	Palets	Nigiri	Wasabi	Maki	Sashimi	Tempura	Gyoza	Puding
1	52.64	19.23	98.37	0.00	12.04	4.26	11.28	0.86
2	13.72	45.52	30.83	0.04	26.82	19.40	27.41	1.75
3	19.95	36.97	41.52	0.30	23.47	42.02	33.98	3.30
4	16.37	38.75	36.83	0.99	23.16	54.43	43.74	10.19
5	13.05	51.30	23.73	3.71	23.81	56.84	55.44	22.12
6	19.13	60.76	29.75	11.45	26.67	56.09	68.99	39.95
7	23.04	66.56	28.70	30.40	36.01	54.76	77.58	49.41
8	23.83	65.94	32.15	58.81	42.46	45.92	82.73	53.56
9	33.86	66.78	33.18	79.49	60.19	43.56	87.35	60.58
10	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Taula 5.9 Percentatge de cops que una carta ha estat utilitzada a cada torn per a l'agent que ha tingut millor puntuació mitjana, el DQL4.

Torn	Palets	Nigiri	Wasabi	Maki	Sashimi	Tempura	Gyoza	Puding
1	36.01	15.96	97.88	0.03	26.53	11.80	0.89	0.00
2	29.23	45.06	34.68	2.01	37.54	24.35	4.82	1.61
3	23.34	42.82	39.17	1.50	34.55	44.19	14.20	4.39
4	23.47	39.64	31.99	3.46	31.72	59.13	29.11	8.33
5	23.83	45.05	18.06	8.77	23.09	66.02	48.08	15.39
6	17.93	50.83	17.39	22.73	17.89	66.77	63.93	27.22
7	14.33	53.52	14.96	44.30	14.71	58.81	72.61	35.60
8	12.25	61.15	20.54	66.38	18.26	49.49	76.86	42.90
9	23.19	76.62	41.00	80.89	40.22	51.05	83.49	61.21
10	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Taula 5.10 Percentatge de cops que una carta ha estat utilitzada a cada torn per a l'agent que ha tingut millor percentatge de victòries, el DQL1.

A les taules 5.9 i 5.10 es pot observar de forma clara l'estratègia que segueixen ambdós agents, que és molt similar. Al torn inicial, pràcticament sempre que tenen oportunitat, utilitzen un Wasabi. Com s'ha comentat prèviament, és la carta que pot proporcionar una recompensa més gran per si mateixa, però necessita d'un Nigiri del valor més alt possible a un torn posterior. Per aquest motiu, després del torn inicial, la utilització del Wasabi decreix molt, i sobretot a la segona meitat de la ronda es converteix en una de les cartes menys usades, ja que si no s'aconsegueix acoplar cap Nigiri, no proporciona cap punt.

Aquesta relació entre els Wasabi i els Nigiri és la causant en part de què els Nigiri siguin la carta més utilitzada al segon torn. L'altra causa és el fet que als primers torns els Nigiri disponibles és probable que siguin els de més valor, mentre que als darrers torns només quedin a la mà Nigiris d'un punt. De tota manera, la immediatesa dels punts que proporcionen els Nigiris els converteix en una de les cartes prioritàries al llarg de tota la ronda.

Entre el tercer i el cinquè o sisè torn, a la fase mitjana de la ronda, la carta més utilitzada és la Tempura. Sembla que els 2.5 punts per carta que proporciona aquest tipus de carta compensen el risc de no aconseguir completar la parella. Aquest risc és molt baix quan la ronda es troba encara a la primera meitat, i queden encara més de 5 cartes per triar. Als darrers torns l'ús de la Tempura decreix, ja que s'utilitzarà només si a la taula hi ha una Tempura pendent de la seva parella, però en el cas contrari és probable que no es pogués completar la parella als pocs torns restants, i per tant la carta no proporcionaria cap punt.

A partir del sisè torn, la carta que pren la prioritat màxima per als dos agents és la Gyoza. En aquests darrers torns, els agents han d'utilitzar una estratègia més conservadora que els faci acumular tants punts de forma segura com sigui possible. Com que a més a més en aquesta fase de la ronda els Nigiri restants seran els de puntuació més baixa, que proporcionen només 1 punt per carta, la millor opció de recompensa directa és la Gyoza, que en el pitjor cas iguala aquest resultat, i si s'acumula més d'una el pot fàcilment duplicar, o fins i tot triplicar en el millor cas.

Cap al final de la ronda, els Maki també augmenten molt les probabilitats de ser utilitzats. La proximitat del final de la ronda i per tant del repartiment dels punts per Makis poden influir en aquesta política.

Pel que fa a la resta de cartes, veiem que els Palets només tenen cert ús a l'inici de la ronda, al primer torn, quan encara queden moltes cartes per jugar, i l'opció de realitzar una jugada doble al segon o tercer torn pot resultar molt beneficiosa. Als darrers torns, però, els Palets resulten totalment inservibles i no proporcionarien cap mena de valor ni puntuació.

Els Pudding no tenen gairebé utilitat a cap dels dos agents. Només a la segona meitat de la ronda s'incrementa una mica la seva probabilitat de ser triats, quan a la mà només hi ha altres cartes com els Palets o el Wasabi que no proporcionen cap valor en aquest punt de la ronda.

Finalment, les cartes Sashimi són del tipus menys utilitzat per ambdós agents, però no obstant això, la seva política de tria és la que provoca que l'agent DQL1 obtingui més

victòries que l'agent DQL4, encara que el DQL4 obtingui puntuacions més altes. Es pot observar com l'agent DQL4 pràcticament no tria mai la carta Sashimi. Gràcies a això, pot concentrar els seus esforços en combinar la resta de cartes de forma òptima, i obtenir-ne una millor puntuació, sense el risc que suposa tenir cartes de Sashimi a la taula i no aconseguir completar el trio per treure'n profit. L'agent DQL1, en canvi, a l'inici de la partida arrisca utilitzant Sashimi, sent aquesta la tercera carta més utilitzada als primers dos torns. Aquest risc li suposa, quan no completa la tripleta, perdre punts respecte a l'algoritme DQL4. Com a contrapartida, l'estrategia del DQL4 té un inconvenient important: com que renuncia a tots els Sashimi, el seu rival els pot acumular més fàcilment, i té més probabilitats d'acumular un trio de Sashimi, que dóna una puntuació de 3.33 punts per carta. En canvi, per als rivals del DQL1, serà més difícil obtenir punts per Sashimi. Per tant, tot i que el DQL4 obtindrà una puntuació per partida més alta, també la mitjana de punts dels seus rivals augmentarà, i la probabilitat de perdre la partida també. En canvi, el DQL1, encara que faci una puntuació mitjana menor, també aconseguirà reduir la puntuació dels seus rivals, i per tant el seu ràtio de vitòries és més gran.

6. Conclusions

6.1 Lliçons apreses

A l'inici del projecte, l'autor no coneixia l'existència de l'aprenentatge per reforç. Gràcies al suggeriment dels consultors, s'ha pogut descobrir aquesta àrea de l'aprenentatge computacional que ofereix tantes possibilitats de creixement professional.

L'estudi i documentació dels algoritmes de RL utilitzats al projecte ha servit com a introducció per a comprendre la base sobre la qual es poden aconseguir construir agents que es comportin d'una manera que es pugui considerar intel·ligent. No obstant això, l'autor és conscient que el coneixement aconseguit gràcies a la realització del projecte és només la punta de l'iceberg de totes les possibilitats que ofereix el món del RL.

Les lliçons que s'aprenen poden venir de dues formes: de decisions correctes, o de decisions errònies. En aquest projecte es poden trobar lliçons de tot dos tipus. Entre les bones decisions, es troba la d'haver generat estructures de dades flexibles que han permès la fàcil gestió dels espais d'estat i d'accions de l'entorn, així com l'estructura d'agents realitzada. El procés d'entrenament és costós en temps d'execució i recursos, però les estructures de suport creades has permès una organització en la creació d'agents òptima. Altres decisions, en canvi, han suposat retards en la planificació o afegir temps de treball addicional per tal de poder complir amb les dates d'entrega. Una mala decisió molt important va ser no assegurar el correcte funcionament de l'entorn abans de començar amb el procés de creació dels agents. Una errada en el codi no va ser descoberta fins després d'haver generat un bon nombre d'agents que havien suposat dies d'execució d'experiments, i que per culpa de l'errada no eren utilitzables. Òbviament la regeneració de tots aquests agents una vegada l'error estava solucionat va suposar un important retard en la planificació.

Finalment, el projecte no només ha servit per recavar coneixement sobre el món del RL, sinó que també per descobrir que estratègies que habitualment l'autor aplica a les partides de Sushi Go són les mateixes que els agents han aconseguit descobrir per si mateixos mitjançant l'entrenament amb tècniques de RL, i per tant queden reforçades com a estratègies vàlides per augmentar la probabilitat de victòria.

6.2 Objectius assolits

Tot i no haver assolit la totalitat d'objectius específics, sí que tots els objectius generals s'han complert. Entre els objectius específics que no s'han pogut realitzar es troben:

- Creació d'una UI Visual: La previsió de temps per crear la UI gràfica era massa optimista. Es va substituir per una UI en mode text que permet a un usuari realitzar partides contra els agents generats.

- Comparació dels agents propis contra els agents de projectes preexistents: La manca d'una estructura estandarditzada en els entorns utilitzats pels projectes preexistents va fer impossible l'adaptació dels seus agents per ser comparats amb els generats al llarg d'aquest projecte.
- Agent basat en policy gradients: Diversos retards a la planificació van fer que la construcció d'agents basats en l'algoritme policy gradients fos descartada.

6.3 Seguiment de la planificació

La planificació ha estat subjecta a diverses modificacions al llarg del projecte. Les causes han estat diverses, però la major part han estat motivades per una previsió massa optimista dels temps de desenvolupament. No obstant això, els objectius que s'han hagut de descartar han estat mínims en comparació amb els objectius assolits.

Pel que fa a la metodologia plantejada des del començament, ha permès dur a terme el projecte de forma satisfactòria. No s'han hagut de realitzar grans ajustos ni modificacions a la línia de treball plantejada inicialment.

6.4 Línies de treball futur

Les possibilitats de millora del projecte són molt grans. Tal com s'ha explicat al punt 6.1, aquest projecte només cobreix una part ínfima de la gran quantitat de tècniques i mètodes que ofereix l'àrea del RL. Aplicant tècniques més avançades es poden generar nous agents amb encara més capacitat de seguir estratègies òptimes al joc, i suposar un repte encara més gran als jugadors humans. A més, el registre de l'entorn a OpenAI Gym facilita la tasca d'integrar l'entorn en tots aquells algoritmes de RL que es vulgui utilitzar.

També els espais d'estats utilitzats proporcionen un marge de millora als resultats obtinguts pels agents generats al projecte. L'espai més ampli utilitzat al projecte no conté tota la informació que rep un jugador en un moment determinat. Un espai d'estats que consideri les cartes que ha tingut el jugador en mans anteriors proporciona una informació molt important de cara a l'estratègia a seguir, i per tant pels agents que són capaços de gestionar espais d'estat molt grans poden aprofitar aquesta informació per obtenir un rendiment encara més alt.

L'entorn implementat també proporciona noves línies de treball per continuar ampliant les capacitats del projecte. El projecte s'ha limitat a realitzar agents enfocats a partides de dos jugadors, però nous projectes poden plantejar la possibilitat de realitzar agents més versàtils que adaptin la seva estratègia al nombre de jugadors als quals s'enfronten. També l'opció de jugar amb les regles de la versió Party de Sushi Go pot aportar nous reptes, i fins i tot l'entorn es podria ampliar amb els nous tipus de cartes que formen part de l'ampliació.

Deixant de banda Sushi Go, totes les tècniques apreses i aplicades poden ser reutilitzades per qualsevol altre entorn a on es vulguin generar agents intel·ligents.

7. Bibliografia

[1] Vincent D. Warmerdam. An OpenAi-like environment for the sushi go card game. https://github.com/koaning/sushigo

[2] Nicolas Arquie. Monte Carlo for Sushi Go (to be used with reinforcement learning) https://github.com/narquie/Sushi_Go_Monte_Carlo

[3] Alexander Soen. Making Tasty Sushi using Reinforcement Learning and Genetic Algorithms. http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2019/paper/ABCs2019_paper_v2_81.pdf

[4] Devir Iberia - Sushi Go!. http://devir.es/producto/sushi-go/

[5] David Poole, Alan Mackworth, Randy Goebel. Computational Intelligence. A Logical Approach. 1998. https://www.cs.ubc.ca/~poole/ci/ch1.pdf

[6] Stuart Russell, Peter Norvig. Artificial Intelligence: A Modern Approach. 3rd Edition. 2010.

[7] Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd Edition. 2017.

[8] OpenAI. https://openai.com/

[9] Deep Mind. Alpha Go. https://deepmind.com/research/case-studies/alphago-the-story-so-far

[10] Zhongheng Zhang. Reinforcement learning in clinical medicine: a method to optimize dynamic treatment regime over time. 2019. http://atm.amegroups.com/article/view/27136/pdf

[11] Hongjie Wu, Ru Yang, Qiming Fu, Jianping Chen, Weizhong Lu, Haiou Li. Research on predicting 2D-HP protein folding using reinforcement learning with full state space. 2019. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-3259-6

[12] Zhenpeng Zhou, Xiaocheng Li, Richard N. Zare. Optimizing Chemical Reactions with Deep Reinforcement Learning. 2017.https://pubs.acs.org/doi/10.1021/acscentsci.7b00492

[13] Hado van Hasselt, Arthur Guez, David Silver. Deep Reinforcement Learning with Double Q-Learning.

[14] Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu,Steven I. Marcus. An Adaptive Sampling Algorithm for Solving Markov Decision Processes. 2005. https://scholar.rhsmith.umd.edu/sites/default/files/mfu/files/cfhm05.pdf?m=1449834091

[15] Bruno Scibilia. How Could You Benefit from a Box-Cox Transformation? https://blog.minitab.com/blog/applying-statistics-in-quality-projects/how-could-you-benefit-from-a-box-cox-transformation