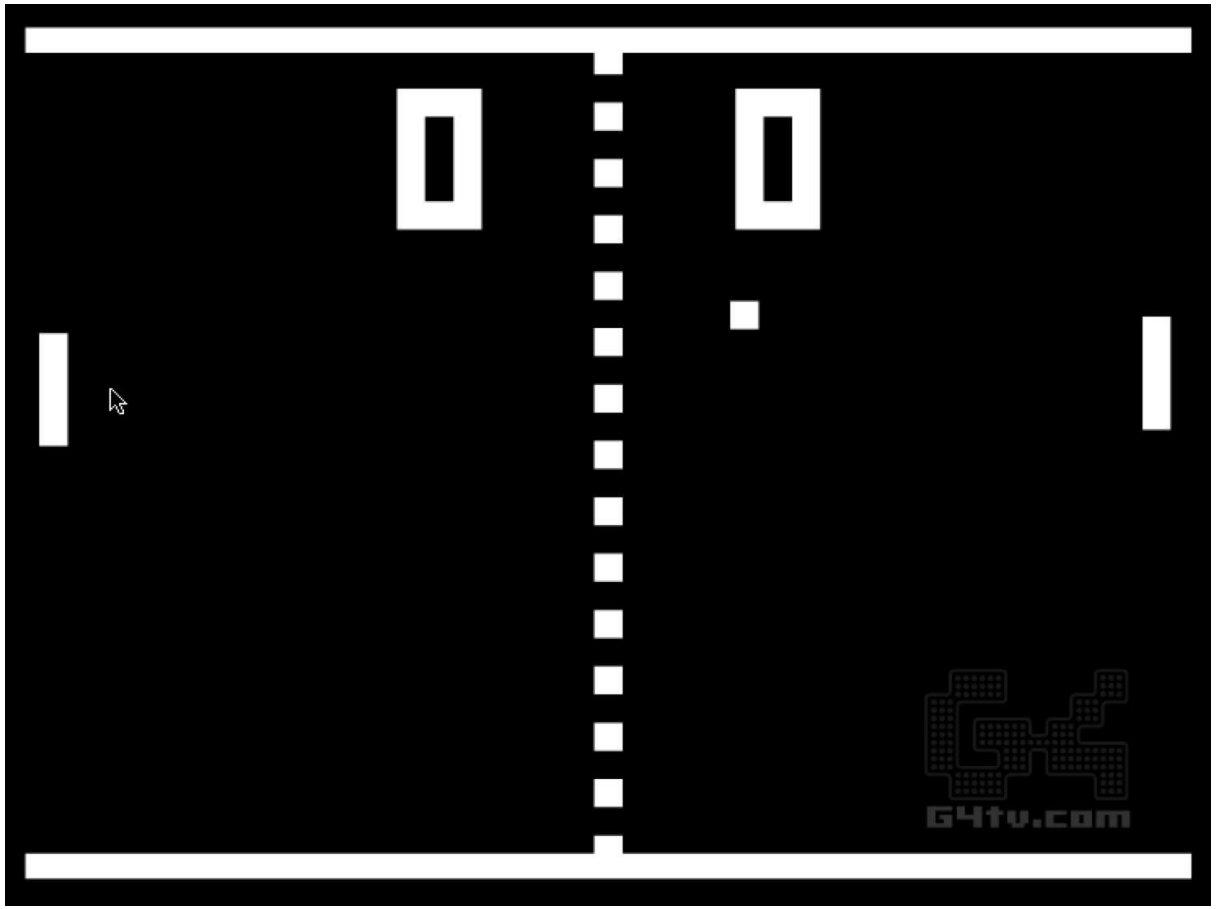
PONG

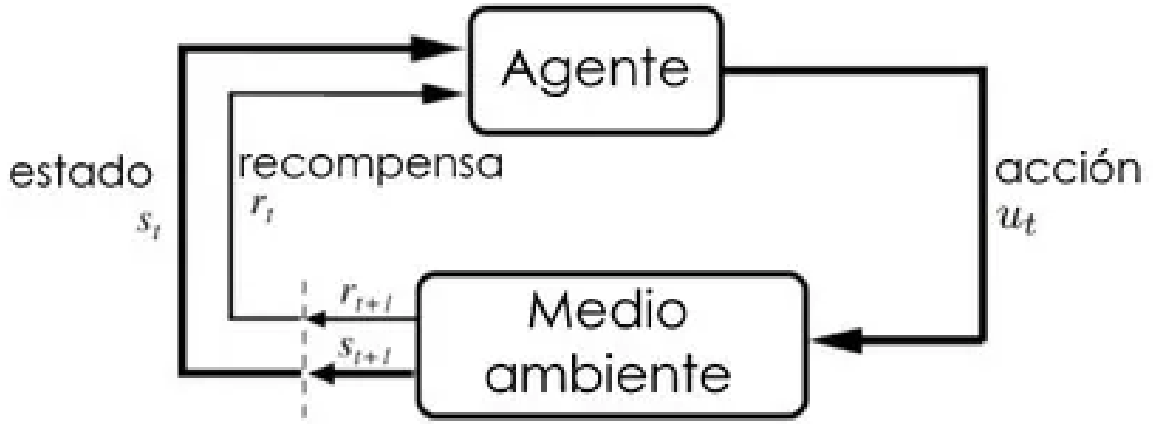
APLICACIÓN DE Q-LEARNING EN EL JUEGO PONG

Ribot, santiago
ribot@frba.utn.edu.ar

PONG



APRENDIZAJE POR REFUERZO

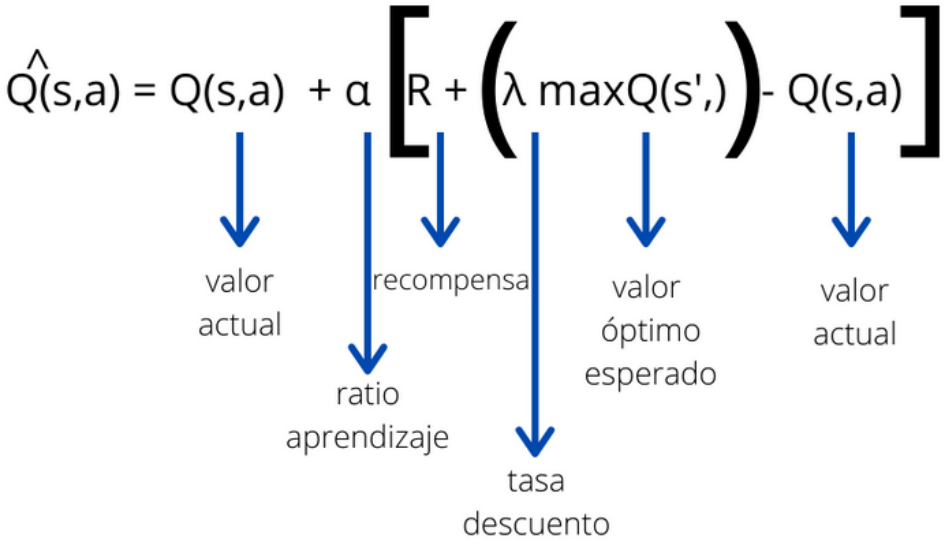


[Figure source: Sutton & Barto, 1998]

Repetir

- Inicializar Q-Table
- Elegir una acción
- Realizar la acción
- Obtener una recompensa
- Actualizar la Q-table

APRENDIZAJE POR REFUERZO



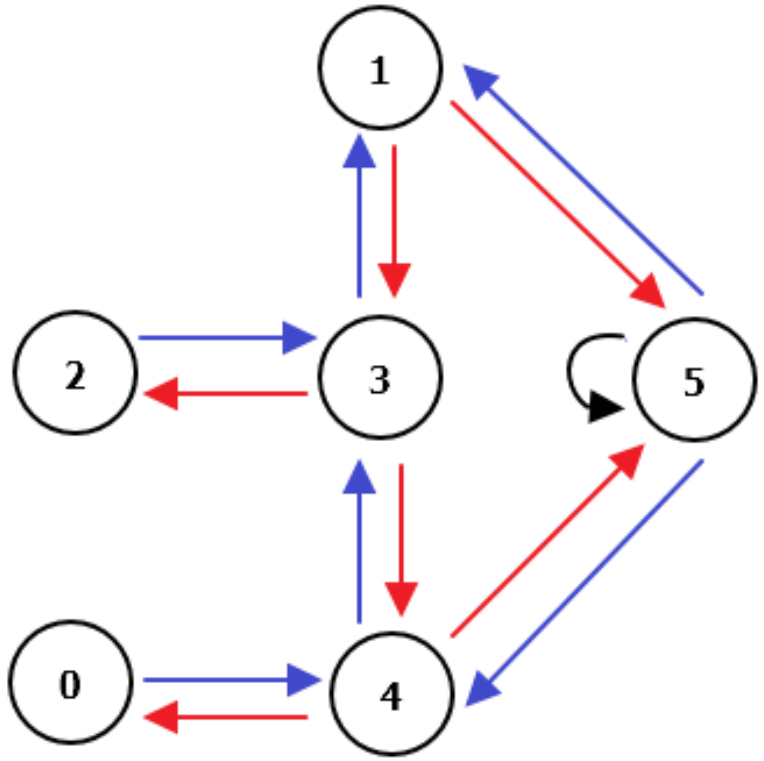
Exploración vs Explotación:

- Ratio de exploración: (0, 1)

Ecuación de Bellman:

- Factor de descuento: (0, 1)
- Ratio de aprendizaje: (0, 1)
- Recompensa: cualquier valor

APRENDIZAJE POR REFUERZO - EJEMPLO

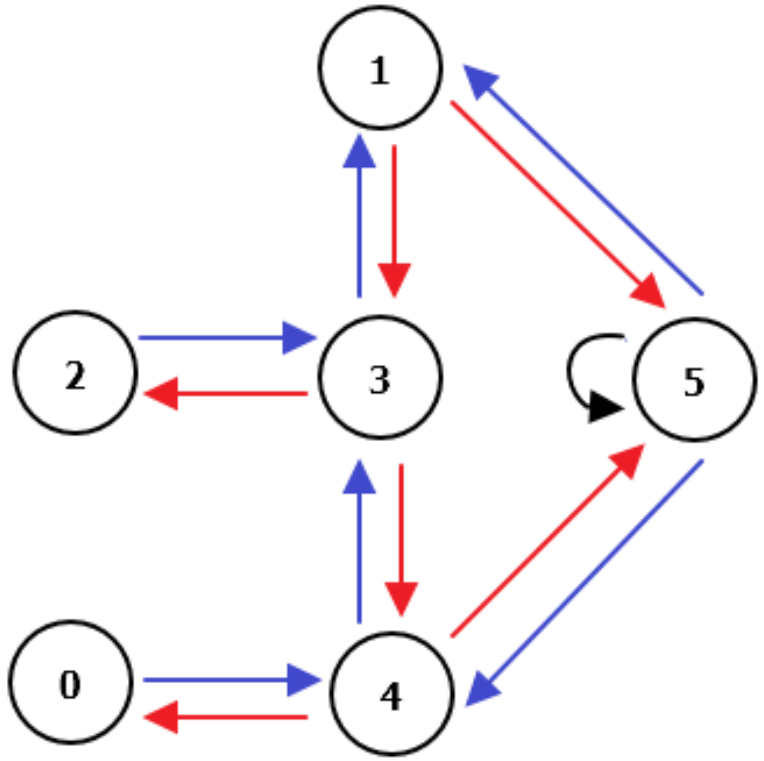


Laberinto de 6 etapas:

- Al llegar a la etapa 5 se gana
- El agente puede iniciar en cualquier etapa
- Los caminos son únicamente los indicados

	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

APRENDIZAJE POR REFUERZO - EJEMPLO



Actualización del valor Q: Ecuación de Bellman

$$Q'(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

	0	1	2	3	4	5
0	0	0	0	0		0
1	0	0	0		0	
2	0	0	0		0	0
3	0			0		0
4		0	0		0	
5	0		0	0		

EL JUEGO

Reglas:

- El agente tiene 3 vidas
- Cantidad máxima de puntos: 200
- Si se cae la pelota pierde 10 puntos
- Si golpea la pelota gana 10 puntos
- Si pierde las 3 vías pierde 20 puntos

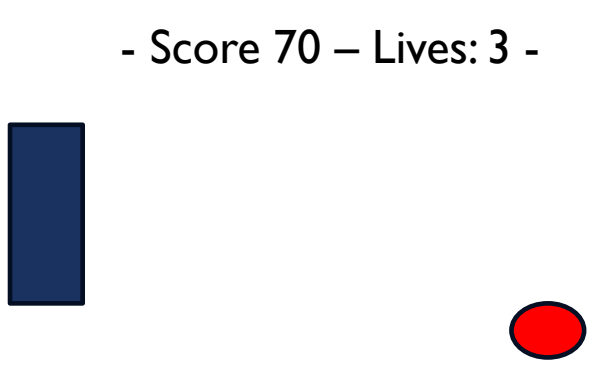
Clase agent:

- Almacena la Q-table
- Tiene las funciones para actualizarla
- Decide que acción tomar a continuación

Clase enviroment:

- Implementa la lógica del juego y las animaciones
- Da las recompensas o castigos
- Aplica la acción del agente y devuelve un nuevo estado

Q-TABLE



El estado queda fijado por tres componentes:

- Posición x de la pelota
- Posición y de la pelota
- Posición de la paleta (solo y)

ACCION:

RECOMPENSA: 0

ESTADO(x) = ESTADO(x) - 1
ESTADO(y) = ESTADO(y) - 1
ESTADO(p) = ESTADO(p) + 1

Q-TABLE

- Score 70 – Lives: 3 -

ACCION: ⬆

RECOMPENSA: 0

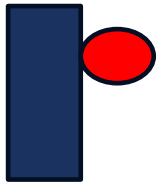
ESTADO(x) = ESTADO(x) - 1
ESTADO(y) = ESTADO(y) - 1
ESTADO(p) = ESTADO(p) - 1

Para el caso de ejemplo te utiliza una pantalla de 50px x 40px:

- Cada step mueve de a 5px
- 10 posiciones en X
- 8 posiciones en Y
- 8 posiciones de la paleta

Q-TABLE

- Score 80 – Lives: 3 -



ACCION: ⬇

RECOMPENSA: 10

ESTADO(x) = ESTADO(x) + 1
ESTADO(y) = ESTADO(y) - 1
ESTADO(p) = ESTADO(p) + 1

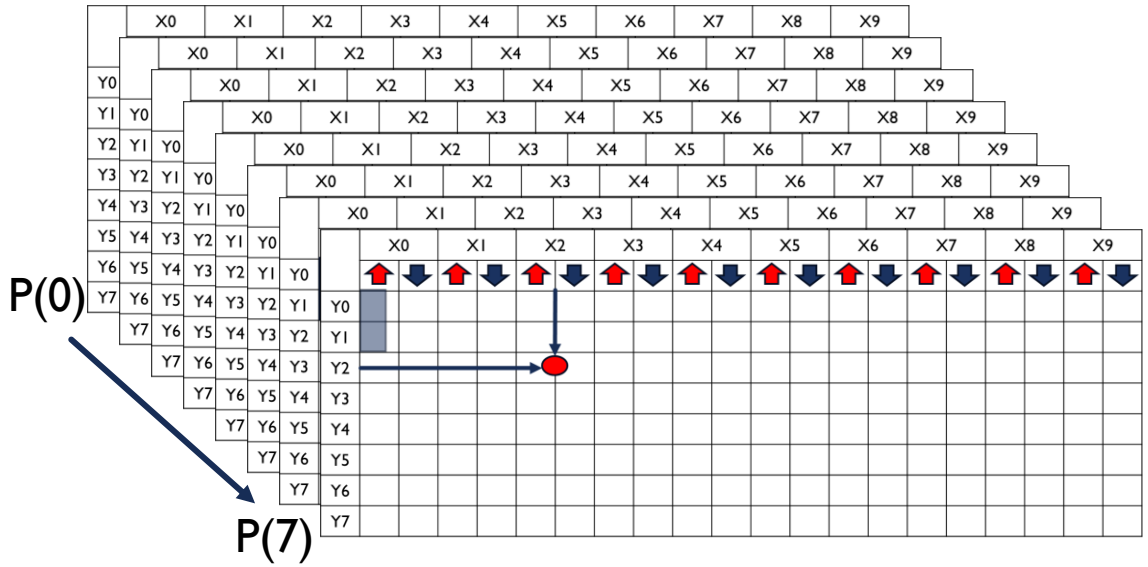
Además, la acción ⬆ o ⬇

- Es lo que modifica el estado P
- Los otros dos estados se modifican con la actualización de cada frame del juego

Q-TABLE

	X0		X1		X2		X3		X4		X5		X6		X7		X8		X9	
Y0																				
Y1																				
Y2																				
Y3																				
Y4																				
Y5																				
Y6																				
Y7																				

Q-TABLE



ESTADO[2][2][7][UP] = Q_VALUE

Q_VALUE = Q_VALUE + **α**(**R** + **λ***MAX(**ESTADO[1][1][6]**) – Q_VALUE)

CLASE AGENT

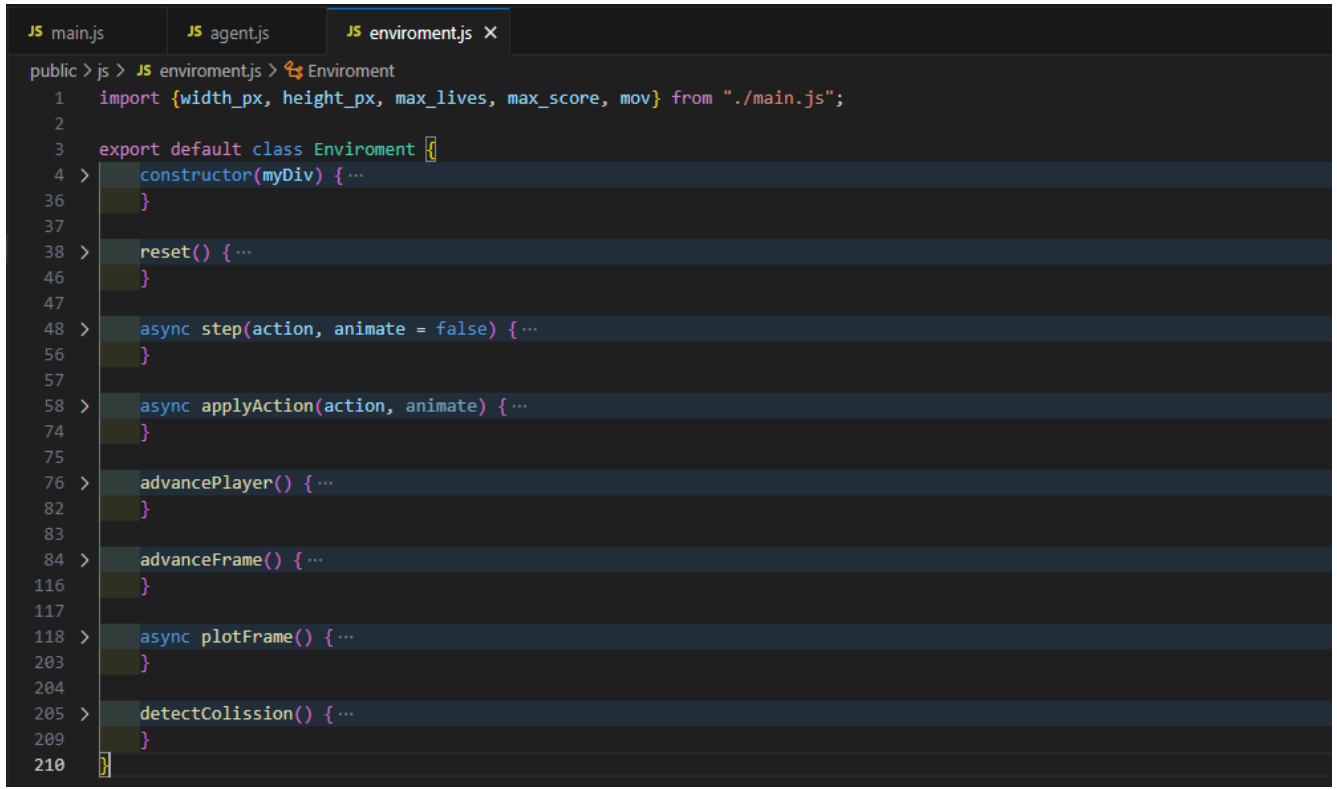
main.js | agent.js | enviroment.js

public > js > agent.js > Agent

```
export default class Agent {
    constructor(enviroment, policy, discountFactor = 0.1, learningRate = 0.1, explorationRatio = 0.9) {···
    }

    getShape(policy) {···
    }

    createQTable(position) {···
    }

    getNextStep(state, enviroment) {···
    }

    randomChoice(opt, state) {···
    }

    update(enviroment, oldState, action, reward, newState, end) {···
    }

    printPolicy() {···
    }

    getPolicy() {···
    }
}
```

```
constructor(enviroment, policy, discountFactor = 0.1, learningRate = 0.1, explorationRatio = 0.9) {

    if(policy)
        this.q_table = policy;
    else{
        this.position = this.getShape(enviroment.policy);
        this.position.push(2);
        this.q_table = this.createQTable(this.position);
    }

    this.discountFactor = discountFactor;
    this.learningRate = learningRate;
    this.explorationRatio = explorationRatio;
}
```

CLASE ENVIROMENT




```
constructor(myDiv) {
    this.actions = ['up', 'down'];
    this.state = [0, 0, 0];
    this.totalScore = 0;

    this.policy = [];
    for (let x = 0; x < height_px/mov; x++) {···
    }

    this.lives = max_lives;
    this.penalty = 0;
    this.paddleHeight = height_px/4
    this.score = 0;
    this.player = height_px/2

    this.x = width_px/2
    this.y = height_px/2
    this.dx = mov
    this.dy = mov
    this.radio = 2.5;
    this.myDiv = myDiv;

    this.interval = null;
}
```

PLAY

```
for(let playedGames = 0; playedGames < rounds; playedGames++) {
    let state = enviroment.reset();
    let reward = null;
    let done = null;
    let iteration = 0;
    let contador = 0;

    while(done != true && iteration < 3000 && enviroment.totalScore < max_score) {
        let oldState = state.slice();
        let nextAction = agent.getNextStep(state, enviroment);

        ({state, reward, done} = await enviroment.step(nextAction, animate));

        if(animate){
            await enviroment.plotFrame();
        }

        if(rounds > 1) {
            agent.update(enviroment, oldState, nextAction, reward, state, done)
        }
        iteration += 1;
    }

    totalReward += enviroment.totalScore;

    if(enviroment.totalScore > max) {
        max = enviroment.totalScore;
        firstMax = playedGames;
    }

    if(playedGames%500 === 0 && playedGames > 1 && !animate) {
        contador += 1;
        console.log(`-- Games: [${playedGames}] -- AvgScore: [${(totalReward/(500*contador)).toFixed(2)}] -- AvgSteps: [${iteration}] -- MaxScore: [${max}] -- GameOfFirstMax: [${firstMax}]`);
        max = 0;
        firstMax = 0;
        totalReward = 0;
    }
}
```

GRACIAS