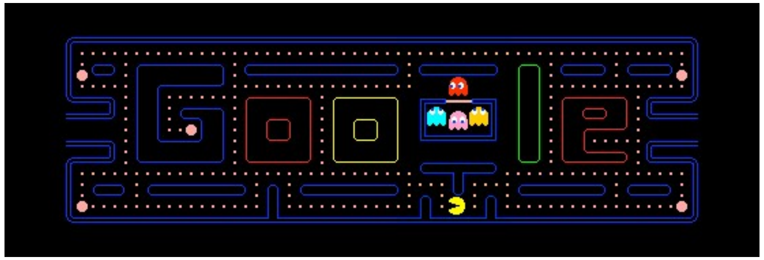


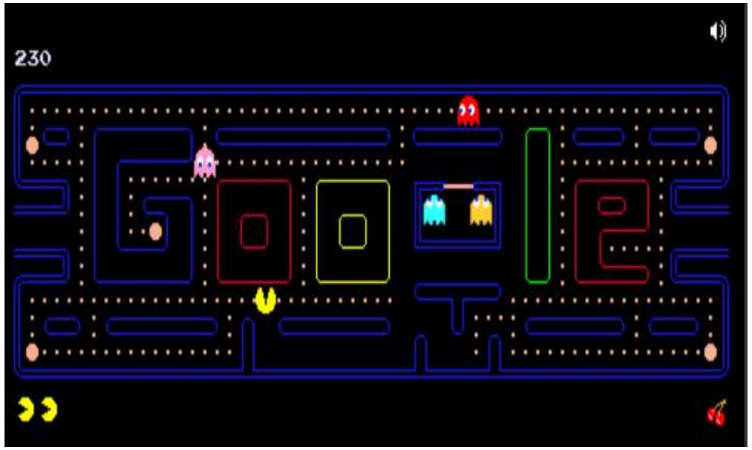
# ROB 311-Task 3
# Q-Learning


Adriana TAPUS & Chuang YU

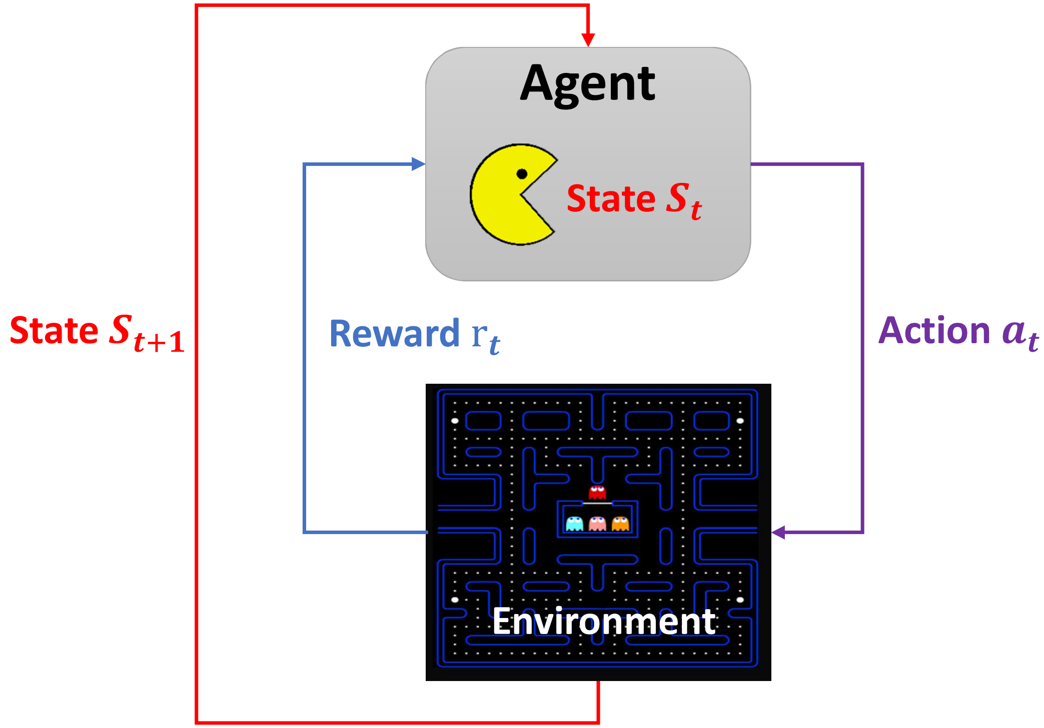
adriana.tapus@ensta-paris.fr & chuang.yu@ensta-paris.fr


09-2020

# 1. Introduction




Pac-Man




**Agent: Player** **Environment: Game**

**State $S_t$: the position at time "$t$", this frame**

**Action $a_t$: $a_t \in$ {"Left", "Right", "Up", "Down"}**

**Reward $r_t$: at time "t", eat one bean, immediate**

**Reward less (little bean) or more (corner big bean).**

**Policy π: It is the probability of taking action.**

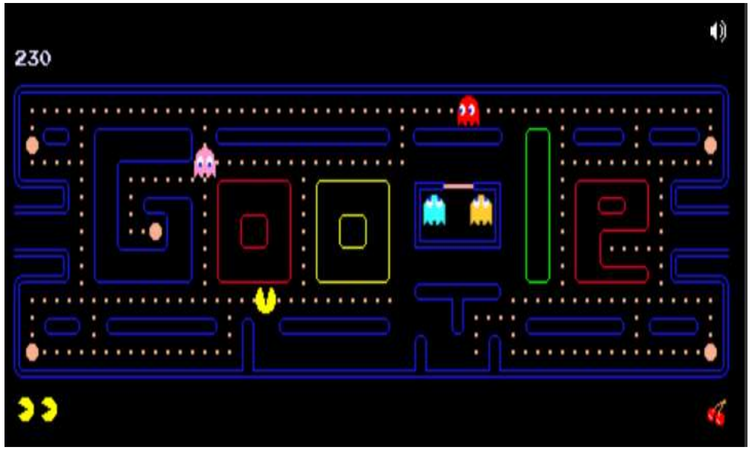
**π ( a | s)= P (A = a|S = s)**

**e.g. π ( Left | s)= 0.4 π ( Right | s)= 0.2**

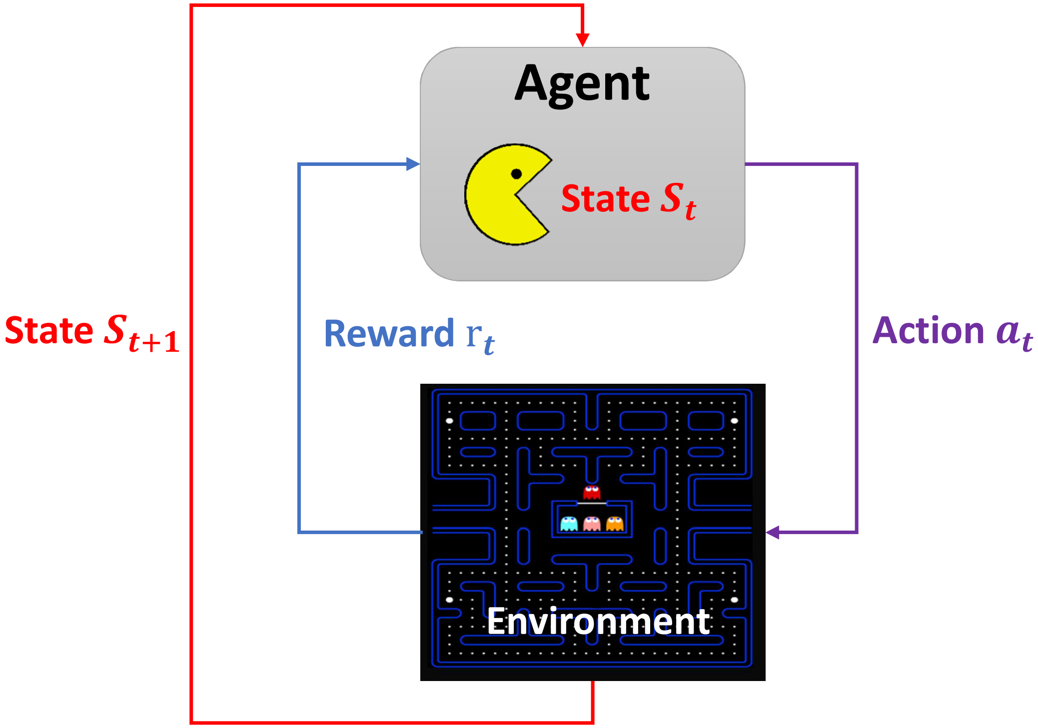
**π ( Up | s)= 0.2 π ( Down| s)= 0.2**

ENSTA IP PARIS

# 1. Introduction




Pac-Man




**State Transition: State $S_t$ → State $S_{t+1}$**

- **PacMan takes Action $a_t$, the state changes**
- $\mathrm{P}(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a)$

# 1. Introduction

**Return : cumulative future reward**

$$U_t = R_t + R_{t+1} + R_{t+2} + \ldots\ldots$$

**Discounted Return : discounted cumulative future reward**

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \ldots\ldots$$

**Action-value function** $Q_\pi(s_t, a_t)$

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$$

**State-value function**

$$V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$$

## 2. Q-Learning

$$V^*(S) = R(s) + max_a \gamma \sum_{S'} T(S,a,S') V^*(S')$$

Value iteration is a method of computing an optimal policy for an MDP (Markov Decision Process) and its value.

**Value iteration** →

$$\begin{cases} Q_{k+1}(s,a) = R(s) + \gamma * \sum_{s'} T(S,a,S') * \boxed{V_k(s')} & (1) \\ V_k(s) = \boxed{\max_a Q_k(s,a)} & (2) \end{cases}$$

Replace $V_k(s')$ in Equation(1) with Equation (2)

$$Q_{k+1}(s,a) = \sum_{s'} T(S,a,s') \left[R(s) + \gamma ** \max_a Q_k(s',a')\right]$$

**Q-Value update**

**(Q-learning)** → $Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha[R(S,a) + \gamma \max_a Q(S',a)]$

https://artint.info/html/ArtInt_265.html

ENSTA IP PARIS

## 2. Q-Learning

### Process of Q-Learning playing Pac-Man

Optimal state-value function $Q^*$

$$S_t \longrightarrow a_t^* = argmax_{a_{all}} Q^*(S_t, a_{all}) \longrightarrow a_t^* \rightarrow S_{t+1},\ r_t$$

Current state

Base on the final Q-value to select optimal action $a_t^*$ who has max $Q^*$ in state $S_t$

# 3. Task 3 Q-learning

## ROB311 – TP3 – Q-Learning and Pac-man

### Introduction

Today, you will implement **Q-Learning** in Python. In particular, you will train an AI player of the famous Pac-Man arcade game. You will work on a set of Python files and libraries provided to you by the **UC Berkeley university**, as part of their ***CS188 – Intro to AI*** course, and you will only have to write the core algorithm of the methods dealing with the Q-Learning and Approximate Q-Learning, based on the equations you have seen in the course.

### Files

The **pacman.zip** archive you have downloaded from the ROB311 course page provides you with the files you need in order to implement the Q-Learning and Approximate Q-Learning AI for the Pac-Man game. The files are taken from the Project 3: Reinforcement Learning page of the *CS188 – Intro to AI* online course. You can read the **Introduction** section of the page if you need extra informations, but we are going to focus only on the files you need in order to implement today's algorithms.

File and classes you have to modify:

- *qlearningAgents.py: QLearningAgent, ApproximateQAgent*

AI/Machine Learning related files and classes (you will have to look through):

- *learningAgents.py: ReinforcementAgent, ValueEstimationAgent*
- *util.py: Counter*
- *game.py: Agent*
- *featureExtractors.py*

**Related PPT-Q Learning in CS 188**

**Project 3: Reinforcement Learning**

ENSTA IP PARIS

# 3. Task 3 Q-learning

PAC-MAN

### Objectives

Implement Q-Learning by modifying the following *QLearningAgent* methods in the ***qLearningAgents.py*** file:

- *__init__()*
- *getQValue()*
- *computeValueFromQValues()*
- *computeActionFromQValues()*
- *getAction()*
- *update()*

# 3. Task 3 Q-learning

### How to run and test your code

Navigate using a terminal to your project folder and run the following commands:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

This command will attempt learning from 2000 training episodes and then test the resulting AI agent (player) on 10 games. You will be able to see the AI playing the 10 test games. The learning is done on the *smallGrid* map and the results of the 10 test games will be displayed in the terminal. The win rate on the 10 test game is supposed to be very high, ideally 100%.

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

This command will show you what happens during the training process for 10 games.

If you want to test your code in other scenarios, use the following command to understand what each of the command line parameters does:

```
python pacman.py --help
```


**Use Layout"SmallGrid"**

Rule

--2 persons in one group

--Dealine: Bedore Monday

Submit:

--Report paper + code

--Github or ENSTA gitlab

or

End!
Question?