50.039 Theory and Practice of Deep Learning

W11-S1 Introduction to Reinforcement Learning

Matthieu De Mari

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

About this week (Week 11)

1. What is **Reinforcement Learning**?
2. What are the key ideas behind **reinforcement learning** and its **framework**?
3. What is the **exploration vs. exploitation tradeoff**?
4. How do we **train** an **RL agent** by exploring, then progressively exploiting?
5. What are some **advanced strategies** in **multi-arm bandit problems**?
6. What are the **Q** and **V functions** for a RL problem?
7. What is **Q-learning** and how can it be implemented in RL problems?

Reinforcement Learning, a definition

Definition (Reinforcement Learning - RL):

Reinforcement Learning (RL) is one of the many machine learning paradigms. Where **DL** attempts to learn from **examples** (i.e. samples in a dataset), **RL** attempts to train an AI from **trial-and-error**.

There is no dataset, only a feedback on the action taken in any given state.

This paradigm challenges the idea that computers and AIs should be learning from datasets.

- **What if the task at hand cannot be solved easily and no dataset can even be generated?**
- **What if the dataset is flawed?**

Reinforcement Learning, a definition

- A Go AI would need to look at a board as input and produce the best move as an output.
- However, Go is a prime example of a problem for which no dataset (board state → best action in given board state) can be generated.
- This has to do with the fact that our understanding of the game of Go is actually… pretty bad.

Reinforcement Learning, a definition

Definition (Reinforcement Learning - RL):

Reinforcement Learning (RL) is one of the many machine learning paradigms. Where **Deep Learning (DL)** attempts to learn from **examples** (i.e. datasets), **RL** attempts to learn from **trial-and-error**.

There is no dataset, only a feedback on the **action** taken in any given **state**.

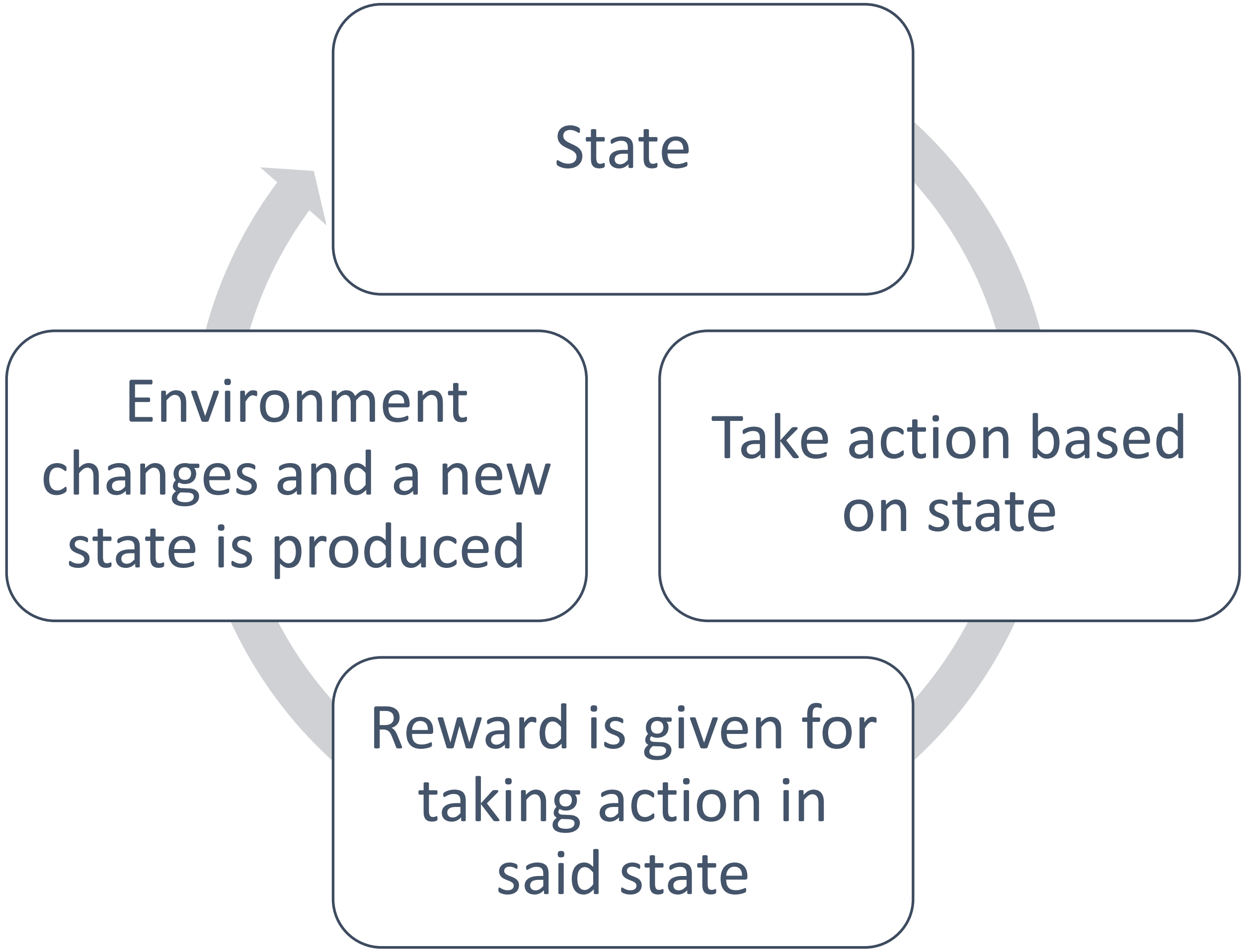
In RL, the AI learns by trying certain **actions** in given **states** and then receives a **feedback** (not always instantaneous) describing whether or not the **action** turned out to be a good or bad decision.

The AI then learns, on-the-fly, from trying out combination of (**states**, **actions**, **feedback**) for a given problem.

A typical RL problem

A typical RL problem can be broken down in **three parts**.

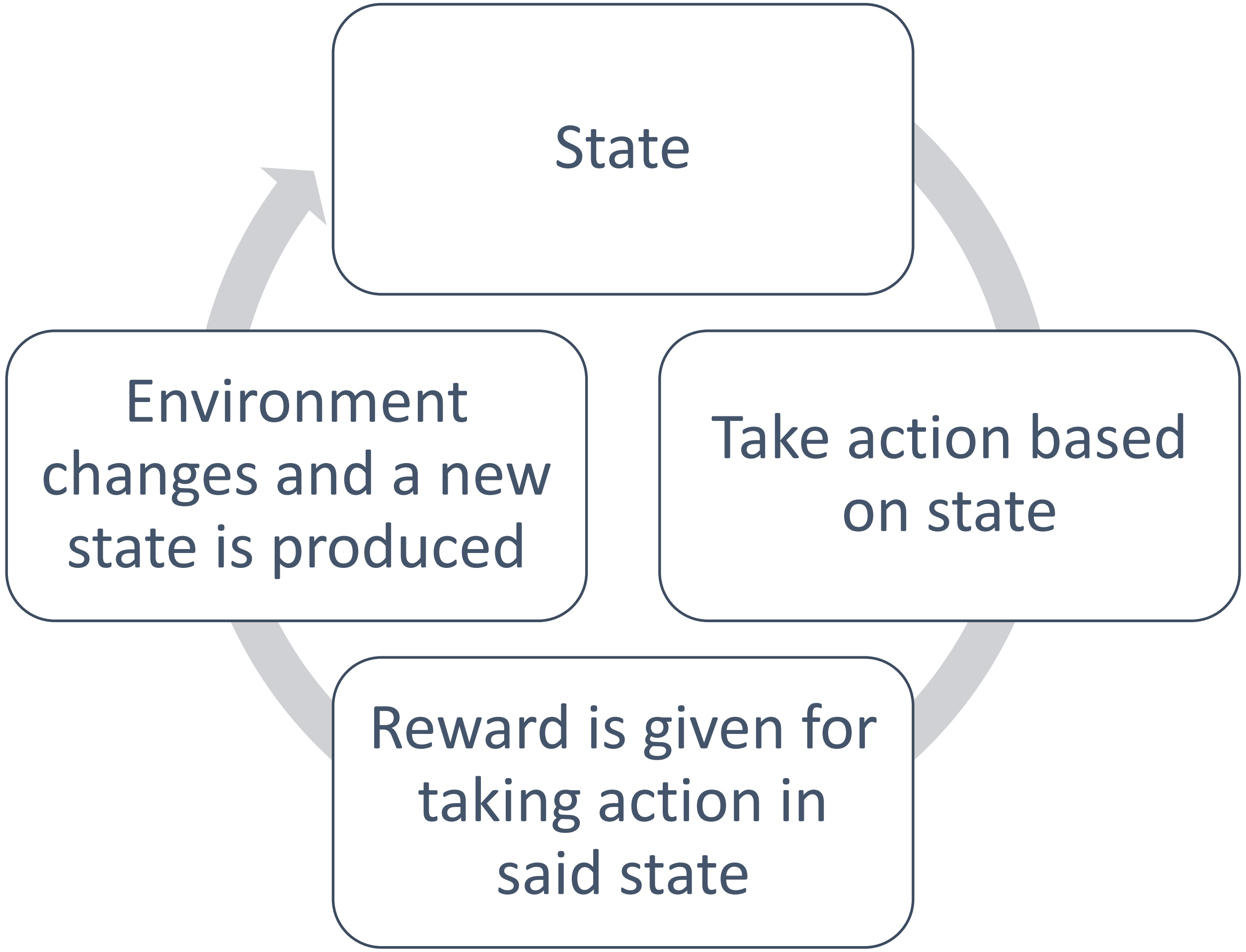
- The AI is given a problem, at time t, in a current **state** $\boldsymbol{s(t)}$.
- The AI then takes an **action**, to answer the problem, in this given state, $\boldsymbol{a(t)}$.
- The **action** has an effect, positive or negative, which is eventually measured by the AI, in terms of **feedback**, $\boldsymbol{R(t, a(t), s(t))}$.



A typical RL problem

A typical RL problem can be broken down in **three parts**.

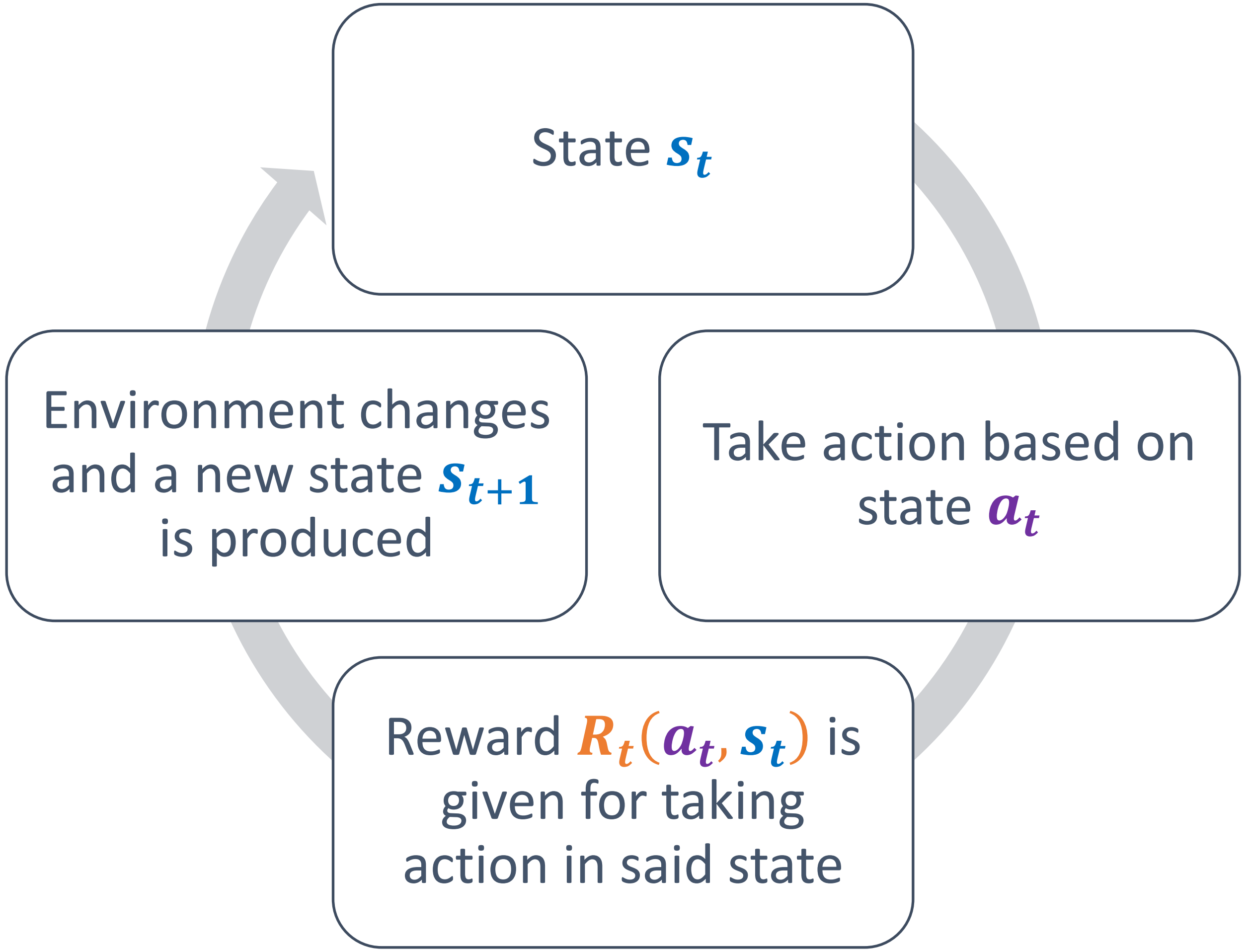
- The AI is given a problem, at time t, in a current **state** $\boldsymbol{s_t}$.
- The AI then takes an **action**, to answer the problem, in this given state, $\boldsymbol{a_t}$.
- The **action** has an effect, positive or negative, which is eventually measured by the AI, in terms of **feedback**, $\boldsymbol{R_t(a_t, s_t)}$.



A typical RL problem

A typical RL problem can be broken down in **three parts**.

- The AI is given a problem, at time t, in a current **state** s_t.
- The AI then takes an **action**, to answer the problem, in this given state, a_t.
- The **action** has an effect, positive or negative, which is eventually measured by the AI, in terms of **reward**, $R_t(a_t, s_t)$.

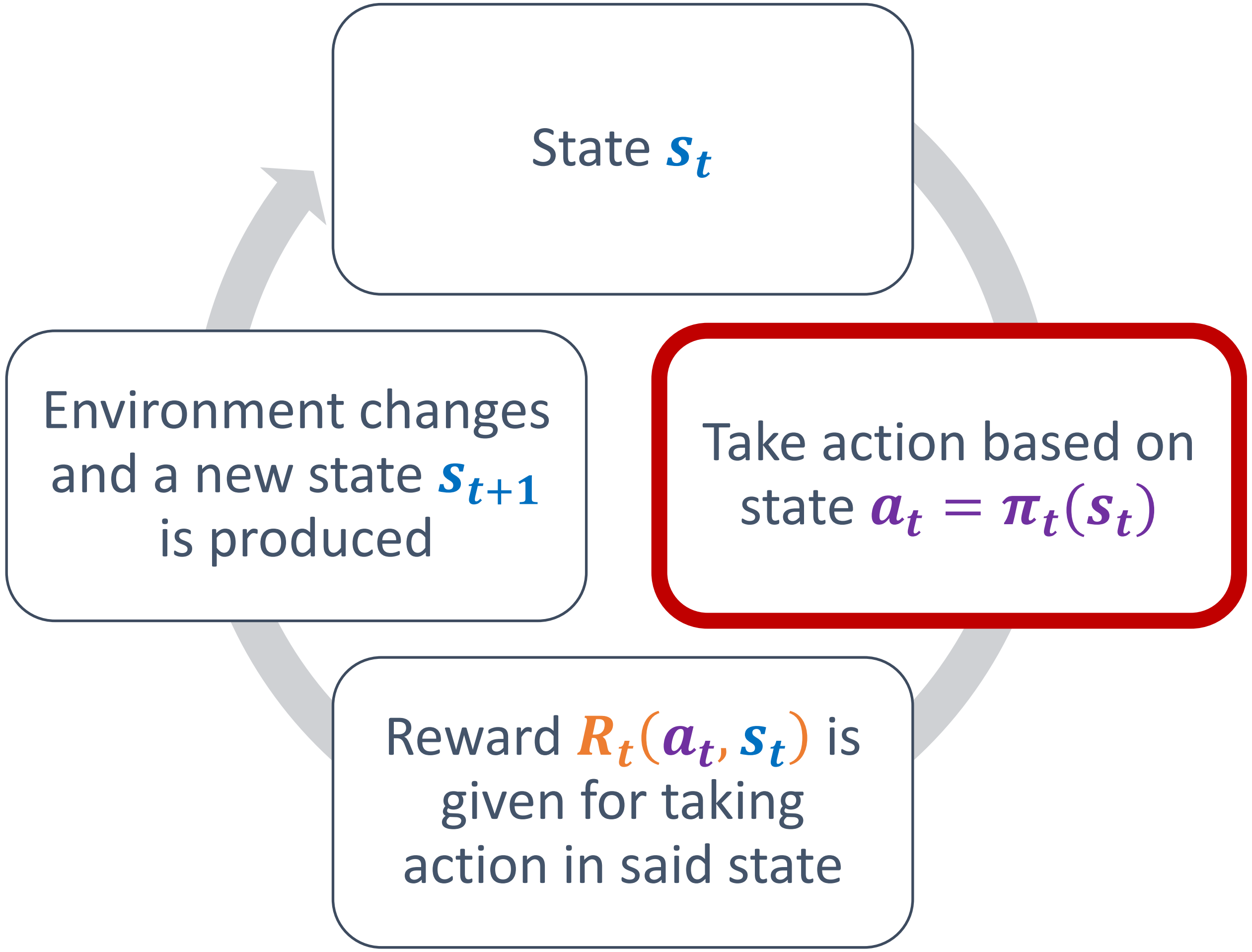


A typical RL problem

Definition (policy):

We call **policy**, the function, which returns the **action** $\boldsymbol{a_t}$, among the set of possible actions $\boldsymbol{A}$, to be taken in any given **state** $\boldsymbol{s_t}$. It is often denoted as $\boldsymbol{a_t = \pi_t(s_t)}$.

When training an AI, with RL, the objective is define the **best policy**, so that the AI knows the best course of action $\boldsymbol{\pi_t(s_t)}$ to use in response for any possible **state** $\boldsymbol{s_t}$.

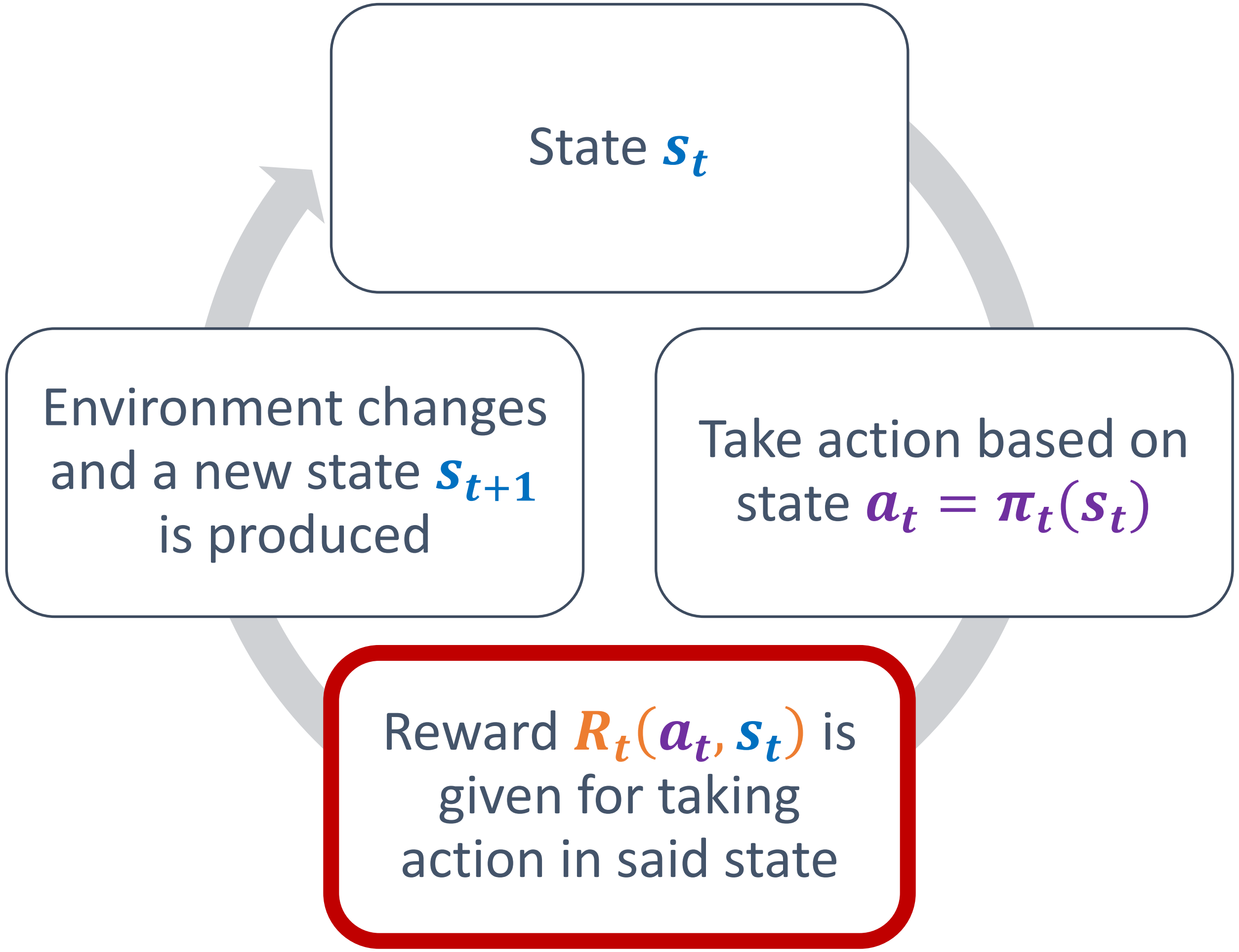


A typical RL problem

Definition (reward and return):

The **reward** function $R_t(a_t, s_t)$ is the "loss" function given to the AI.

The objective of the AI is then to find the best policy $\pi_t(s_t)$…



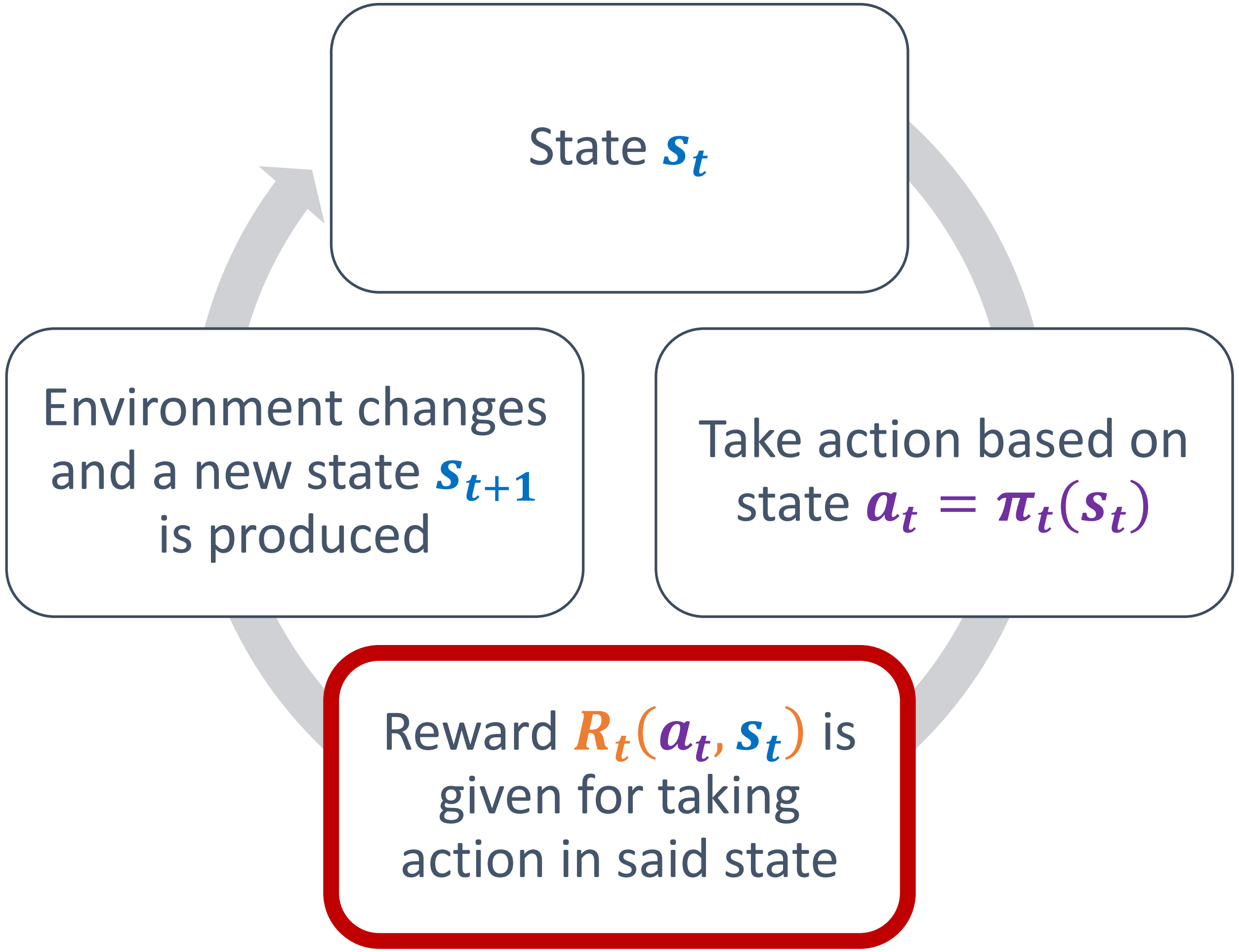
A typical RL problem

Definition (reward and return):

The **reward** function $R_t(a_t, s_t)$ is the “loss” function given to the AI.

The objective of the AI is then to find the best policy $\pi_t(s_t)$, that is, the policy, which maximizes the **return** G_t, i.e. the gains that can be expected in the future, if action $\pi_t(s_t)$ is taken in present state s_t.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$



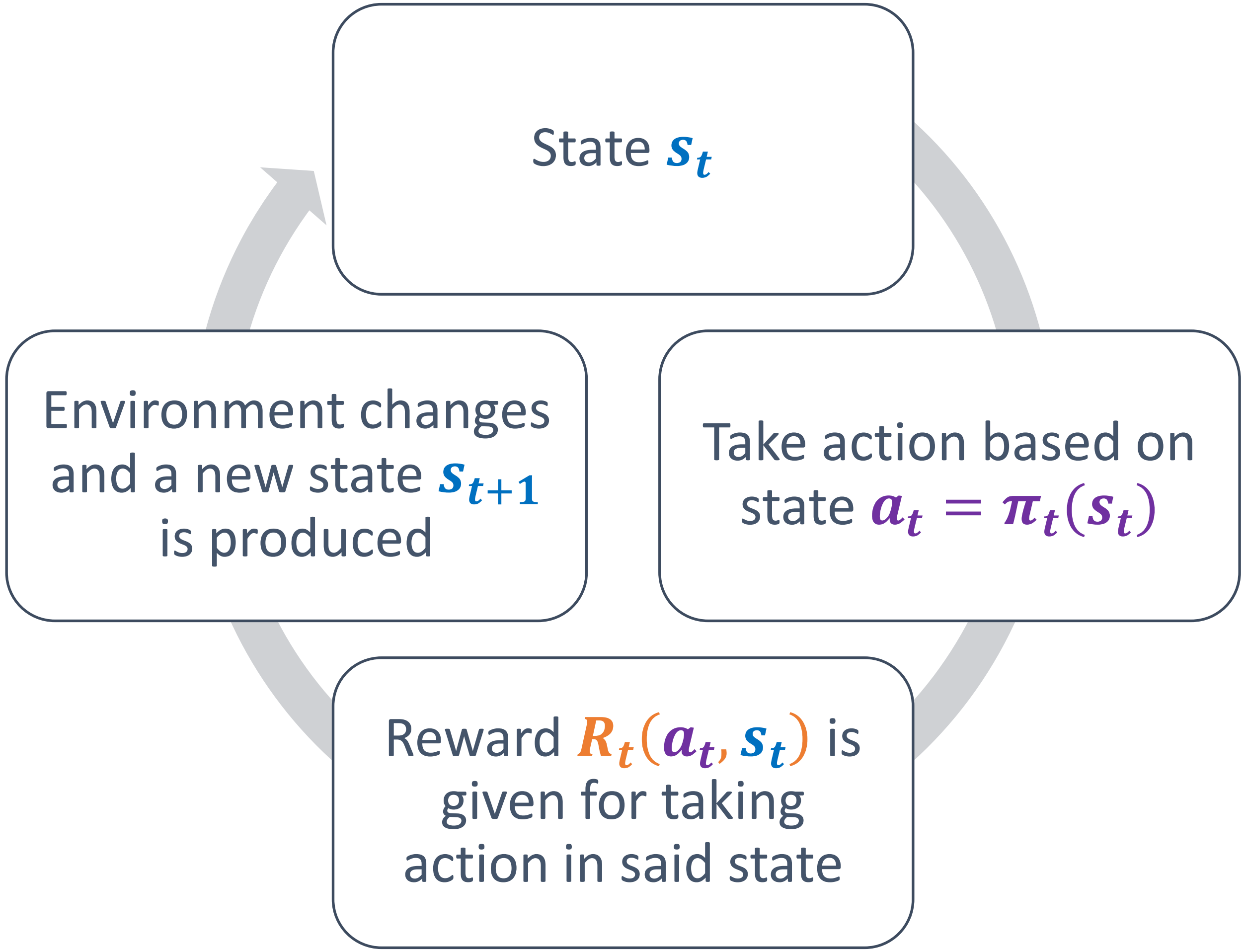
A typical RL problem

Definition (reward hypothesis):

The RL framework relies on the **reward hypothesis**.

This states that *"Any goal can be formalized as the outcome of maximizing a cumulative reward (or gain) function G_t of some sort".*

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$



A first toy problem

The Random Candy Machines problem

Restricted

Candy Machines are **deterministic**

A standard candy machine often work as follows.

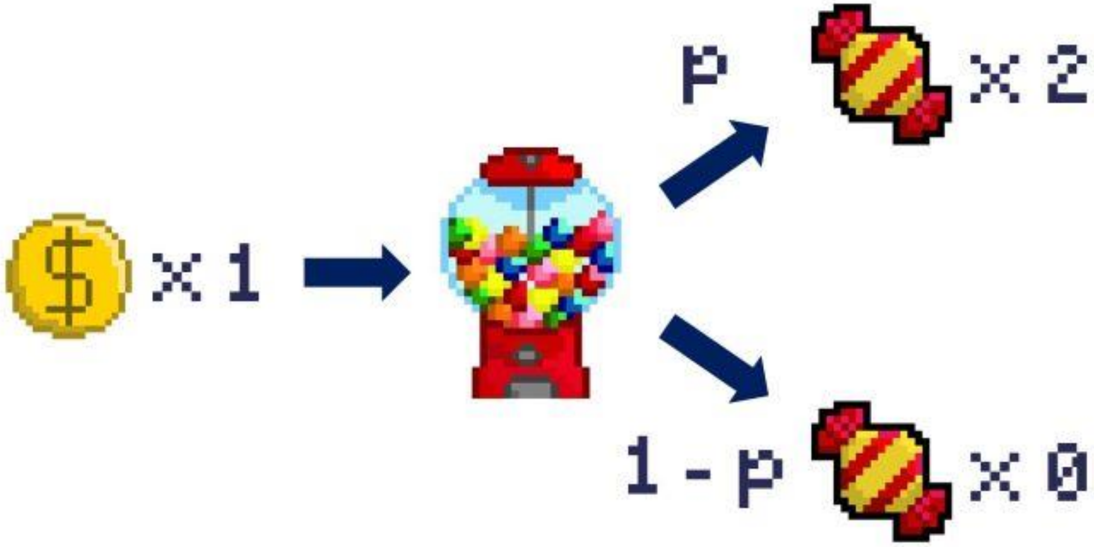
- You put one coin in the machine and obtain a candy as a result.
- Its behavior is **deterministic**, i.e. it is **100% predictable**.
- It will ALWAYS return a candy, whenever you put a coin in the machine.
- (We also assume it never runs out of candy…)

Restricted

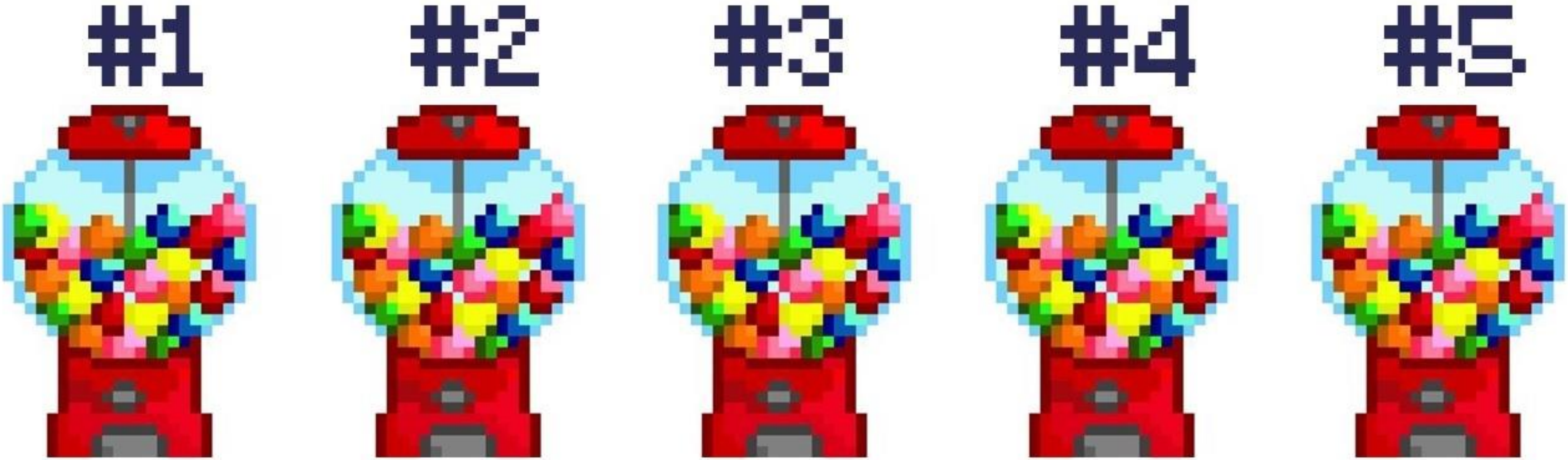
What if machines were **random**?

Let us now consider a world where candy machines are **random machines** instead.

- When you put a coin in, it might give you two candies… Or nothing at all.
- Depending on a **probability** $\boldsymbol{p}$.
- For instance, if $p = 80\%$, you will receive
  - two candies 80% of the time,
  - and no candies 20% of the time.

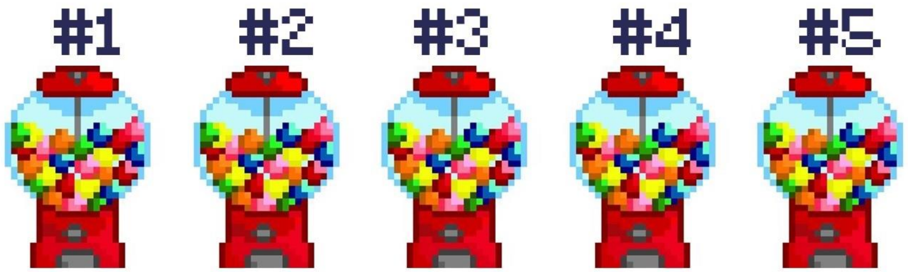


A simple problem?



Restricted

A simple problem?



Let us consider the following problem.

- You have $\mathbf{N = 10000}$ **coins**.
- You are facing **5 machines**: machine #1 is a **deterministic** machine, machines #2-5 are **random** machines.
- Each **random** machine #2-5 has its own independent **winning probability** (p_2, p_3, p_4, p_5) and these will not change over time.
- **You do NOT know** what the probabilities (p_2, p_3, p_4, p_5) are, but you can use some of your coins to test the machines, and estimate the hidden probabilities (p_2, p_3, p_4, p_5).

Big question: what should be your **strategy** to **maximize** the number of candies you will obtain after using your 10000 coins?

Reusing the RL formalism

We start with a bit of formalism.

- Let us denote $\boldsymbol{t}$, the **index of the coin to be played**.
$$\boldsymbol{t \in \{1, \dots, 10000\}}$$
- The **set of actions** is simply
$$\boldsymbol{A = \{1, 2, 3, 4, 5\}}$$
- We could define our **state**, at time t, as our **current estimates of the expectations** $\boldsymbol{e_i(t)}$ of each machine $\boldsymbol{i} \in \boldsymbol{A}$.
$$\boldsymbol{s_t = (e_1, e_2, e_3, e_4, e_5)}$$

Reusing the RL formalism

Ultimately, we get the feeling that the best strategy will follow this type of scheme…

- We will probably **try our machines to guess/estimate which one is the best**.

Reusing the RL formalism

Ultimately, we get the feeling that the best strategy will follow this type of scheme…

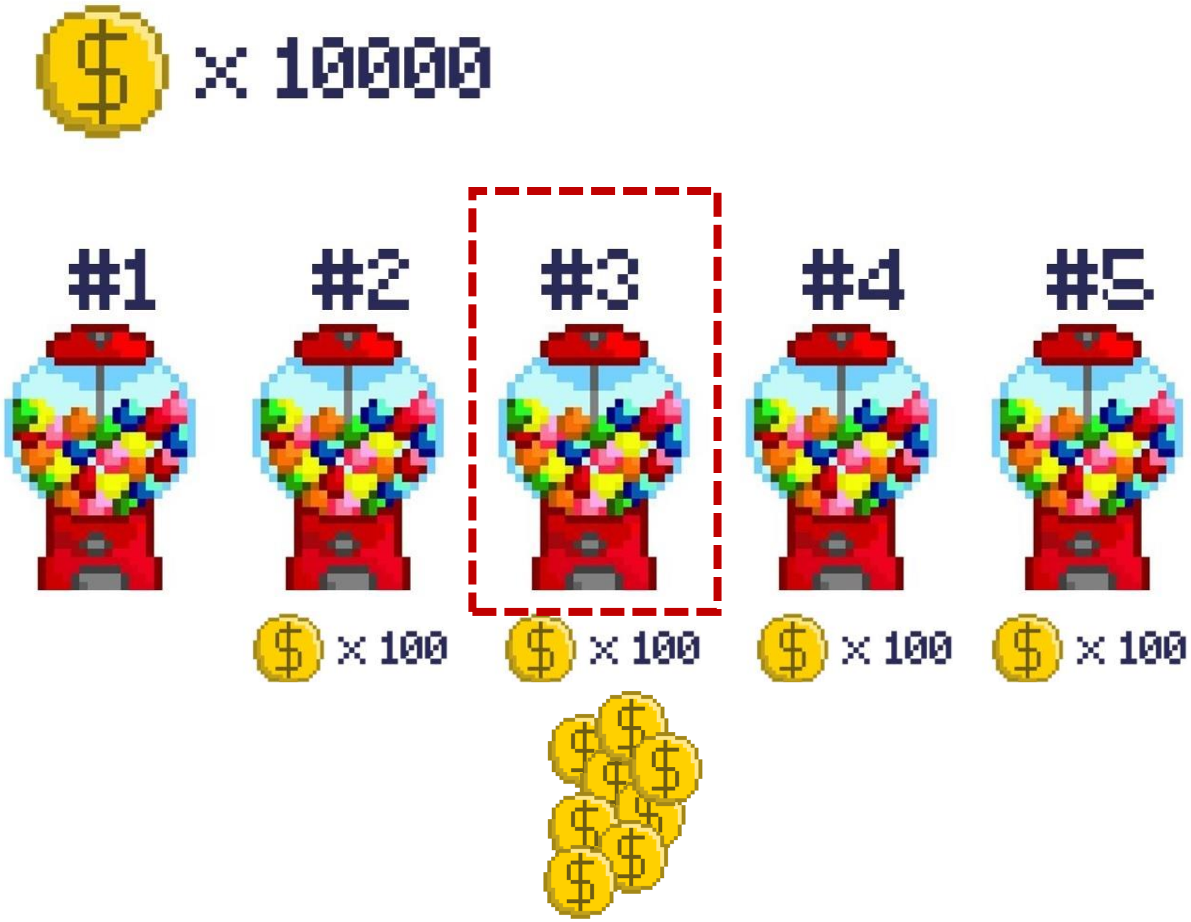
- We will probably **try our machines to guess/estimate which one is the best**.

$e_1 = 1$ $e_2 = 0.8$ $\boldsymbol{e_3 = 1.6}$ $e_4 = 1.4$ $e_5 = 1.1$

Reusing the RL formalism

Ultimately, we get the feeling that the best strategy will follow this type of scheme…

- We will probably **try our machines to guess/estimate which one is the best**.
- And eventually, we will **use all our coins on that “best” machine**, to maximize our number of candies.



Exploration and exploitation tradeoff

Definition (exploration and exploitation):

The phase during which, you try out possible actions, even though they might not be optimal, in order to acquire knowledge about the problem is called **exploration**.

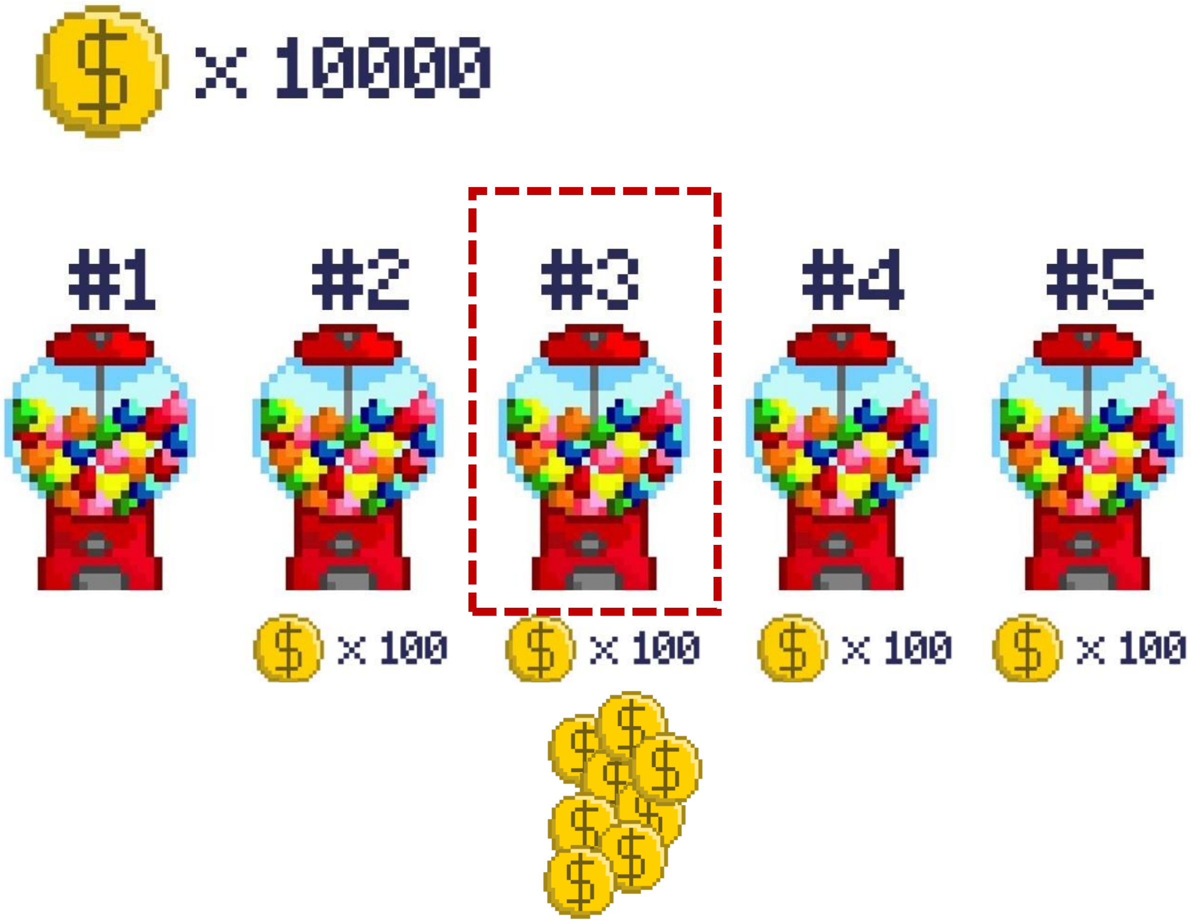
$e_1 = 1$ $e_2 = 0.8$ $e_3 = 1.6$ $e_4 = 1.4$ $e_5 = 1.1$

Exploration and exploitation tradeoff

Definition (exploration and exploitation):

The phase during which, you try out possible actions, even though they might not be optimal, in order to acquire knowledge about the problem is called **exploration**.

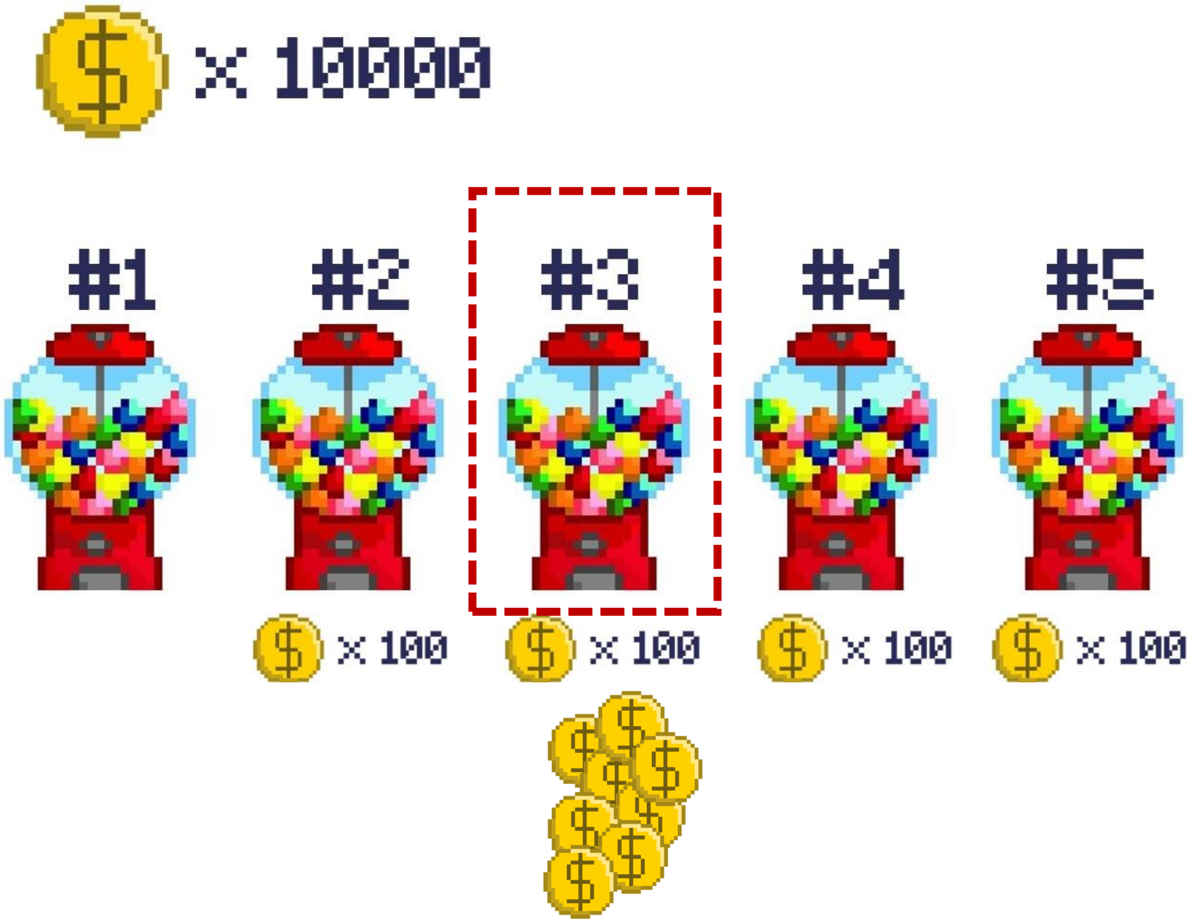
The second phase, where you rely on your acquired knowledge, and play what you feel is the best strategy, is called **exploitation**.



Exploration and exploitation tradeoff

Definition (exploration vs. exploitation tradeoff):

A good RL-based AI, needs to smartly combine **exploration** and **exploitation** phases.

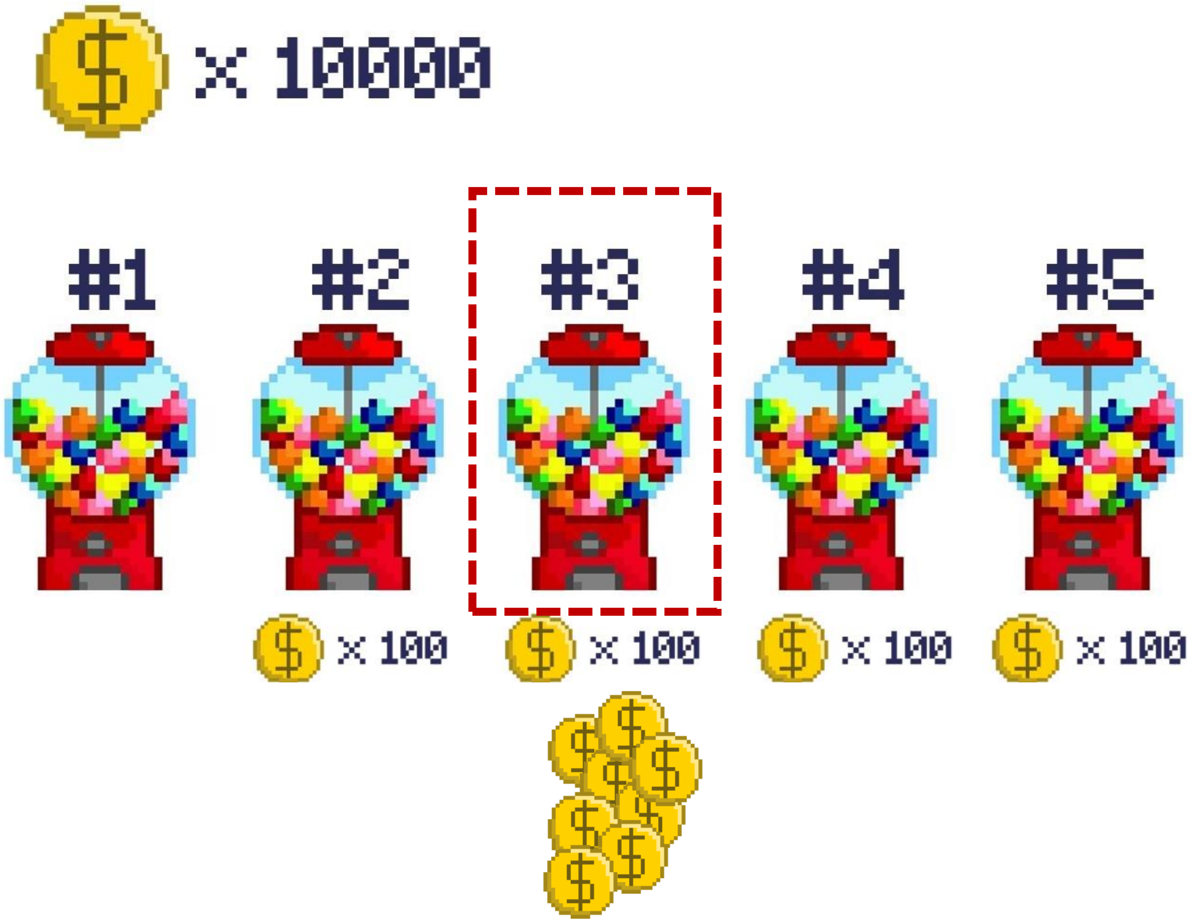


Exploration and exploitation tradeoff

Definition (exploration vs. exploitation tradeoff):

A good RL-based AI, needs to smartly combine **exploration** and **exploitation** phases.

- **Too much exploration?** You have wasted coins trying out bad machines.

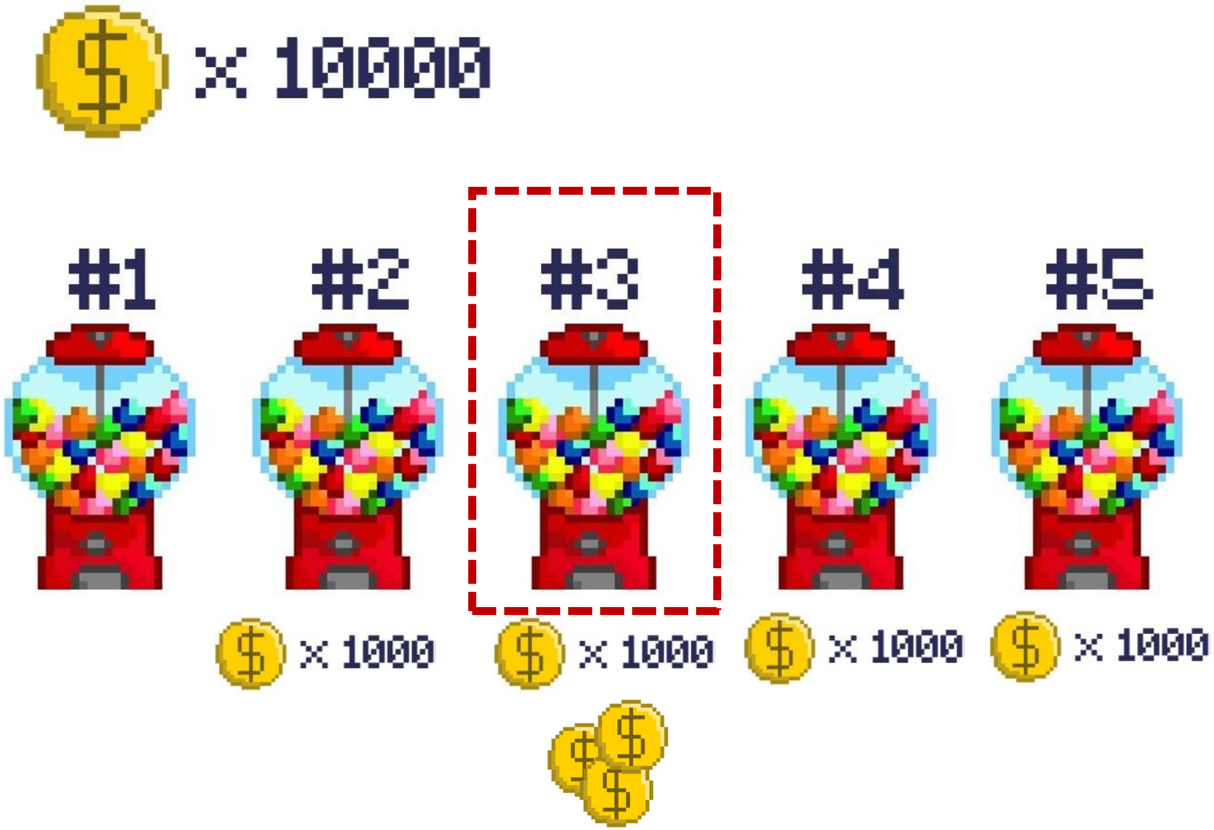


Exploration and exploitation tradeoff

Definition (exploration vs. exploitation tradeoff):

A good RL-based AI, needs to smartly combine **exploration** and **exploitation** phases.

- **Too much exploration?** You have wasted coins trying out bad machines.

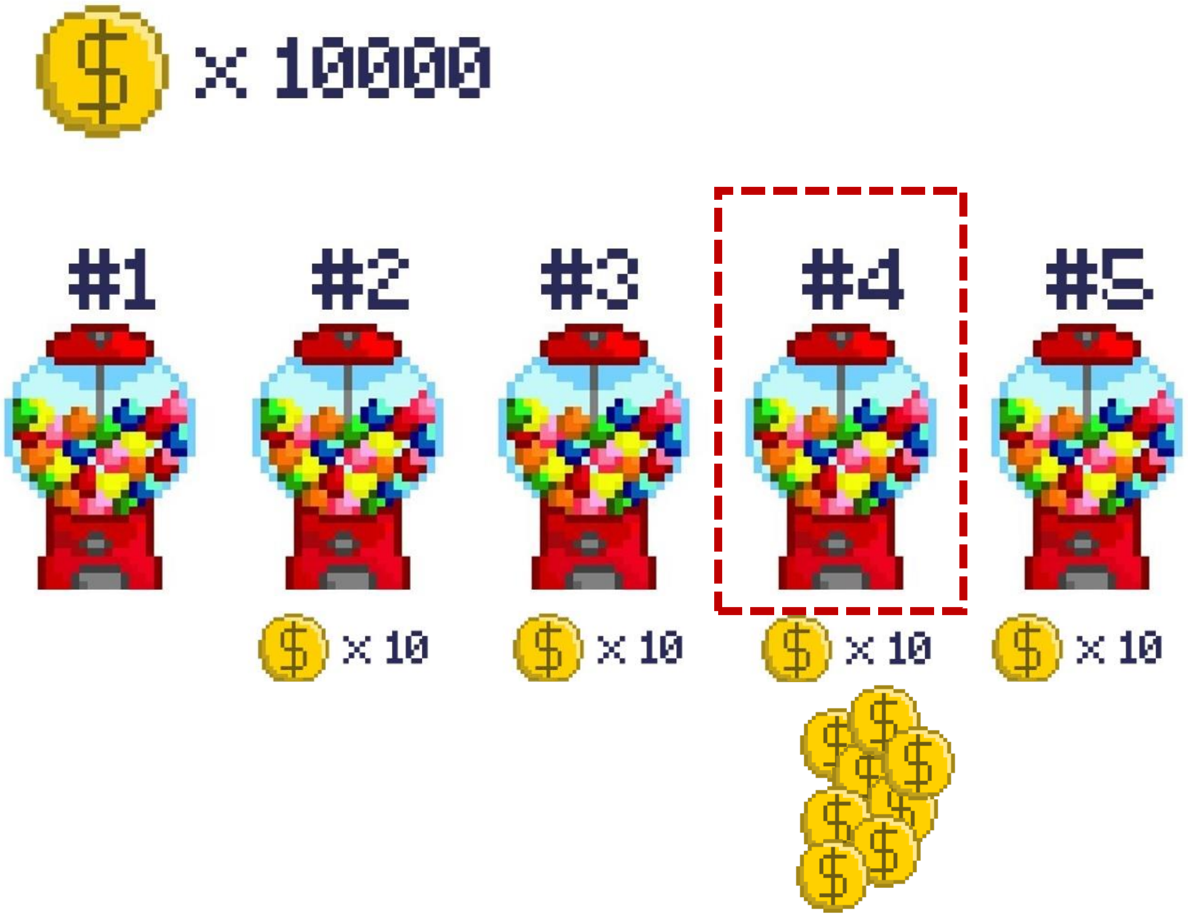


Exploration and exploitation tradeoff

Definition (exploration vs. exploitation tradeoff):

A good RL-based AI, needs to smartly combine **exploration** and **exploitation** phases.

- **Too much exploration?** You have wasted coins trying out bad machines.
- **No enough exploration?** You might end up choosing the wrong machine as the "best" one.

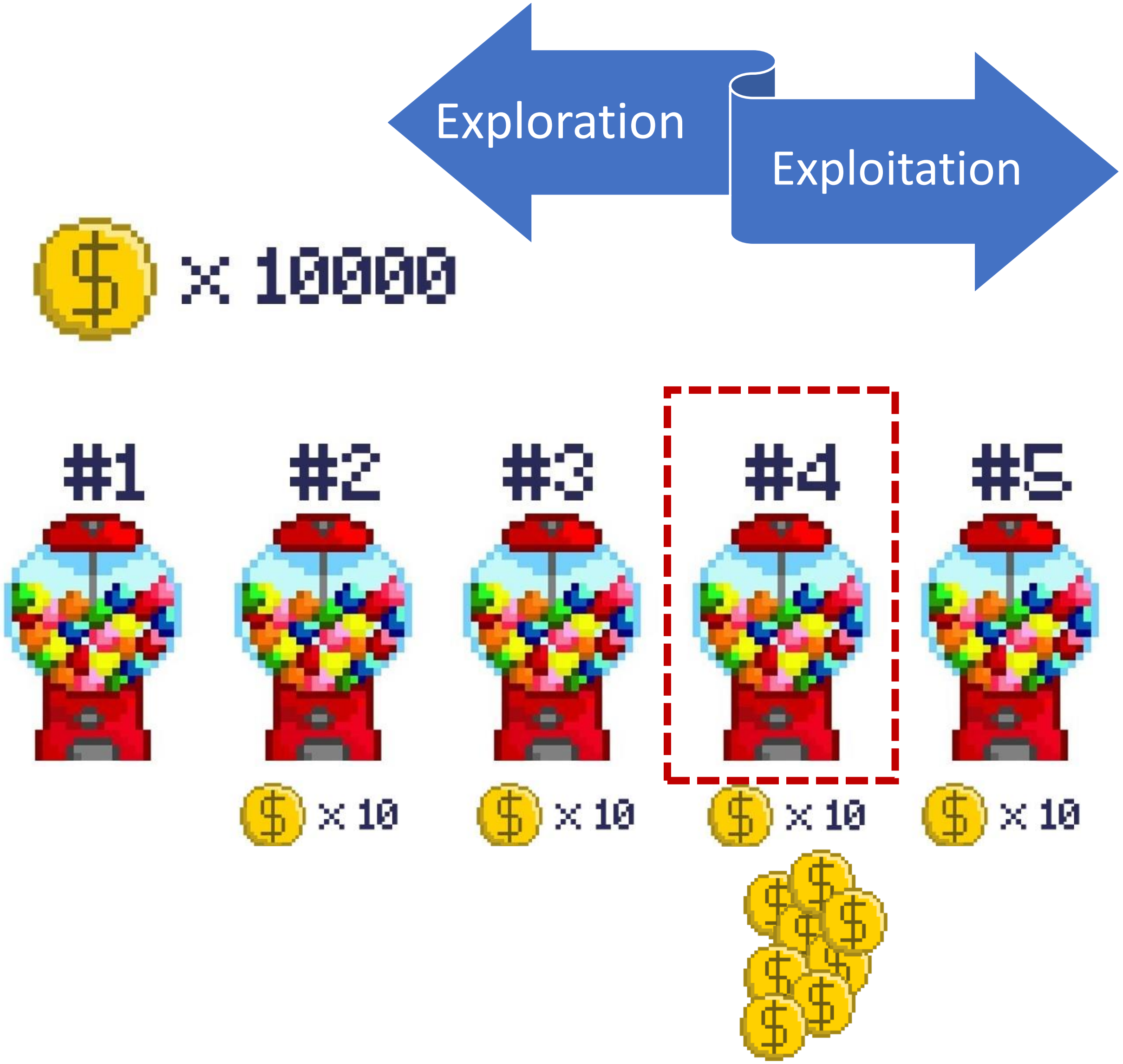


Exploration and exploitation tradeoff

Definition (exploration vs. exploitation tradeoff):

A good RL-based AI, needs to smartly combine **exploration** and **exploitation** phases.

- **Too much exploration?** You have wasted coins trying out bad machines.
- **No enough exploration?** You might end up choosing the wrong machine as the “best” one.



Reusing the RL formalism

Back to our problem…

- Let us denote $\boldsymbol{t}$, the **index of the coin to be played**.

$$\boldsymbol{t} \in \{\mathbf{1}, \dots, \mathbf{10000}\}$$

- The **set of actions** is simply

$$\boldsymbol{A} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}\}$$

- We could define our **state**, at time t, as our **current estimates of the expectations** $\boldsymbol{e_i(t)}$ of each machine $\boldsymbol{i} \in \boldsymbol{A}$.

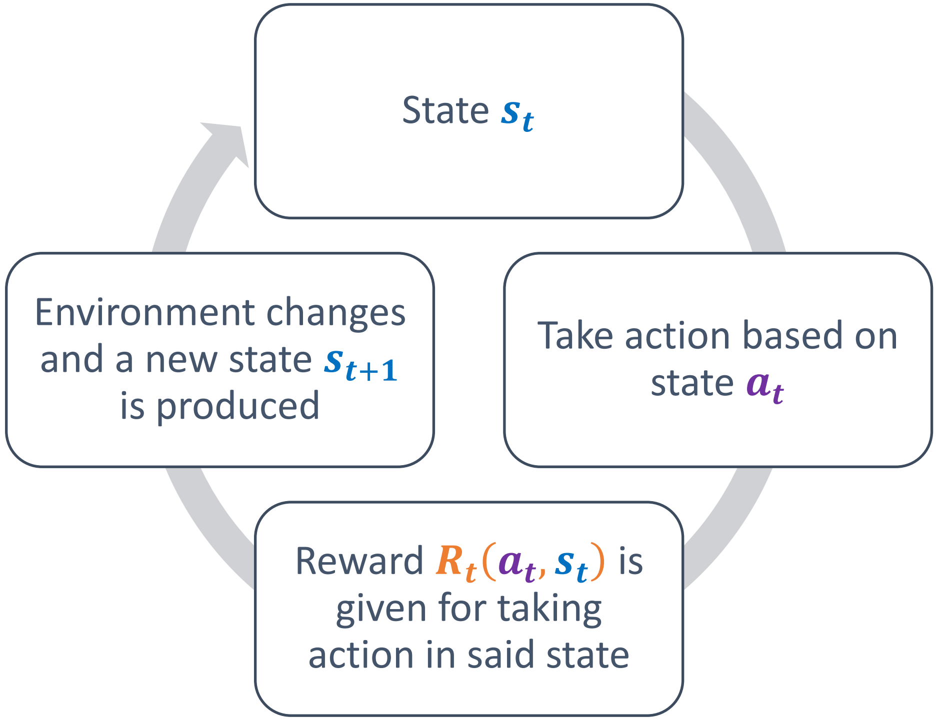
$$\boldsymbol{s_t} = (\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \boldsymbol{e_4}, \boldsymbol{e_5})$$

Reusing the RL formalism

Definition (agent):

In RL, we refer to the AI, as an agent. At each step, the agent:

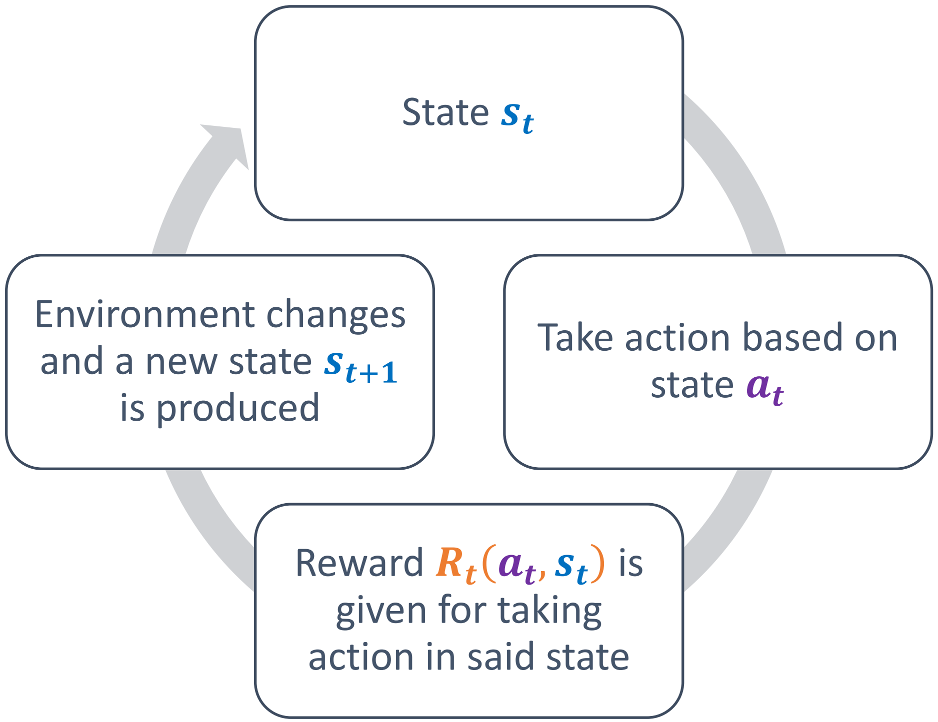
- Looks at the current **state** s_t,
- Then takes an **action**, in this given state, a_t.
- The **action** has an effect, which is eventually measured in terms of reward, R_t.



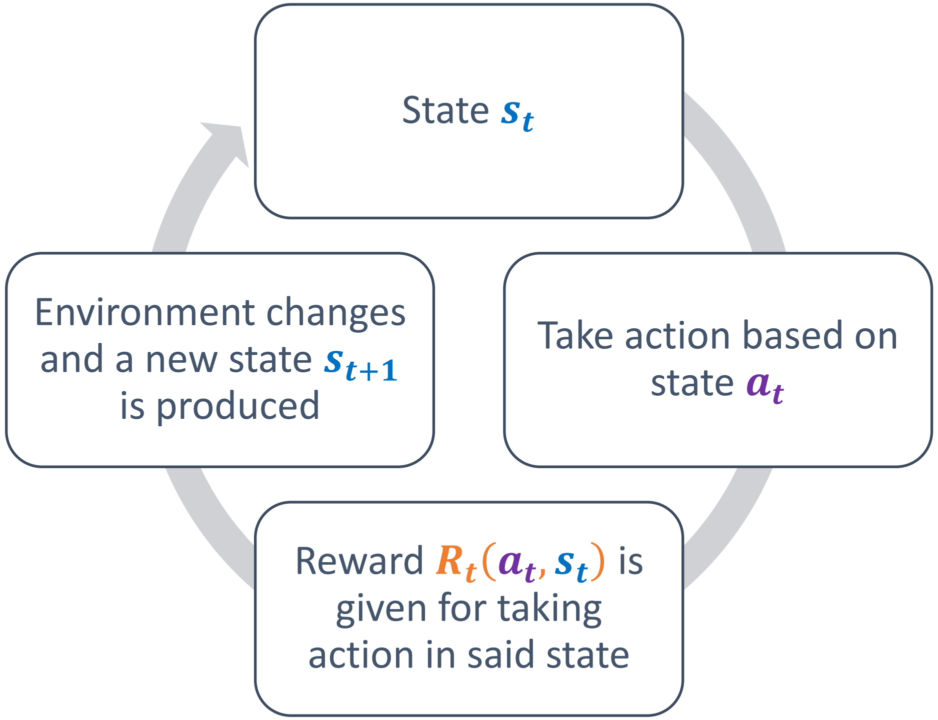
Reusing the RL formalism

Action and reward:

- The **action** $\boldsymbol{a_t = i}$, here, simply consists of **using coin $\boldsymbol{t}$, on machine $\boldsymbol{i}$**.
- The **reward** $\boldsymbol{R_t}$ is then defined **as the amount of candies obtained with coin $\boldsymbol{t}$**, randomly defined with the hidden winning probability of that machine.



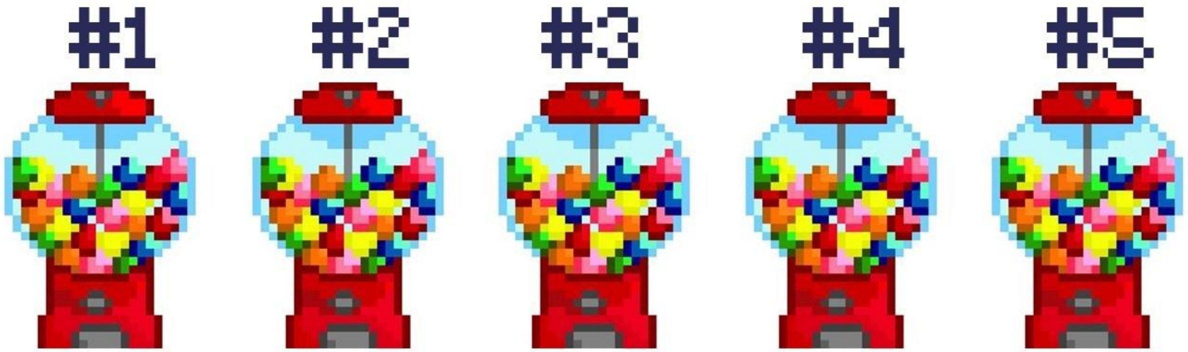
Reusing the RL formalism



State update:

- On each action, update the probability estimates, as

$$\forall i \in A, e_i = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$



This mechanism is used to update the **state** $s_t \rightarrow s_{t+1}$, after each round of the game t.

Reusing the RL formalism

Million dollar question: what is then the optimal policy $\boldsymbol{\pi}^*$, which maximizes the number of candies we can hope to obtain?

Mathematically speaking, this is equivalent to the following optimization problem

$$\pi^* = \arg\max_{\pi} \left(E\left[\sum_{t=1}^{10000} R_t(a_t = \pi_t(s_t), s_t) \right] \right)$$

Starting with three reference strategies

Three reference strategies can be considered to address this problem.

- **The deterministic strategy:** we always play the deterministic machine, and ignore the random ones.

$$\forall t, s_t, \qquad \pi_t(s_t) = 1$$

In a sense, that is the "lowest-risk" strategy, but it clearly misses the opportunity to play some "good" random machines.

Starting with three reference strategies

Three reference strategies can be considered to address this problem.

- **The naive random strategy:** we randomly decide on the machine to play, with uniform probability (20% for each of the five machine).

$$\forall t, \pi_t(s_t) = \cdots ?$$

Starting with three reference strategies

Definition (stochastic policy):

We can define a **stochastic policy**, where $\pi_t(a|s)$ is the probability to play action a in state s at time t.

$$\pi_t(a|s) = \mathrm{P}[a_t = a | s_t = s]$$

Starting with three reference strategies

Three reference strategies can be considered to address this problem.

- **The naive random strategy:** we randomly decide on the machine to play, with uniform probability (20% for each of the five machine).

Following the stochastic policy definition, $\pi_t(\mathrm{a}|\mathrm{s})$ is then defined as:

$$\forall s, \forall a, \qquad \pi_t(a|s) = \frac{1}{5}$$

Starting with three reference strategies

Three reference strategies can be considered to address this problem.

- **The deterministic strategy (v2):** we always play the deterministic machine and ignore the random ones.
- We can rewrite the deterministic strategy using the stochastic policy definition as well.
- (Note: any deterministic policy can be rewritten as a stochastic one).

$$\forall s, \qquad \pi_t(\mathrm{a}|\mathrm{s}) = \begin{cases} 1 & if\ a = 1 \\ 0 & else \end{cases}$$

Starting with three reference strategies

Three reference strategies can be considered to address this problem.

- **The perfect knowledge strategy:** Let us assume we know about the hidden probabilities of the random machines.

In that case, we can always play the machine with the highest probability (or expectation in terms of candies).

Using the stochastic policy formalism,

$$\forall s, \qquad \pi_t(a|s) = \begin{cases} 1 & if\ a = \underset{\mathrm{i} \in A}{\operatorname{argmax}}(1, 2p_2, 2p_3, 2p_4, 2p_5) \\ 0 & else \end{cases}$$

This is the upper bound performance strategy, as we cannot do better than this. It is however “cheating”, as it requires prior knowledge.

Performance of reference strategies

- Consider the following game, which includes six machines (one deterministic and five random machines), with parameters shown below.

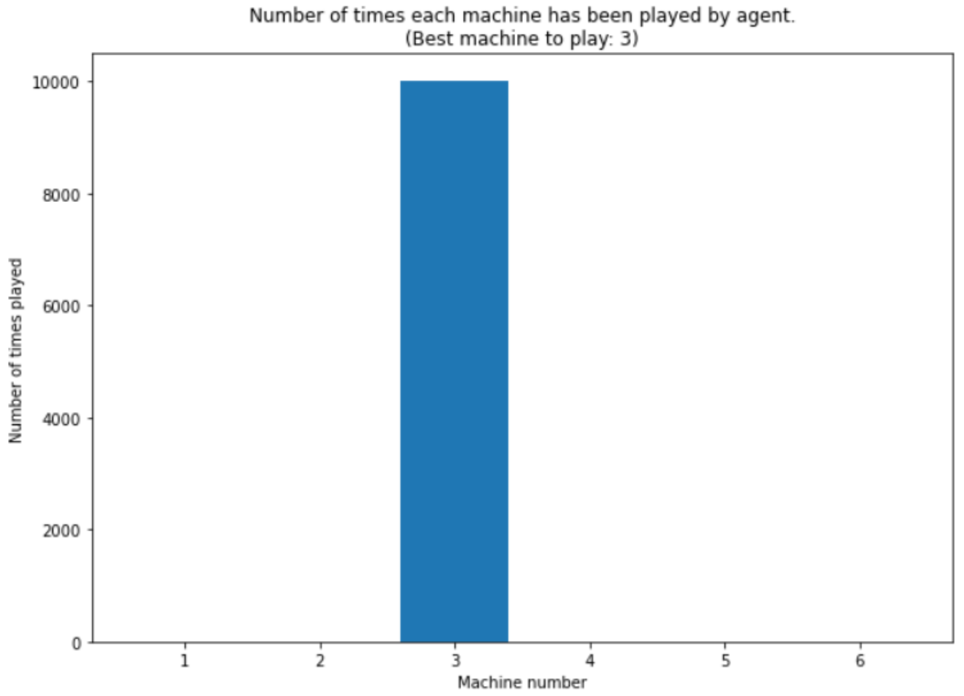
Machine_number	Machine_type	Cost	Return_win	Return_loss	Win_probability
1	deterministic	1	1	1	1
2	random	1	2	0	0.873429
3	random	1	2	0	0.968541
4	random	1	2	0	0.869195
5	random	1	2	0	0.530856
6	random	1	2	0	0.232728

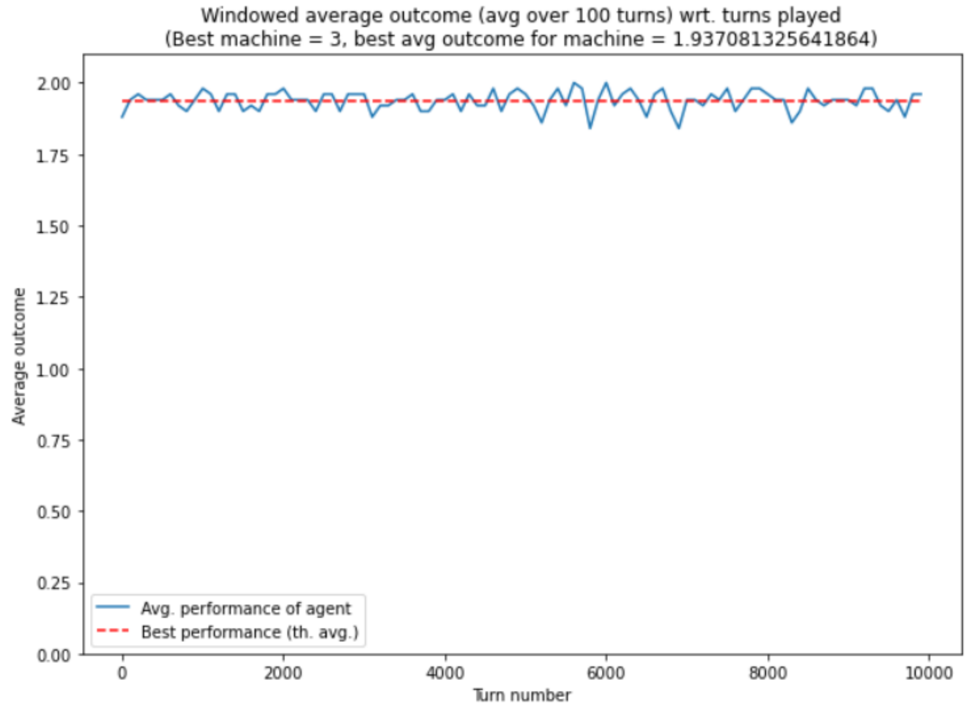
- The best machine to play is obviously **machine 3**, here.

Performance of reference strategies

As expected, the **perfect knowledge strategy** then always plays machine 3.

- It defines the **theoretical upper bound** for all strategies, i.e. the maximal number of candies one can hope to obtain.
- Unfortunately, the perfect knowledge strategy assumes it knows the hidden probabilities.
- All other strategies will not have access to such knowledge!

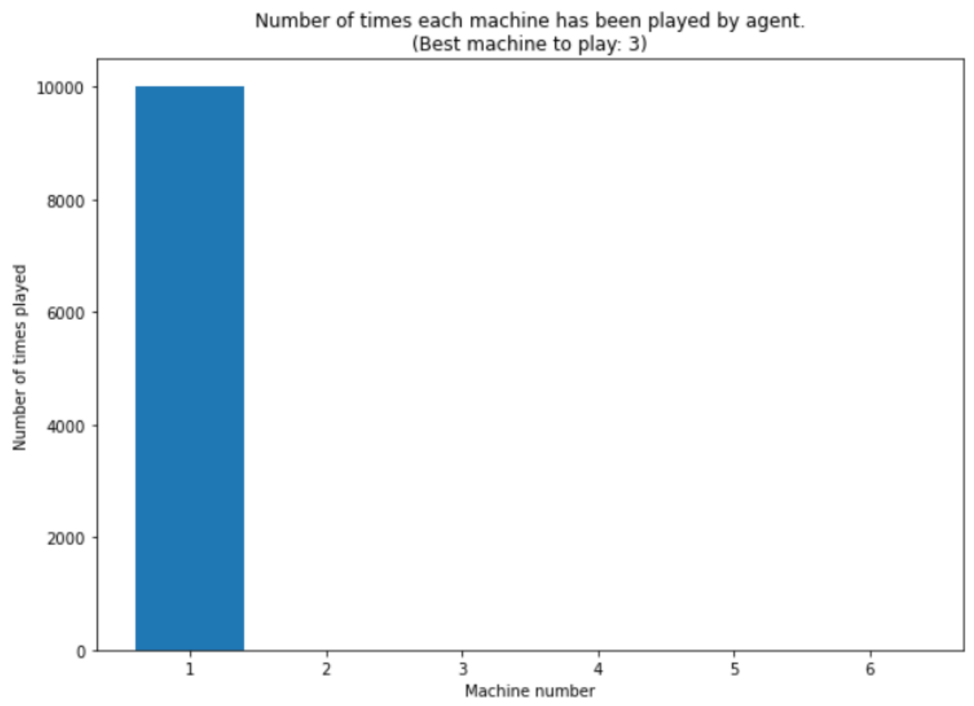


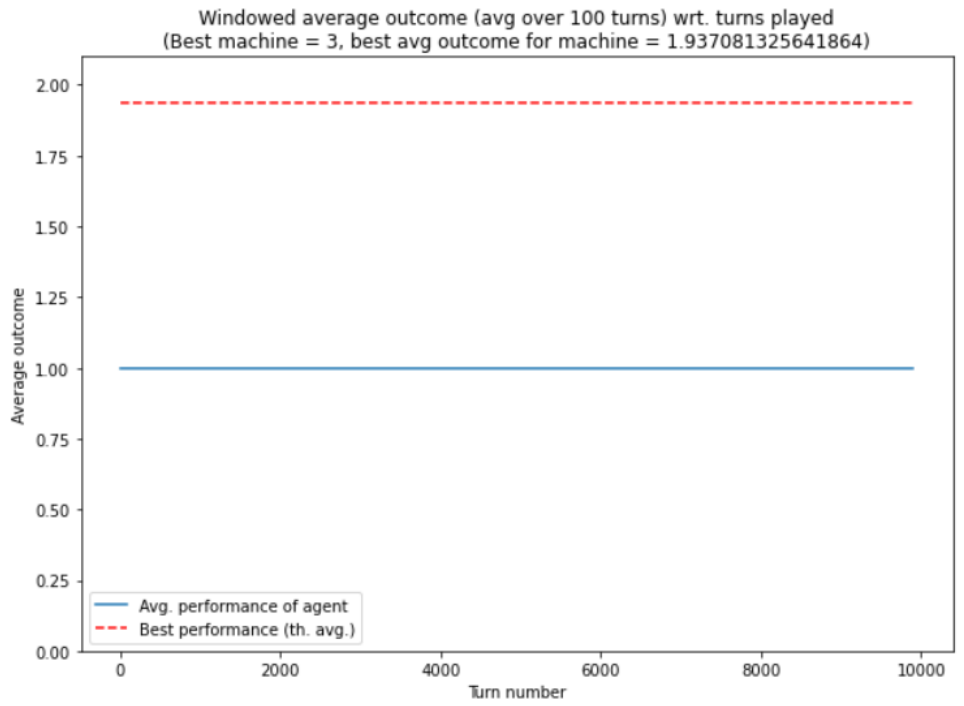


Performance of reference strategies

As expected, the **deterministic strategy** always plays on machine 1.

- Its performance is very low compared to the perfect knowledge one.
- Because, it misses on the opportunity of winning more candies by playing some random machines, with higher yield.
- It defines a **lower bound** for the performance.

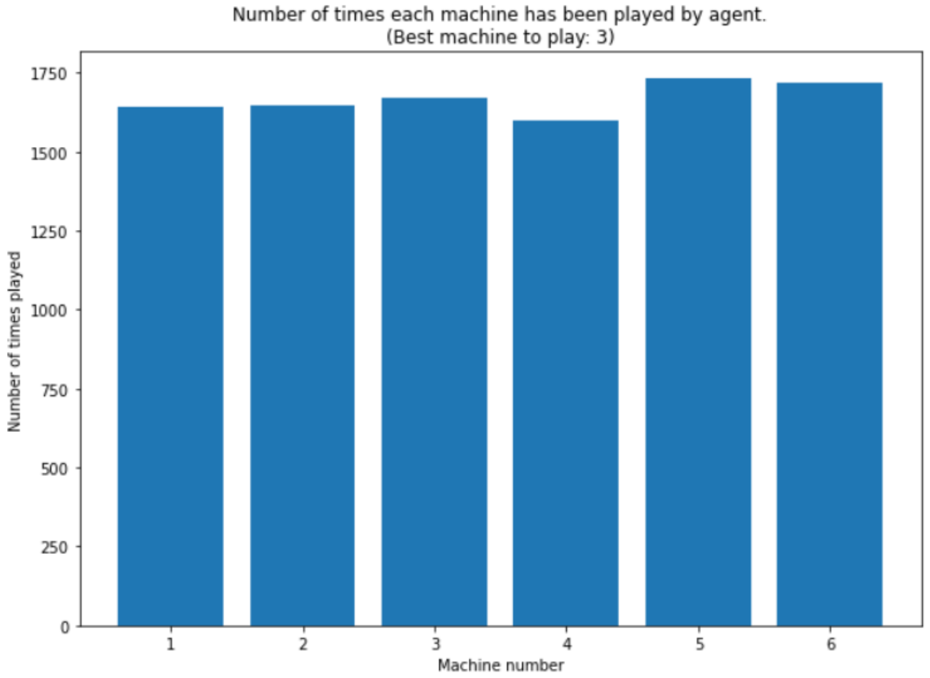


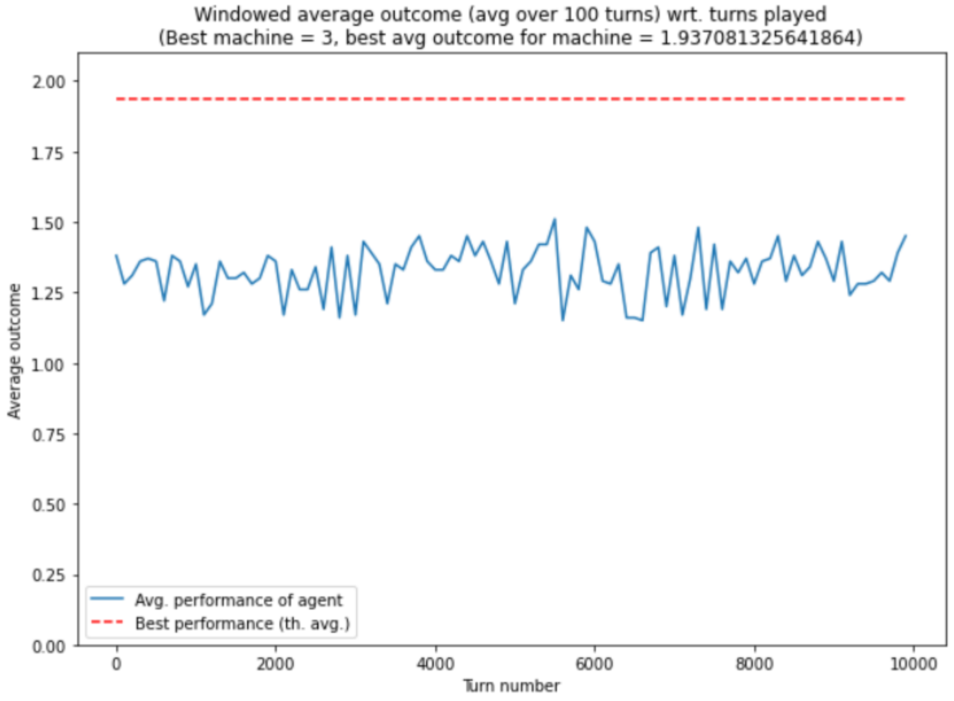


Performance of reference strategies

The **naive random strategy**, on the other hand, plays all six machines, roughly the same number of times.

- **Note:** it performs slightly better than the deterministic one. But this is due to the fact that the random machines were - on that particular run - initialized with probabilities/yields roughly better than the deterministic one.



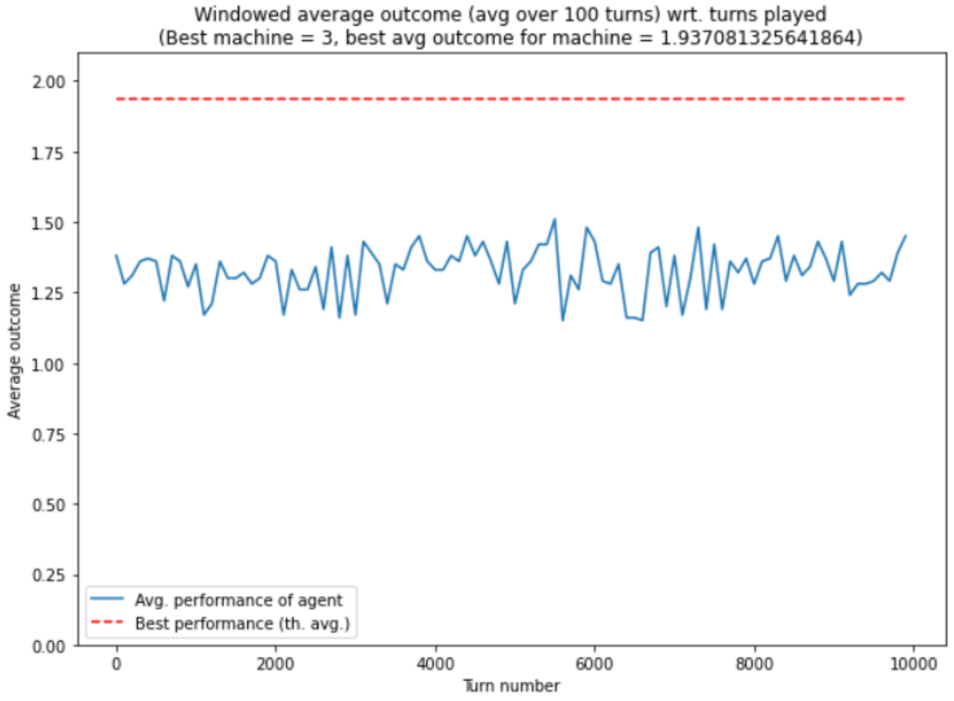


Performance of reference strategies

Machine_number	Machine_type	Cost	Return_win	Return_loss	Win_probability
1	deterministic	1	1	1	1
2	random	1	2	0	0.873429
3	random	1	2	0	0.968541
4	random	1	2	0	0.869195
5	random	1	2	0	0.530856
6	random	1	2	0	0.232728

The **naive random strategy**, on the other hand, plays all six machines, roughly the same number of times.

- **Note:** it performs slightly better than the deterministic one. But this is due to the fact that the random machines were - on that particular run - initialized with probabilities/yields roughly better than the deterministic one.



Performance of reference strategies

Overall, our deterministic and basic random strategies struggle because

- The deterministic one refuses to explore before exploiting, as it plays it “safe”, using the deterministic machine only.
- It then misses on the opportunity to play random machines, which might have better performance.

- The naive random strategy explores all machines but is unable to exploit any knowledge it might obtain from the exploration.
- More advanced strategies will have to combine exploration and exploitation in an intelligent manner.

Exploring and exploiting (strategy #1)

Definition (*ϵ-first strategy*):

In the **ϵ-first strategy**, we have two phases.

- **Exploration:** use a proportion $\epsilon \in [0,1]$ of our 10000 coins, randomly distributed over all the random machines, with a uniform distribution.
- **Exploitation:** then, use all remaining coins on the machine with the highest estimate $e_i(\mathrm{t})$.

$$\forall i \in A, e_i(t) = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$

Exploring and exploiting (strategy #1)

Definition (*ϵ*-first strategy):

The ***ϵ*-first strategy**, has the following stochastic policy.

$$\forall s, \forall a \in A, \qquad \pi_t(a|s) = \begin{cases} 1/5 & if\ t \leq 10000\epsilon \\ 1 & \begin{array}{c} if\ t > 10000\epsilon \\ and\ a = \arg\max_{i \in A}(e_i(t)) \end{array} \\ 0 & else \end{cases}$$

$$with, \forall i \in A, e_i(t) = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$

Exploring and exploiting (strategy #1)

Definition (*ϵ*-first strategy):

The **ϵ-first strategy**, has the following stochastic policy.

$$\forall s, \forall a \in A, \qquad \pi_t(a|s) = \begin{cases} \boxed{1/5 \qquad\qquad if\ t \leq 10000\epsilon} \\ 1 \qquad \begin{matrix} if\ t > 10000\epsilon \\ and\ a = \arg\max_{i \in A}(e_i(t)) \end{matrix} \\ 0 \qquad\qquad else \end{cases}$$

Exploration phase, using the naïve random strategy from earlier

$$with, \forall i \in A, e_i(t) = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$

Exploring and exploiting (strategy #1)

Exploitation phase, similar to the perfect knowledge strategy, but using our estimates to make a decision.

Definition (*ϵ*-first strategy):

The *ϵ*-first strategy, has the following stochastic policy.

$$\forall s, \forall a \in A, \qquad \pi_t(a|s) = \begin{cases} 1/5 & if\ t \leq 10000\epsilon \\ 1 & \begin{array}{c} if\ t > 10000\epsilon \\ and\ a = \arg\max_{i \in A}(e_i(t)) \end{array} \\ 0 & else \end{cases}$$

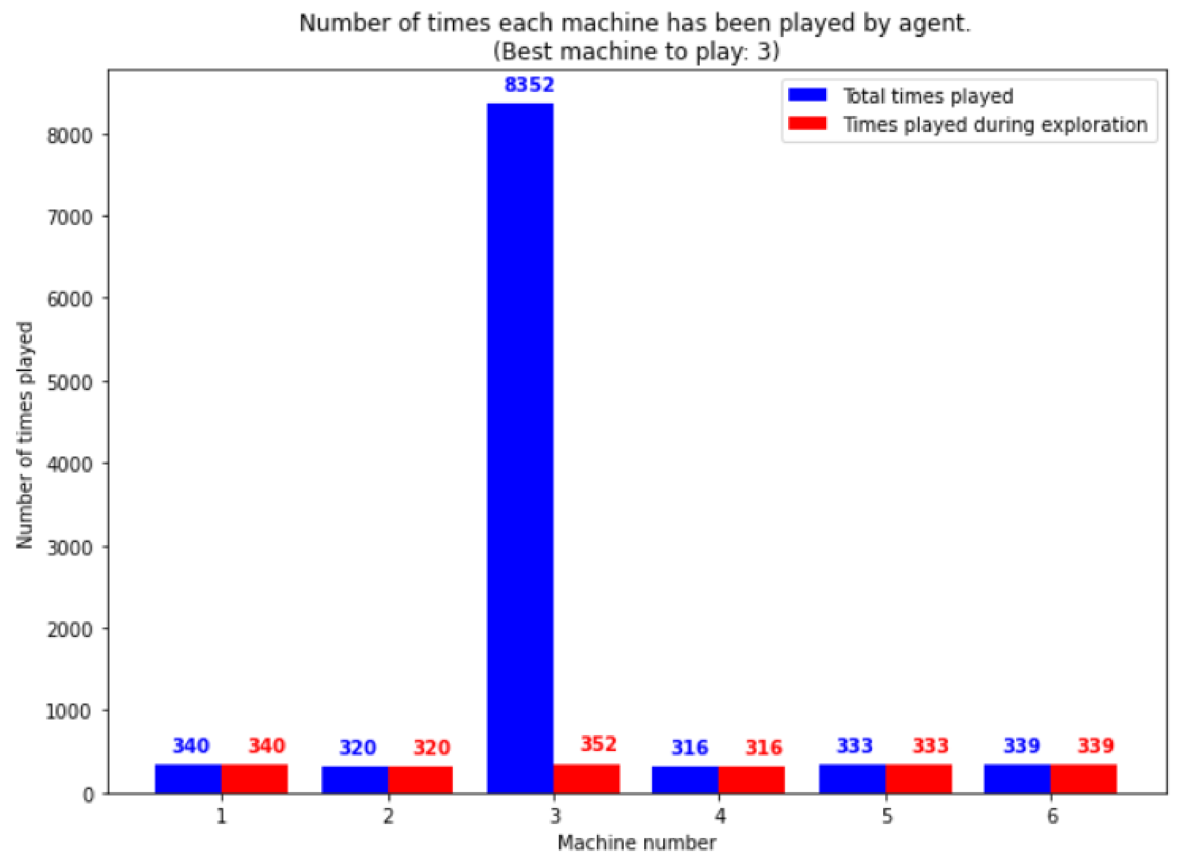
$$with, \forall i \in A, e_i(t) = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$

Exploring and exploiting (strategy #1)

Definition (ϵ-first strategy):

In the ϵ-**first strategy**, we have two phases.

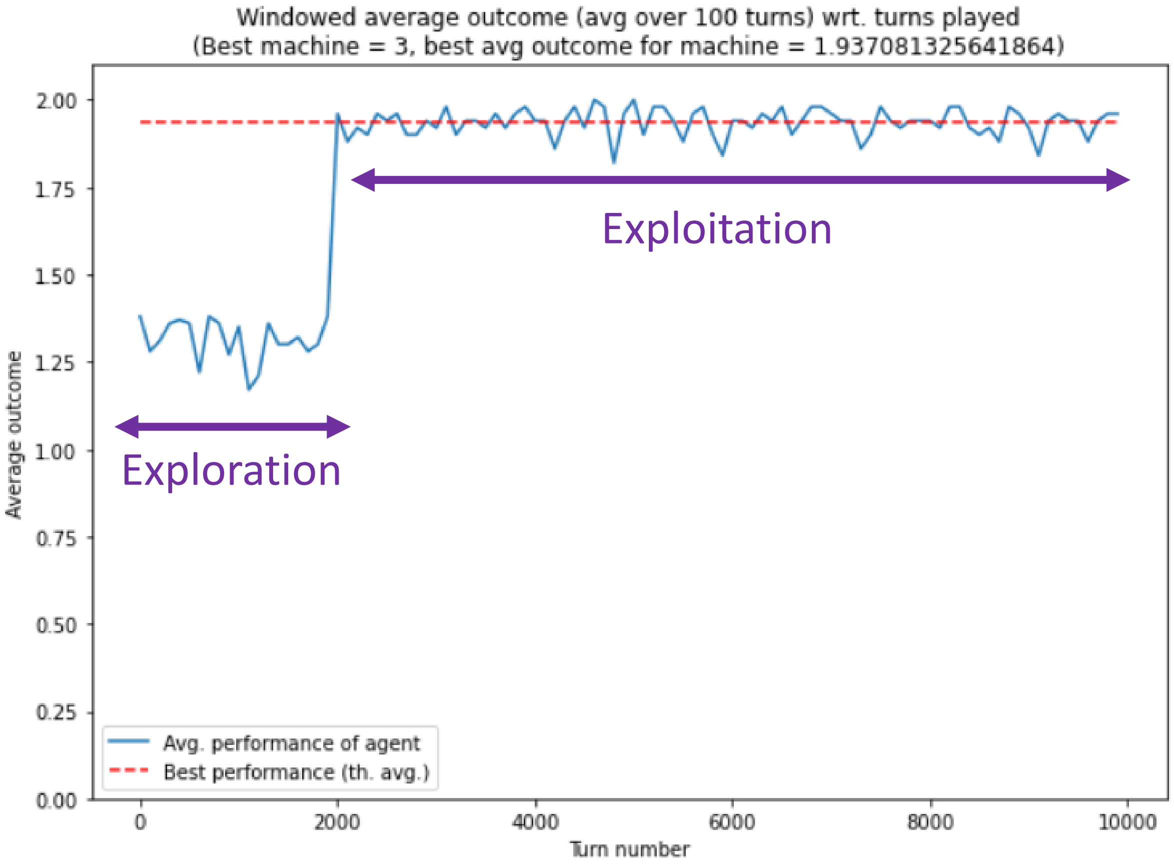
- **Exploration:** use a proportion $\epsilon \in [0,1]$ of our 10000 coins, randomly distributed over all the random machines, with uniform distribution.
- **Exploitation:** then, use all remaining coins on the machine with the highest estimate $e_i(t)$.



Exploring and exploiting (strategy #1)

The ***ϵ*-first strategy**, explores by using 2000 coins.

After the exploration phase, it is then able to identify the best machine immediately and play it with the remaining coins.

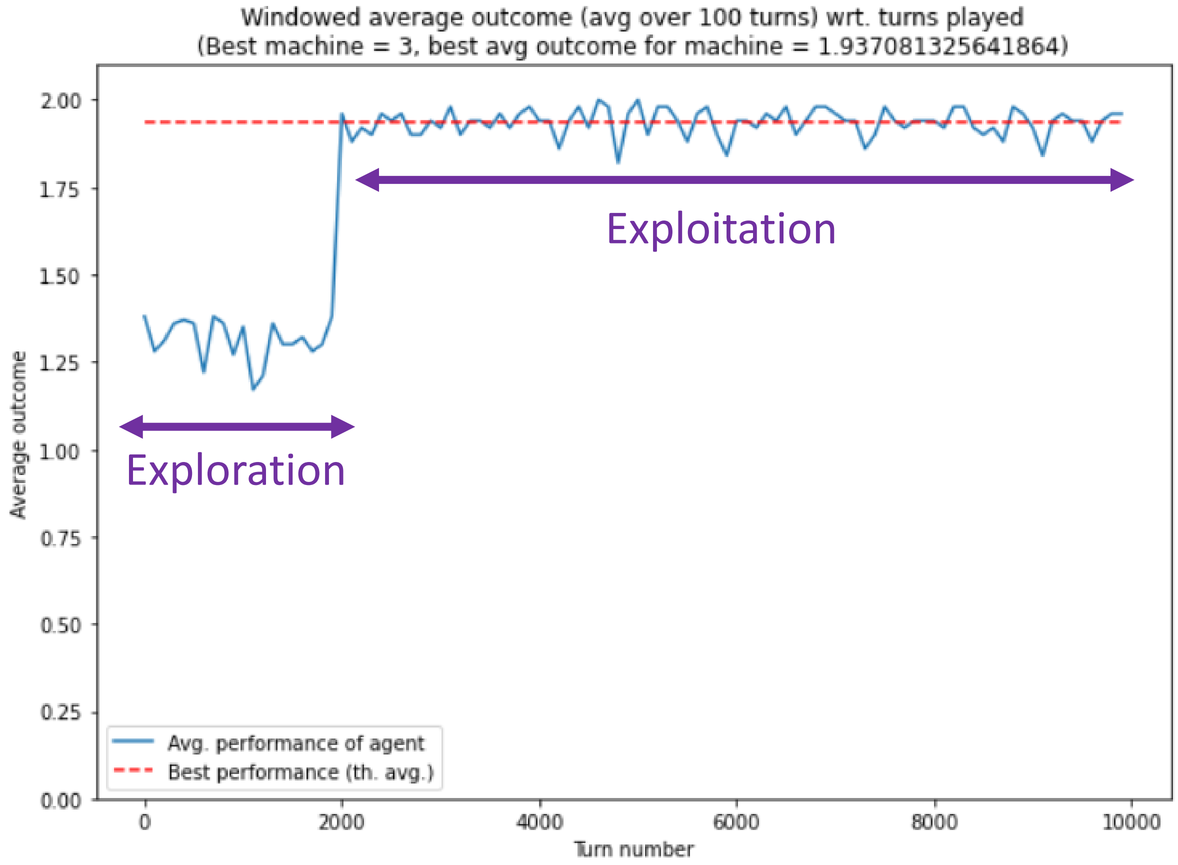


Exploring and exploiting (strategy #1)

The ϵ**-first strategy**, explores by using 2000 coins.

After the exploration phase, it is then able to identify the best machine immediately and play it with the remaining coins.

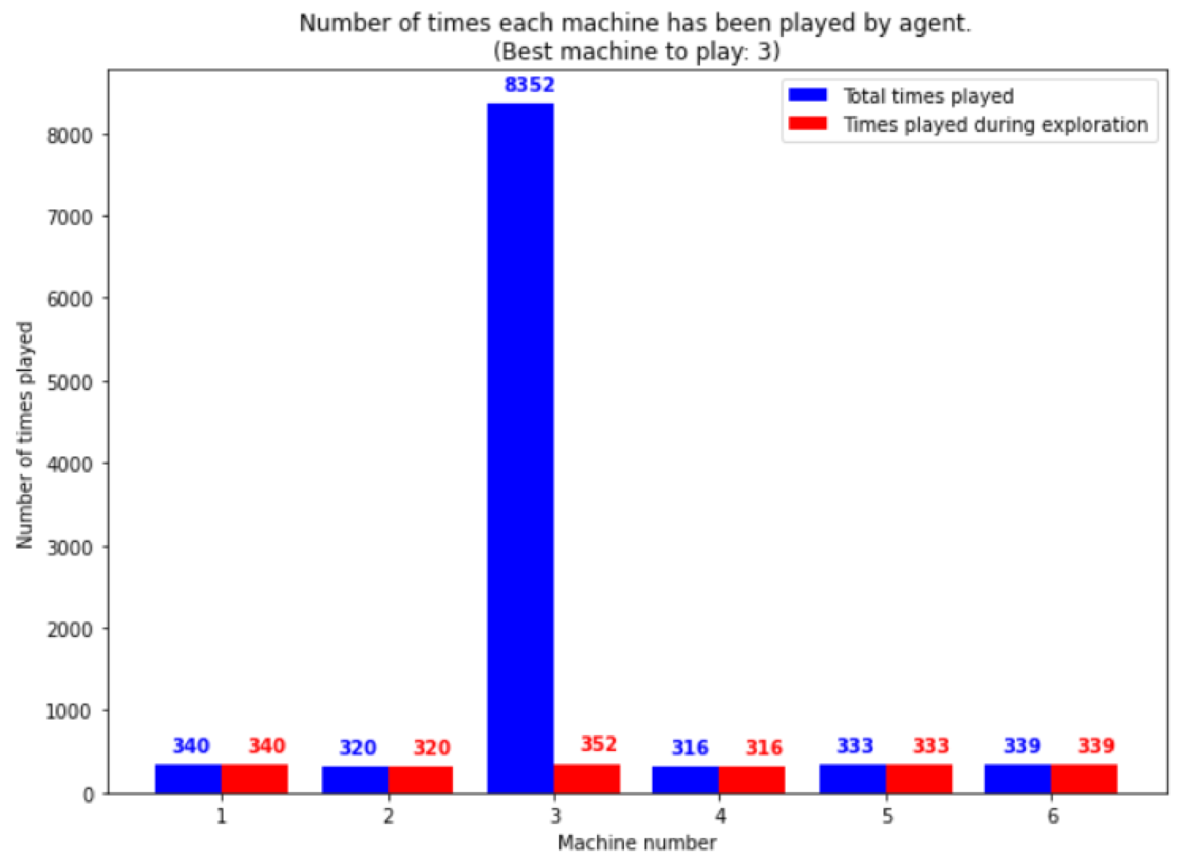
Ultimately, the objective is to explore for a while, and then go into exploitation, eventually matching the performance of the upper bound strategy.



Exploring and exploiting (strategy #1)

Problem: during the exploration phase, we keep on using coins on machines, which have shown poor results.

- It would be preferable to explore, by giving more coins to machines which have shown good results.
- Dropping machines, which feel bad, therefore not wasting any more coins on them.



Exploring and exploiting (strategy #2)

Definition (*ϵ-first strategy with softmax exploration*):

In the *ϵ-first strategy with softmax exploration*, we have two phases.

- **Exploration:** use a proportion $\epsilon \in [0,1]$ of our 10000 coins, randomly distributed over all the random machines, with softmax distribution.

$$\forall t \leq 10000.\epsilon, \forall i \in A, \qquad \pi_t(i|s) = \frac{\exp\big(e_i(t)\big)}{\sum_{k \in A} \exp\big(e_k(t)\big)}$$

- **Note:** if we have never played the machine before, initialize it as $e_i(t) = 1$, which is the same value as the deterministic machine.

Exploring and exploiting (strategy #2)

Definition (*ϵ-first strategy with softmax exploration*):

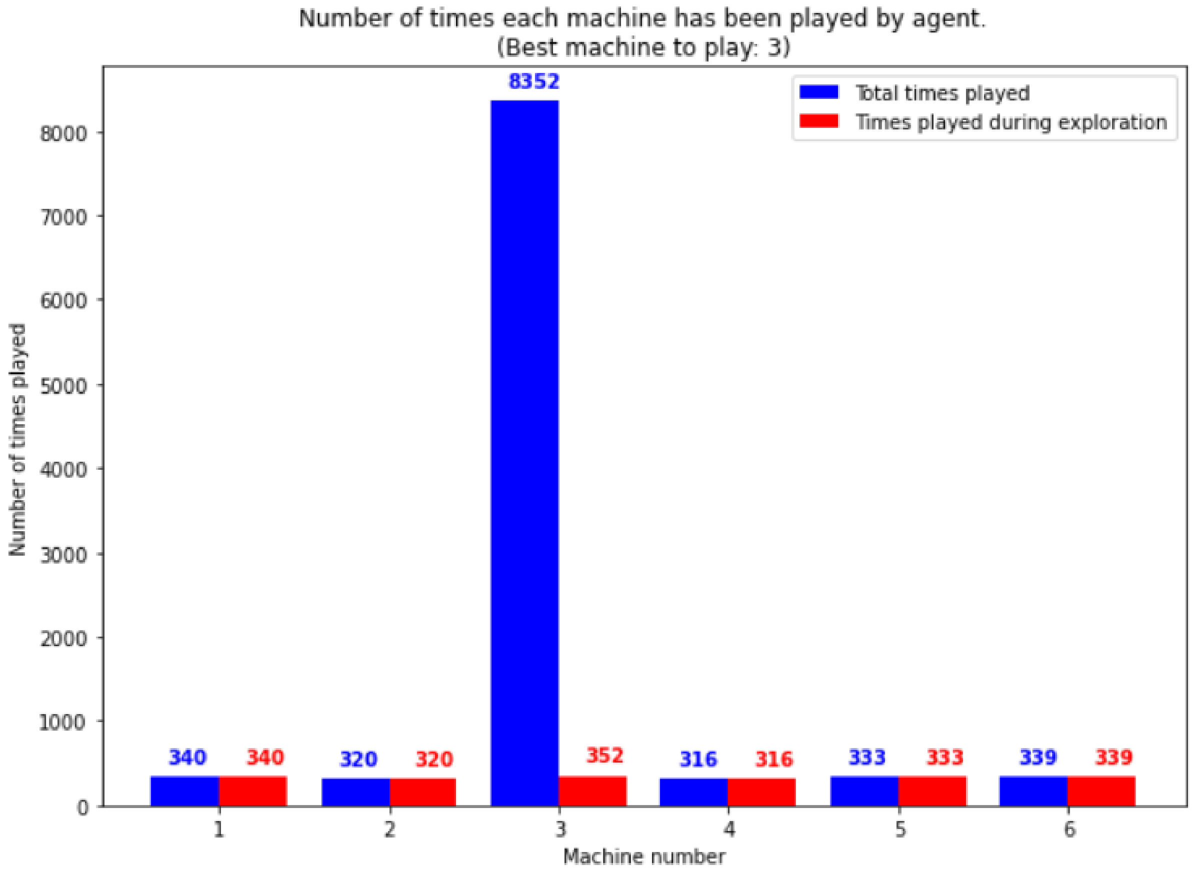
In the **ϵ-first strategy with softmax exploration**, we have two phases.

- **Exploration:** use a proportion $\epsilon \in [0,1]$ of our 10000 coins, randomly distributed over all the random machines, with softmax distribution.

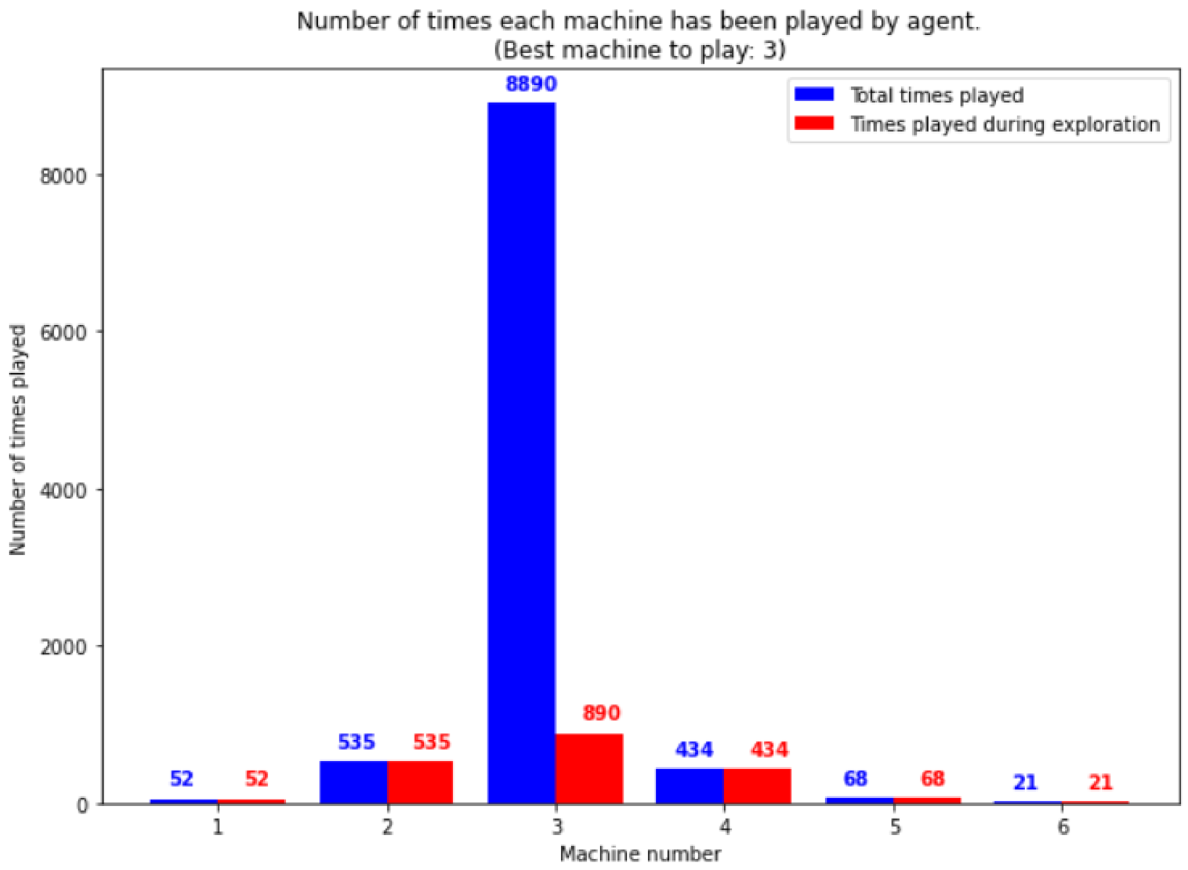
$$\forall t \leq 10000.\epsilon, \forall i \in A, \qquad \pi_t(i|s) = \frac{\exp\left(e_i(t)\right)}{\sum_{k \in A} \exp\left(e_k(t)\right)}$$

- **Exploitation:** then, use all remaining coins on the machine with the highest estimate $e_i(\mathrm{t})$.

Exploring and exploiting (strategy #2)

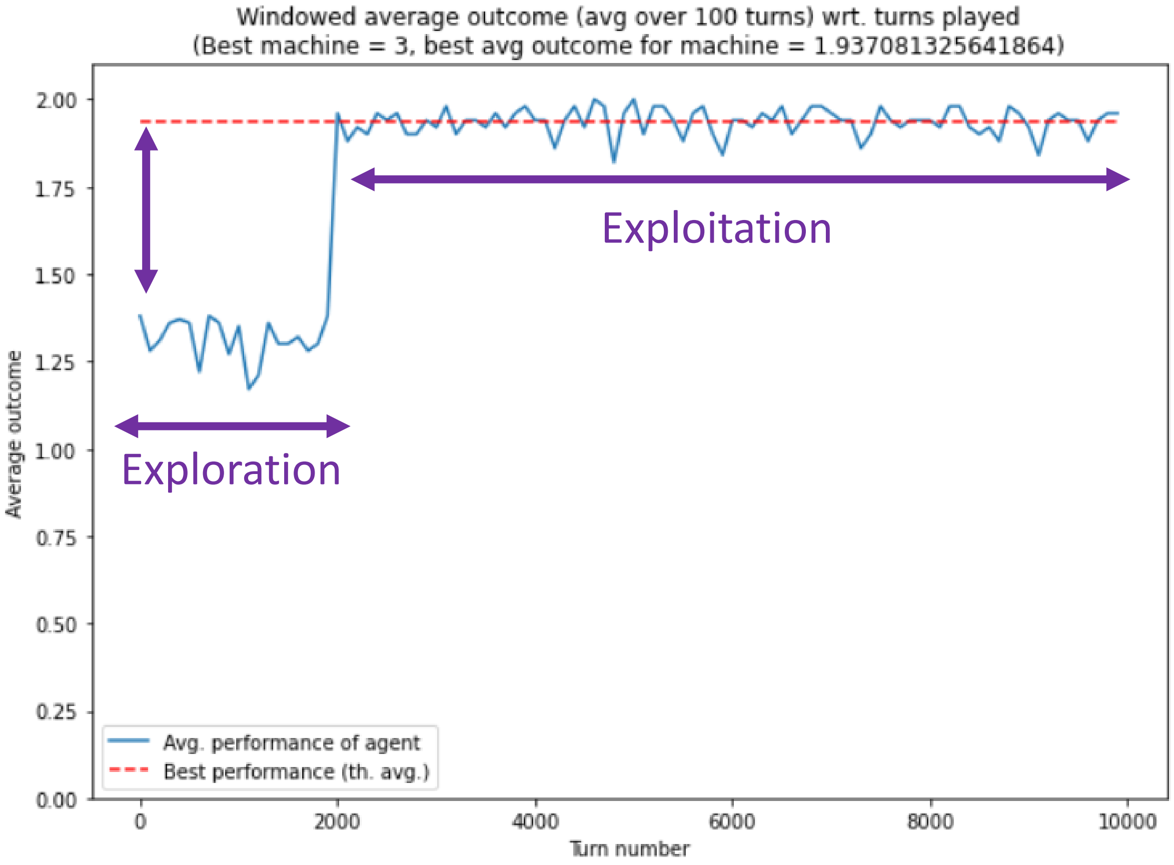


ϵ-first strategy

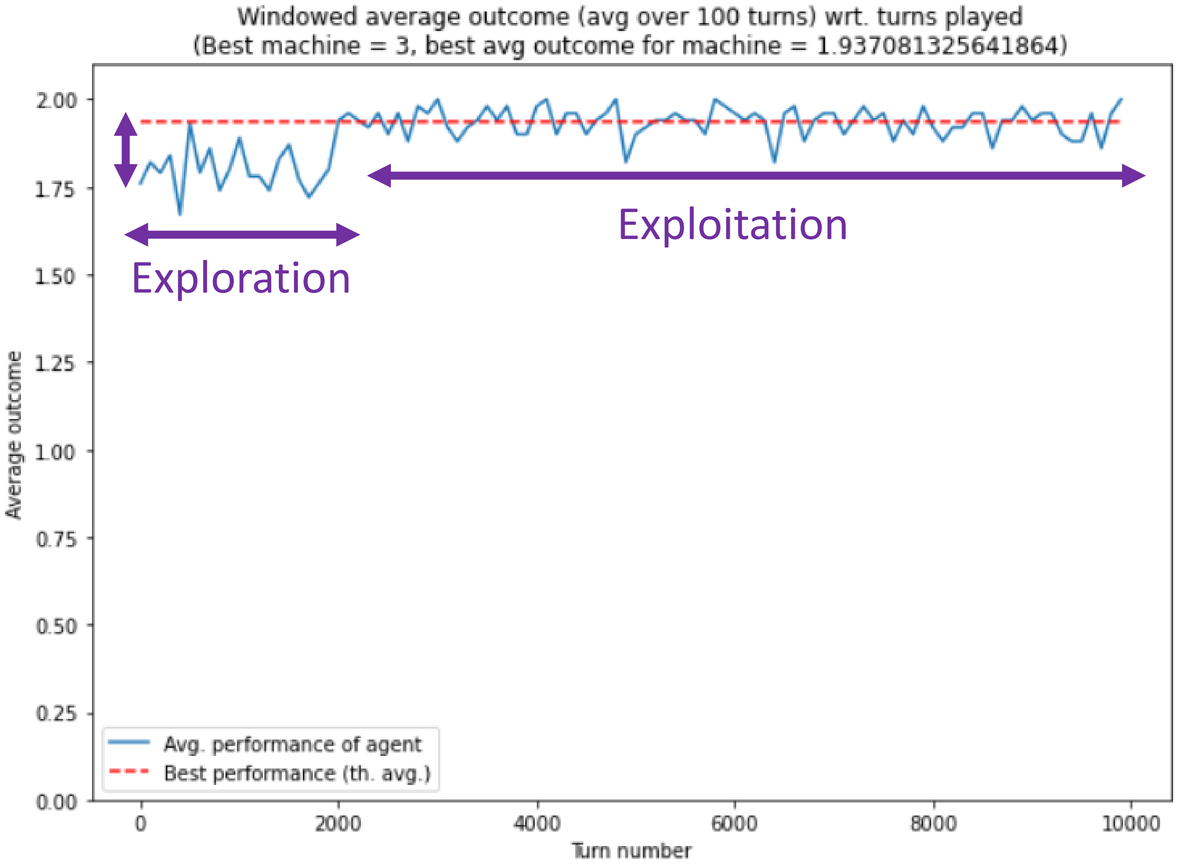


ϵ-first strategy with softmax exploration

Exploring and exploiting (strategy #2)



ϵ-first strategy



ϵ-first strategy with softmax exploration

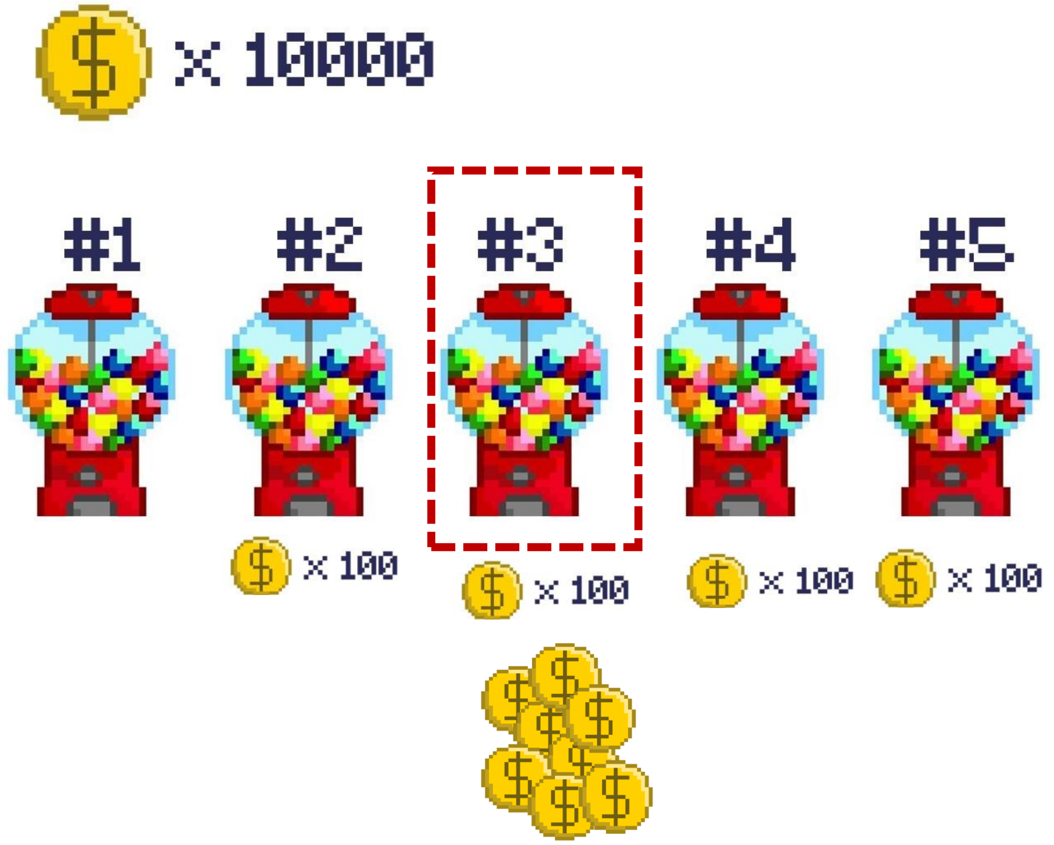
On regret and "cost of knowledge"

Definition (regret):

Unfortunately, we will always be wasting coins to explore, i.e. understand that some machines are bad and should not be played during the exploitation phase.

We define the regret, as the **quantity of candies we have lost**, during the exploration phase.

It is typically the **performance difference between a given strategy and the perfect knowledge one**.



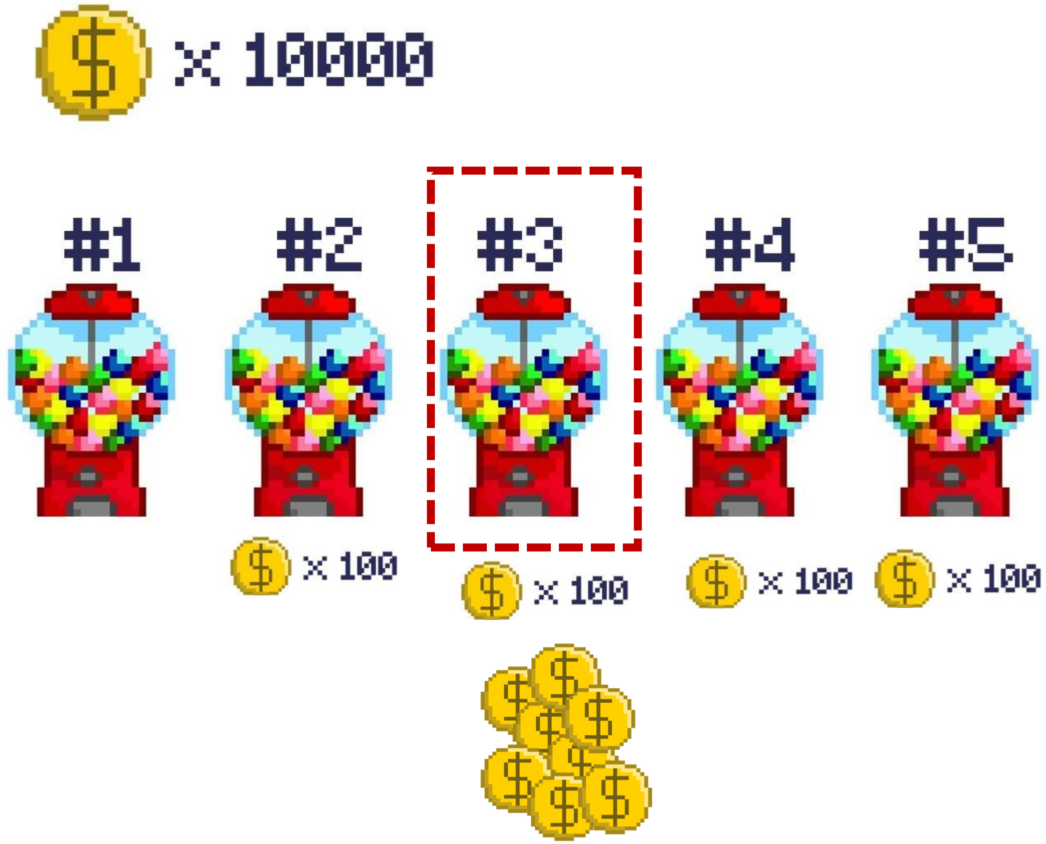
On regret and "cost of knowledge"

Definition ("cost of knowledge"):

The regret is the number of candies we have to **waste to obtain information** about the hidden probabilities of the machines.

Objective: strictly equivalent to find the strategy, which minimizes regret.

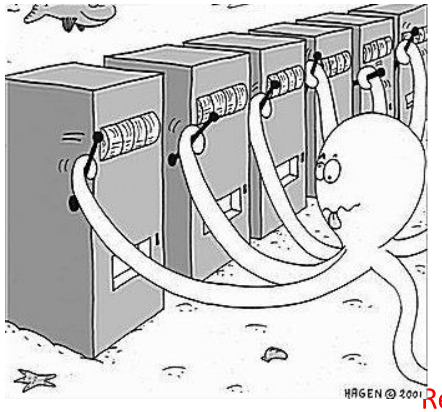
The regret cannot be zero, and **its minimal value** is the "cost of knowledge", what one must pay to acquire the missing knowledge.



Some more advanced concepts and strategies

Definition (multi-armed bandit problem):

Our random candy machine problem is a well-known mathematical problem, commonly referred to as the **multi-armed bandit problem**.

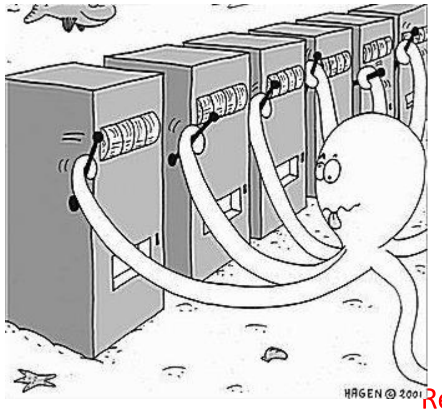


Restricted

Some more advanced concepts and strategies

Definition (multi-armed bandit problem):

Our random candy machine problem is a well-known mathematical problem, commonly referred to as the **multi-armed bandit problem**.



- It consists of is a problem in which a fixed limited set of resources must be allocated between alternative choices in a way that maximizes their expected gain.
- Each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to each possible choice.

Restricted

Some more advanced concepts and strategies

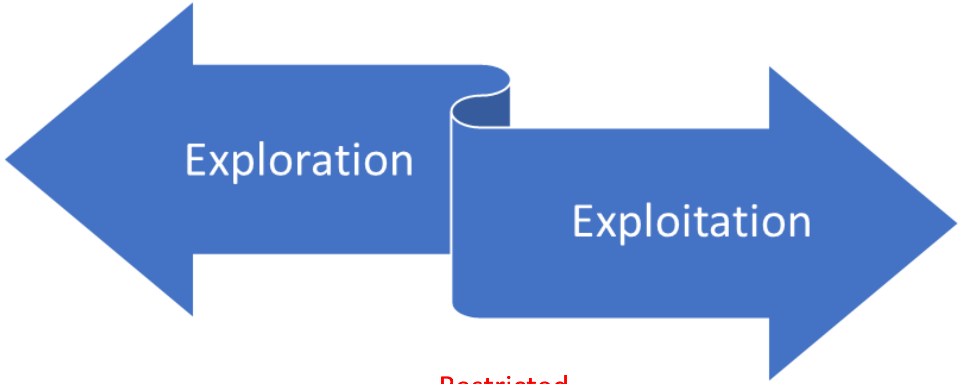
Some advanced questions

- **How to decide of the value ϵ, in the ϵ-first strategies?**

If ϵ is large, waste of coins, high regret during exploration phase.

If too low, risk of missing the “best” machine, and regret will build up during exploitation phase.

So, basically, ϵ plays the role of a **hyperparameter** here?



Some more advanced concepts and strategies

Some advanced questions

- **Should the length of the exploration phase be decided beforehand or on-the-fly, based on what is seen during the exploration?**

If we manage to identify the “best” machine early, no point in exploring machines anymore, we should transition immediately into exploitation.

If we realize we are not confident about the “best” machine to use at the end of the fixed exploration phase, maybe better to do another round of exploration?

Or even better, smoothly transition between exploration and exploitation.

Exploring and exploiting (strategy #3)

Definition (*ϵ*-greedy strategy):

In the **ϵ-greedy strategy**, we define a value $\epsilon \in [0,1]$, which decreases over time. At each time t, we decide on exploration/exploitation as:

$$\phi_t = \begin{cases} exploration & with\ prob.\ \epsilon \\ exploitation & with\ prob.\ 1-\epsilon \end{cases}$$

- **Exploration:** choose action among all the random machines, with softmax distribution over the estimates $e_i(\mathrm{t})$.

Exploring and exploiting (strategy #3)

Definition (*ϵ*-greedy strategy):

In the **ϵ-greedy strategy**, we define a value $\epsilon \in [0,1]$, which decreases over time. At each time t, we decide on exploration/exploitation as:

$$\phi_t = \begin{cases} exploration & with\ prob.\ \epsilon \\ exploitation & with\ prob.\ 1-\epsilon \end{cases}$$

- **Exploitation:** use coin on the machine with the current highest estimate $e_i(\mathrm{t})$.

Exploring and exploiting (strategy #3)

Definition (*ϵ*-greedy strategy):

In the **ϵ-greedy strategy**, we define a value $\epsilon \in [0,1]$, which decreases over time. At each time t, we decide on exploration/exploitation as:

$$\phi_t = \begin{cases} exploration & with\ prob.\ \epsilon \\ exploitation & with\ prob.\ 1-\epsilon \end{cases}$$

- **Decay on ϵ over time:** the value of ϵ starts with 1, and progressively decays over time, progressively becoming 0.

Exploring and exploiting (strategy #4)

Definition (Upper Confidence Bound strategy):

In the **Upper Confidence Bound strategy**, we do not use the estimates of the expectation of the machine $e_i(t)$.

$$\forall i \in A, e_i(t) = \frac{\sum_{k=1}^{t} \delta_{a_t=i} R_t}{\sum_{k=1}^{t} \delta_{a_t=i}}$$

Exploring and exploiting (strategy #4)

Definition (Upper Confidence Bound strategy):

In the **Upper Confidence Bound strategy**, we do not use the estimates of the expectation of the machine $e_i(t)$.

Instead, we choose the machine i, with the highest upper bound for the confidence interval of the statistical evaluation for machine i.

$$\forall t, a_t = \arg\max_{i \in \mathrm{A}} \left[e_i(t) + c\sqrt{\frac{\log(t)}{N_t(i)}} \right]$$

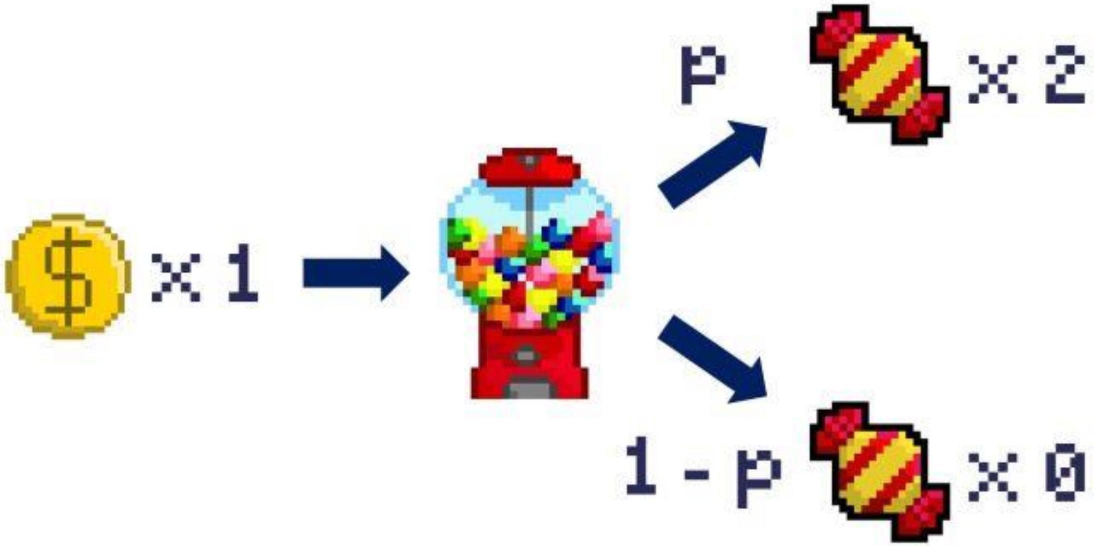
With c a parameter defining the amount of exploration and $N_t(i)$ the amount of time we have played machine i, before time t.

Non-stationarity in problems

So far, we have assumed that all parameters, describing the system (e.g. the hidden probabilities of machines) were **stationary**.

This means that they do not change over time.

- But… What if they did?
- **Example:** machines increase their win probability when you lose and vice-versa.



→ Would then have to learn the dynamics of the probabilities (how they vary) on top of everything else!

Nor

So fa
parar
(e.g.
mach

This r
chan

- But
- **Exa** **arn the**
  thei **ties**
  lose :

12-2012

Examining Slot Machine Play with Varying Percentages of Losses Disguised as Wins

Johnna L. Dunning
jld13731@siu.edu

Conclusion

1. What is Reinforcement Learning?
2. What are the key ideas behind reinforcement learning and its framework?
3. What is the exploration vs. exploitation tradeoff?
4. How do we train an RL agent by exploring, then progressively exploiting?
5. What are some advanced strategies in multi-arm bandit problems?
6. What are the Q and V functions for a RL problem?
7. What is Q-learning and how can it be implemented in RL problems?

Restricted

Learn more about these topics

Out of class, for those of you who are curious

- [TheBibleOfRL] R. **Sutton** et al., “Reinforcement learning: An Introduction, 2nd edition”, 2018.

Restricted