# A New Reinforcement Learning Algorithm for a Game-Playing Agent

Thanh D Le

## Introduction

Reinforcement learning is one of the best approaches to machine learning that resembles human experience and which also enables adaptation in unknown contexts very well. There are powerful tools to help computers understand and learn completely unfamiliar environments, but many of the algorithms in common use have weaknesses. Our research addresses one of the weaknesses by combining a reinforcement learning approach called Q-learning with another type of machine learning algorithm called Adaptive Resonance Theory (ART).

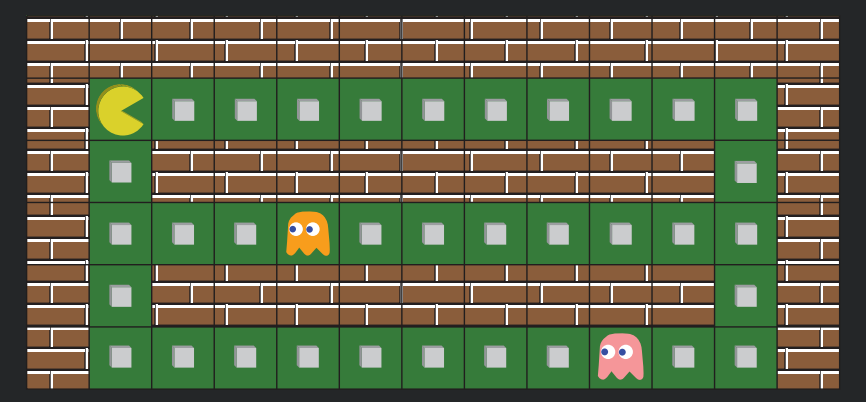
## Game Description

### *Ms. Pac-man:*

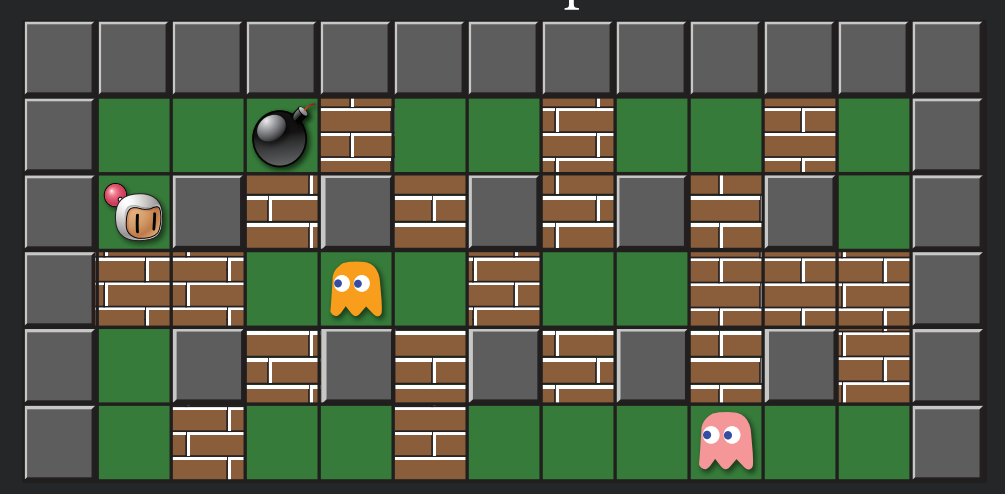
- Environment with walls, ghosts (enemies) and palettes.
- The agent available moves are UP, DOWN, LEFT and RIGHT of the squares adjacent to its position that are not wall squares.
- The agent's goal is to collect all the palettes on the board without getting caught by the ghosts.
- The game is over when all the palettes are collected by the agent (considered as a win) or the agent is captured by a ghost (considered as a lost).

### *Bomber-man:*

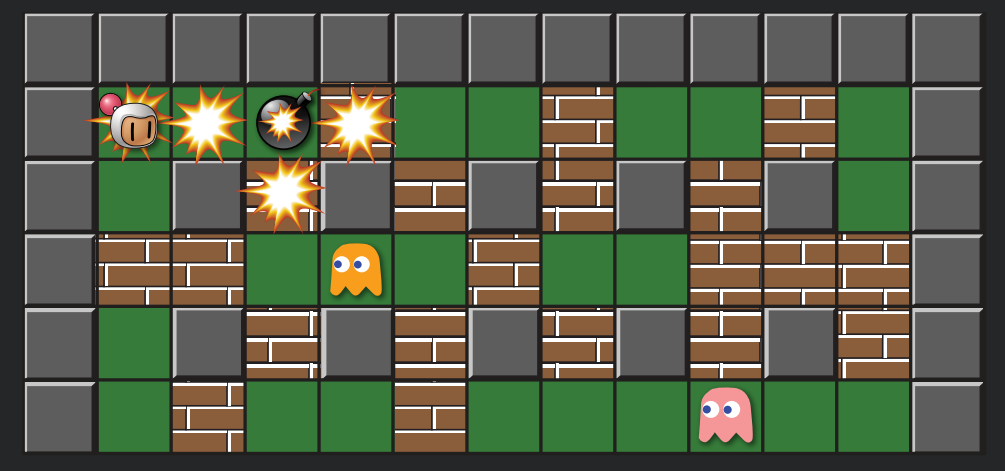
- Environment with iron walls, brick-blocks, and enemies.



- The agent available moves are UP, DOWN, LEFT, RIGHT, STANDSTILL, and PLANT BOMB (at the standing position). It cannot go pass walls, brick-blocks and bombs that it planted.



- One bomb is allowed at the time, and bombs can destroy the brick-blocks, the enemies and the agent itself but not the iron walls.
- The agent's goal is to use its bomb to destroy all



  the blocks and not get caught by the enemies and not be destroyed by its own bomb.
- The game is over when the agent successfully destroys all the brick-blocks (considered as a win) or gets caught by the enemies, or destroyed by its own bomb (considered as a lost).

## Method

### *Q-Learning*

The idea behind Q-Learning is considering the best action that might produce the best result in the future, the best expected value of the reward the agent will receive when playing that action. The expected value function of current state-action is then updated using the following formula:

$$Q(s_t,a)=Q(s_t,a) + \alpha \times [r + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a)]$$

Where:

*Q(s,a)*: The expected value of the state s when playing the action a.
*α:* The agent's learning rate
*γ:* The discount rate when calculate the expected value of the best expected reward received in the future.
*r:* The reward received in the state $s_t$ when the action a is played.

### *Adaptive Resonance Theory (ART)*

The idea behind adaptive resonance theory is the recognition of an input pattern by comparing with recognized group of clustered categories based on the similarities between them
The process of clustering the r^th input pattern is complement coding. The input vector is represented as

$$I_r=(a_1,a_2\ldots a_n,1 - a_1,1 - a_2,\ldots 1 - a_n)$$

Where: a_i is the i^(th) element of the input pattern represented by a value between 0¬ and 1.
After complement coding, vector I_r is used to calculate the category choice using all the

$$T_j(I_r) = \begin{cases} \dfrac{M}{\alpha + 2M} & \text{if node } j \text{ is uncommitted} \\ \dfrac{|I_r \wedge w_j|}{\alpha + |w_j|} & \text{if node } j \text{ is committed} \end{cases}$$

committed and uncommitted nodes:

Where:

$w_j$: is the weight vector of the j^th node.
α: is the choice parameter $\alpha \in (0,\infty)$.
$T_j$: is the value of the choice value of the current input pattern when calculate using the

$$\frac{|I_r \wedge w_{jmax}|}{|I_r|} \geq \rho$$

j^th node.
Suppose $j_{max}$ gives us the maximum T value, we will use category $j_{max}$ to make the vigilance test, which is:
(ρ is the vigilance parameter)
If the test is passed, then we can update the weight vector of the matched category by:

$$w_{jmax} = w_{jmax} \wedge I_r$$

If the matched node is uncommitted, we set that node to committed and add a new uncommitted node to the set of categories.
If node $j_{max}$ does not satisfy the vigilance criterion, we disqualify the node by setting the $T_{jmax}(I_r) = -1$, and use the next best node until we find a node that satisfy the vigilance test.

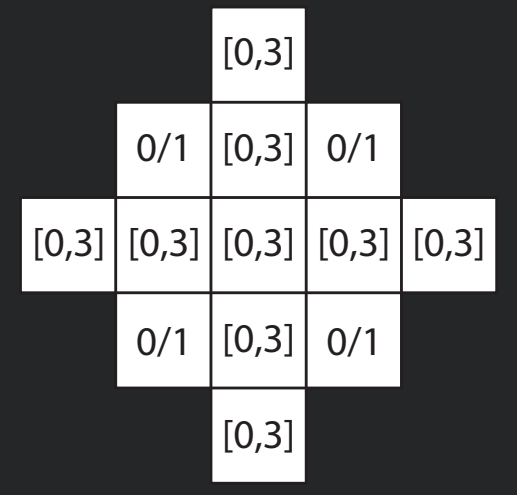
*Note*: Fuzzy Operation AND
Suppose we have two vectors $A = ( a_1, a_2, ..., a_n )$ and $B = ( b_1, b_2, ..., b_n )$ then:

$$\begin{aligned} A \wedge B &= ( a_1, a_2, ..., a_n ) \wedge ( b_1, b_2, ..., b_n ) \\ &= ( \min(a_1,b_1), \min(a_2,b_2), ..., \min(a_n,b_n) ) \end{aligned}$$

## Implementation & Result

### *Ms. Pac-man (Q-Learning)*

The most important problem we appeared during the implementation of Ms. Pac-man is how we represent a state of the game. Suppose we have an 5×5 environment, each square could have one of the four following states: Empty, palette, ghost or wall square. Therefore, we have $4^{25}=1.126\times10^{15}$ different states in this game, which cannot be store in any kind of known data structured.



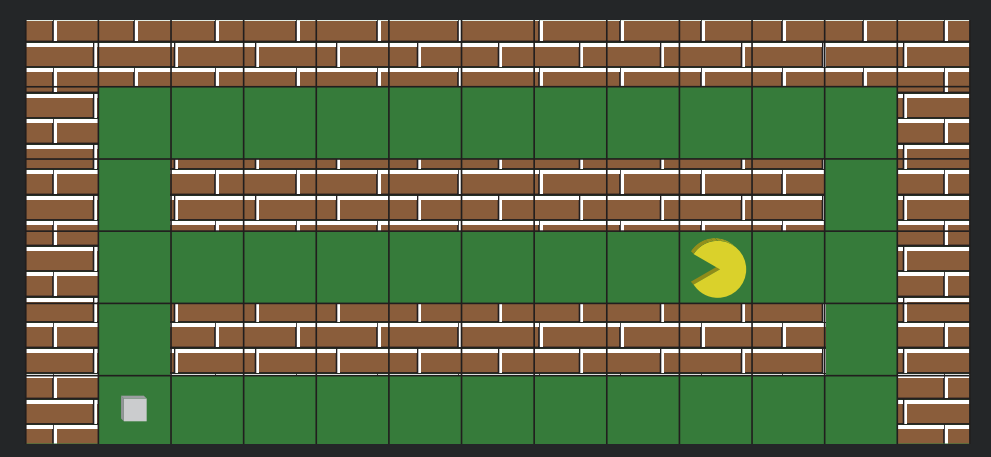
We solve this problem by giving the agent only a small area of vision around it.

Using this method of presentation, there are $2^4\times4^9=4.2$ million states, which is much smaller that the number of states calculated previously.

The reward system of Ms. Pac-man is as follow:

- The agent is reward a high positive value for winning the game, while it receives a high penalty (negative reward) when dies to ghosts.
- Each palette collected is rewarded with an amount of reward (smaller than winning reward and losing penalty).
- A small penalty is given to the agent each move.

The agent were able to escape from ghosts (knowing that ghosts are dangerous objects) and go after the palette in its vision. However, in a bigger environment, when the chance of the agent get chased by the ghost to a palette is much lower, the agent seems to get stuck in an area where there is nothing around it (no ghosts and no palette).



*The situation where there is no ghost and palette inside the agent's vision*

We came up with an extra penalty to give the agent a reinforcement when it is in such a situation, which is a penalty if the agent going back and forth a position. After the implement of the new penalty, the agent was no longer stuck in when there is nothing around it.

### *Ms. Pac-man (Q-Learning + ART)*

The problem with by means of Q-Learning alone in the previous implementation was the amount of information of the environment given to the agent is not enough for it to advance to a bigger, more complicated environment. This is where ART comes in and save the day. We represent the state by a feature vector with different aspect of the environment, and the similar patterns are clustered into a same category.

The most important information is the information of the closest ghost to the agent (since that is the most dangerous object) and the nearest palette (the goal of the agent). Our first way of represent these feature was the exact position of the closest ghost and the nearest palette. The result of this implementation was worse than applying Q-Learning alone in the same number of episodes of learning, using the exact location of the nearest ghost and palette was not general enough, since the agent will not be able to learn from similar situations.

To solve the problem mentioned above, we came up with a new way of representation of the environment which is the following vector:

$$I = (x_{ghost}, dir_{ghost}, x_{palette}, dir_{palette})$$

Where:

$x_{ghost}$: the distance between the agent and the nearest ghost
$dir_{ghost}$: the direction to the nearest ghost
$x_{palette}$: the distance between the agent and the nearest palette
$dir_{palette}$: the direction to the nearest palette

After the new representation was implemented, the agent did much better than the previous illustration of the state, and the outcome was almost the same as the implementation using Q-Learning alone in a small environment. But the result is now much better comparing to the Q-Learning method when the environment is much bigger. Specifically, the agent was able to move in the direction of the nearest palette even when no palette around it. Moreover, we did not have to add a new penalty to restrain the agent from going back and forth like the previous implementation when we only used Q-Learning.

### *Bomber-man*

Bomber-man is the next step in understanding the power of Q-Learning when combining with ART. We used the same implementation of Q-Learning and ART and apply them to a new, more complicated environment, where the agent not only have more actions each move, but it has to learn about timing since after planting a bomb, a few moves later.
Rewards and penalties as follow:

- The agent is given a high value reward when it wins a game and a high penalty when it loses a game.
- Each block destroyed will give the agent a positive reward (smaller than winning reward and losing penalty).
- Each action the agent does give it a small penalty.

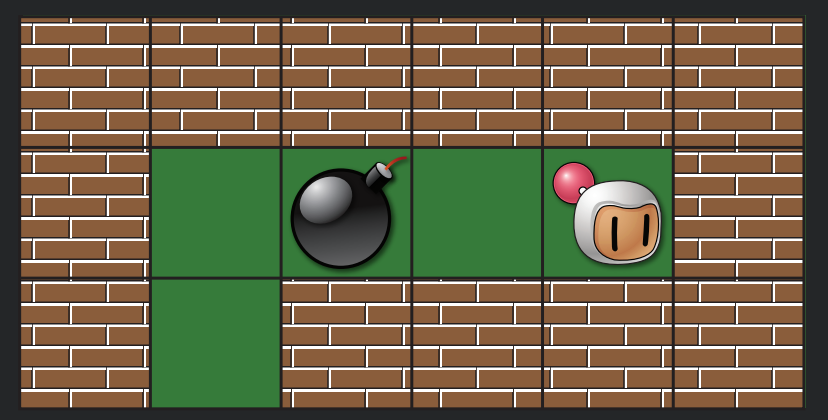
The first way to describe a state of the game was the following vector:

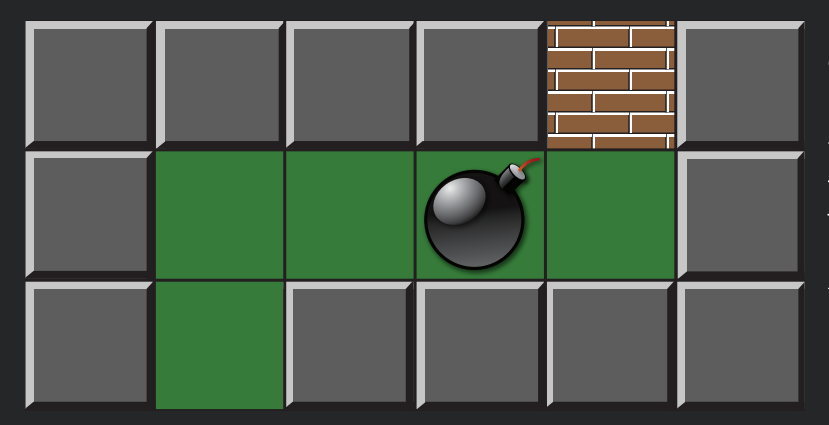
$$I=(x_{enemy}, dir_{enemy}, x_{block}, dir_{block}, x_{bomb}, dir_{bomb}, t_{bomb}, dir_{escape})$$

Where:

$x_{enemy}$: the distance between the agent and the nearest reachable enemy
$dir_{enemy}$: the direction to the nearest reachable enemy
$x_{block}$: the distance between the agent and the nearest brick-block
$dir_{block}$ the direction to the nearest brick-block
$x_{bomb}$: the distance between the agent and the bomb (since there is only one bomb available at once)
$dir_{bomb}$: the direction to the bomb
$t_{bomb}$: the time of the bomb
$dir_{escape}$: the direction to the furthest possible position from the bomb

The first problem is when the bomb is planted in the square like the following picture, and the bomb explosion size is 2, the agent sometime still goes to the right even though that square is a guaranteed dead.





The second problem is the agent try to destroy the block in the position in the following picture, where 'B' the position of the block we want to destroy and 'X' is the position where the agent planted the bomb

Previously, we calculate the distance between the agent and the objects simply by Breath First Search (BFS) since the distances between two adjacent squares are one. However, the same distance could give us different situations, so we need to use a different method. We switched to Ford-Bellman algorithm to calculate the minimum distance between the agent and objects, and the graph's weights no longer ones only, but when the agent makes a turn, the weight of the path is much longer.

Using the mentioned method of representation, the agent becomes much more aware of the environment and successfully plant bombs and escapes the bomb as expected, and its ability to escape the ghost is still very effective. After only thousands of episodes of training, the agent was even able to develop a strategy to play the game, which is trying to eliminate the ghost before destroying the brick-block. It tries to plant the bomb to separate itself from the enemy until the enemy dies by the bomb; it is then proceed to destroy the brick-blocks.

## Conclusion

Combining Q-Learning and ART open ups a lot of possibilities for artificial intelligence, helping the agent learn in different game environments and setups, using the same method of implementation, and only change in data representation. And with a lot of training, the agent could possibly beat many complicated games, even real time trategy games.