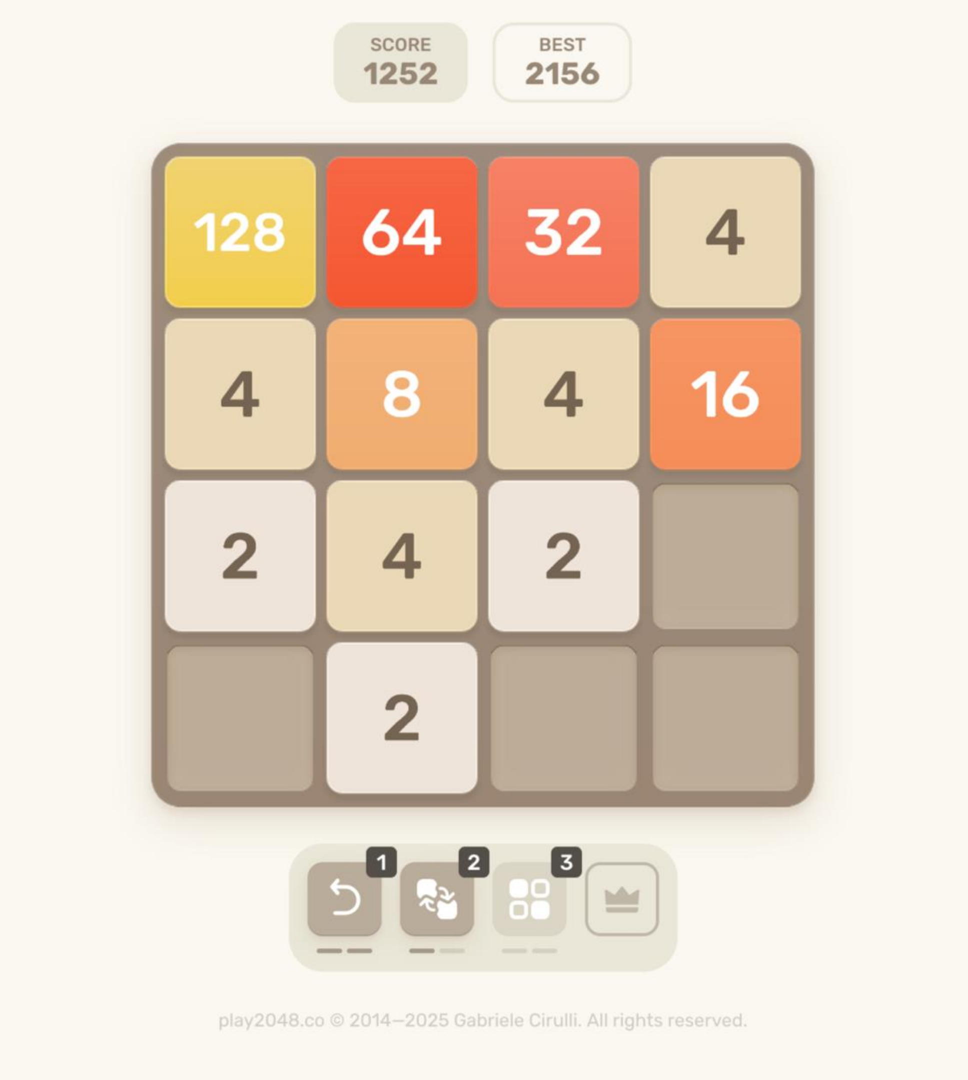
2048

Reinforcement Learning final project

01

THE GAME MECHANICS

A small intro to 2048



THE GRID

4 x 4 grid

Each tile can be either

- Empty (0)
- A power of 2

GAME MECHANICS

MOVE

Move the tiles up, down, left or right

MERGE

Merge two concurrent tiles of the same value

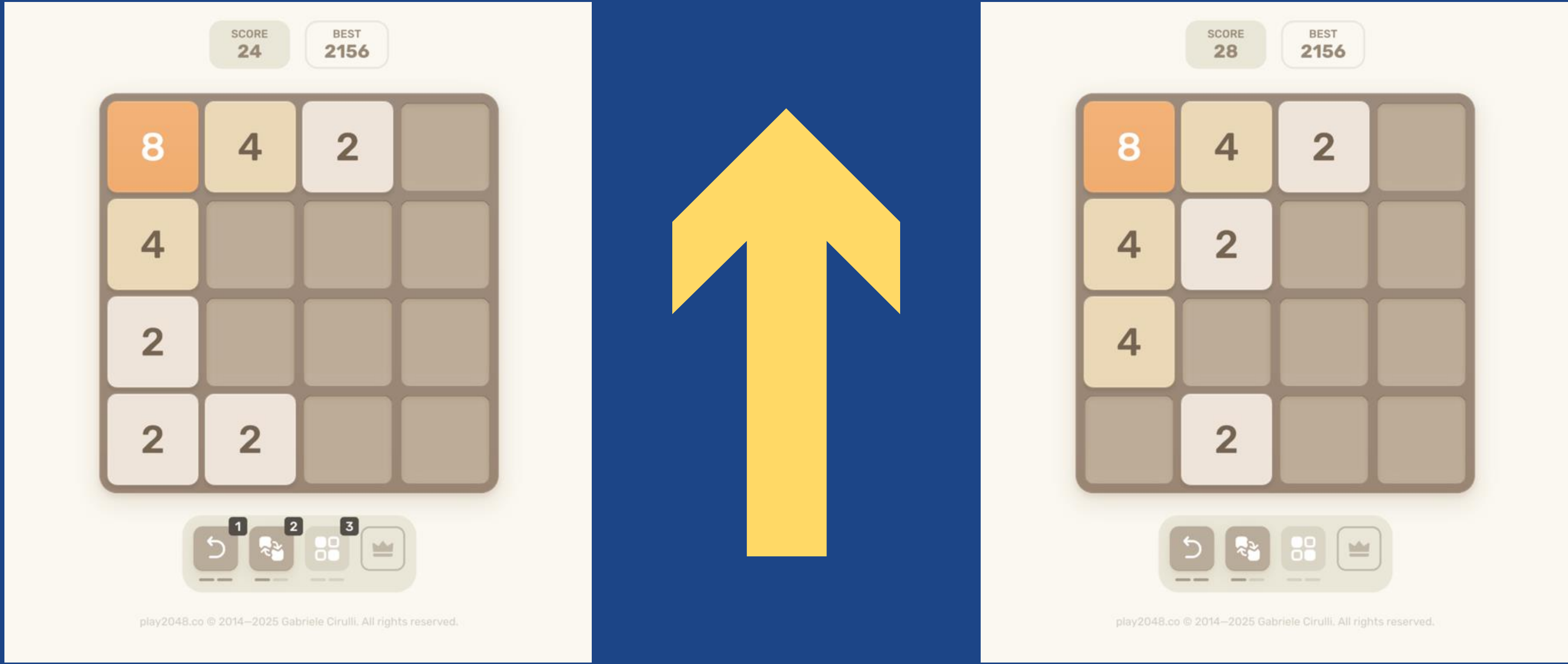
NEW TILE

Add a new tile in a random position after each move

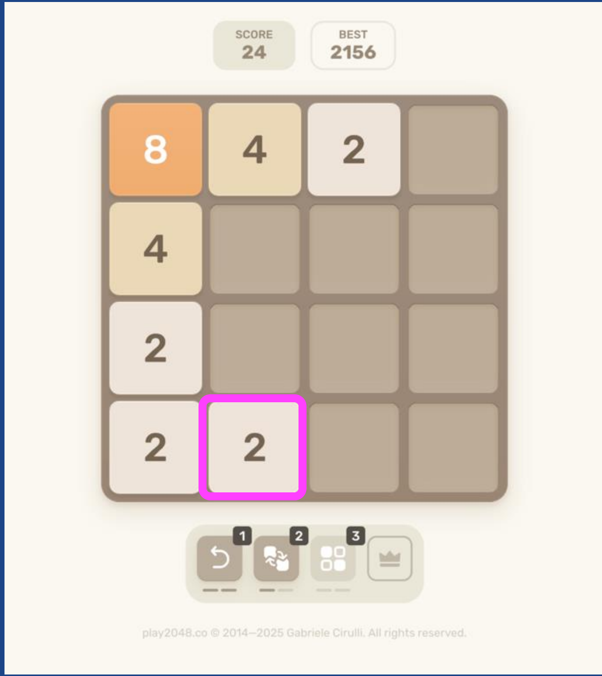
REWARD

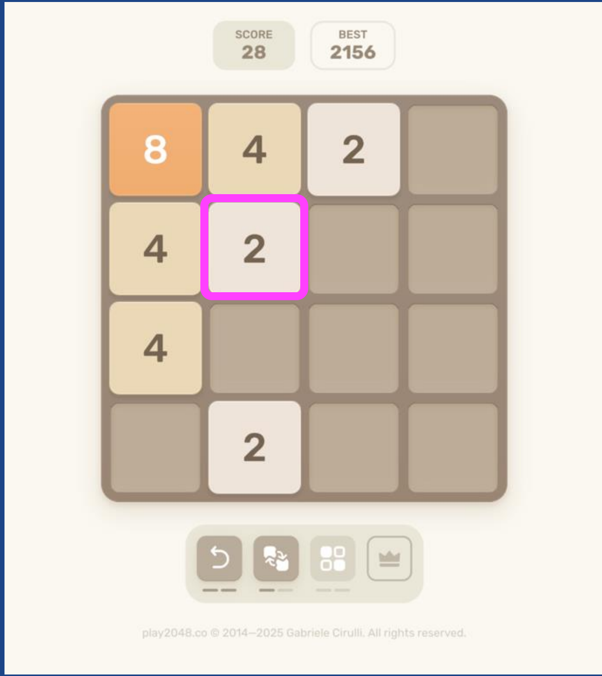
Get the sum of the merged tiles as reward

ACTIONS

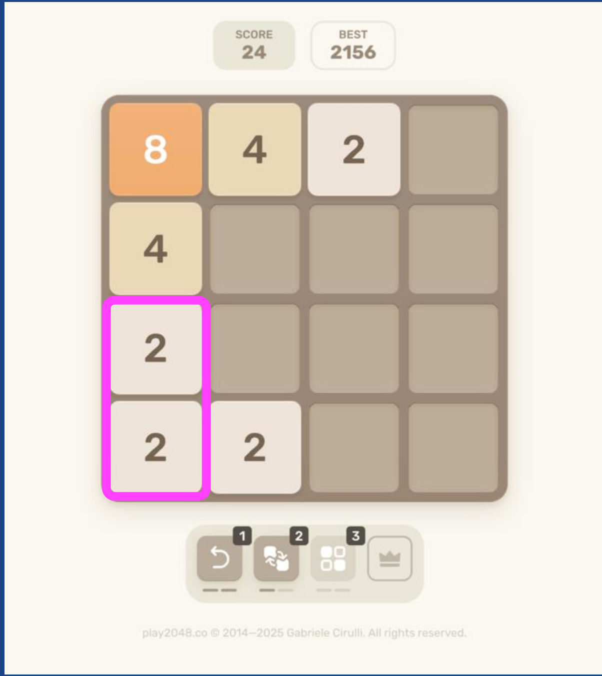

SCORE
24
BEST
2156
8 4 2
4
2
2 2
1 2 3
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.
SCORE
28
BEST
2156
8 4 2
4 2
4
2
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

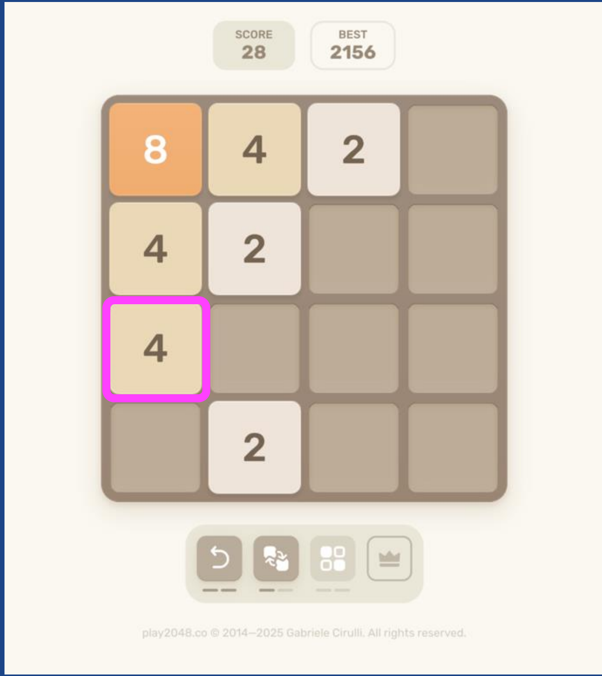
MOVE


SCORE
24
BEST
2156
8
4
2
4
2
2
2
1
2
3
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

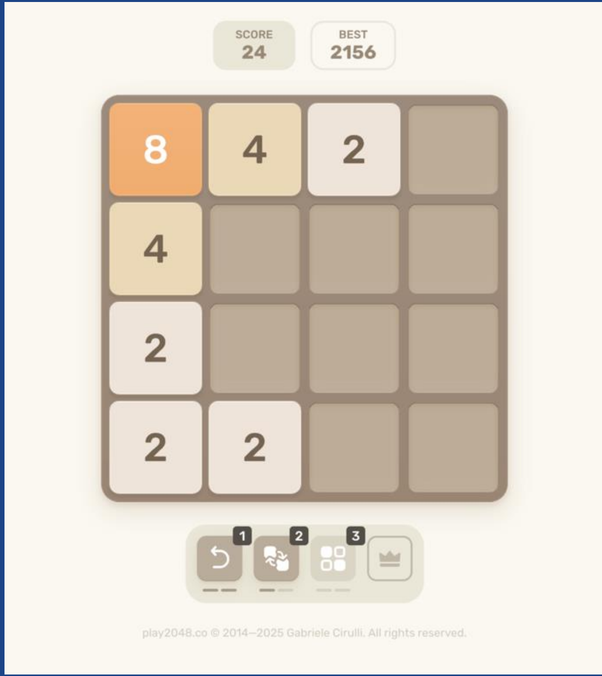

SCORE
28
BEST
2156
8
4
2
4
2
4
2
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

MERGE


SCORE
24
BEST
2156
8 4 2
4
2
2 2
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

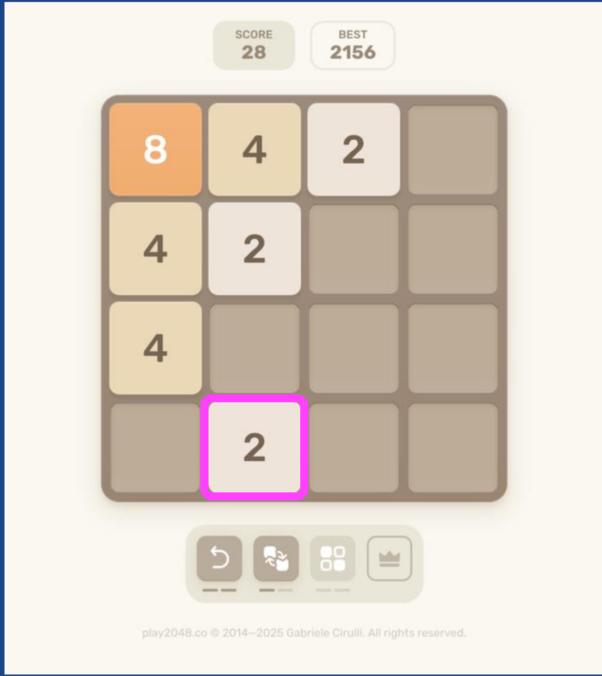

SCORE
28
BEST
2156
8 4 2
4 2
4
2
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

NEW TILE

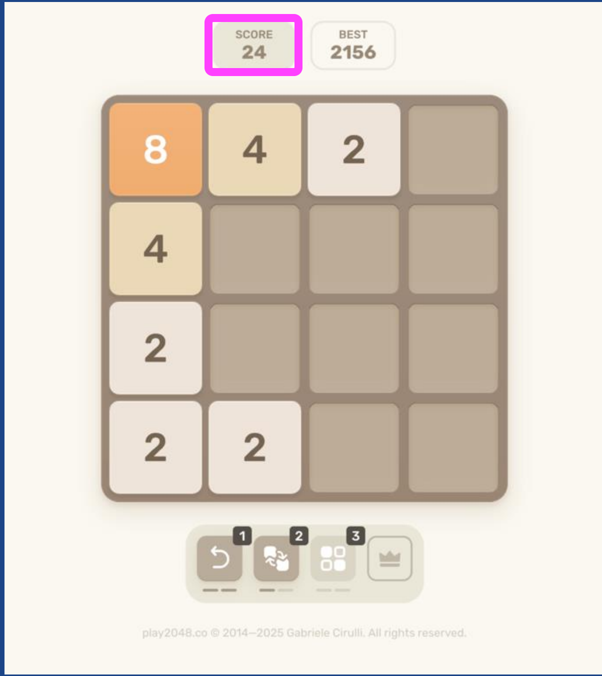


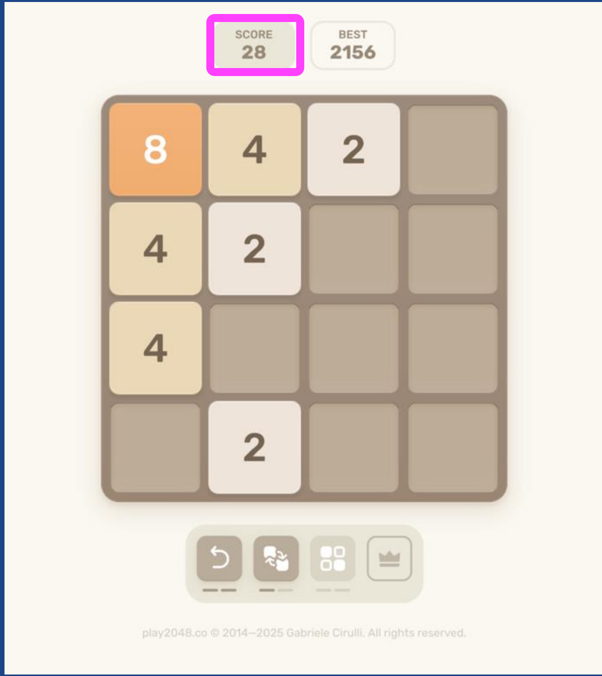
2 - 90%

4 - 10%

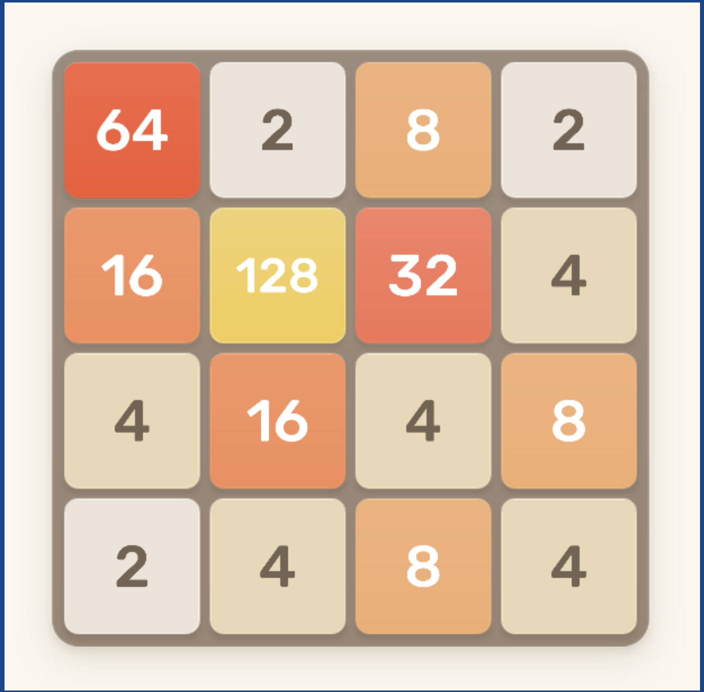


REWARD


SCORE
24
BEST
2156
8
4
2
4
2
2
2
1
2
3
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.


SCORE
28
BEST
2156
8
4
2
4
2
4
2
play2048.co © 2014–2025 Gabriele Cirulli. All rights reserved.

GAME OVER



02

SOLVING STRATEGIES

FINITE MARKOV DECISION PROCESS

STATE SPACE

16 cells that can assume 12 values each
(0, 2, …, 2048)

ACTION SPACE

4 actions: up, down, left and right

REWARD

Sum of the merged tiles, -10 each time the move does not change the grid

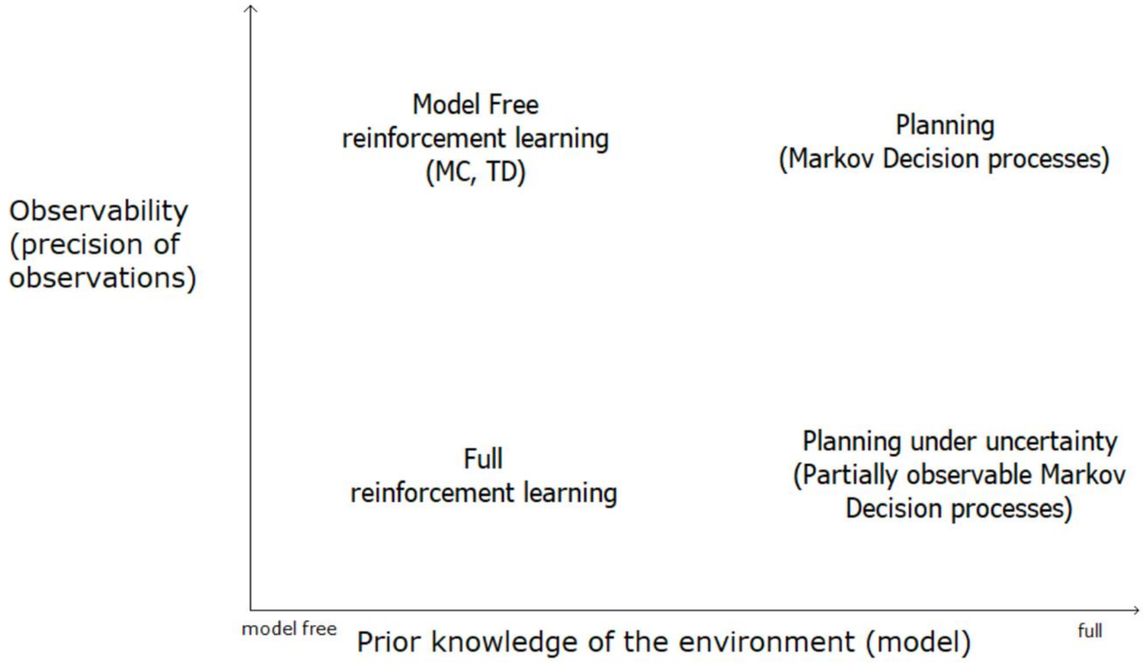
TRANSITIONS

Stochastic, due to adding the new tile in a random position

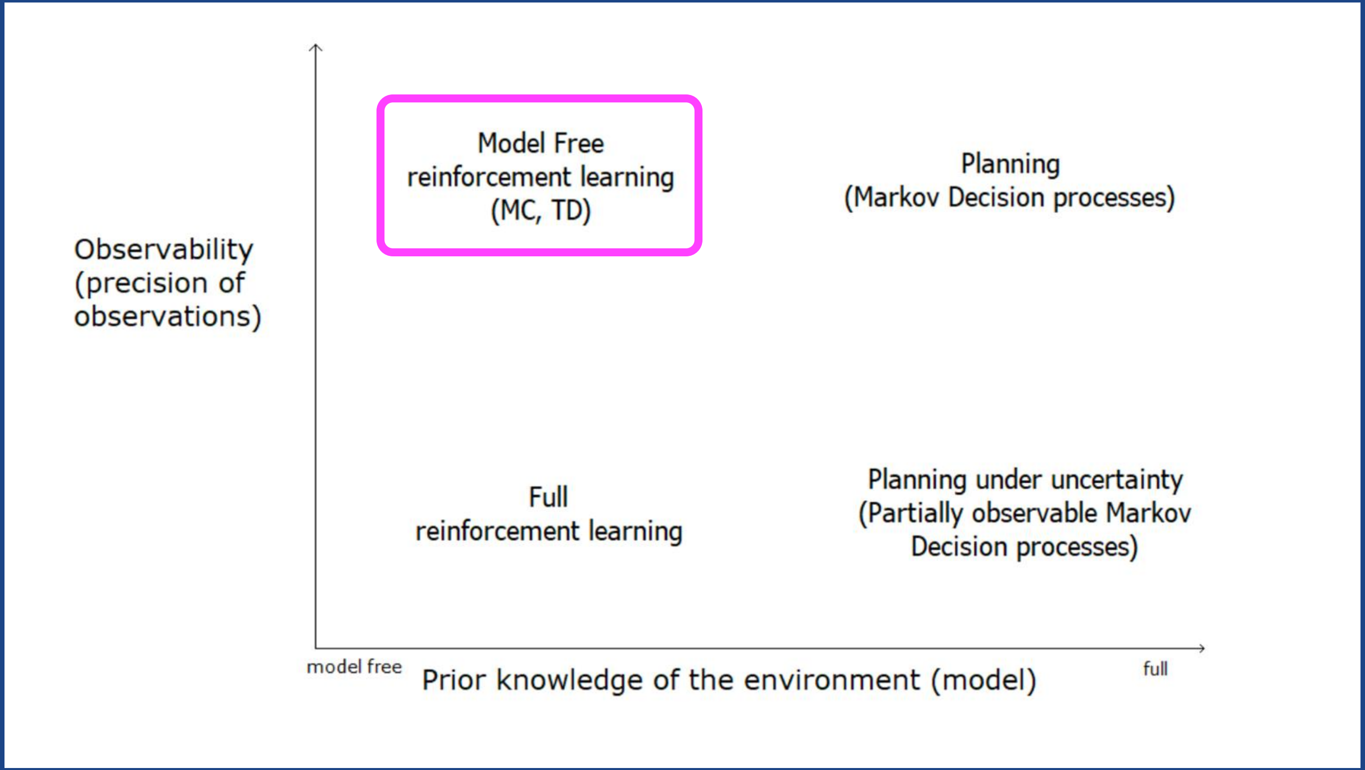
SOLVING STRATEGIES

1. Stochastic transitions
2. Fully observable states

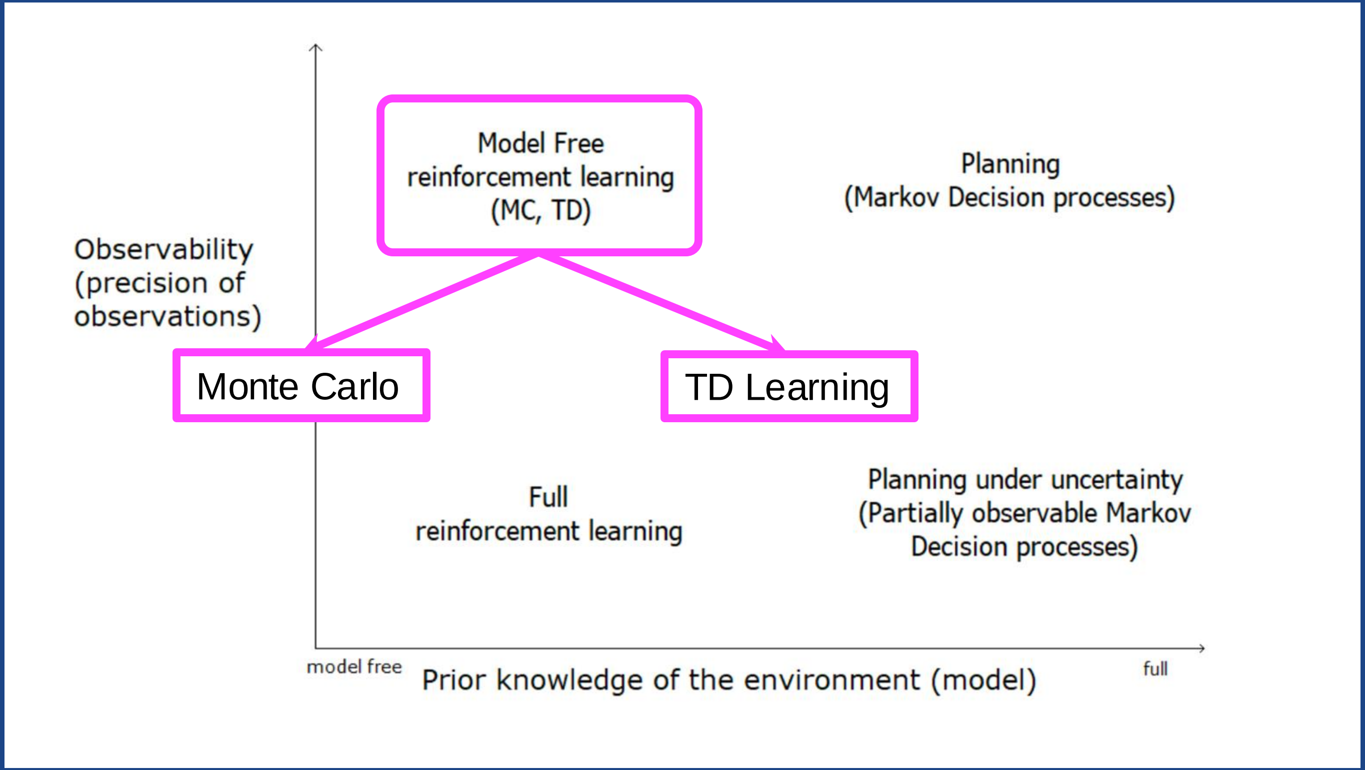
OBSERVABILITY AND KNOWLEDGE



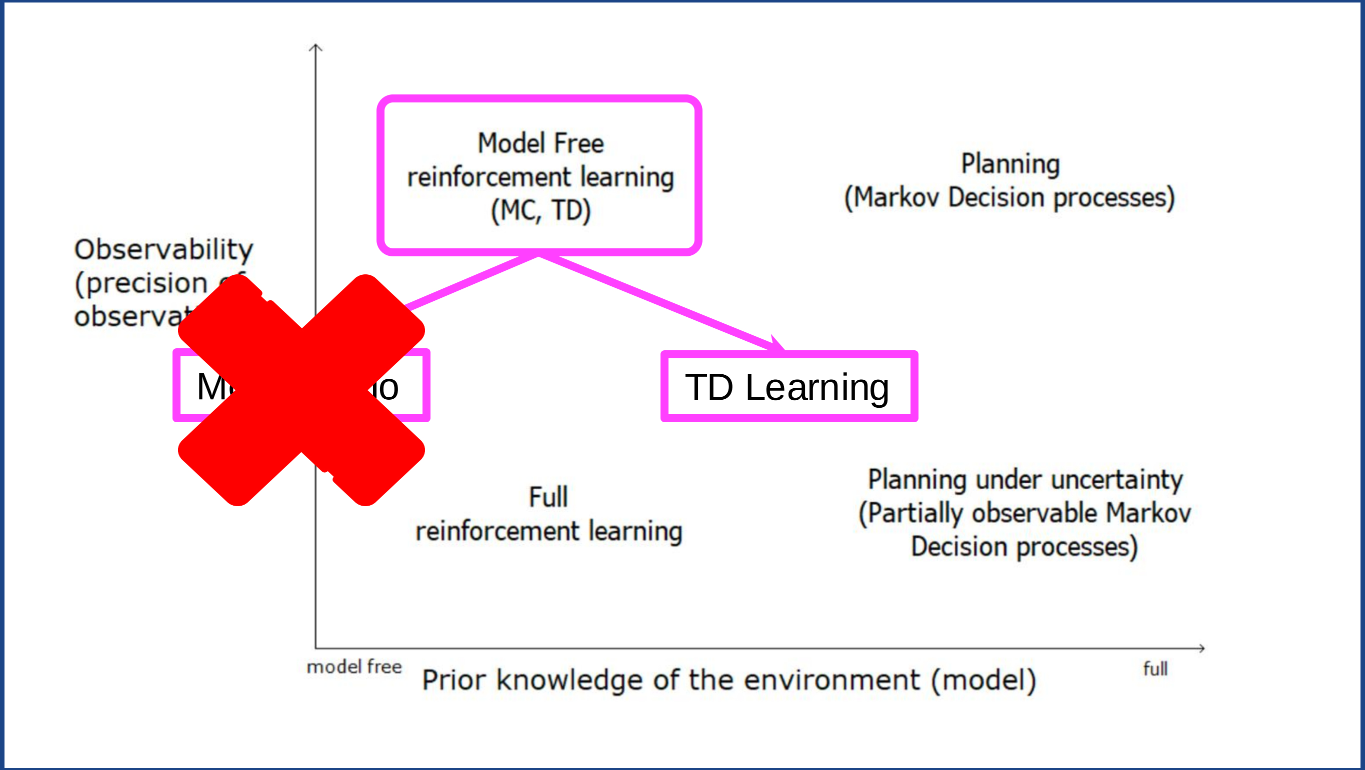
OBSERVABILITY AND KNOWLEDGE



OBSERVABILITY AND KNOWLEDGE



OBSERVABILITY AND KNOWLEDGE



03

Q-LEARNING

Q-LEARNING

Initialize $Q(s,a)$ arbitrarily

Repeat (for each episode):

  Initialize s

  Repeat (for each step of episode):

    Choose a from s using policy derived from Q

    Take action a, observe r, s'

    Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

    $s \leftarrow s'$;

  Until s is terminal

Q-LEARNING

Initialize $\boxed{Q(s,a)}$ arbitrarily

\# states x # actions

Repeat (for each episode):

Initialize s

Repeat (for each step of episode):

Choose a from s using policy derived from Q

Take action a, observe r, s'

Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$s \leftarrow s'$;

Until s is terminal

STATE SPACE SIZE

12

STATES

$0, 2^1, 2^2, \ldots, 2^{11}$

16

CELLS

In the 4 x 4 grid

10^{17}

STATE SPACE SIZE

04

DEEP Q-LEARNING

KEY POINT

Q(s, a) and Q(s’, a’) are calculated using neural networks, called **policy network** and **target network** respectively

THE NEURAL NETWORKS

Target Network

$$Q(s,a) \leftarrow \boxed{Q(s,a)} + \alpha[r + \gamma \max_{a'} \boxed{Q(s',a')} - \boxed{Q(s,a)}]$$

Policy Network

THE NEURAL NETWORKS

The neural networks
are trained using a
batch of past games

THE NEURAL NETWORKS

ARCHITECHTURE

The two networks have the same architecture

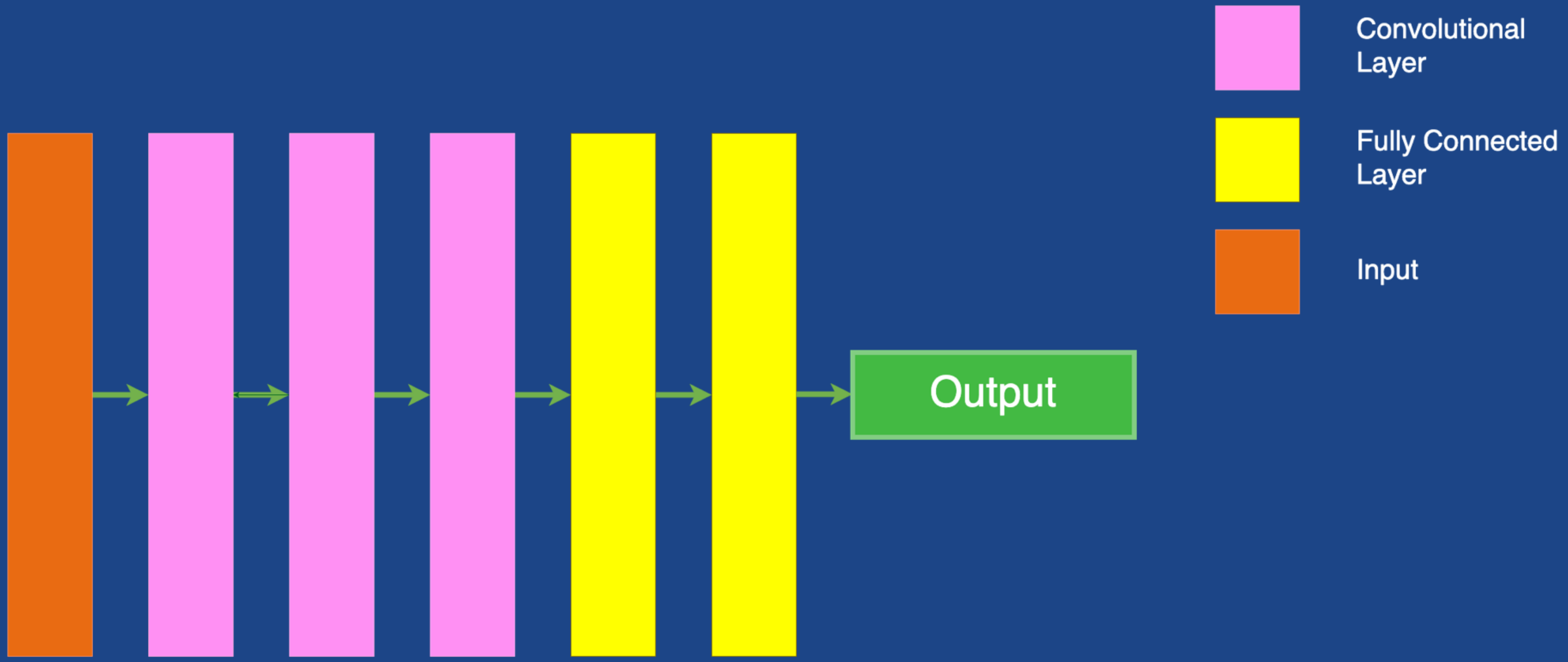
UPDATES

- The weights of the policy network are updated after each episode
- The weights of the target network are updated every **m** episodes

THE NEURAL NETWORKS

- 3 x convolutional layers with 128 channels
- Batch normalization layer after each convolutional layer
- 2 x fully connected layer with 128 channels
- ReLU activation function

THE NEURAL NETWORKS



POLICY

Epsilon-greedy policy
with decaying epsilon

POLICY

WITH PROBABILITY ε

Perform a random action

WITH PROBABILITY $1 - \varepsilon$

Choose the action that maximizes $Q(s', a')$

OPTIMIZATION

- 1000 episodes
- 100 epochs of optimization
- ADAM optimizer with 1e-5 learning rate
- Size of the replay buffer: 50000
- γ: 0.99
- ε: starting from 0.9, decays of a factor of 0.999, minimum value of 0.01
- Batch size: 64
- One-hot encoding for training

DEEP Q-LEARNING

Algorithm 1 Training Loop

Initialize policy and target networks with the same weights: $\theta \leftarrow \theta_{\text{target}}$
Initialize replay buffer $\mathcal{D}$ with capacity N
Set discount factor γ, learning rate, target update frequency C, exploration rate ϵ, minimum epsilon ϵ_{min} and decay rate ϵ_{decay}
for N episodes **do**
  Start the game with just one random tile on the board
  while game is not over **do**
    Choose action a using ϵ-greedy policy:
    if random number $< \epsilon$ **then**
      Choose a random action a
    else
      Choose $a = \arg\max_{a'} Q(s, a'; \theta)$
    end if
    Take action a and get reward r, next state s', and *game over* status
    if state = next state and the game is not over **then**
      reward $\leftarrow$ reward - penalty
    end if
    Store transition (s, a, r, s', *game over*) in replay buffer $\mathcal{D}$
    Update state $s \leftarrow s'$
  end while
  Train the policy network
  Decay exploration rate as $\epsilon \leftarrow \max(\epsilon_{min}, \epsilon \cdot \epsilon_{decay})$
end for

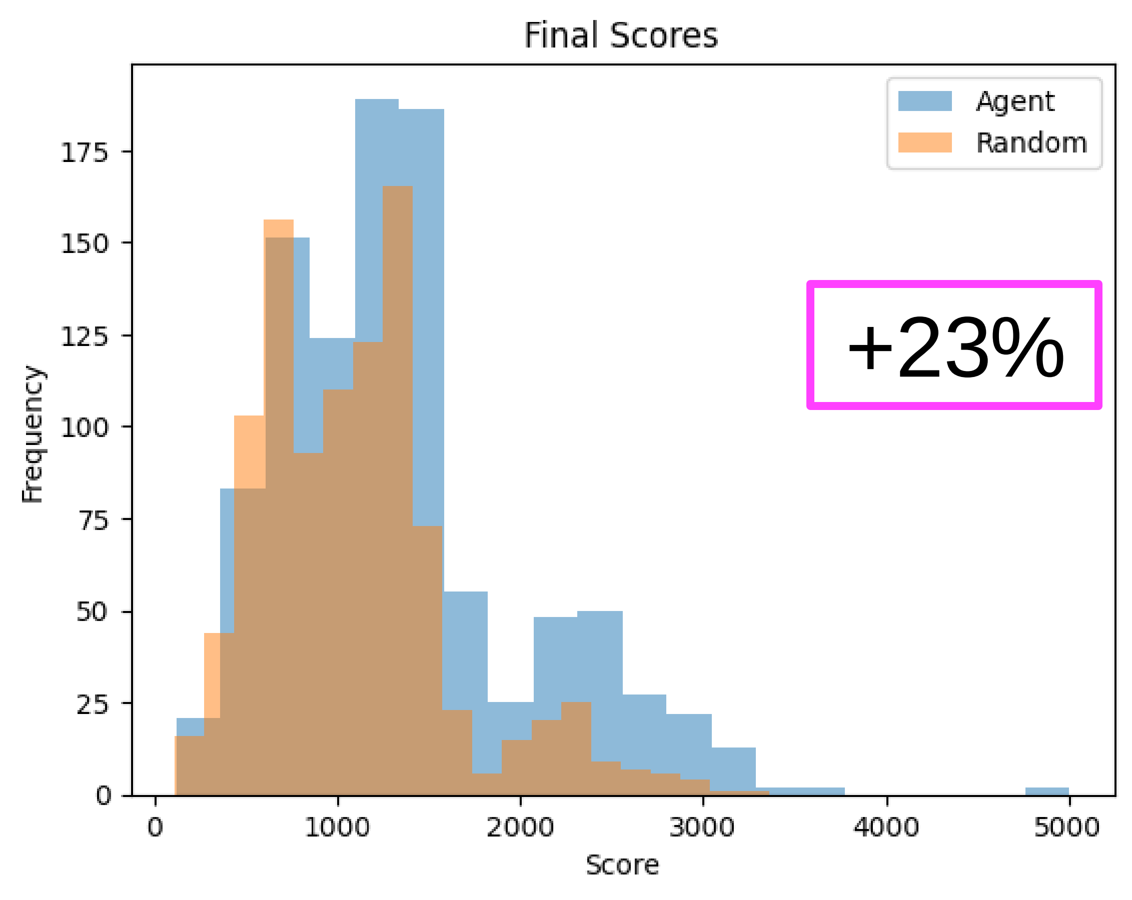
DEEP Q-LEARNING

Algorithm 2 Neural Network Training

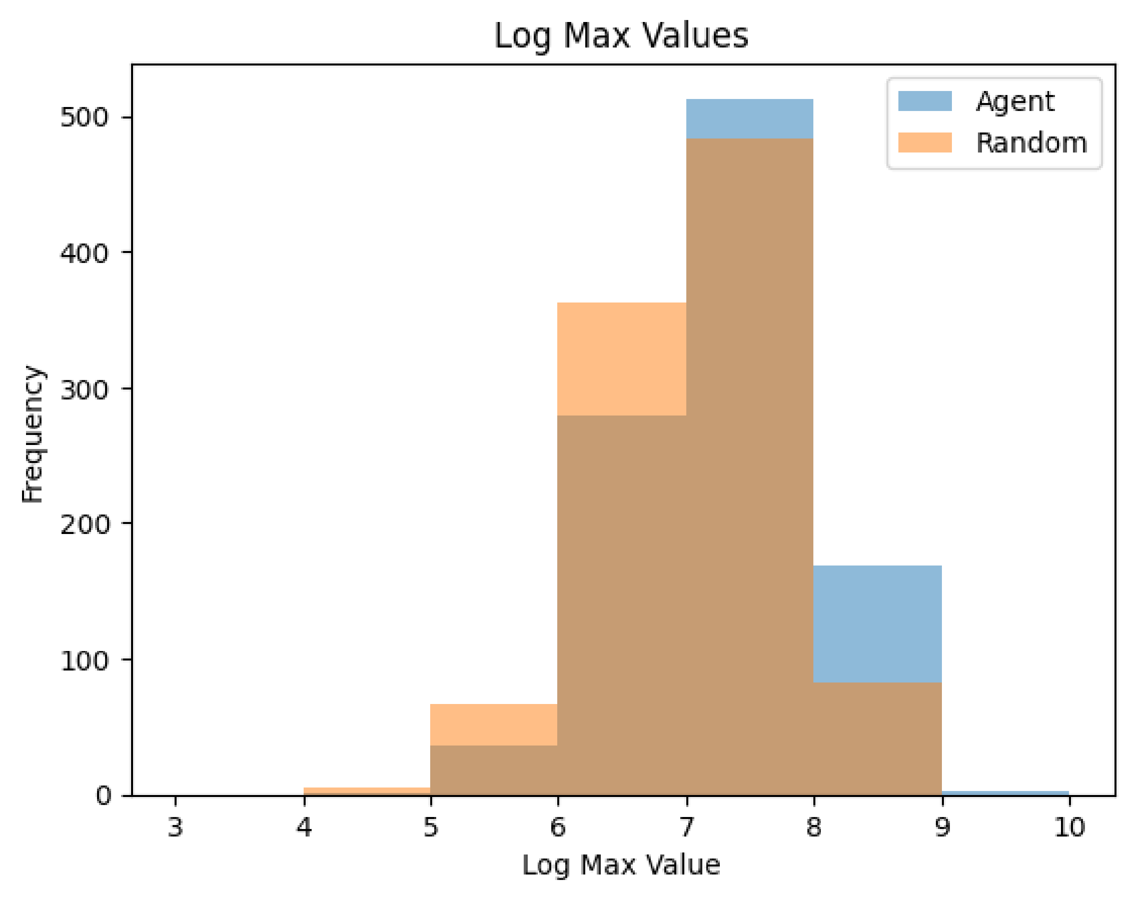
for M epochs **do**
    Sample a batch of B transitions $(s, a, r, s', \text{done})$ from replay buffer $\mathcal{D}$
    Compute current Q-values: $Q(s, a; \theta)$ using the policy network
    Compute target Q-values $Q(s', a'; \theta_{\text{target}})$ using the target network:
    if done is True **then**
        $y = r$
    else
        $y = r + \gamma \cdot \max_{a'} Q(s', a'; \theta_{\text{target}})$
    end if
    Compute loss between $Q(s, a; \theta)$ and y
    Perform backward pass and update Q-network weights θ
    if episode % C == 0 **then**
        Update target network weights: $\theta_{\text{target}} \leftarrow \theta$
    end if
end for

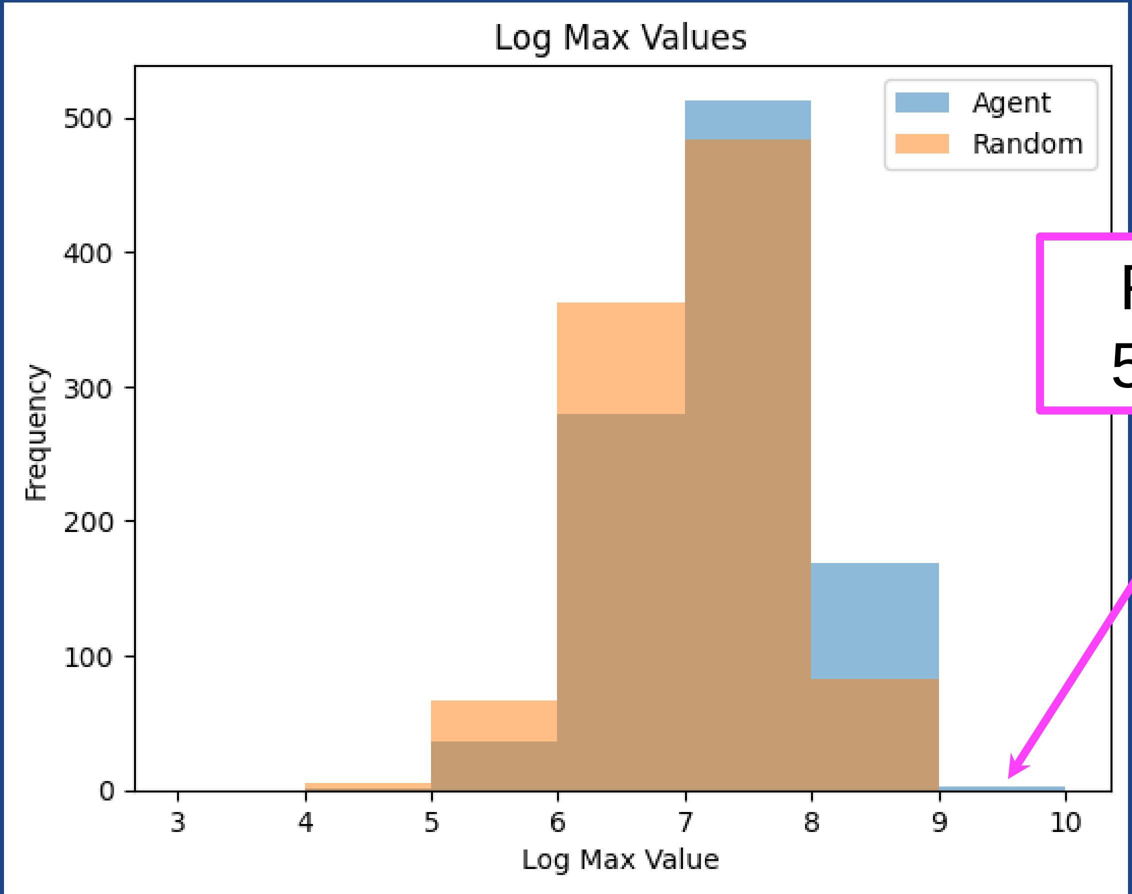
05

RESULTS

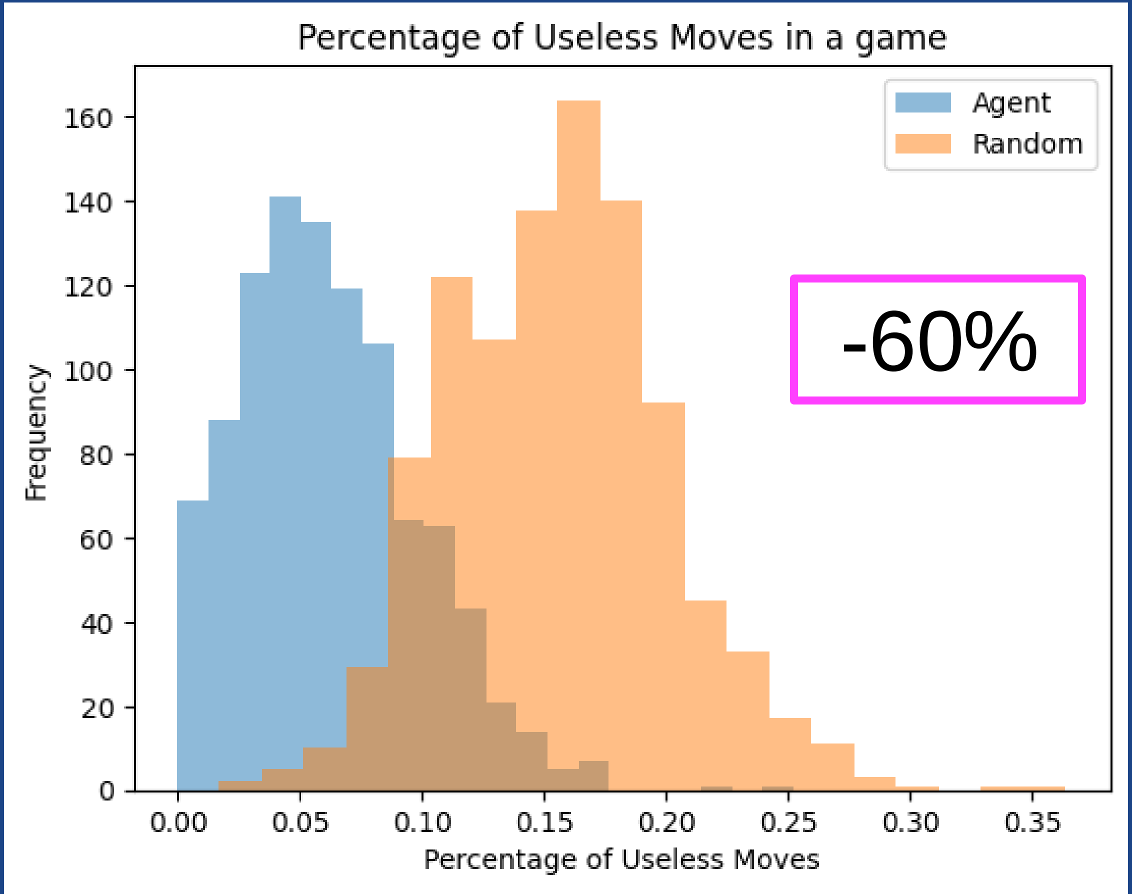


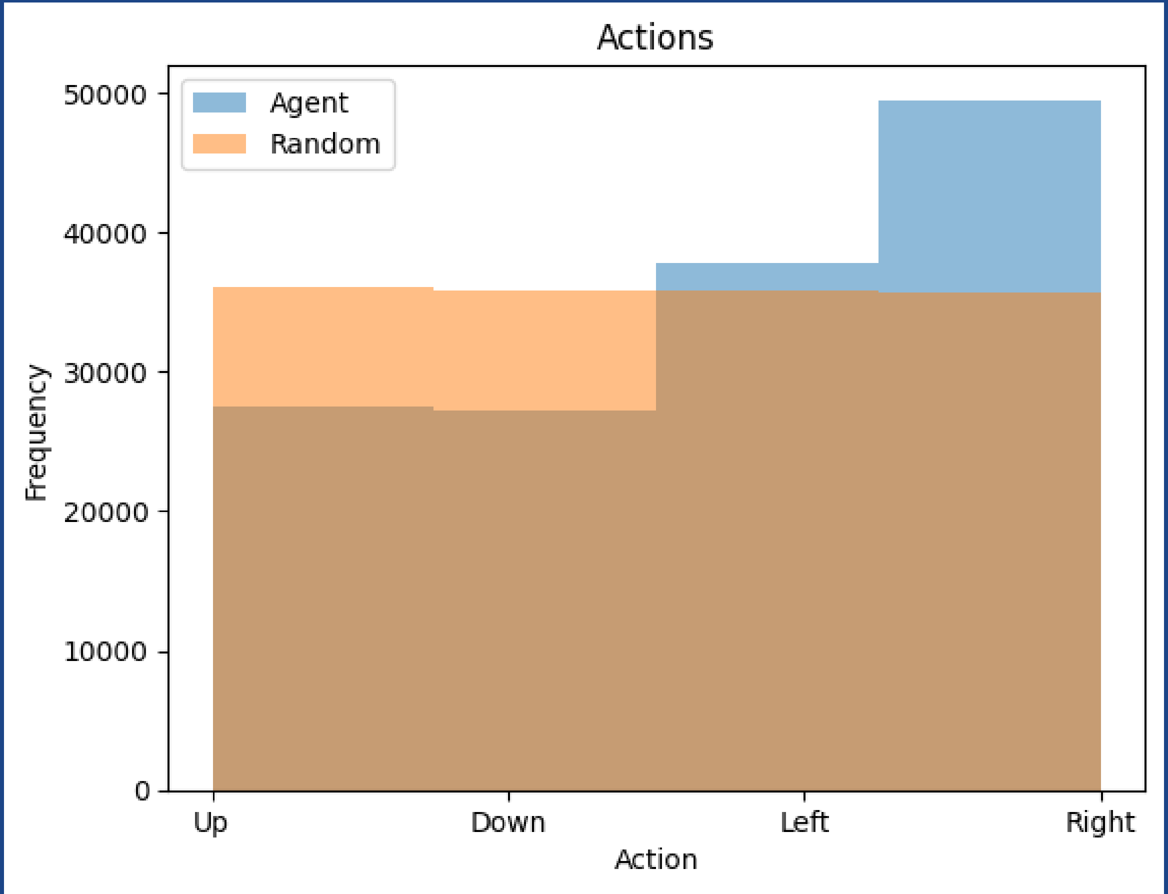
Final Scores
Agent
Random
+23%
Frequency
0
25
50
75
100
125
150
175
0
1000
2000
3000
4000
5000
Score


Log Max Values
Agent
Random
Frequency
0
100
200
300
400
500
3
4
5
6
7
8
9
10
Log Max Value



Reaches 512 twice


Percentage of Useless Moves in a game
Agent
Random
-60%
Frequency
0
20
40
60
80
100
120
140
160
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Percentage of Useless Moves

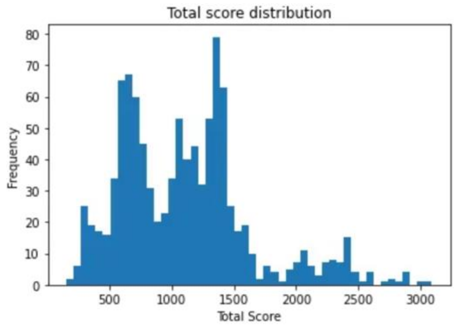


Actions
Agent
Random
50000
40000
30000
20000
10000
0
Frequency
Up
Down
Left
Right
Action

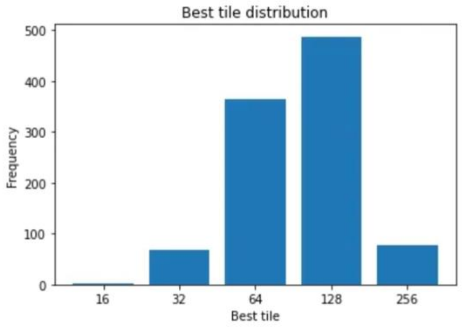
06

SIMILAR WORKS

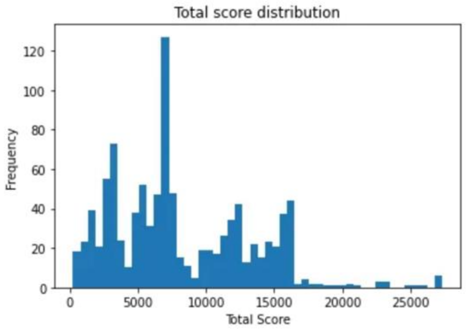
Playing 2048 with Deep Q-Learning by Lok Hin Chan



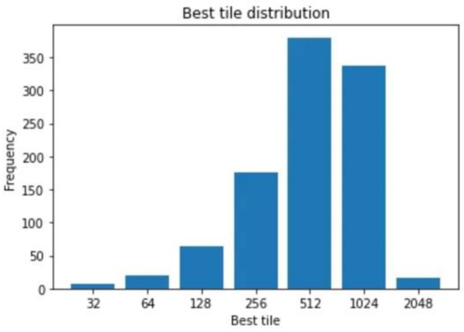
Source by the Author



Source by the Author



Source by the Author



Source by the Author

Playing 2048 with Deep Q-Learning by Lok Hin Chan

```
class ConvBlock(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(ConvBlock, self).__init__()
        d = output_dim // 4
        self.conv1 = nn.Conv2d(input_dim, d, 1, padding='same')
        self.conv2 = nn.Conv2d(input_dim, d, 2, padding='same')
        self.conv3 = nn.Conv2d(input_dim, d, 3, padding='same')
        self.conv4 = nn.Conv2d(input_dim, d, 4, padding='same')

    def forward(self, x):
        x = x.to(device)
        output1 = self.conv1(x)
        output2 = self.conv2(x)
        output3 = self.conv3(x)
        output4 = self.conv4(x)
        return torch.cat((output1, output2, output3, output4), dim=1)

class DQN(nn.Module):

    def __init__(self):
        super(DQN, self).__init__()
        self.conv1 = ConvBlock(16, 2048)
        self.conv2 = ConvBlock(2048, 2048)
        self.conv3 = ConvBlock(2048, 2048)
        self.dense1 = nn.Linear(2048 * 16, 1024)
        self.dense2 = nn.Linear(1024, 4)

    def forward(self, x):
        x = x.to(device)
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = nn.Flatten()(x)
        x = F.dropout(self.dense1(x))
        return self.dense2(x)
```

Playing 2048 with Deep Q-Learning by Lok Hin Chan

The following parameters are used to train the model to play 2048:

- Size of memory buffer: 50000
- Optimizer: Adam
- Learning Rate: 5e-5
- γ: 0.99
- ε: Starting from 0.9 and decays by a factor of 0.9999 for each episode until it reaches 0.01
- Batch size: 64

Playing 2048 with Deep Q-Learning by Lok Hin Chan

Finally after training the model for 20000 episodes, some of the episodes does manage to reach tile 2048, for example the one below:

06

FUTURE IMPROVEMENTS

POSSIBLE IMPROVEMENTS

- Penalize distance between high-value tiles
- Trying Double Q-Learning, where we separate action selection and evaluation
- Prioritized Experience Replay
- n-step TD learning instead of 1-step: target q-value calculated over n-steps
- Hyperparameter tuning

THANKS!

Credits:

Playing 2048 with Deep Q-Learning by Lok Hin Chan