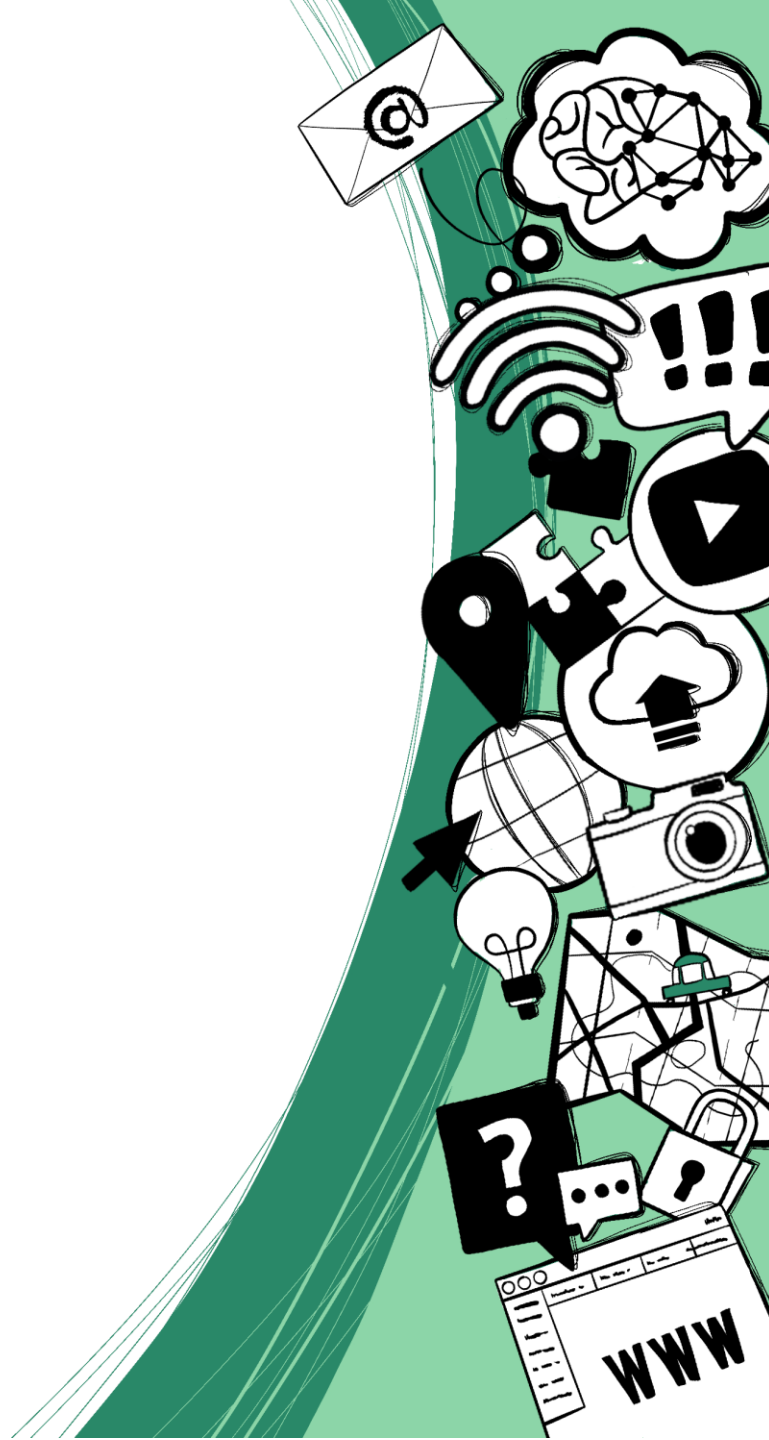




# Reinforcement Learning



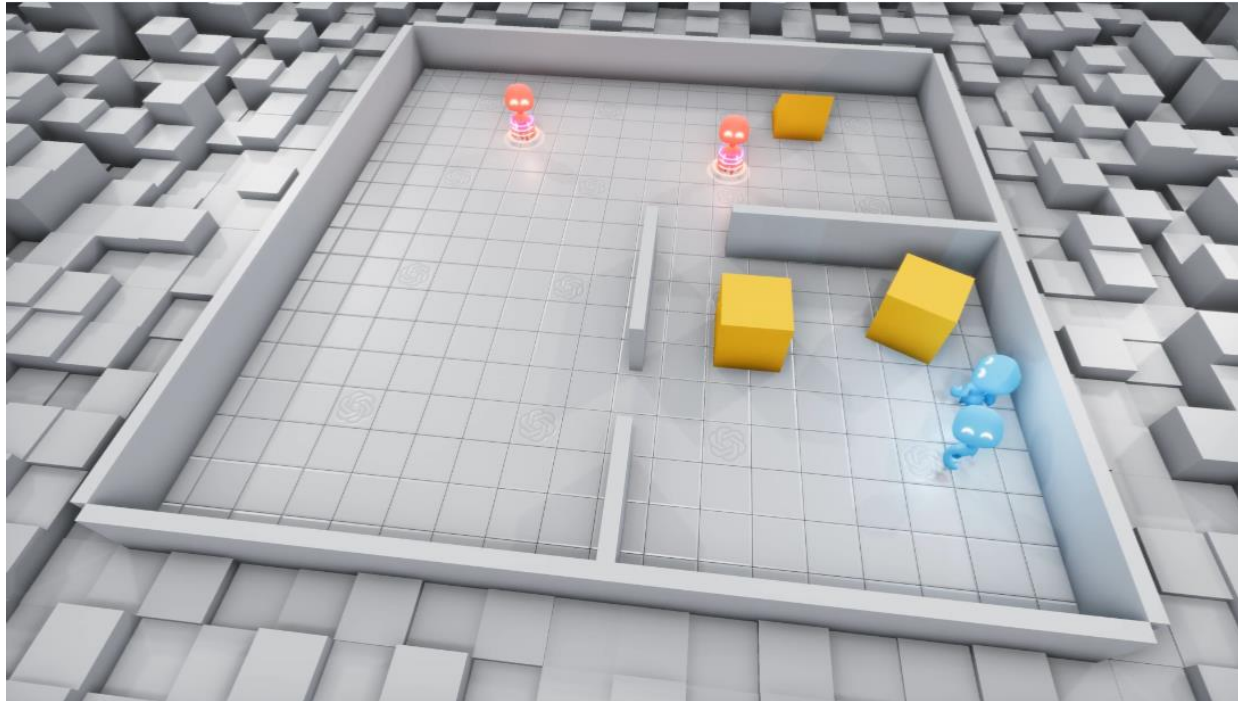
# Examples



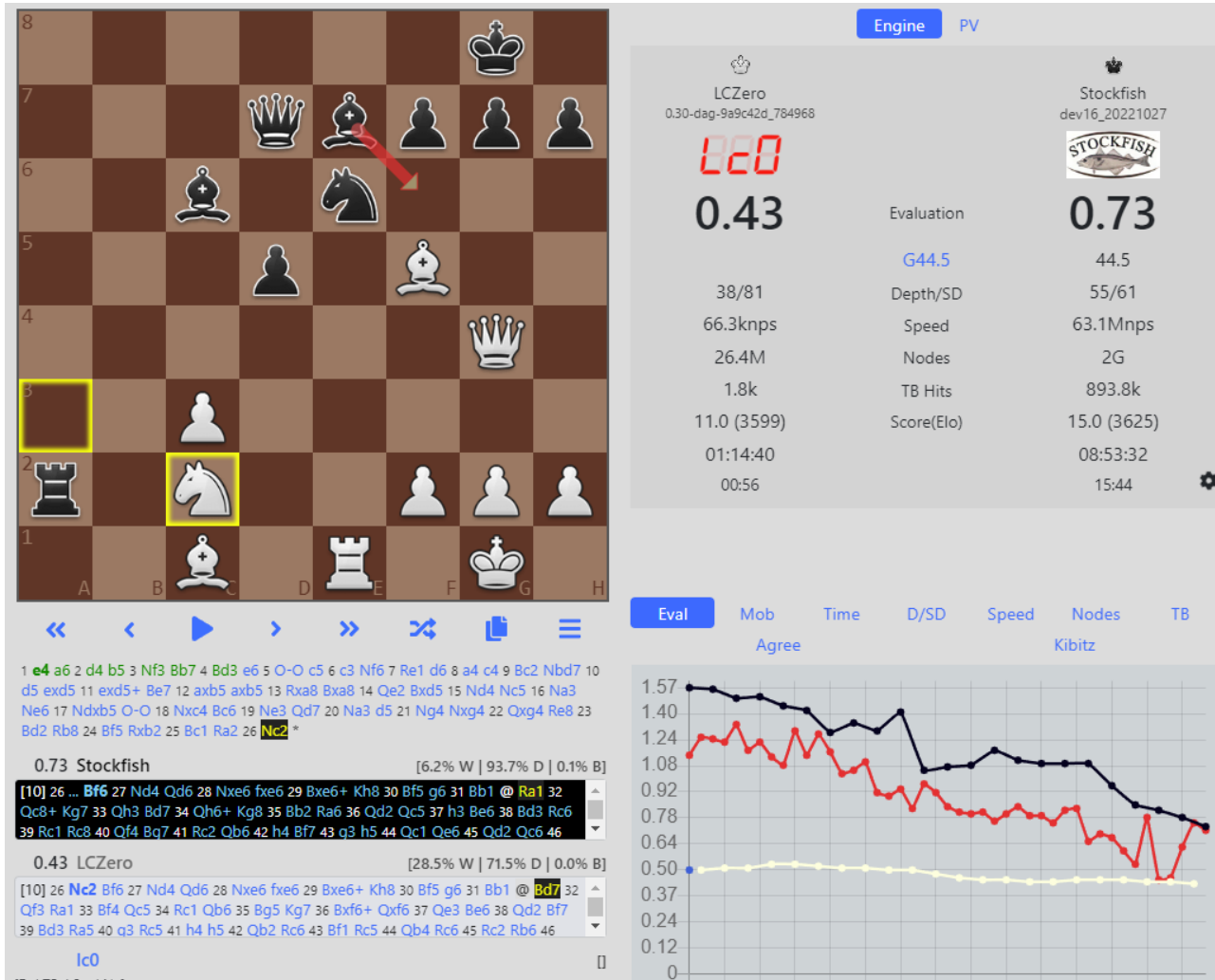


# OpenAI Hide and Seek

<https://openai.com/blog/emergent-tool-use/>



# Leela Chess Zero (Lc0)



The screenshot displays a chess engine interface with the following components:

- Chessboard:** Shows a chessboard with pieces. A red arrow points to a knight on e6, and a yellow box highlights a knight on b2.
- Engine Comparison:**

Engine	Score	Evaluation	Depth/SD	Speed	Nodes	TB Hits	Score(Elo)	Time
LCZero	0.43	G44.5	38/81	66.3knps	26.4M	1.8k	11.0 (3599)	01:14:40
Stockfish	0.73	44.5	55/61	63.1Mnps	2G	893.8k	15.0 (3625)	08:53:32
- Performance Graph:** A line graph showing performance metrics over time for both engines. The y-axis ranges from 0 to 1.57. The x-axis represents time. The black line (Stockfish) starts at approximately 1.57 and fluctuates downwards. The red line (LCZero) starts at approximately 1.1 and fluctuates downwards, ending near 0.4.
- Game History:**

1 e4 a6 2 d4 b5 3 Nf3 Bb7 4 Bd3 e6 5 O-O c5 6 c3 Nf6 7 Re1 d6 8 a4 c4 9 Bc2 Nbd7 10 d5 exd5 11 exd5+ Be7 12 axb5 axb5 13 Rxa8 Bxa8 14 Qe2 Bxd5 15 Nd4 Nc5 16 Na3 Ne6 17 Ndx5 O-O 18 Nxc4 Bc6 19 Ne3 Qd7 20 Na3 d5 21 Ng4 Nxc4 22 Qxg4 Re8 23 Bd2 Rb8 24 Bf5 Rxb2 25 Bc1 Ra2 26 **Nc2\***

0.73 Stockfish [6.2% W | 93.7% D | 0.1% B]

[10] 26 ... Bf6 27 Nd4 Qd6 28 Nxe6 fxe6 29 Bxe6+ Kh8 30 Bf5 g6 31 Bb1 @ Ra1 32 Qc8+ Kg7 33 Qh3 Bd7 34 Qh6+ Kg8 35 Bb2 Ra6 36 Qd2 Qc5 37 h3 Be6 38 Bd3 Rc6 39 Rc1 Rc8 40 Qf4 Bg7 41 Rc2 Qb6 42 h4 Bf7 43 g3 h5 44 Qc1 Qe6 45 Qd2 Qc6 46

0.43 LCZero [28.5% W | 71.5% D | 0.0% B]

[10] 26 **Nc2** Bf6 27 Nd4 Qd6 28 Nxe6 fxe6 29 Bxe6+ Kh8 30 Bf5 g6 31 Bb1 @ **Bd7** 32 Qf3 Ra1 33 Bf4 Qc5 34 Rc1 Qb6 35 Bg5 Kg7 36 Bxf6+ Qxf6 37 Qe3 Be6 38 Qd2 Bf7 39 Bd3 Ra5 40 g3 Qc5 41 h4 h5 42 Qb2 Rc6 43 Bf1 Rc5 44 Qb4 Rc6 45 Rc2 Rb6 46







# Alpha Star

14:32  
Catalyst LE

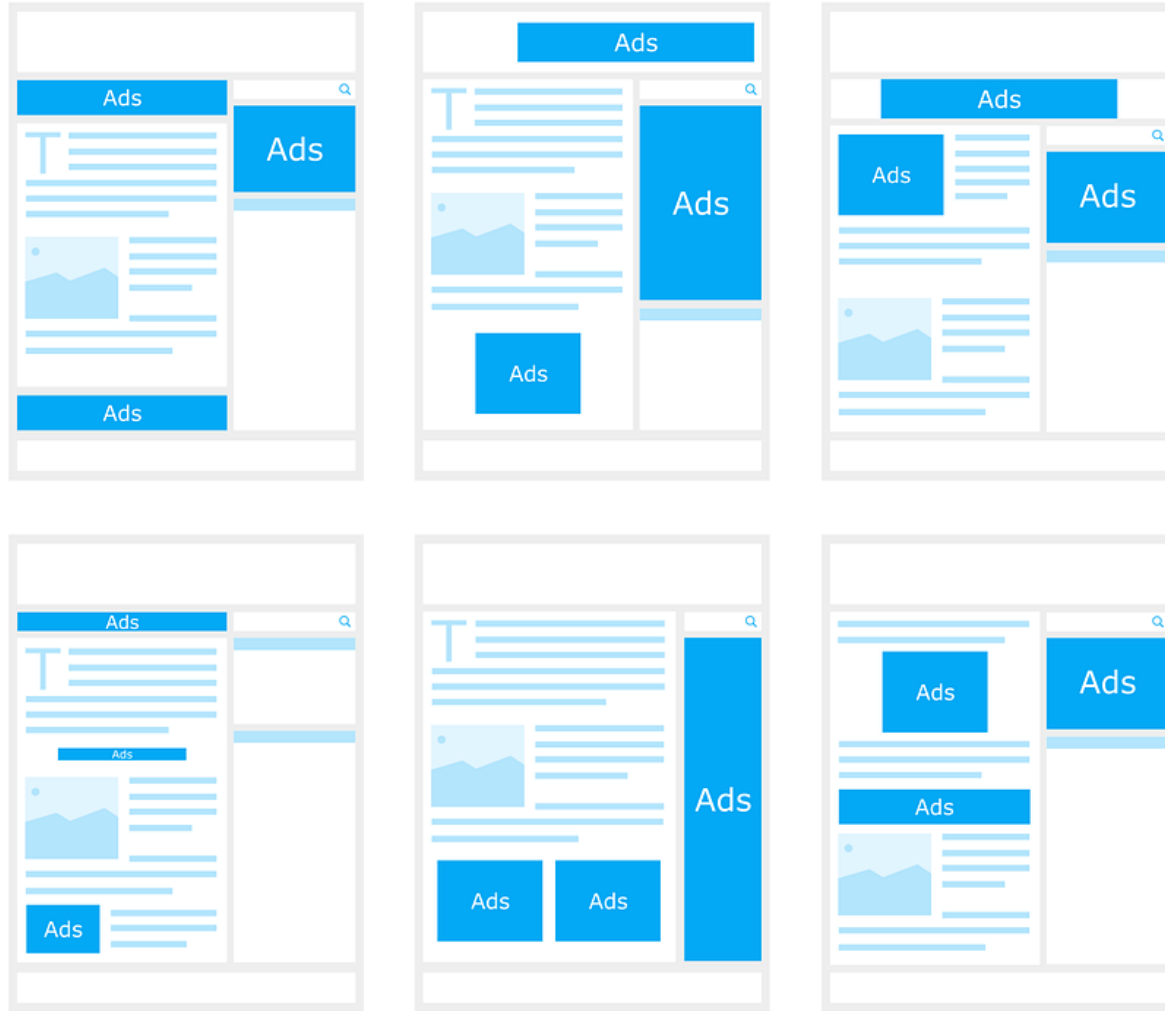
Player	Supply	Minerals	Gas	Workers	Army	APM	Production
AlphaStar	177 / 200	945 +2015	758 +873	64	113	940	2 Pylons
LiquidTLO	147 / 172	335 +1595	442 +1030	61	86	1377	2 Spawning Pools

<https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii>





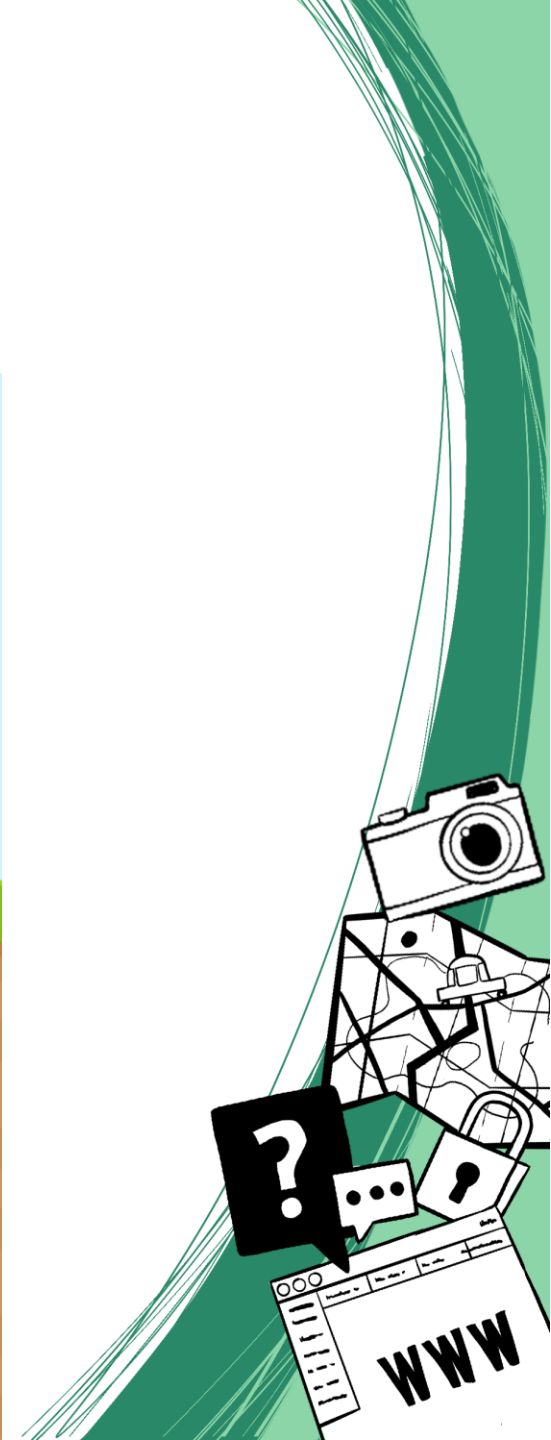
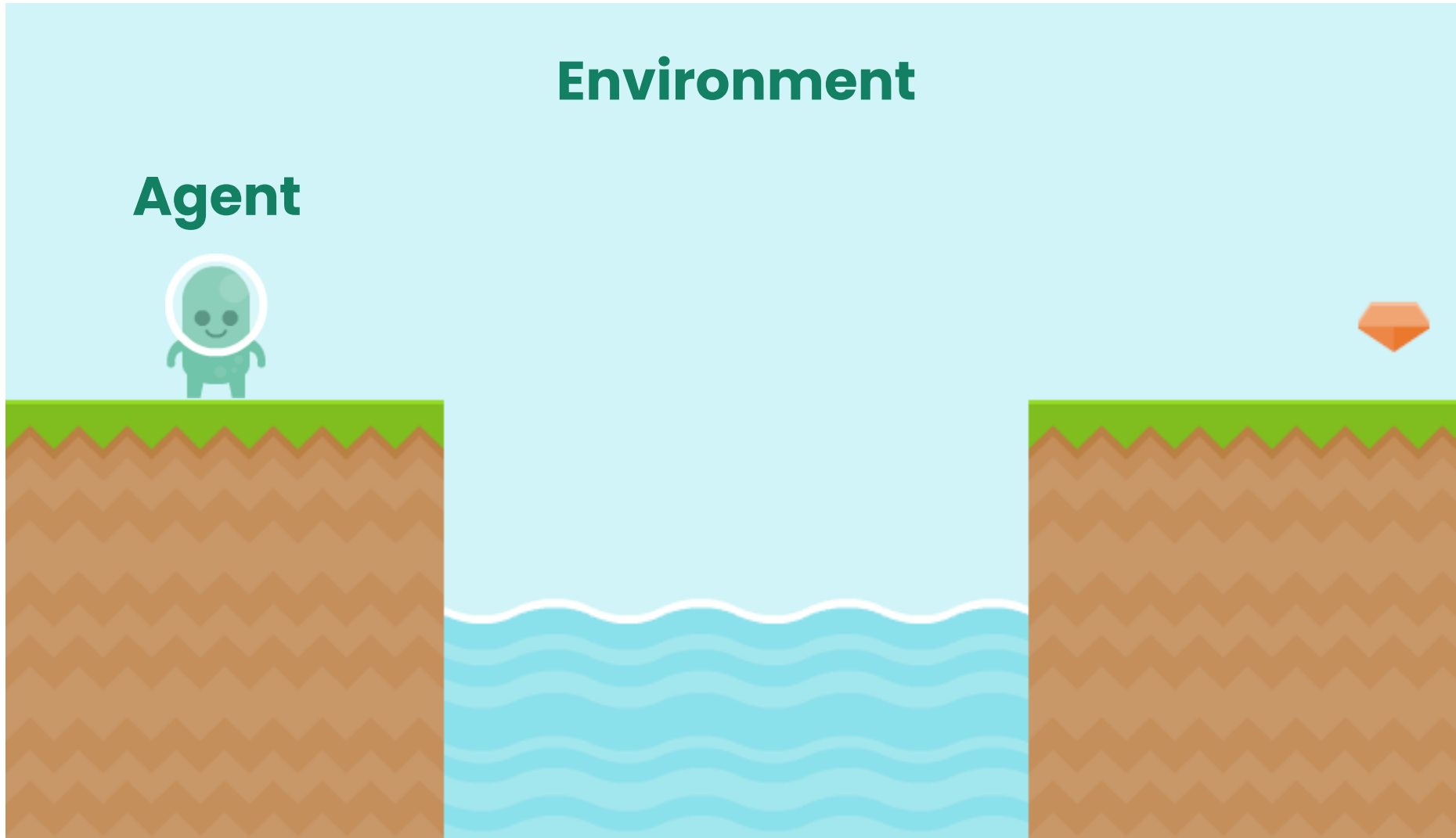
# Custom Advertisement



# RL Basics

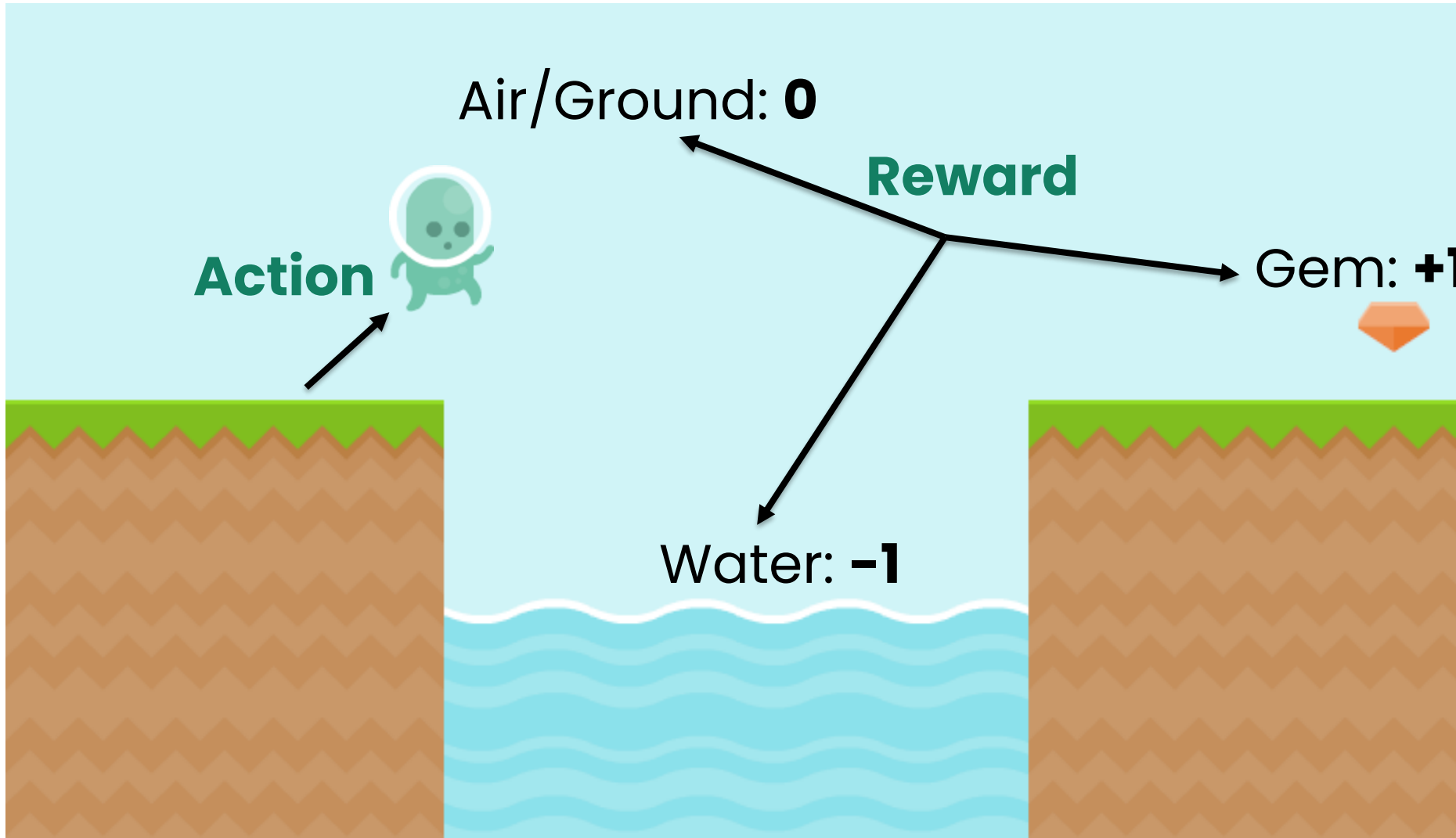


# What we have

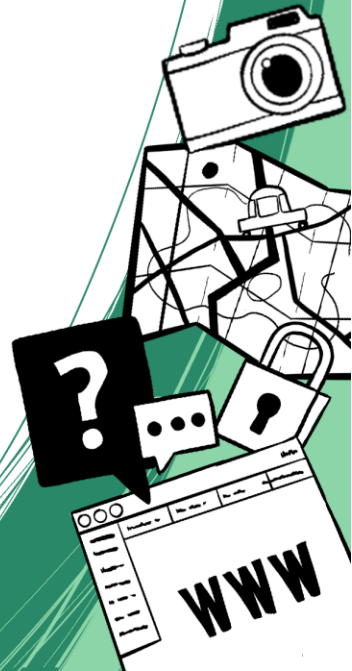
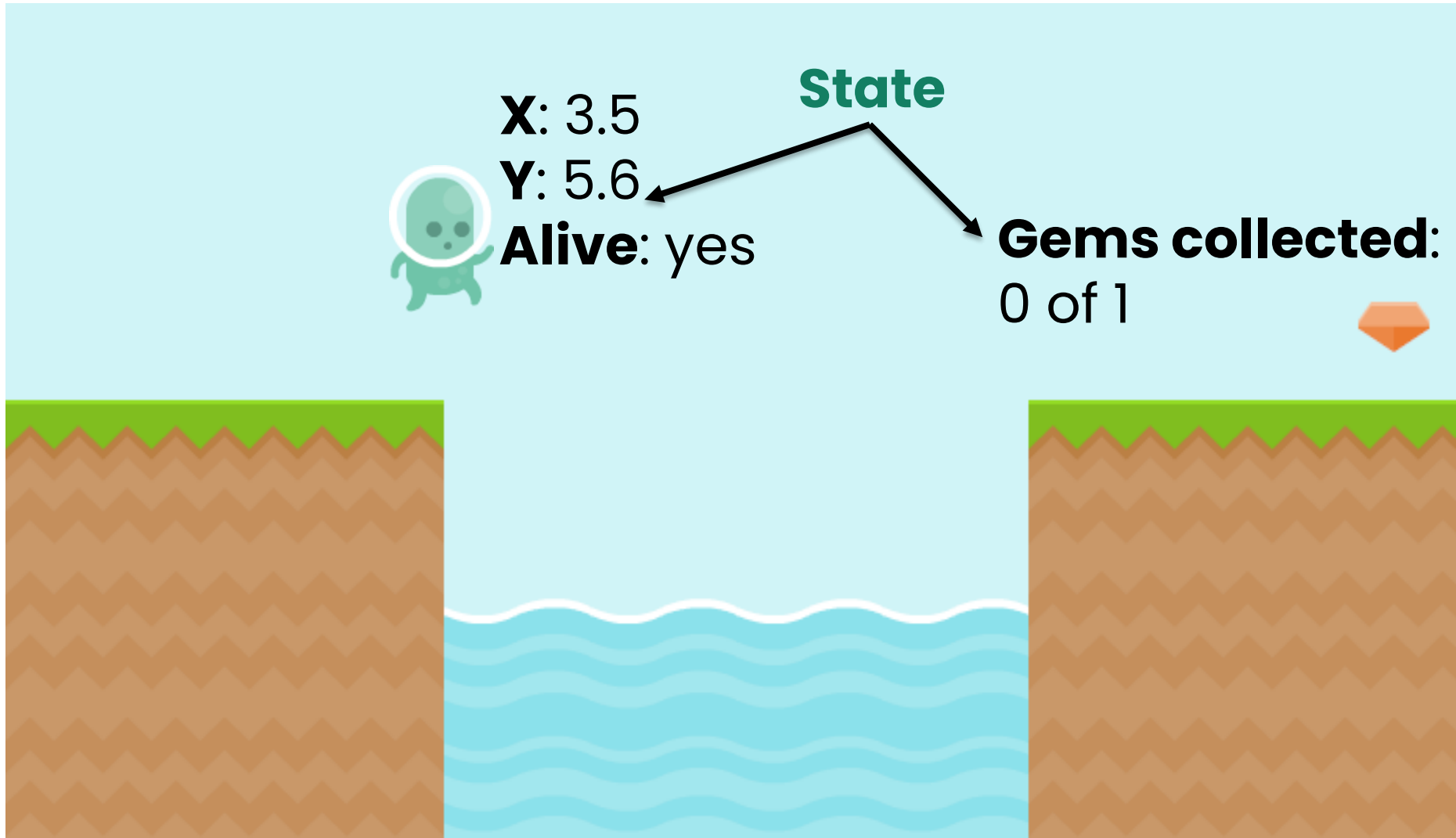




# What we have



# What we have





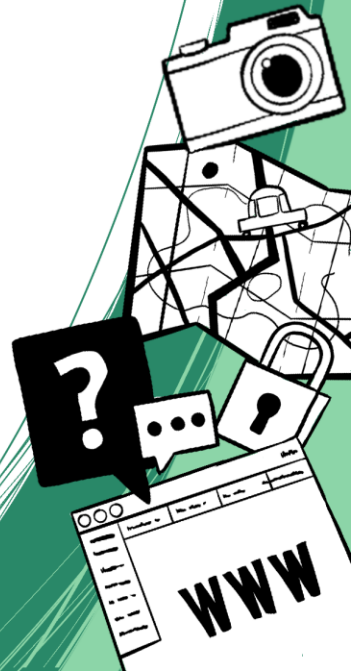
# Example - TicTacToe





# Example - TicTacToe

Agent

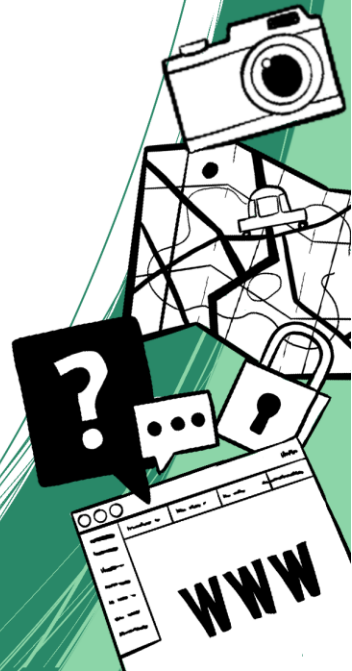




# Example - TicTacToe

## Agent

- Players (X, O)





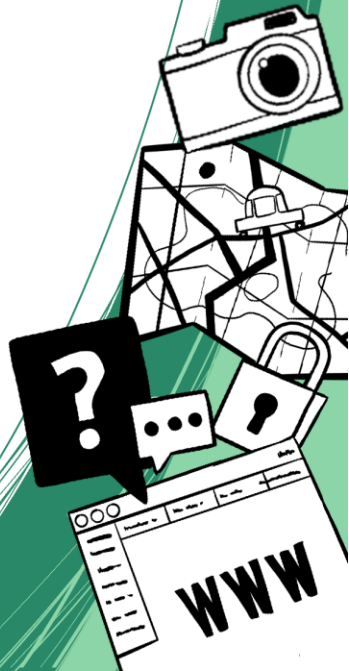


# Example - TicTacToe

## Agent

- Players (X, O)

## State

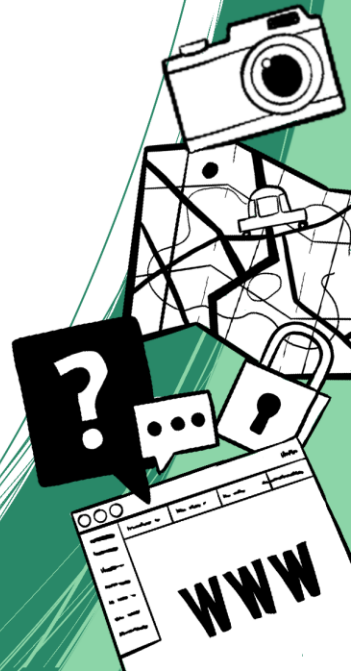
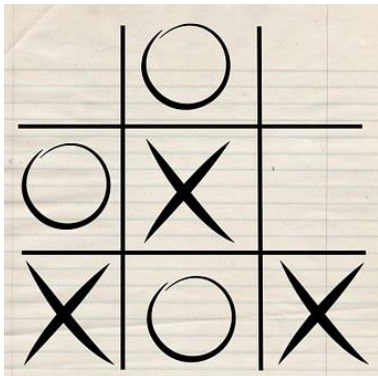


# Example - TicTacToe

## Agent

- Players (X, O)

## State



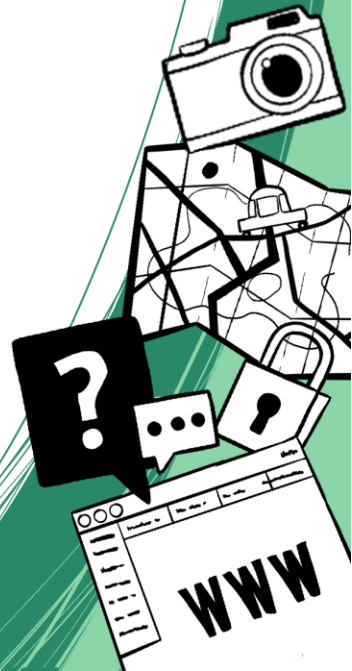
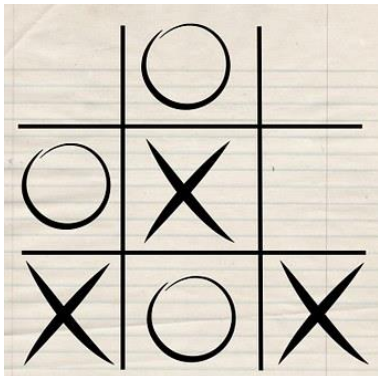
# Example - TicTacToe

## Agent

- Players (X, O)

## Actions

## State

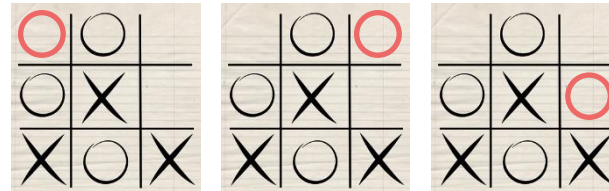


# Example - TicTacToe

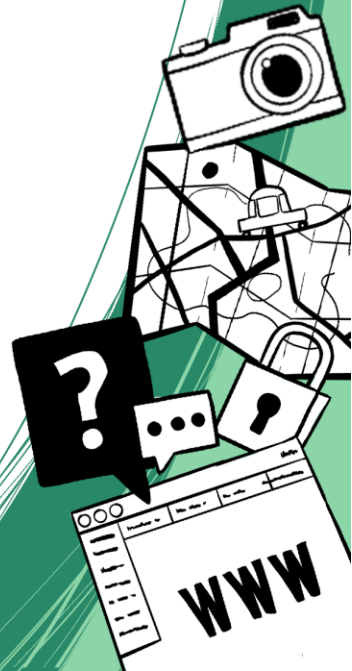
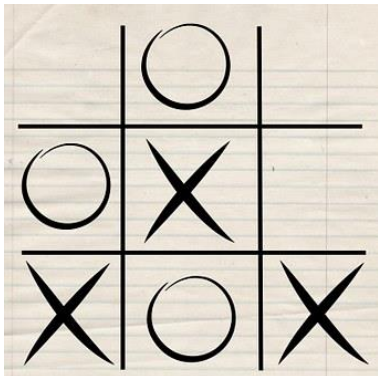
## Agent

- Players (X, O)

## Actions



## State

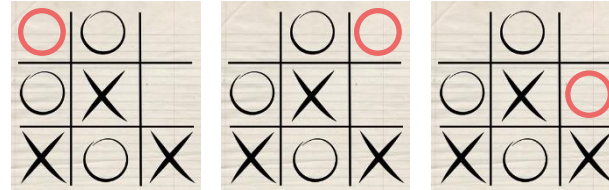


# Example - TicTacToe

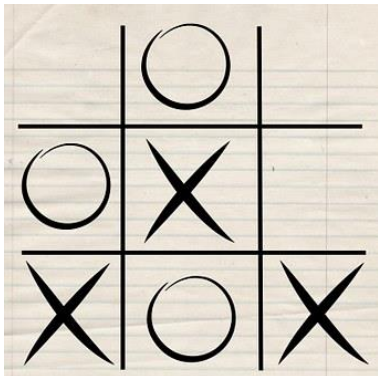
## Agent

- Players (X, O)

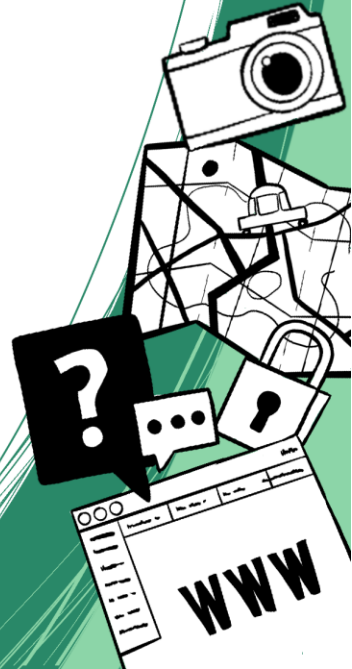
## Actions



## State



## Rewards



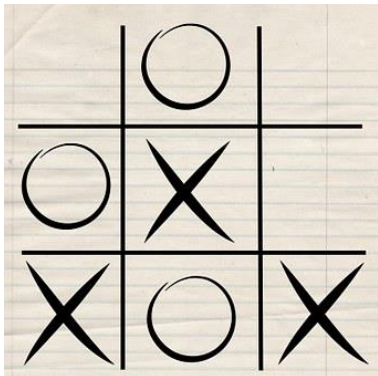


# Example - TicTacToe

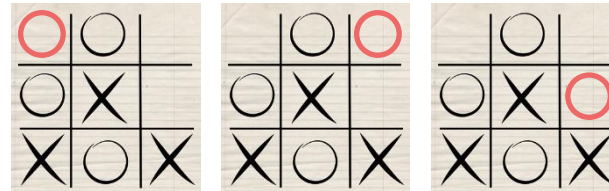
## Agent

- Players (X, O)

## State



## Actions



## Rewards

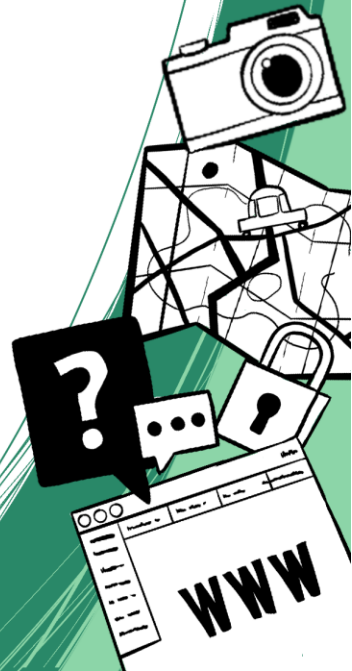
- Won: **+1**
- Lost: **-1**
- Else: **0**



# Further Examples

What are **agents**, **state**, **actions** and possible **rewards** in...

- ...Leela Chess Zero
- ...OpenAI Hide and Seek
- ...Custom Advertisement



# Q-Learning

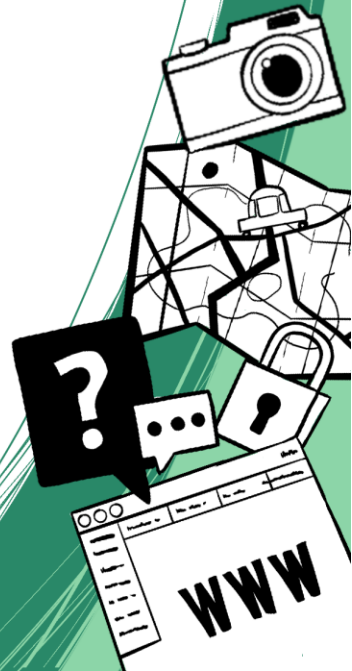
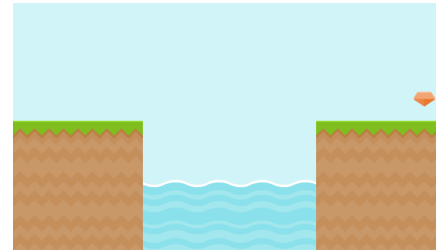


# Reinforcement Learning Loop

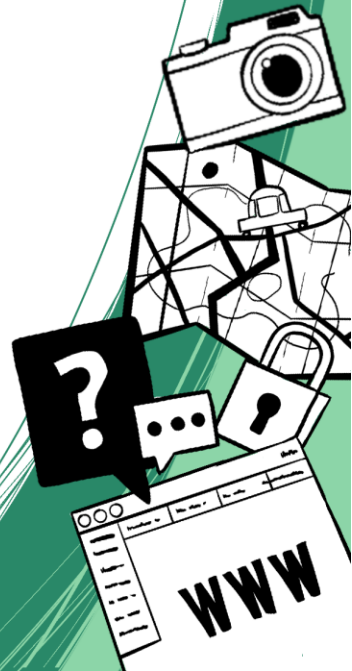
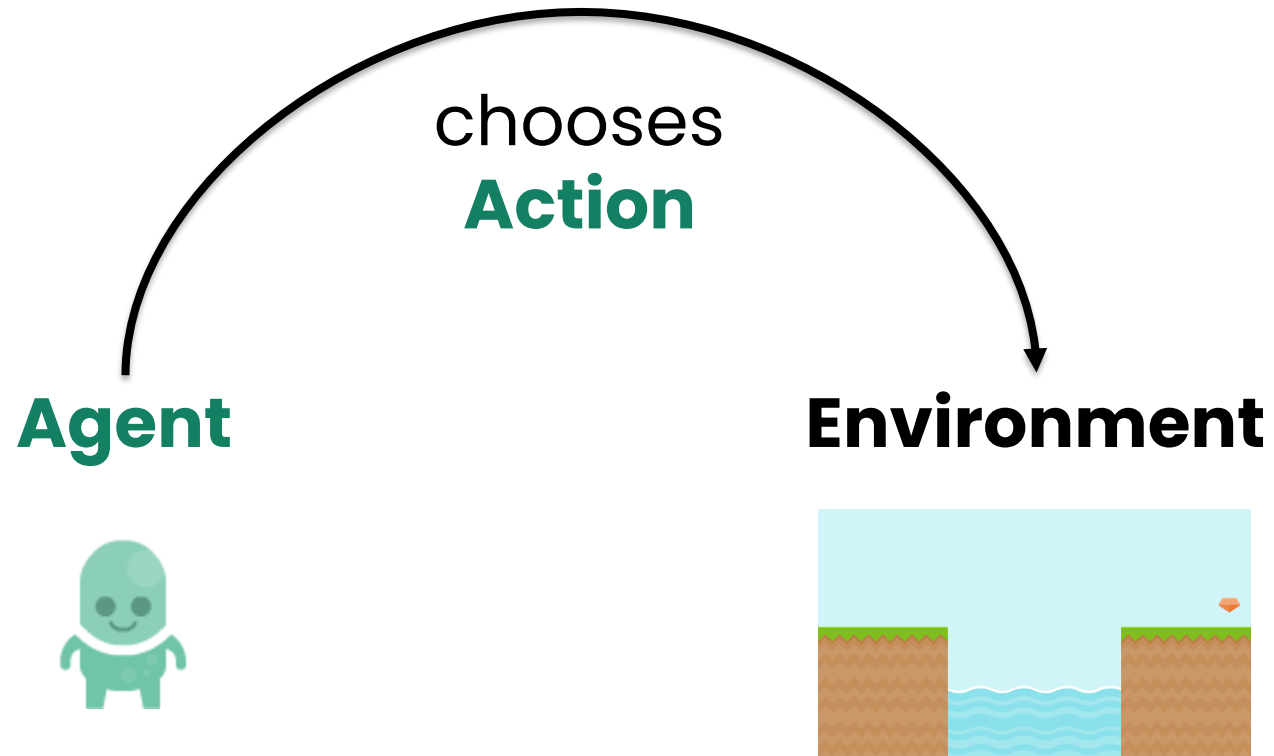
**Agent**



**Environment**

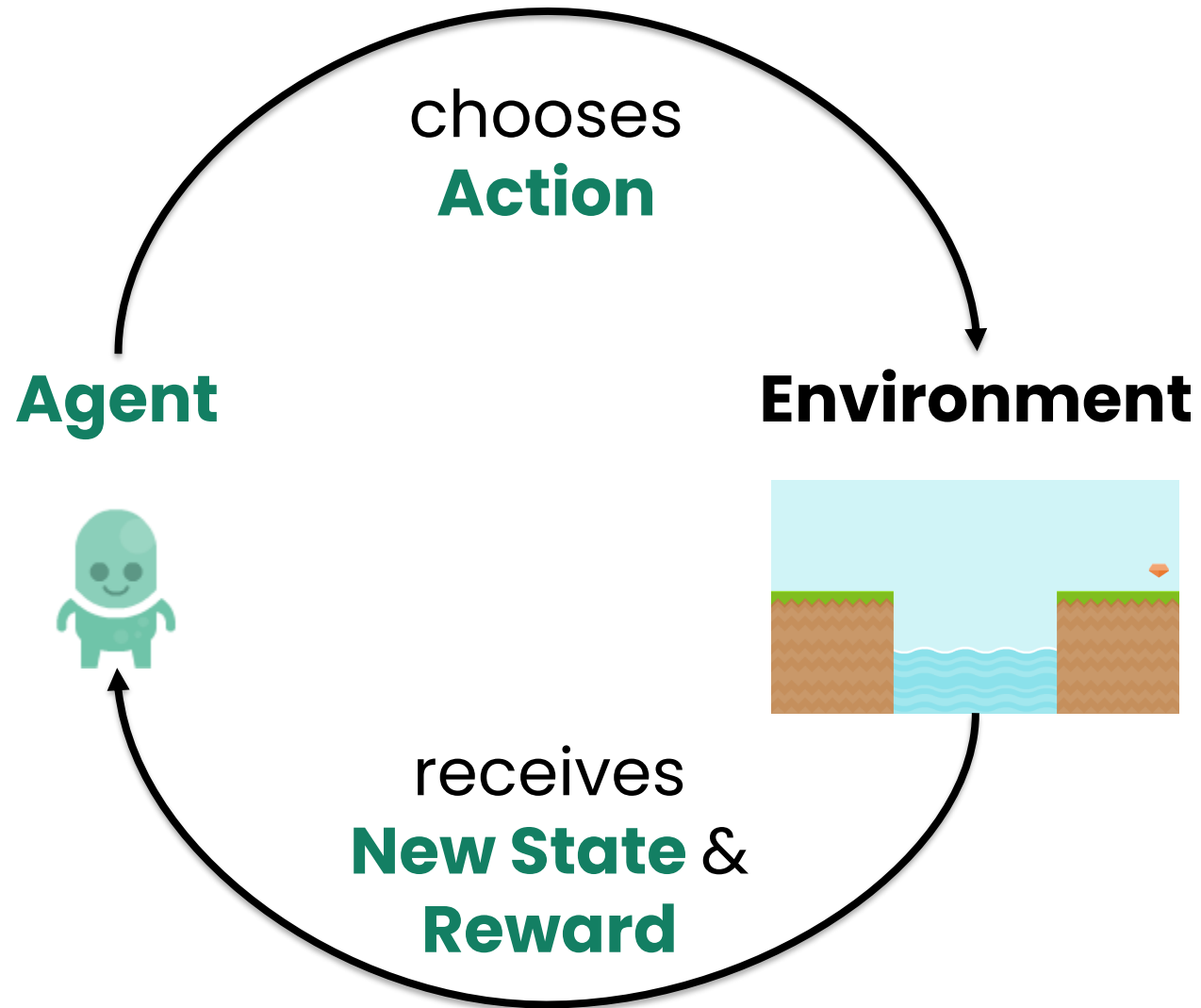


# Reinforcement Learning Loop



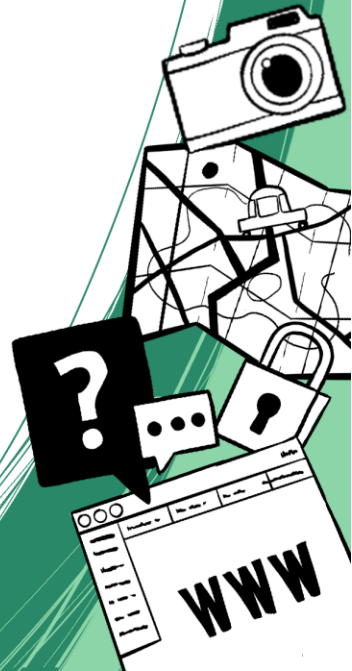


# Reinforcement Learning Loop



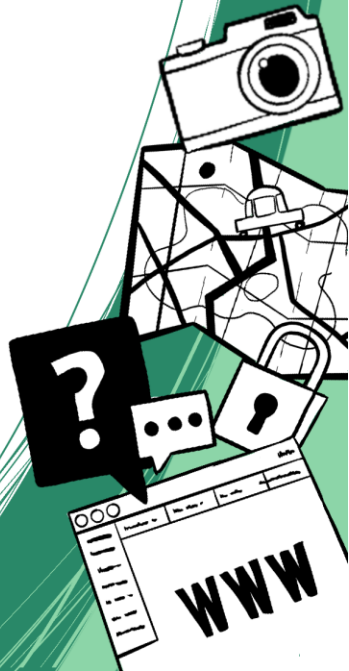
# Q-Learning

- For each **state** we store a **quality value (Q-value)** for each **action** (how good the action is given the state)



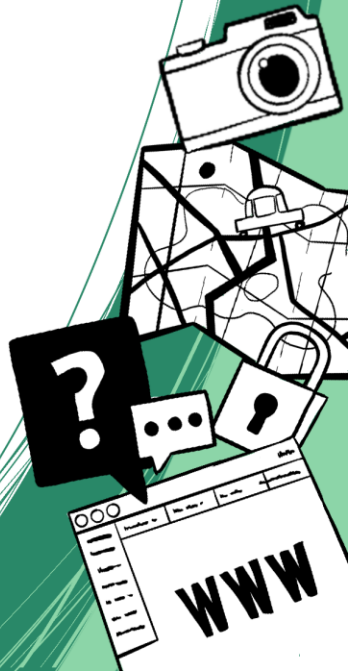
# Q-Learning

- For each **state** we store a **quality value (Q-value)** for each **action** (how good the action is given the state)
- Then, whenever a **state** is reached, choose the **action** with the highest **Q-value**



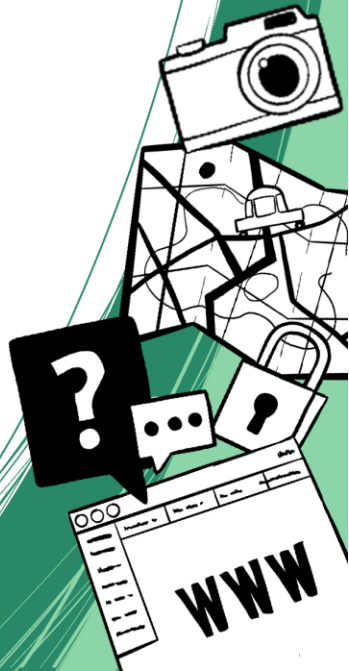
# Q-Learning

- For each **state** we store a **quality value (Q-value)** for each **action** (how good the action is given the state)
- Then, whenever a **state** is reached, choose the **action** with the highest **Q-value**
- Finally **increase/decrease** the **Q-value** in regards to the **reward** after the **action** was performed



# Q-Learning

- For each **state** we store a **quality value (Q-value)** for each **action** (how good the action is given the state)
- Then, whenever a **state** is reached, choose the **action** with the highest **Q-value**
- Finally **increase/decrease** the **Q-value** in regards to the **reward** after the **action** was performed
- For small scenarios, this can be stored in a table (**Q-table**)



# Q-Table example

State	Step left	Step right	Jump left	Jump right
	0	1	-1	0
	-1	0	-1	1
...				



# Q-Table example

State	Step left	Step right	Jump left	Jump right
	-0.2	1	-1	0
	-1	0.5	-1	1
...				





# Coin Game

Try it on a **real example!**

