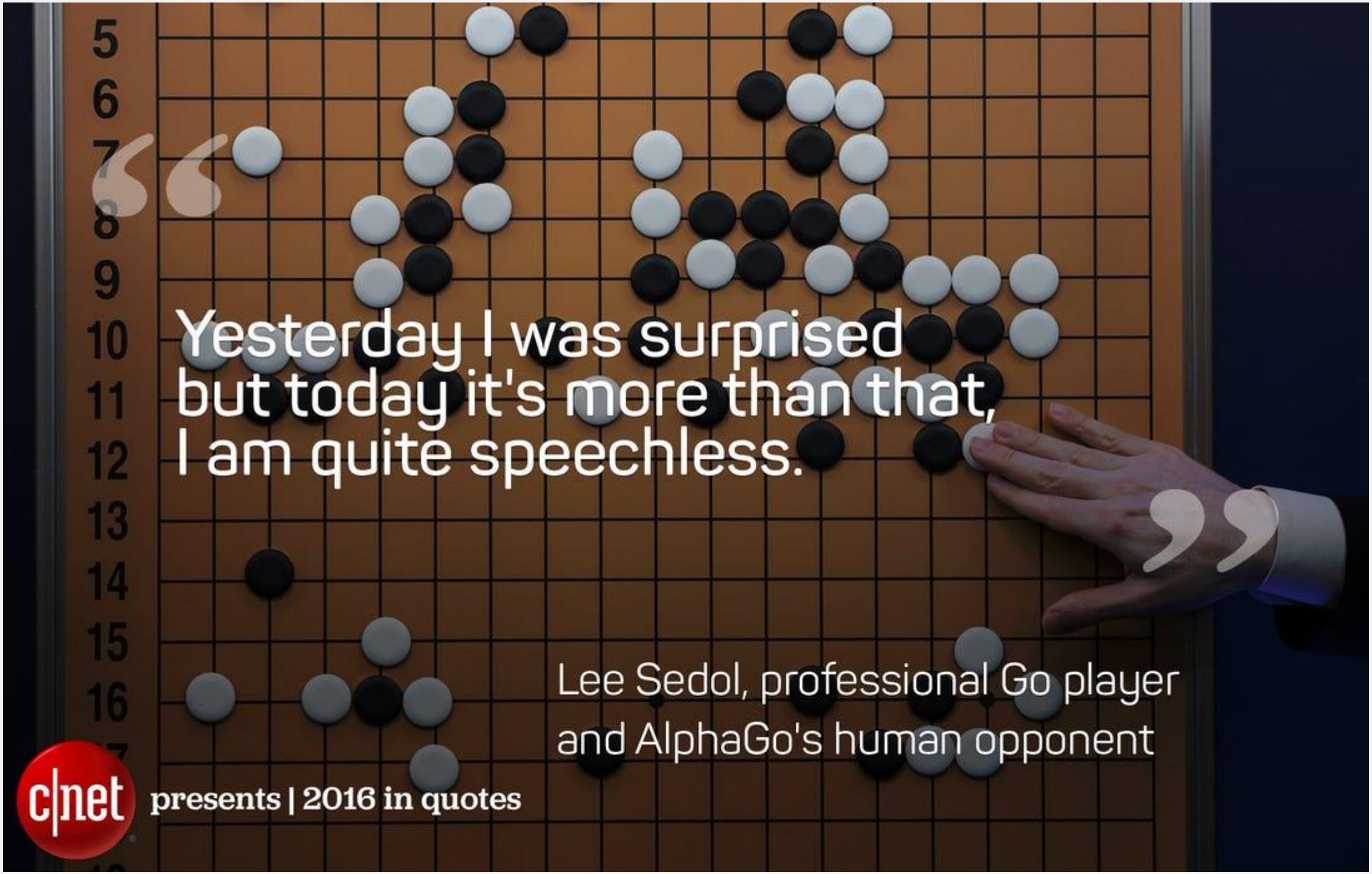
INTRODUCTION TO AI – REINFORCEMENT LEARNING AND GENETIC ALGORITHMS

Philip Mortimer


Yesterday I was surprised
but today it's more than that,
I am quite speechless.
Lee Sedol, professional Go player
and AlphaGo's human opponent
c|net
presents | 2016 in quotes

SUMMARY

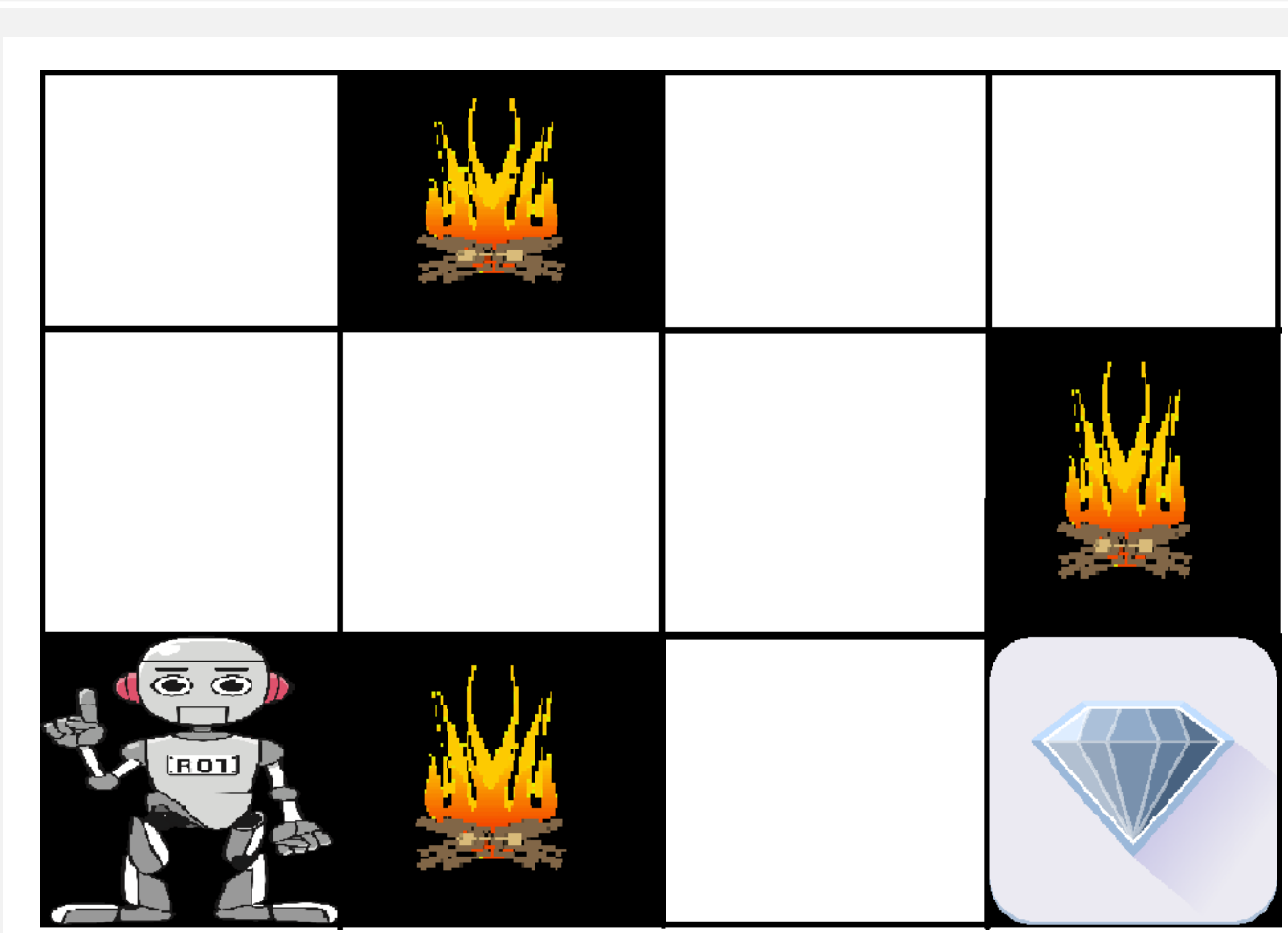
- We discussed unsupervised learning
- Unsupervised learning is a term that covers a wide range of algorithms that draw inferences from **unlabelled** data
- K-means clustering is a way of dividing a dataset into k groups. Datapoints within groups are likely to share similar attributes
- Autoencoders use data as both an input and as a training label
- Autoencoders consist of two neural networks: an encoder and a decoder
- The encoder compresses the original input. The decoder converts the compressed representation to the original format

SUMMARY

- Autoencoders can be applied to image compression and image enhancement
- Discussed principal component analysis (PCA)
- This is a method that converts the attributes of a dataset into a different set of attributes in such a manner that as much information as possible is encoded in the first principal component
- As much remaining information is then encoded in the second principal component and so on
- This allows us to not store the z least valuable principal components, thus significantly reducing the dimensionality of our data

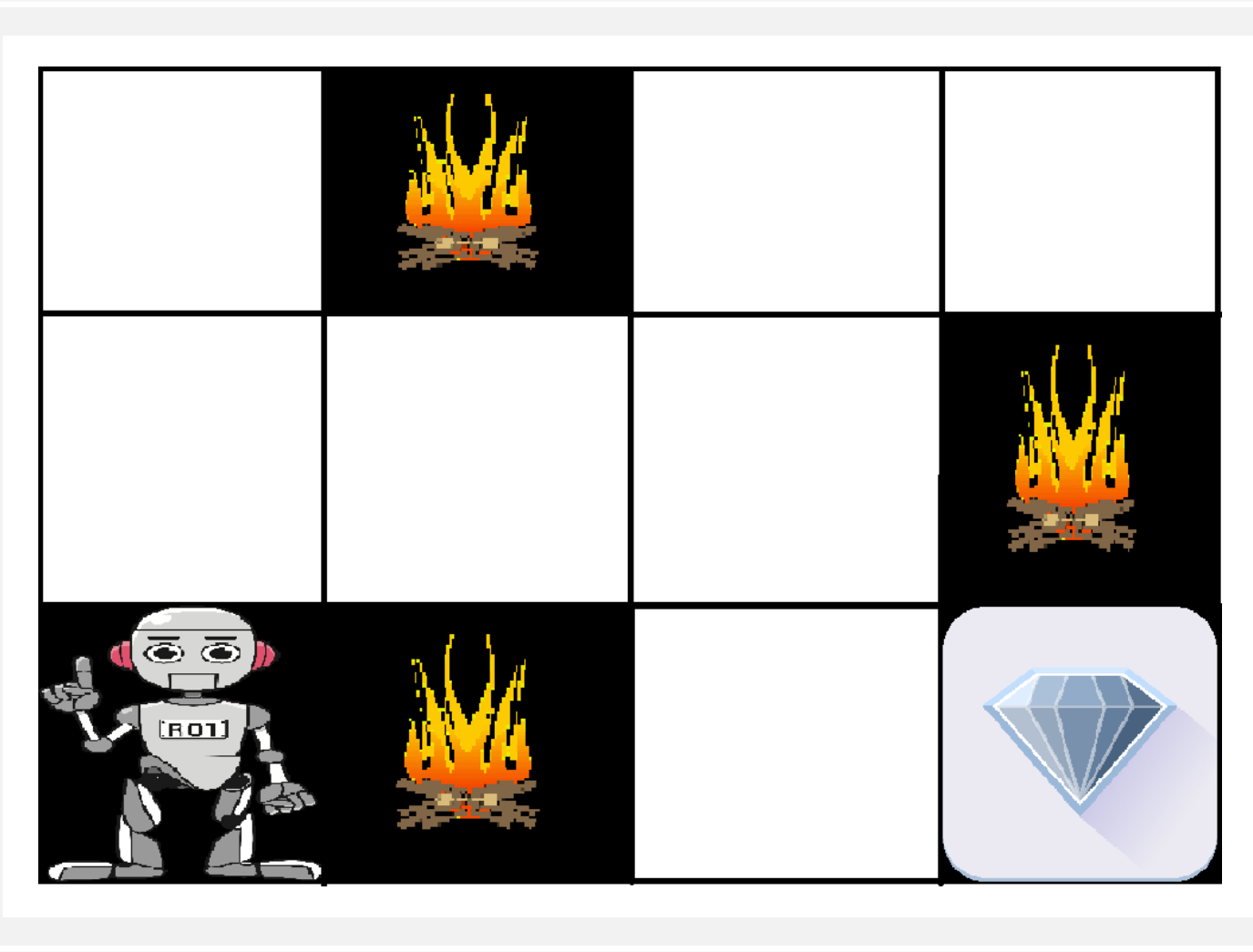
REINFORCEMENT LEARNING

- Reinforcement learning is different to other forms of learning as it does not require any sort of dataset (labelled or not)
- Reinforcement typically involves an **agent**, an **environment** and a set of goals
- Take this example of a robot in a maze, with some obstacles and a reward



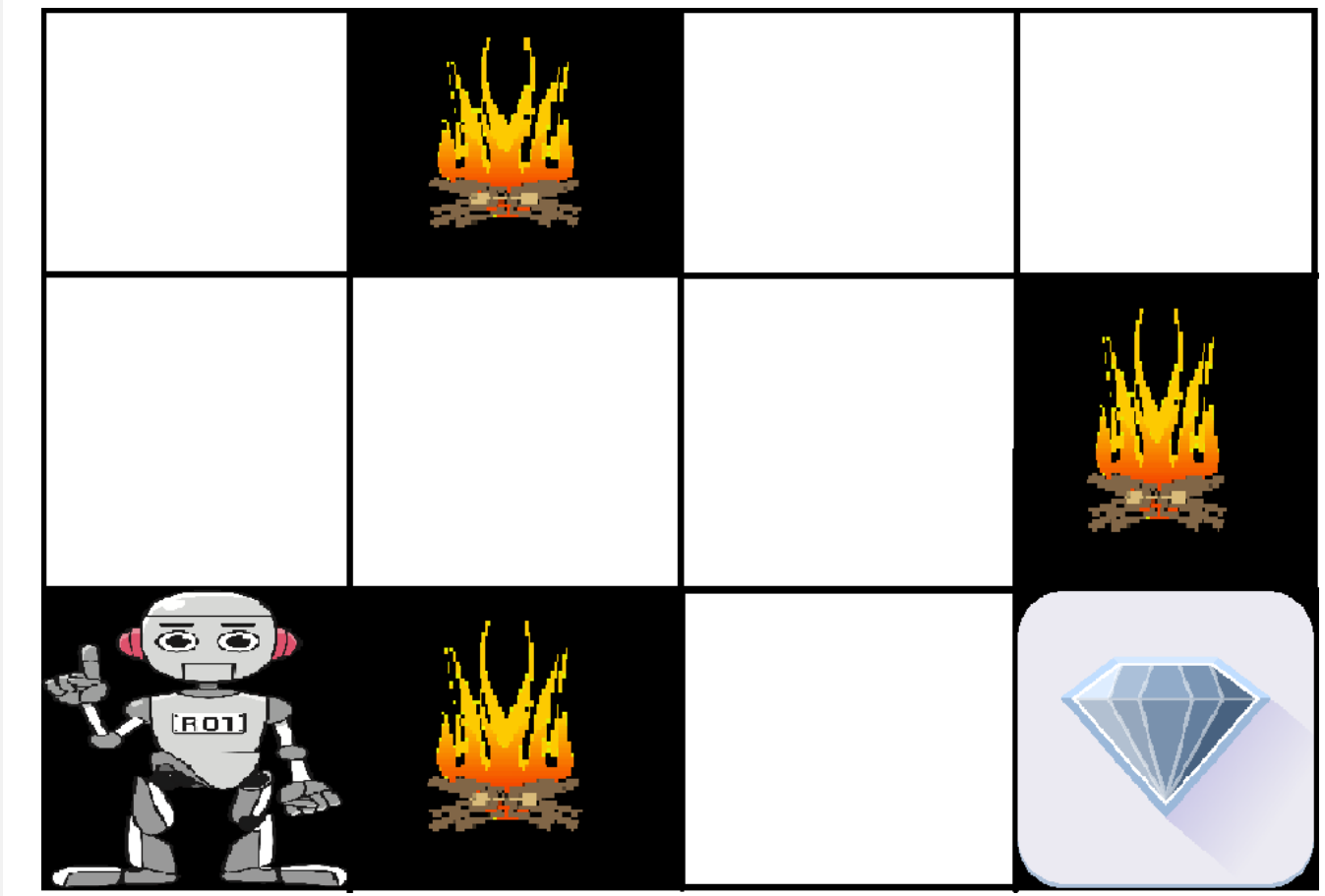
REINFORCEMENT LEARNING

- In this case our agent is the robot and our environment is the maze
- This may be a real world maze, or it may be a very large maze simulation on a computer
- In either case, we may not able to compute a solution to the problem
- Supervised learning is unsuitable as we can't generate data and corresponding labels for optimal moves



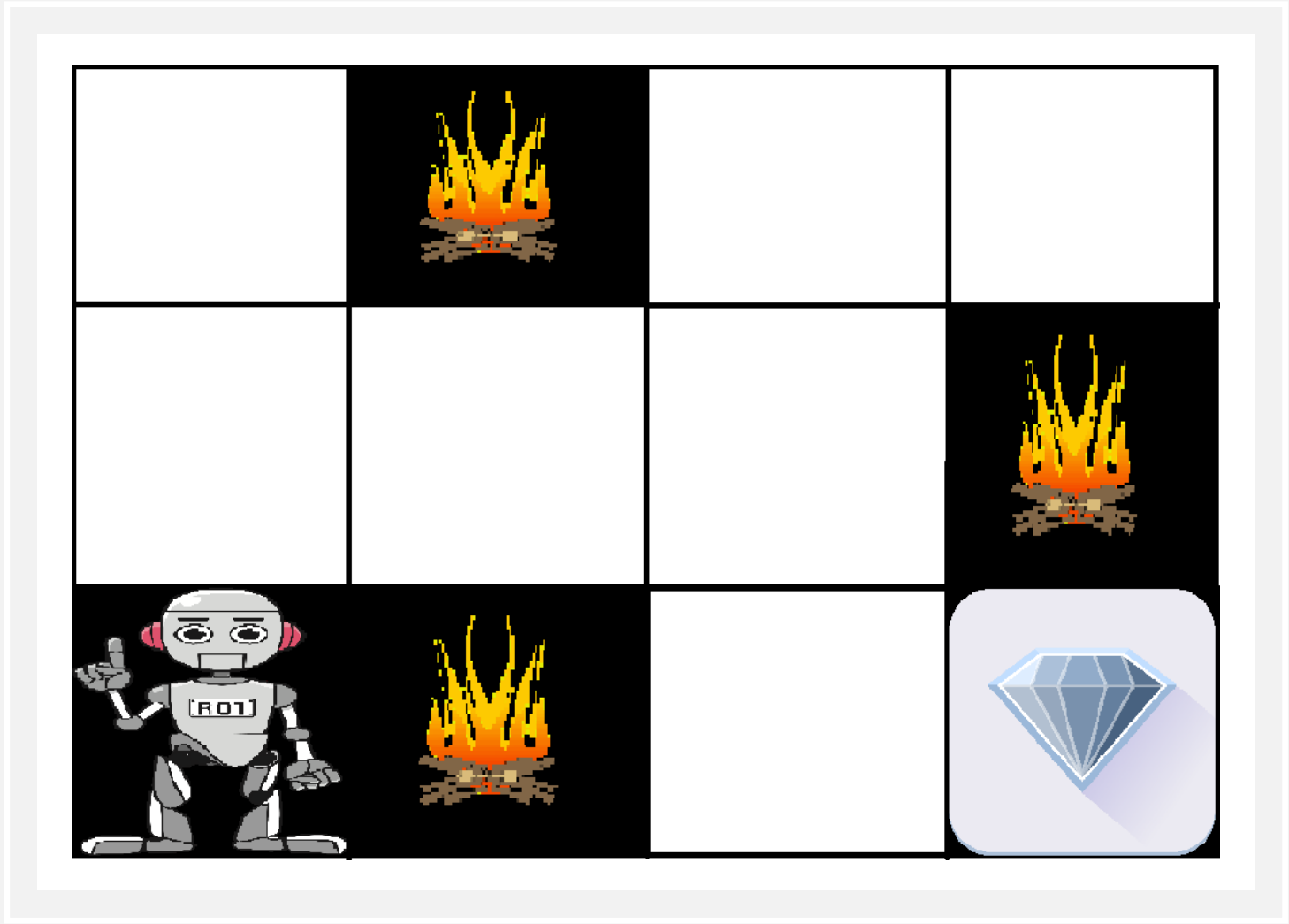
REINFORCEMENT LEARNING

- Unsupervised learning is also not suitable as meaningful data analysis / collection is computationally impossible
- As the problem space may be unknown or too large to enumerate over, what we do instead is define a set of actions that have scores associated with them
- For example, we may say that hitting an obstacle results in -100 points



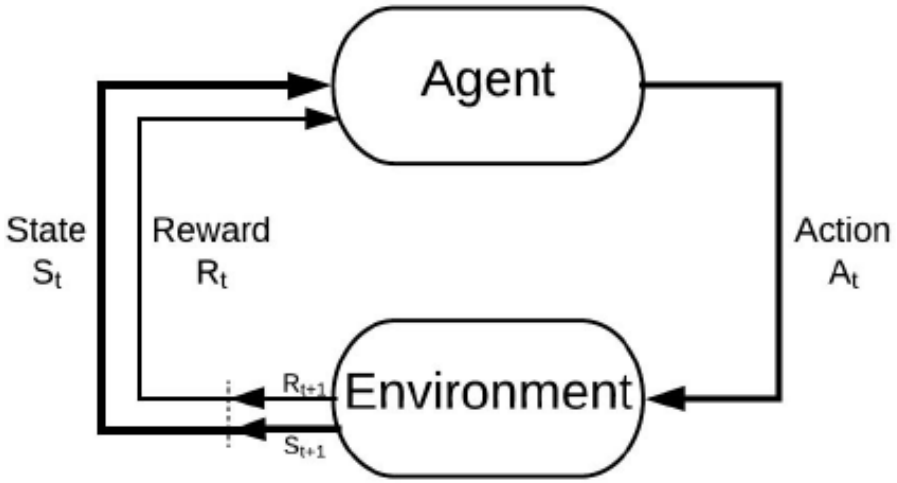
REINFORCEMENT LEARNING

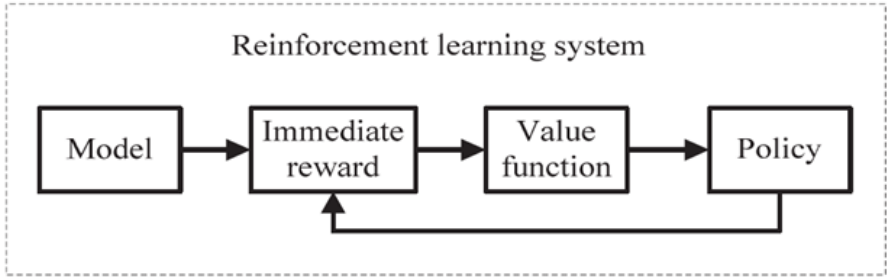
- Finding a diamond may result in a score of +100 points
- We then take actions and reward our AI for doing good things and punish it for doing bad things
- Reinforcement learning (RL) is often compared to training a dog. If we throw a stick and the dog retrieves it, we may reward it with a treat. Conversely, if the dog is badly behaved, we may choose to not reward it
- Over time, our dog learns to behave well



RL

- The idea is that RL involves an environment state and a sequence of actions that can be taken for each state
- A new state can be fully described by the previous state and the action taken at that state
- After we take our actions, we can reward or punish or agent depending on our reward policy





REINFORCEMENT LEARNING

REINFORCEMENT LEARNING

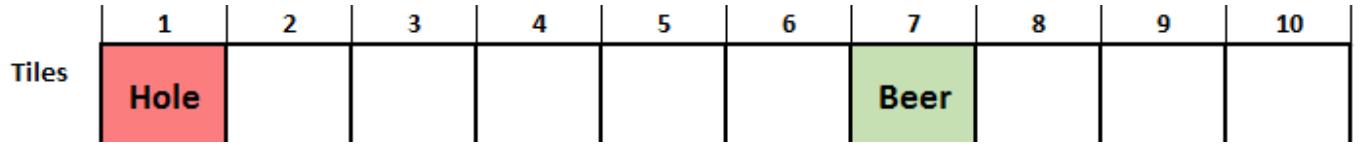
- Reinforcement learning can be broken down into two broad categories: **model-based** and **model-free**
- A model is simply something that takes a state and an action and converts it into a new state.
- In a game of chess, we already have the model (we know exactly how each move should change the state)
- However, we may have a robot that is operating in the real world. If the set of actions it can take are sufficiently complex, we wouldn't be able to code a perfect model (without knowing all of the laws of physics). In this case, we simply have a current state and a set of actions we can take
- Note that in model-free, we still can take an action and get a future state

MODEL-FREE LEARNING

- The most common algorithm that doesn't require a model, is called Q learning
- Q learning requires no prior knowledge
- Q learning creates a table which stores all possible actions and then which actions lead to various environment states
- All we need to know is the start state and the set of possible actions.
- To introduce this, we will look at a simple example

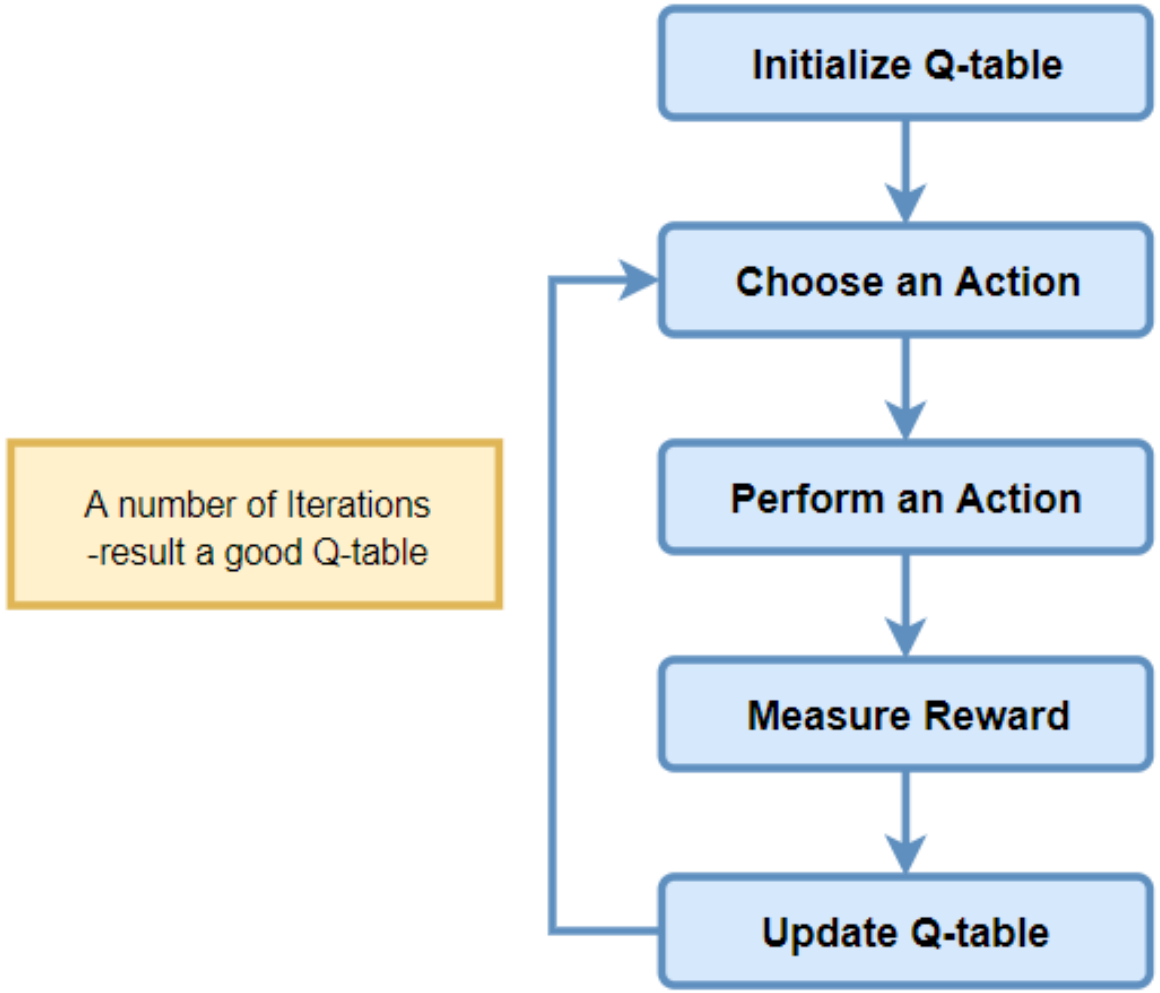
Q LEARNING

- Let's say we have a ball that can be at any point along this one dimensional line
- The goal is to reach the goal (in this case labelled "beer")
- If we fall in the "hole" area, we lose
- In our case, we may be in three states: winning, losing or undetermined.
- Let's say that "winning" actions give us a score of +100. "Losing" is -100 and "undetermined" is 0.
- This is our **reward policy**



Credit: https://itnext.io/reinforcement-learning-with-q-tables-5f11168862c8

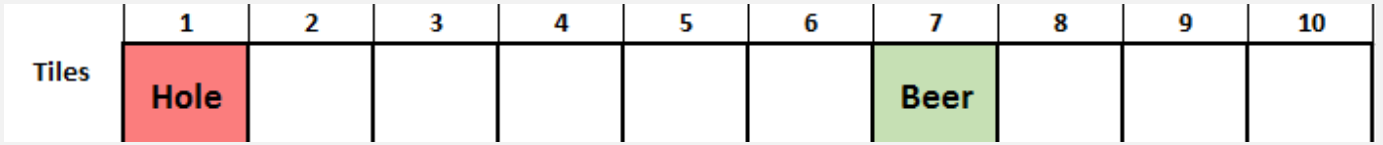
ALGORITHM


Initialize Q-table
Choose an Action
Perform an Action
Measure Reward
Update Q-table
A number of Iterations
-result a good Q-table

INITIALISE Q TABLE

- We need to initialise our Q table. This table stores our maximum expected reward for taking a certain action at a certain state
- We initialise each item in our table to zero

	Action	
State	Left	Right
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

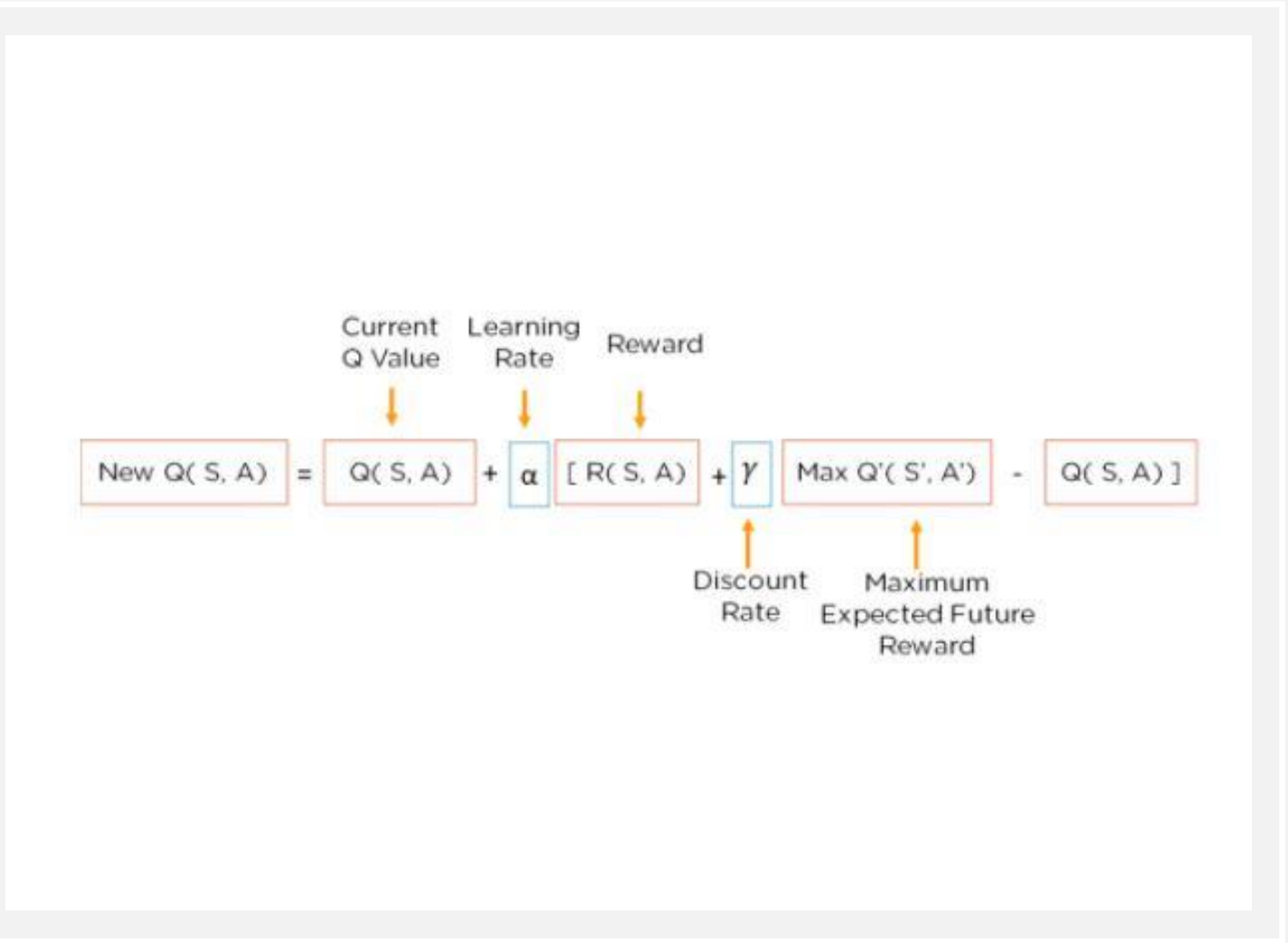


PLAYOUT

- We know play lots of random game playouts
- Each time we are at a given move, we update our Q table using a formula known as the **Bellman** equation

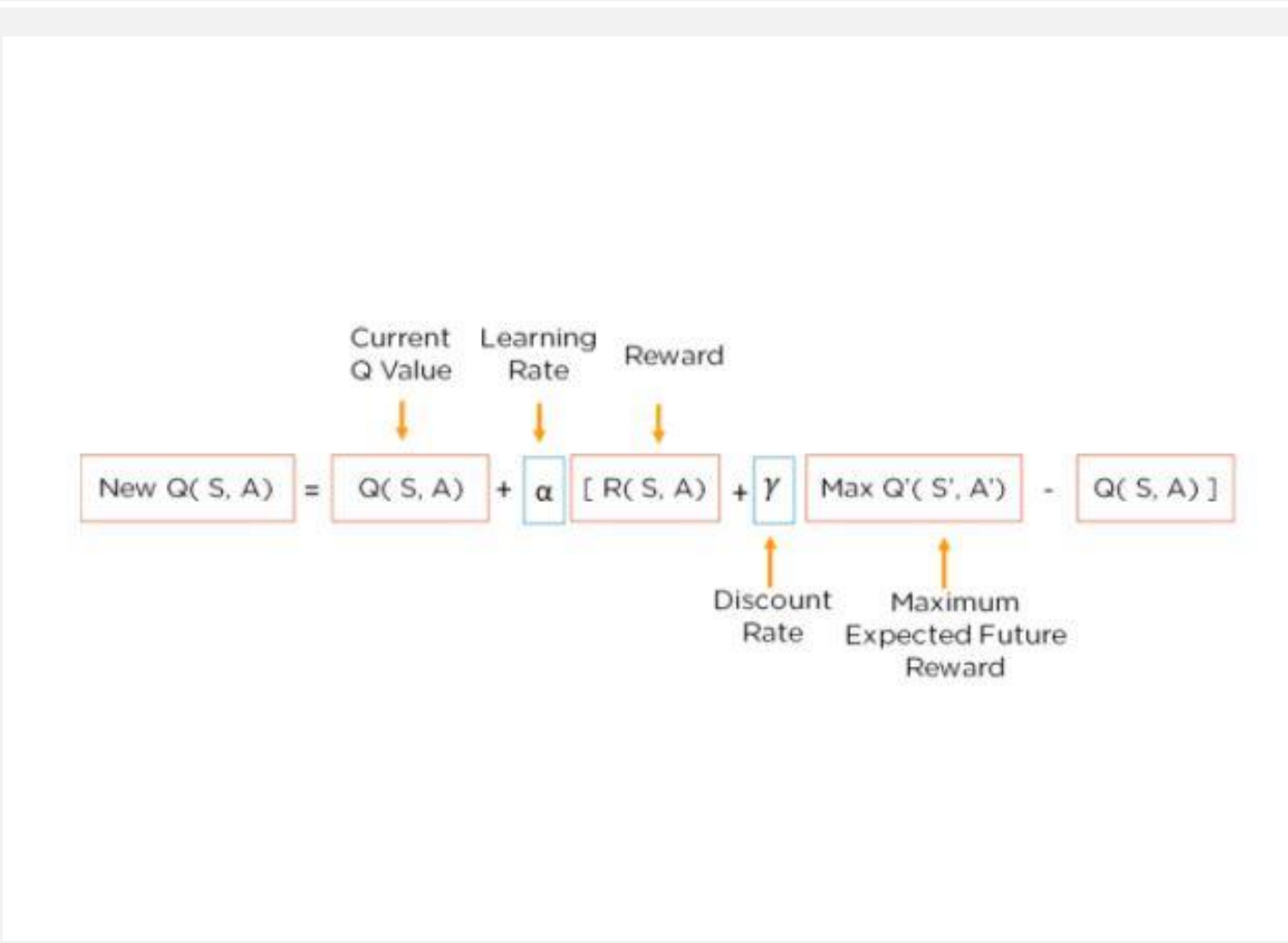
BELLMAN EQUATION

- We are at a state and we select an action semi-randomly using an epsilon greedy approach
- This means that we are more likely to choose "good actions" but also have a chance of exploring other states
- As training goes on, we are less likely to explore
- Note Q(S, A) just denotes an item in our table where S is the state and A is the action
- Learning rate and discount rate are hyperparameters



BELLMAN EQUATION

- R(S,A) is our reward policy for the action in the current state (+100 for a win, -100 for a loss etc.)
- Max Q'(S',A') refers to the maximum value in the q table for the new state S'. S' = the new state resulting from taking action S at state A.
- Thus, we are finding the best possible value for the next state after taking the action



Q-LEARNING

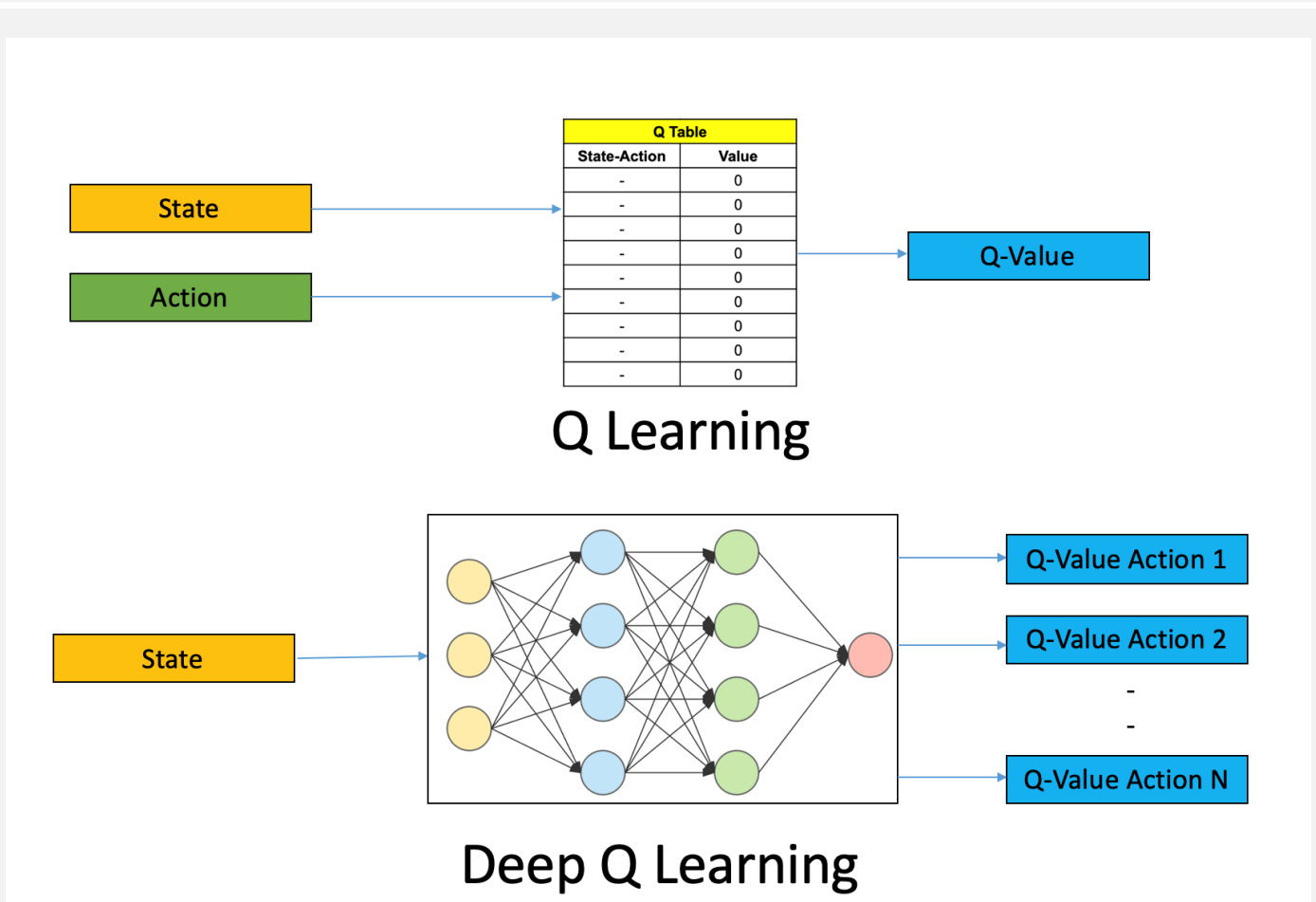
Through iteratively running this algorithm, we train an agent to play the game

When using this agent in the real world, we will select the value from the Q table for the current state that leads to the best future reward

	Action	
State	Left	Right
0	0	0
1	-100	65.61
2	59.049	72.9
3	65.61	81
4	72.9	90
5	81	100
6	0	0
7	100	81
8	90	72.9
9	81	0

Q-LEARNING

- Q-Learning makes produces it's own estimate of the actual model, as well as optimal policy
- This makes it very prone to errors and also less effective than supervised approaches
- It also takes up a lot of memory and becomes tricky to use for very large state spaces
- Deep Q learning is a modern solution to this issue, which makes it make more feasible for real world use

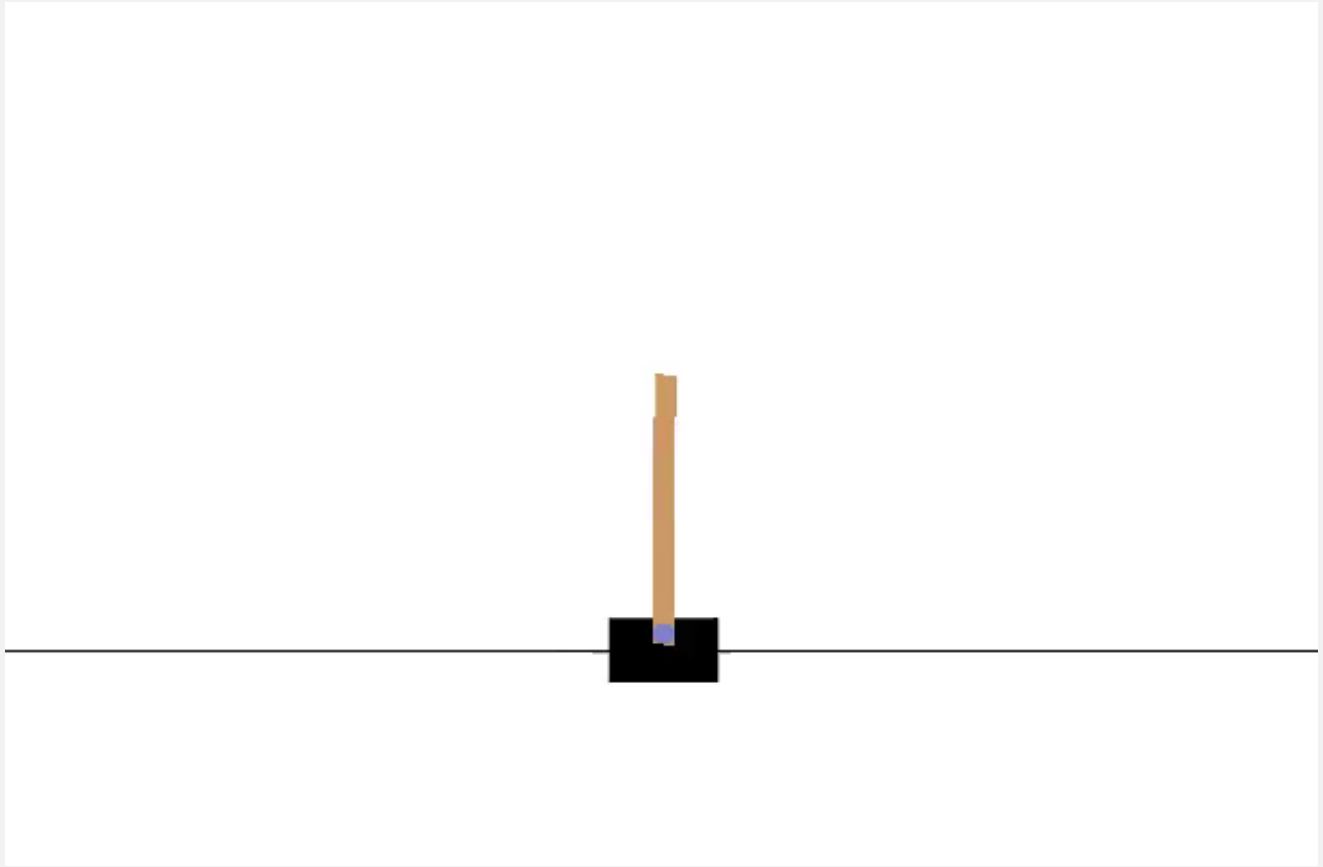


DEEP Q LEARNING

- The details are complicated, however the long and short is that it performs very well and is a versatile algorithm
- The idea is that we have a deep neural network that replaces our table

DEEP Q LEARNING

- We can train a deep q model. I have written code to do so here: https://github.com/philipmortimer/AI-Course/tree/main/Programs/Part%207/cartpole_solver
- Here's the output of a trained deep q model on Cart Pole game



MONTE CARLO TREE SEARCH

- Monte Carlo Tree Search (MCTS) is a tree search algorithm often used to search a game tree (which is what the minimax algorithm does)
- MCTS is very useful for games with a high branching factor, games where heuristics are hard to define and games where imperfect information / probability is present
- Given infinite compute time, it will make an optimal decision (converge to minimax)
- MCTS has been successfully applied to games like Go, which (until recently) has been an area of failure for AI research

MONTE CARLO TREE SEARCH

- MCTS involves **four** phases

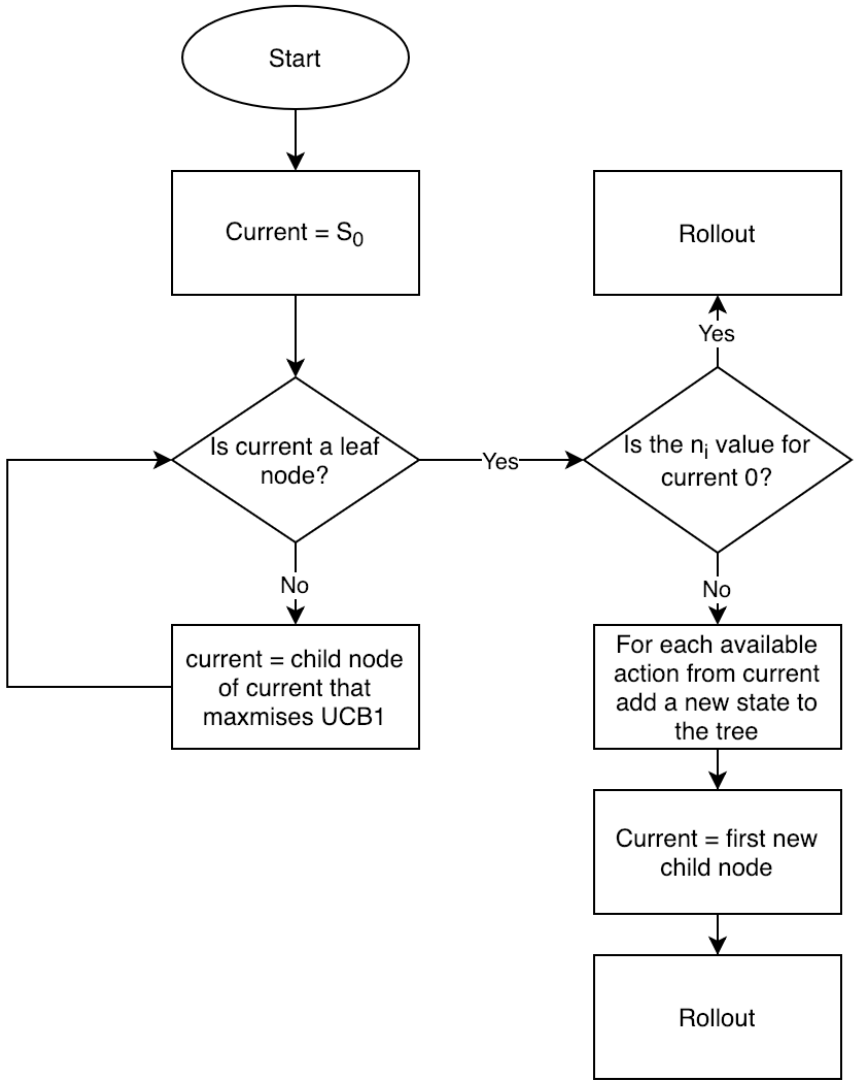
1. **Selection** – Select nodes from the root that represent good "moves"
2. **Expansion** – If our selected leaf node is not a terminal node, generate all possible future nodes and select one. Only do this if the node has already been explored
3. **Simulation (rollout)** – Run a simulation (playout) of random moves from the selected move and store the eventual game result
4. **Backpropagation** – Update the current move sequence with the result

UCT

- For our selection, we use a formulae known as UCT to evaluate the score of a given node. For all children nodes, we chose the node that maximises the UCT formula
- The idea behind UCT is to balance exploring states that we haven't looked at a lot with states that seem to lead to better outcomes
- X_j is win ratio of child
- C is constant used to balance between exploration and exploitation
- n is number of times parent node has been visited
- n_j is the number of times the child node has been visited
- A common value for C is $\sqrt{2}$

$$UCT_j = X_j + C * \sqrt{\frac{ln(n)}{n_j}}$$

MCTS


Start
Current = S_0
Is current a leaf node?
Yes
No
current = child node of current that maxmises UCB1
Is the n_i value for current 0?
Yes
No
Rollout
For each available action from current add a new state to the tree
Current = first new child node
Rollout

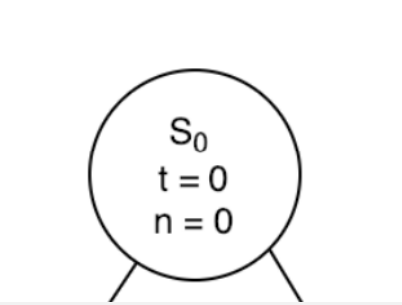
MCTS

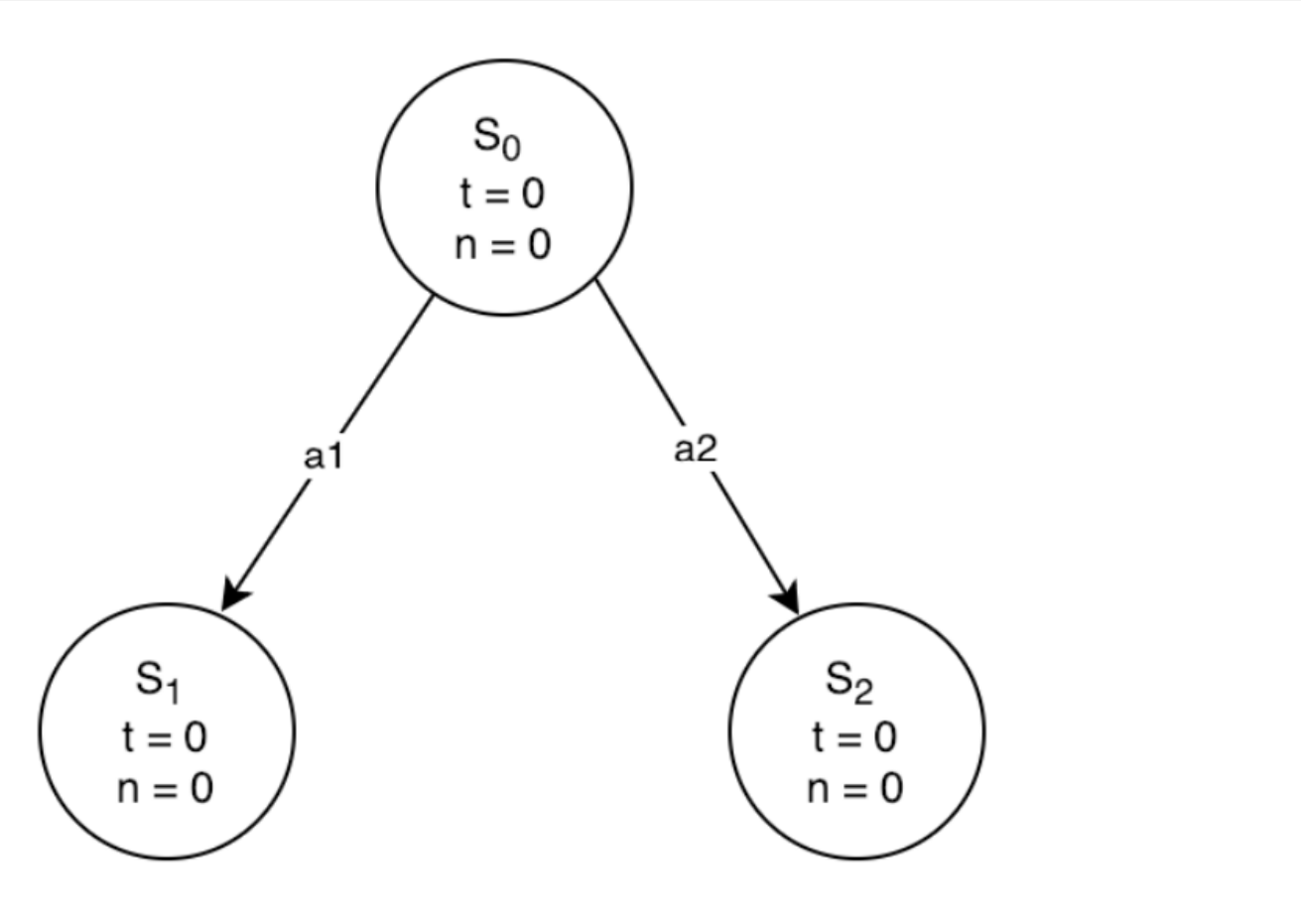
- Let's look at an example. In this example, we'll say $C = 2$
- For each generated node, we keep track of a few variables.
- We track the total score of the state. This variable is t. In practice this means that if the rollout leads to a win, we increment t by 1. If it leads to a loss, we decrement it by 1. And if it's a draw, we don't change it
- We also keep track of a variable n, which denotes how many time the state has been visited
- Hence our average reward is simply (t / n) (i.e. $X_j = \frac{t_j}{n_j}$)

Credit: https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/

MCTS

- We start with our root state S_0. We may imagine that each state represents a chess board, for example.

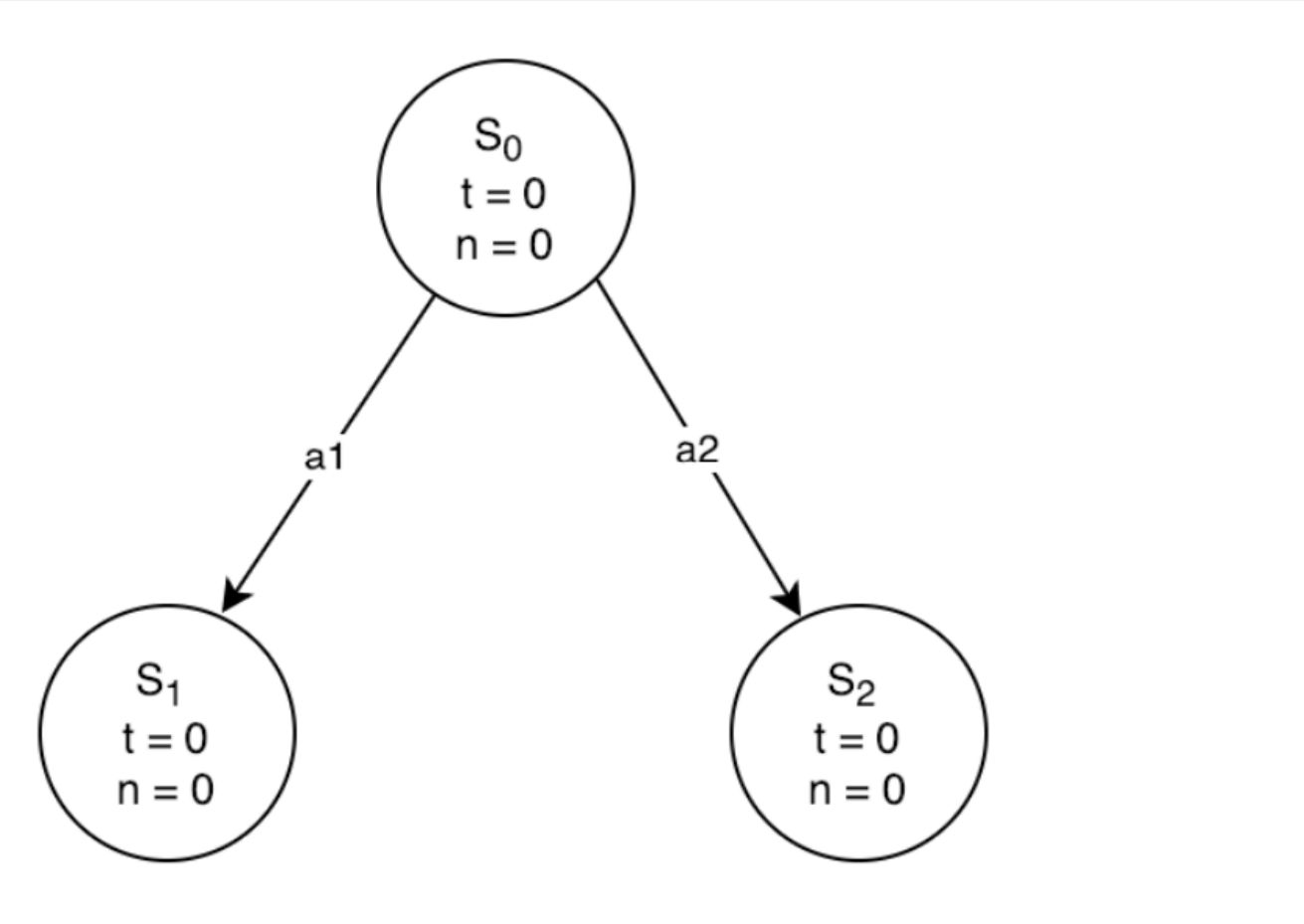




MCTS

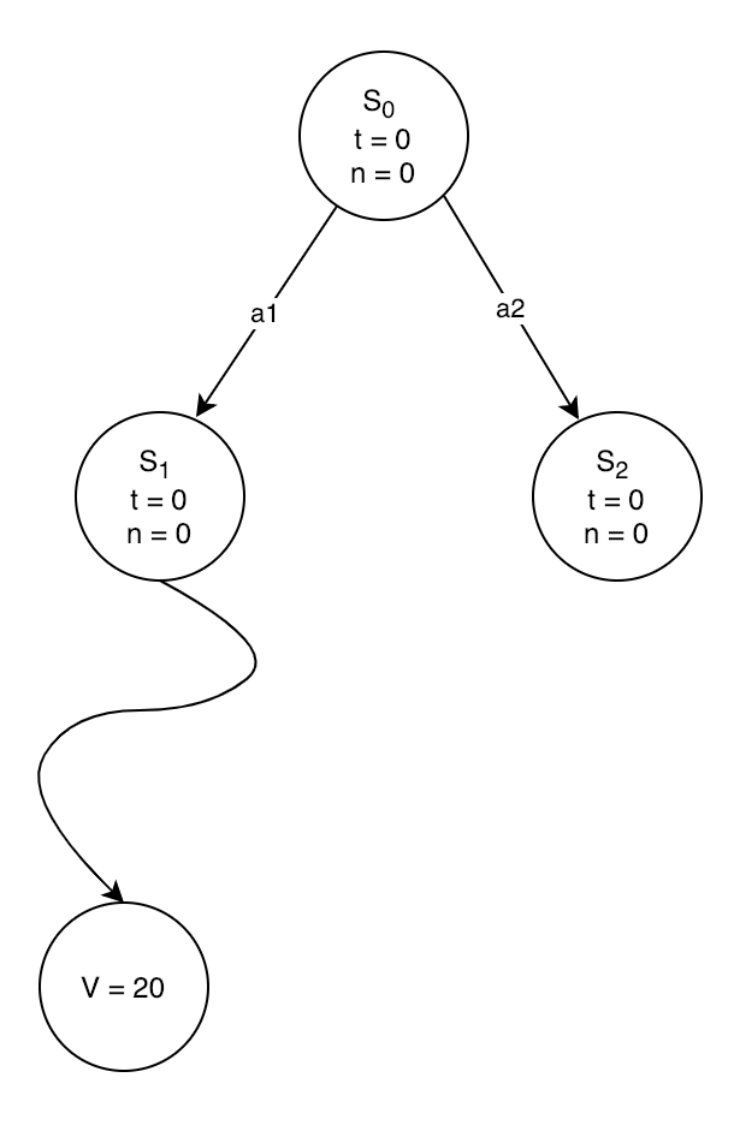
- As this is the root node, we perform the **expansion** phase.
- Thus, we generate all possible actions from this state
- In this case, we find that there are two possible actions, each leading to two different states
- We add these to our game tree, with values of n and t initialised to 0

$$UCB1 = V_i + 2\sqrt{\frac{\ln N}{n_i}}$$



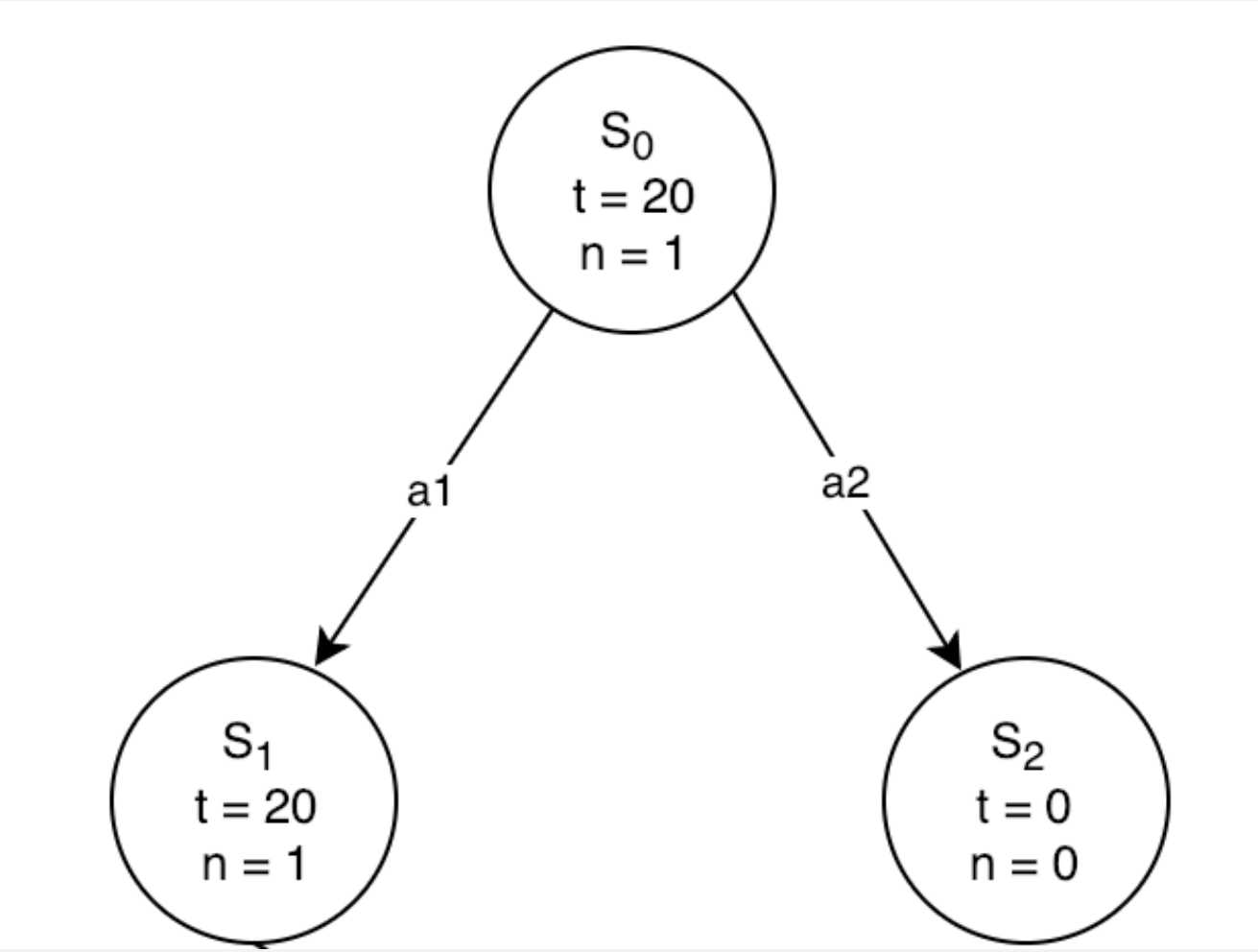
MCTS

- Now, we have to choose which node to select (we are in the **selection** phase)
- We apply the UCT formula to each node and select the node that maximises this value
- In this case $UCB(S_1) = UCB(S_2)$
- We treat unexplored nodes as having a UCB of +infinity
- In this case, we randomly select one of the two states. Let's choose S_1



MCTS

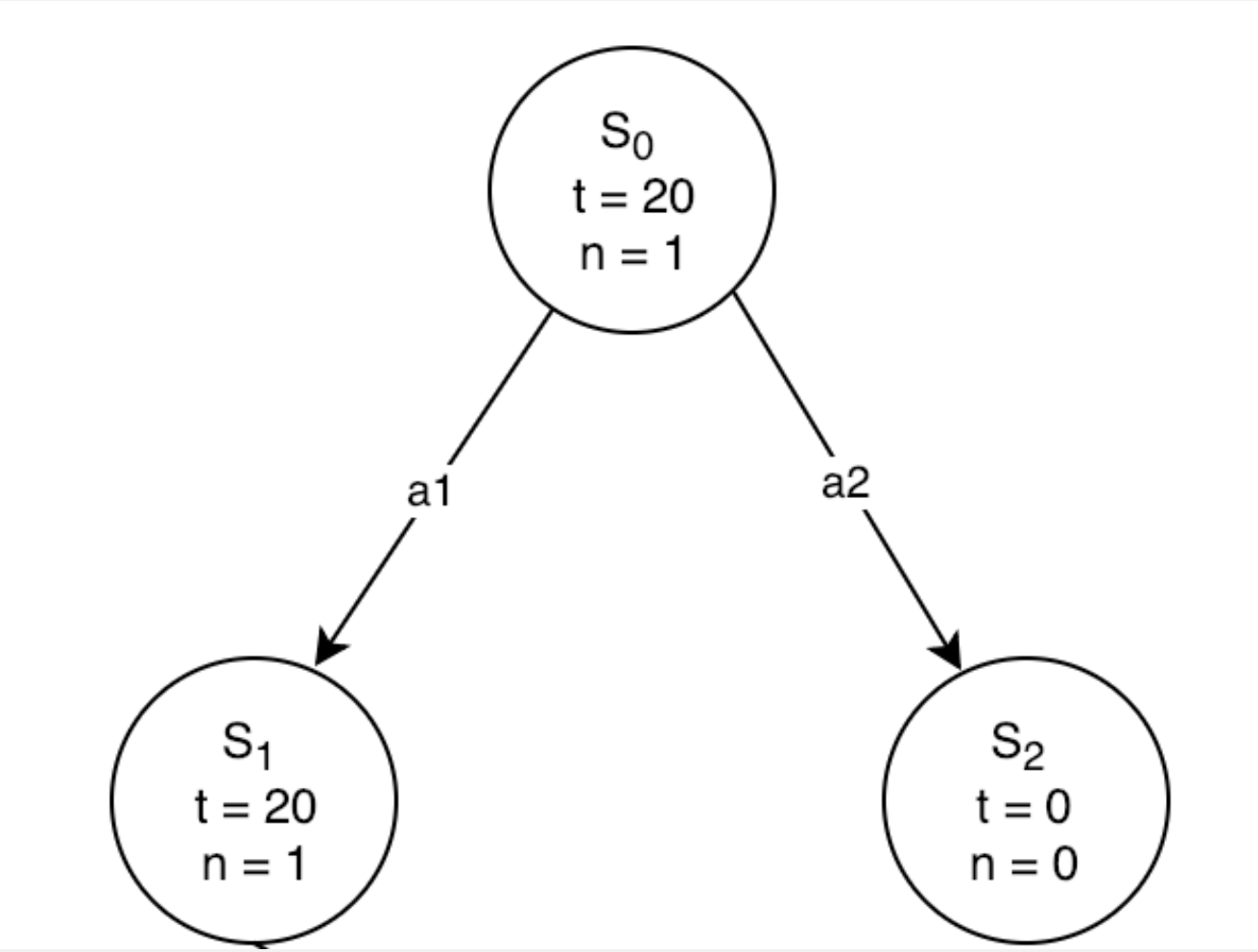
- We have selected S_1. This is used in the **rollout** phase
- We start at S_1 and take random actions until we reach a terminal state.
- This terminal state is assigned a score given our reward policy
- In the case of chess (and this example), a score of 20 may represent a win for the root player



MCTS

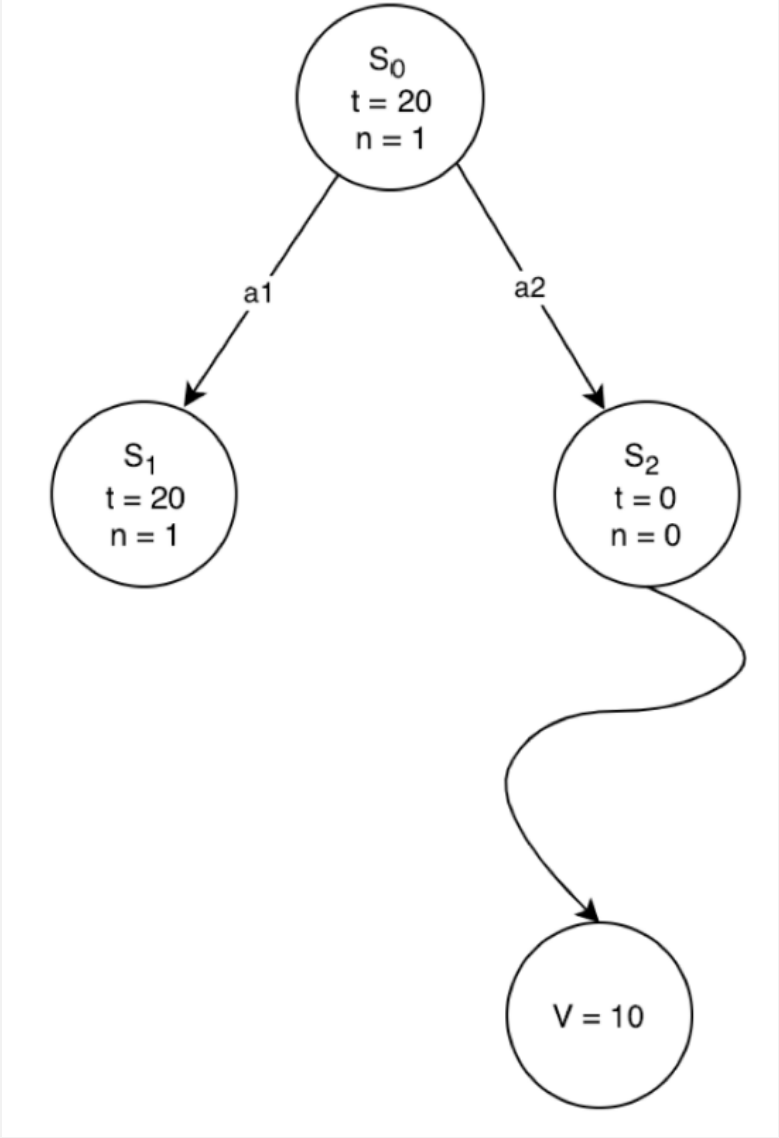
- After the rollout phase, we propagate the score back up the tree
- This is the **backpropagation** phase
- All moves selected in the sequence that lead to the root node of the expansion phase have n incremented by 1.
- Additionally, t is incremented for each of those nodes by the rollout reward
- We have now completed our first iteration

$$UCB1 = V_i + 2\sqrt{\frac{\ln N}{n_i}}$$



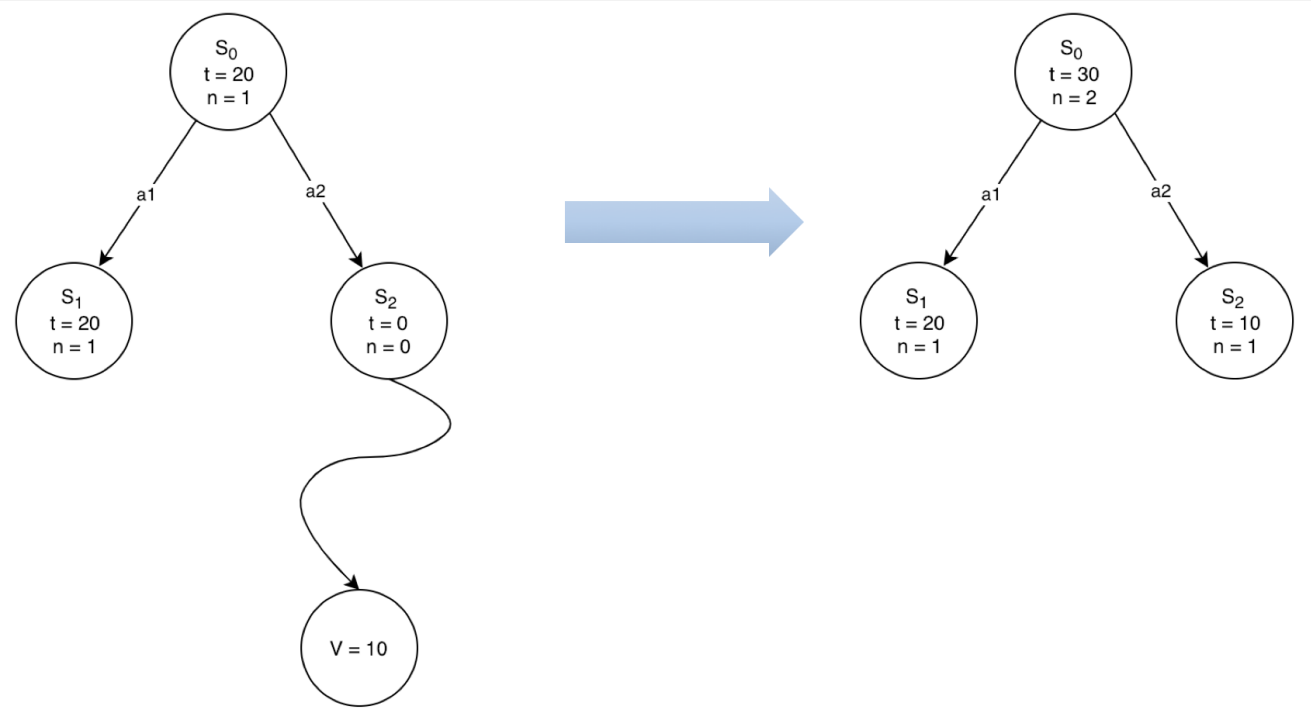
MCTS

- We now return to the root of the tree and enter the selection phase again
- Applying our UCT formula, we find that $UCT(S_1) = \frac{t_1}{n_1} + 2\sqrt{\frac{\ln n_0}{n_1}} = \frac{20}{1} + 2\sqrt{\frac{\ln 1}{1}} = 20$ and that $UCT(S_2) = \infty$
- In the selection phase, we always choose to continue down the tree for the child that maximises the UCT
- Thus, we select S_2



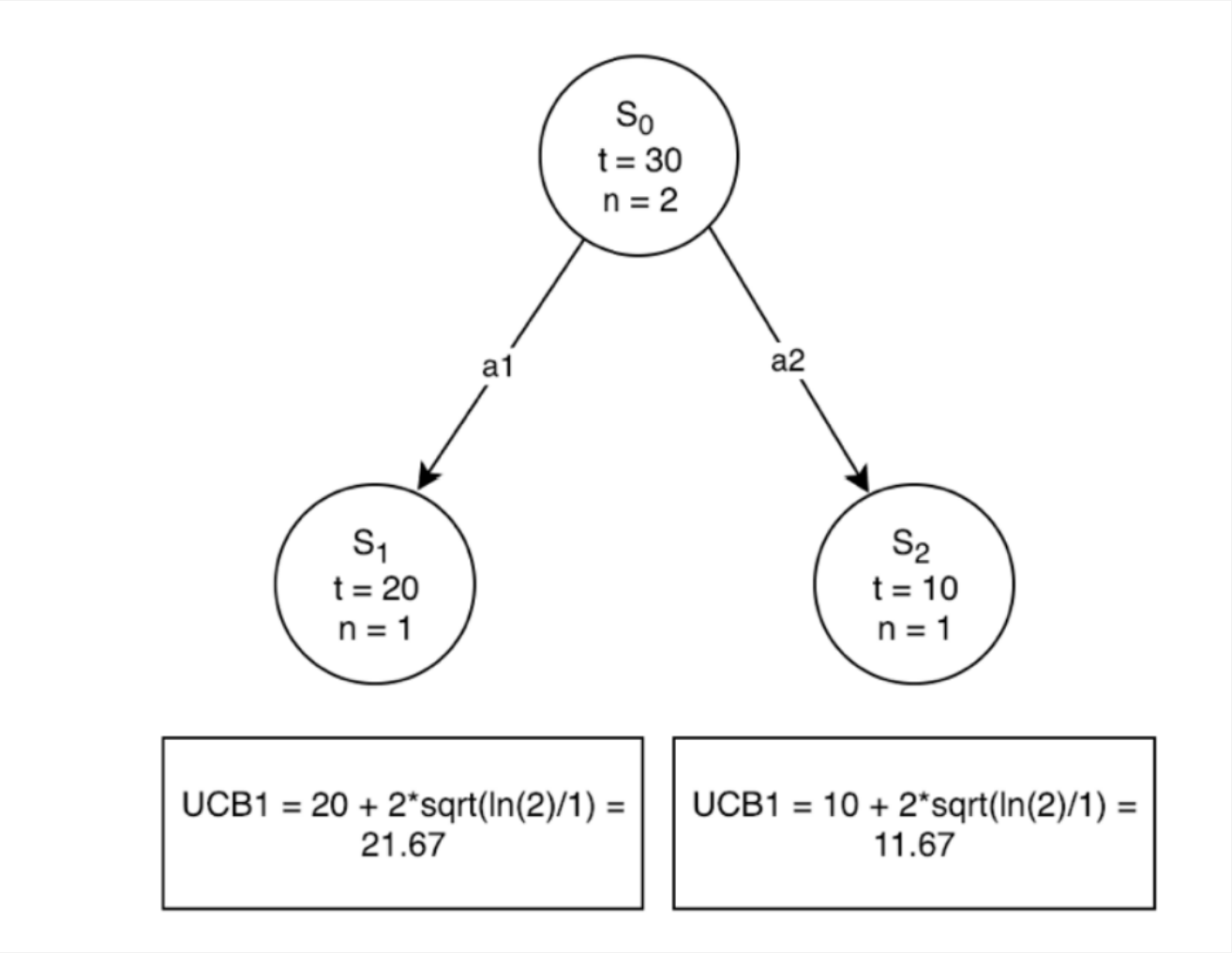
MCTS

- As S_2 is a leaf node that has not yet been visited, we once more commence the **rollout** phase
- This means that, starting at state S_2 we take random actions until we reach a terminal state



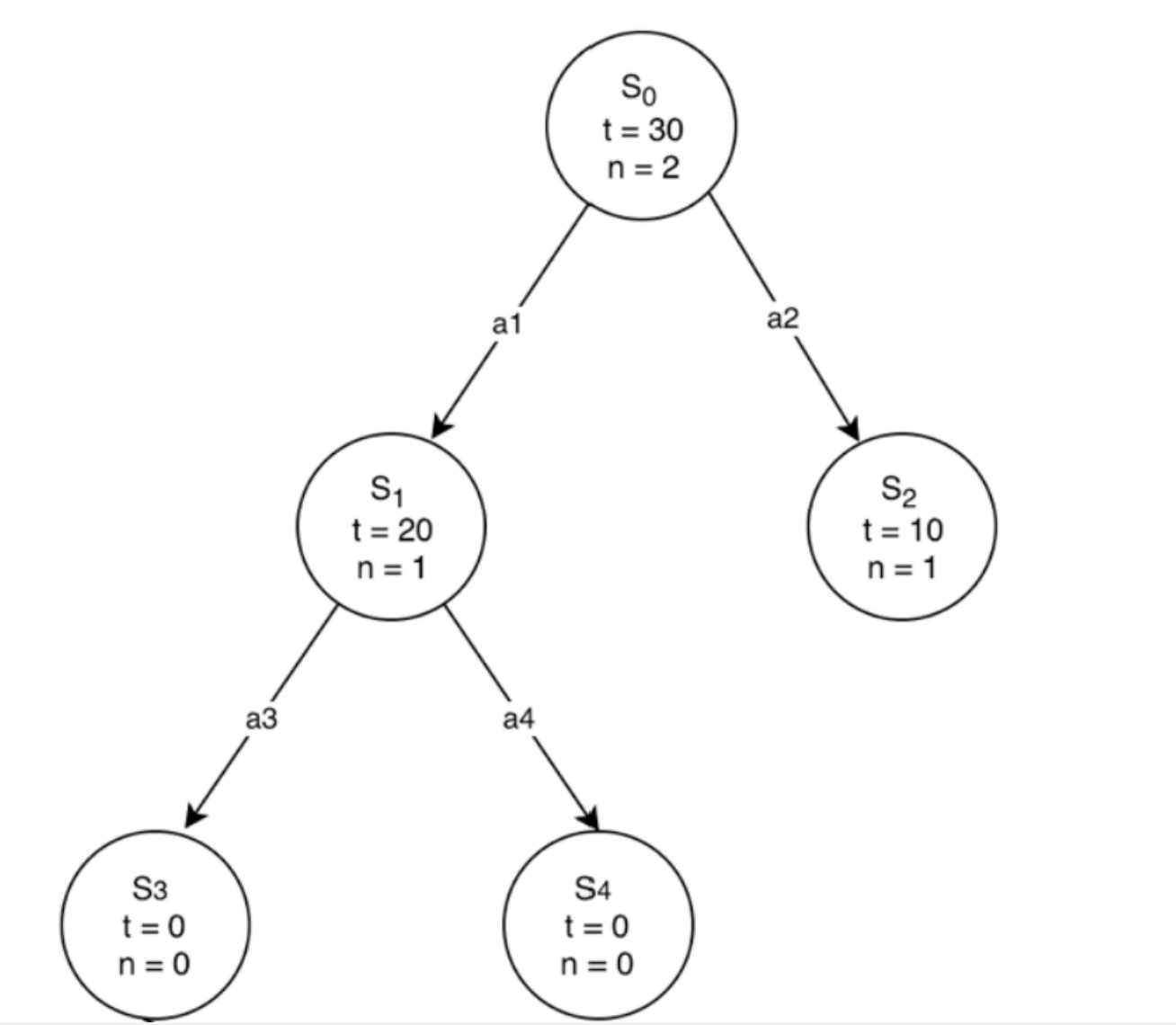
MCTS

- The rollout result is propagated back through the tree, with all states that were in the move sequence that lead to the rollout node being updated
- In this case, the move sequence was $S_0 \rightarrow S_2 \rightarrow Rollout$. Thus, for S_0 and S_2, we say n = n + 1 and that t = t + 10
- This is the end of the second iteration



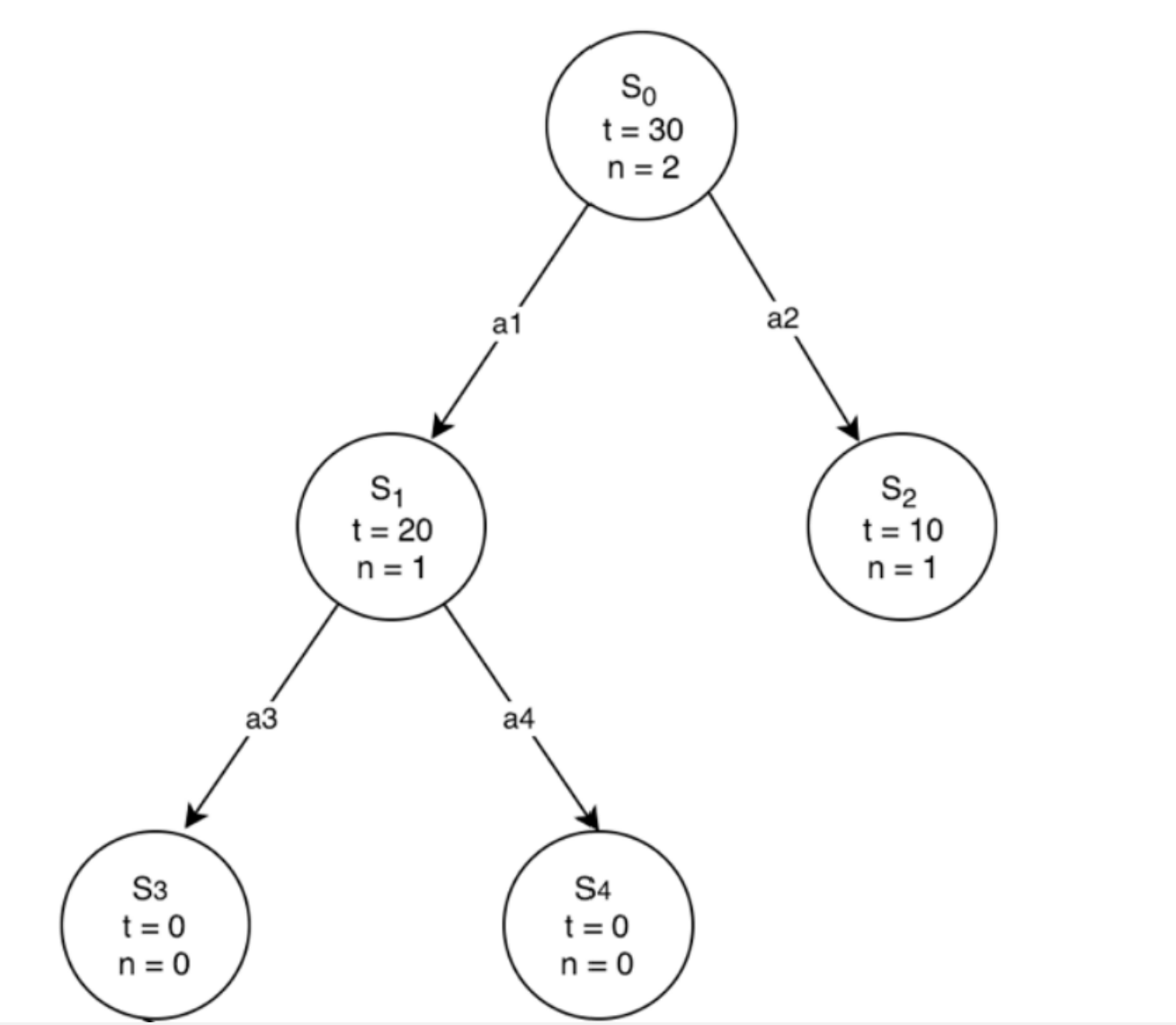
MCTS

- We return to the root node and continue with the selection phase
- We select the child node that maximises the UCT score
- In this case, $UCT(S_1) \cong 21.67$ and $UCT(S_2) \cong 11.67$
- Thus, we select S_1



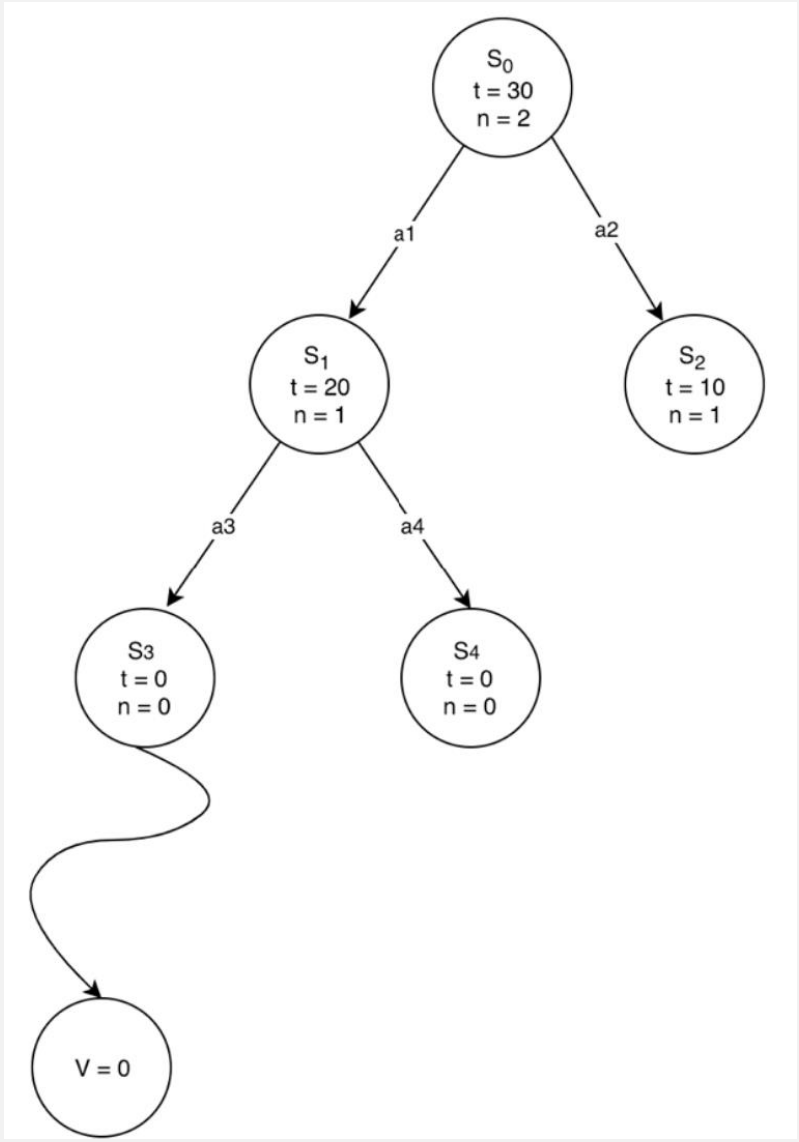
MCTS

- We now reach S_1 which is a leaf node
- As S_1 has already been visited at least once, we commence the **expansion** phase
- We, find that two actions can be taken from S_1, giving us two new states: S_3 and S_4



MCTS

- We now move to selection phase, applying our UCT formula to S_3 and S_4
- We find that they have the same UCT score (namely ∞). Thus, we randomly choose one
- In this case, let's choose S_3
- We now begin the rollout phase for S_3



MCTS

- This is the rollout phase for S_3
- We play random moves from S_3 until we achieve a terminal state

MCTS

- We can perform MCTS for as many iterations (/ amount of time) as we wish
- With more time / iterations, our AI is more likely to pick the best move and find an optimal strategy
- MCTS forms the core of the AlphaGo algorithm.
- Alpha Go is an AI system that beat the world champion of the board game Go.
- Go is similar to chess but hadn't been solved by AI in the same way that chess had until Deep mind produce Alpha Go in 2016
- Like chess, Go is a perfect information game with two players: White and Black, however there are more possible states and heuristic are much harder to develop

ALPHA GO

- Alpha Go used 30 million Go games from human players to train two neural networks that estimate the value of Go boards and also actions to take in a given board state.
- It then uses these networks to optimise the selection and rollout phases of the MCTS. In the rollout phase, the neural network is used to make moves that look good (i.e. not just purely random moves)
- In the selection phase, the neural network is used to select moves that are more promising
- It then plays against itself to continually improve both of these neural networks using reinforcement learning
- This lead to the strongest Go player ever seen up until that point

DEEP MIND

- Deep mind have improved Alpha Go multiple times. They now have a general algorithm that can be applied to any board game (regardless of perfect or imperfect information)
- This new algorithm only uses reinforcement learning (and thus requires no human dependent data)
- This new algorithm, has beaten the best AI systems in chess, Go, shogi and many other games

GENETIC ALGORITHMS

- Genetic Algorithms refer to a set of algorithms that emulate biological processes to solve optimisation problems
- In order to get an overview of this, field we will look at one of the most common ones, titled NEAT

NEAT

- NEAT stands for NeuroEvolution of Augmenting Topologies
- This algorithm is used to alter artificial neural networks in an unsupervised fashion
- The algorithm is **roughly** as follows:

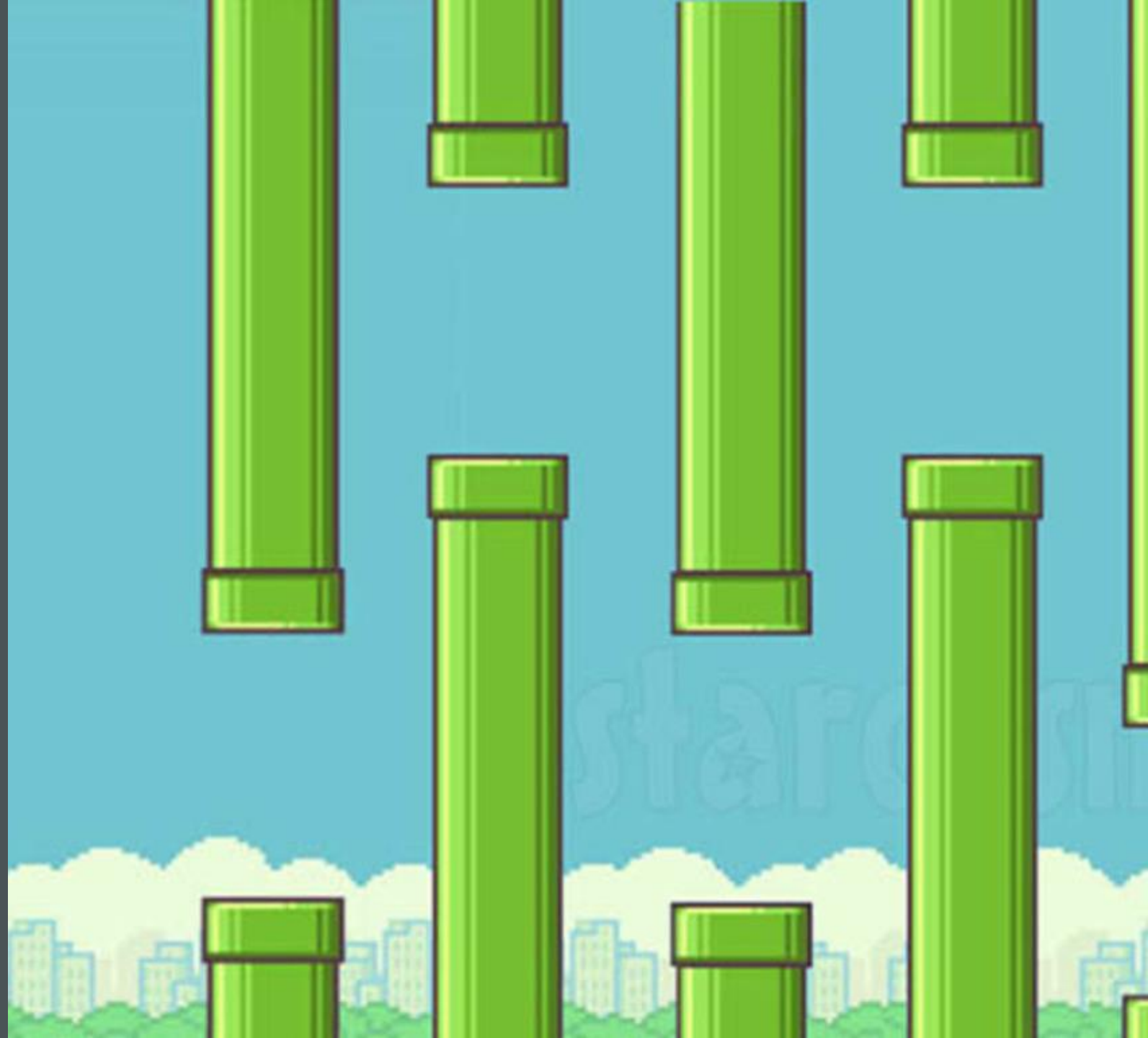
1. Initialise X artificial neural networks with random weights and biases
2. Let each network play the current state
3. Evaluate how well each network did using a fitness function
4. Pick the top n% of networks and splice the weights and biases of these networks together to create X ANN's
5. Repeat until stopping criteria

NEAT

- The idea is roughly mimicking that of natural selection. The best genes lead to increased survival odds, which increases the chances of "genes" being passed on to the next generation
- This is commonly applied in video games
- Let's take the video game "flappy bird" as an example

FLAPPY BIRD

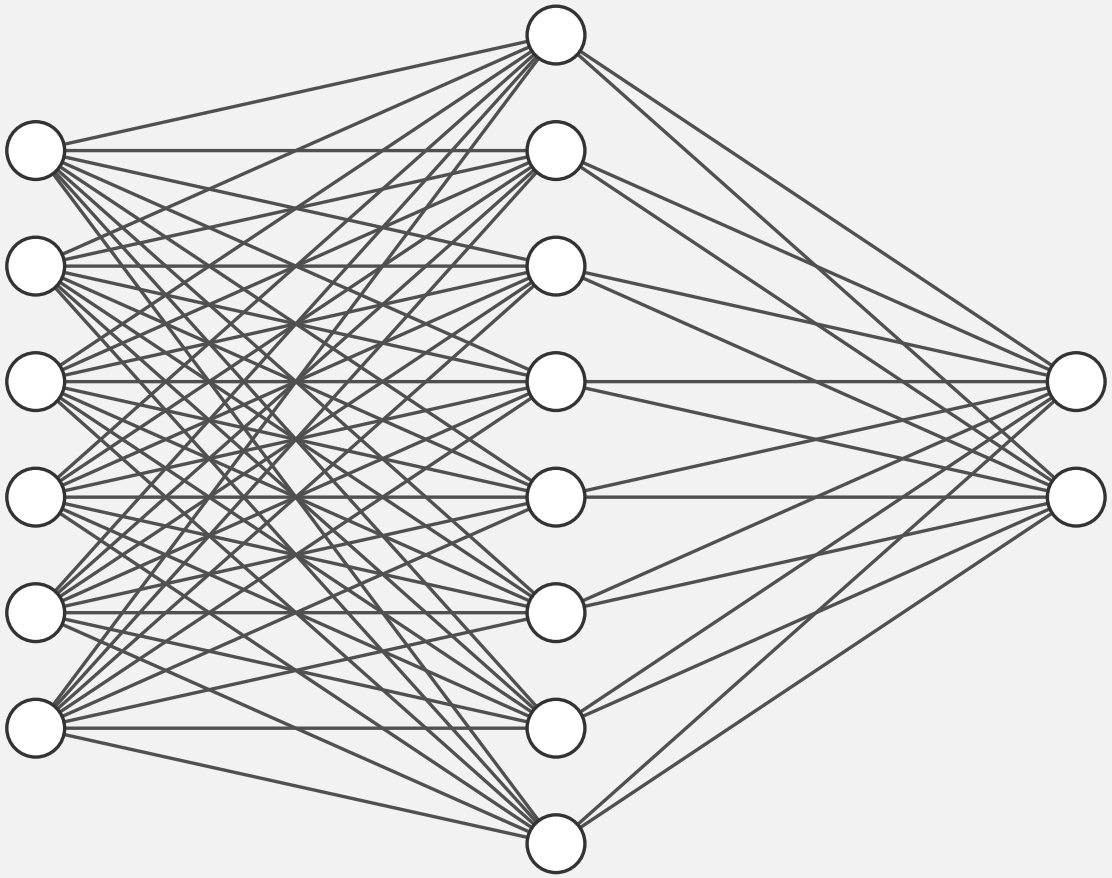
- In the game, you can either tap the screen or not tap the screen
- If you don't tap the screen, the bird will fall down
- If you do tap the screen, the bird will move up (i.e. flap their wings)
- The goal is to avoid collision with the pipes for as long as possible to maximise your score
- The game can be played here: https://flappybird.io/



FLAPPY BIRD

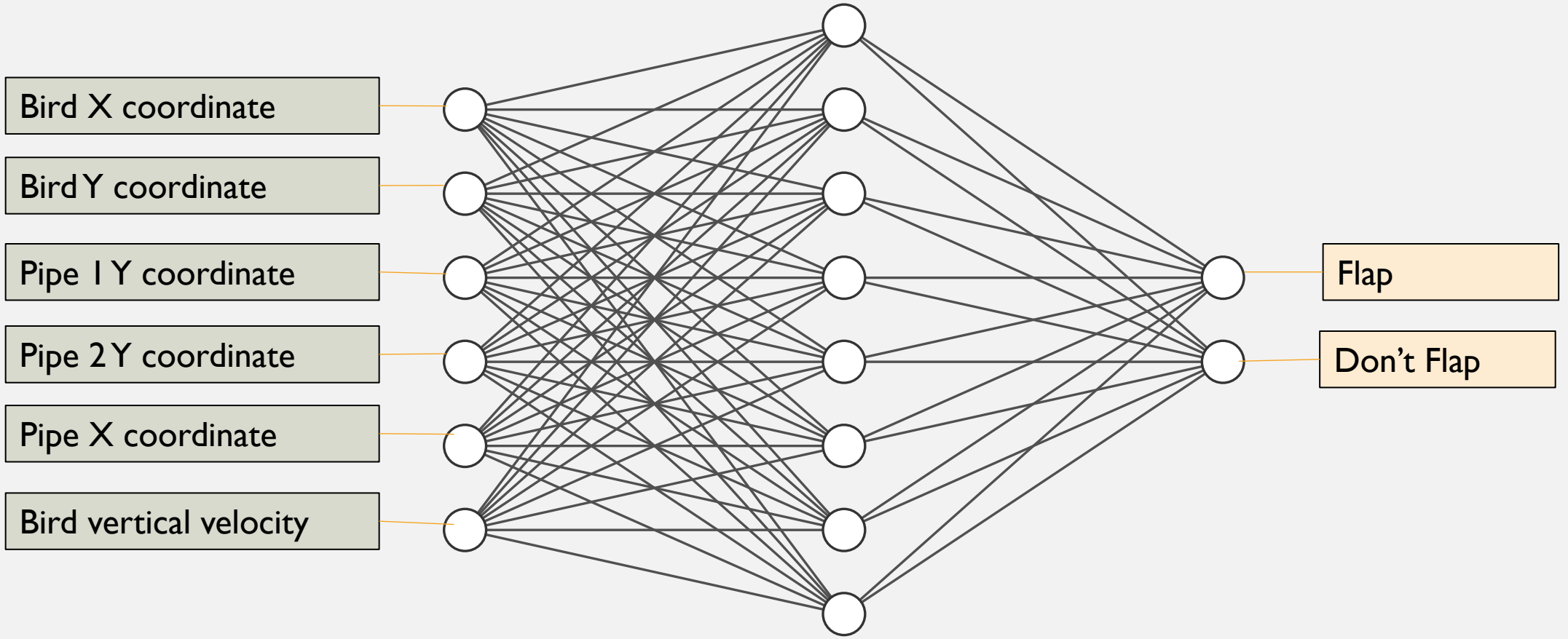
- Let's design a neural network for our bird agent.
- First, let's consider what data we want to have as an input to the network
- We will want to know our location, our vertical velocity, the location of the closest lower pipe and the location of the upper closest pipe.
- For each location, we will use both the x and y coordinate
- As this is a simple task to perform, we will want a small neural network
- This allows real time performance which is critical

FLAPPY BIRD



Input Layer $\in \mathbb{R}^6$ Hidden Layer $\in \mathbb{R}^8$ Output Layer $\in \mathbb{R}^2$

FLAPPY BIRD


Bird X coordinate
Bird Y coordinate
Pipe 1 Y coordinate
Pipe 2 Y coordinate
Pipe X coordinate
Bird vertical velocity
Flap
Don't Flap
Input Layer ∈ ℝ⁶
Hidden Layer ∈ ℝ⁸
Output Layer ∈ ℝ²

FLAPPY BIRD

- Let's have 100 agents per generation
- Initially, each agent will have a randomly initialised set of weights and biases
- For our fitness function, we will use the horizontal distance travelled
- This means that agents that clear more pipes will be more likely to "reproduce" and pass their genes onto future generations
- After each generation, we will take the weights and biases from the top 10 neural networks and splice them together to produce 99 new agents
- Splicing is complicated but is roughly analogous to how two parents pass their genes on to a child (the precise formula can be found here: http://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf)
- The final agent for the new generation will be identical to the best performing agent from the previous generation
- We repeat this for a certain amount of time

GA

- Let's look at our flappy bird AI in action.
- Please download the folder found here: https://github.com/philipmortimer/AI-Course/tree/main/Programs/Part%207/11.3_neuroevolution_tfjs.js
- Simply double click the "index.html" file and open it in your web browser (e.g. chrome or edge).
- You don't need to have pre-installed anything (like python or Java)!

Credit: https://github.com/CodingTrain/website/tree/main/Courses/natureofcode/11.3_neuroevolution_tfjs.js

FLAPPY BIRD

- As you can see, the genetic algorithm is highly effective and learns to play flappy bird very quickly in real time!
- As with all forms of ML (included reinforcement learning), there are many variants of genetic algorithms
- Although, of course this was a very simple problem

ML TECHNIQUES

- It can be difficulty to decide what ML technique is suitable for a given problem
- Traditionally, the following order of effectiveness is true: supervised, unsupervised, reinforcement then genetic algorithms.
- However, this is too generic and the simple answer is that it takes experience to know which type(s) of learning to apply
- Alpha Go, represents a breakthrough as the algorithm exceeds any results obtainable from supervised techniques, for example.

SUMMARY

- We looked at reinforcement learning, a machine learning technique that has some agent interact with an environment to learn
- We discussed q learning, where a q table is generated to assess how good a given move probably is for a given state
- We looked at the application of deep neural networks as a way to replace this table for large state spaces
- We discussed Monte Carlo Tree search, an advanced search algorithm
- MCTS balances taking good actions with evaluating actions that have not been taken often to calculate the best action to take in a given state
- It can be thought of as a more versatile extension of the concepts introduced in by the minimax algorithm

SUMMARY

- We discussed how Alpha Go combined deep neural networks with MCTS to be the first computer to exceed human skill at the incredibly complex board game Go
- Discussed genetic algorithms – algorithms used to solve optimisation problems by emulating biological processes.
- We discussed the NEAT algorithm which creates a set of agents to tackle a problem. It selects the best agents to combine their genes for the next generation
- This approach allowed us to produce a neural network that achieves perfect flappy bird play

QUESTIONS

- Please ask any questions about this lesson **OR** any part of the course
- This is the last lesson, so now is the time to discuss anything you wish to talk about!

FURTHER READING

Q learning: https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

Deep Q Learning: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/#:~:text=In%20deep%20Q%2Dlearning%2C%20we,is%20generated%20as%20the%20output.

MCTS: *https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/*

Alpha Go: https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc085319 https://www.youtube.com/watch?v=WXuK6gekU1Y

Genetic Algorithms: https://www.youtube.com/watch?v=9zfeTw-uFCw&t=107s https://www.youtube.com/watch?v=cdUNkwXx-I4

END OF COURSE

- This is the end of my introduction to AI
- Thanks for watching!
- Hopefully, you should have a high level overview of a range of common AI techniques (even if you don't understand all of the low level maths)
- You now understand neural networks, supervised learning, deep learning, unsupervised learning, reinforcement learning, genetic algorithms, graph traversal (DFS, BFS etc.), minimax and AI design principals
- You now have all the tools you need to tackle an AI problem of your choosing
- Remember, if you get stuck, always use **Google**. This will more often than not resolve your issues

NEXT STEPS

- If you wish to continue pursuing machine learning, I would recommend attempting to code some of the projects I discussed in this course.
- For example:

1. A tic tac toe playing minimax AI using alpha beta pruning. For an extension, implement iterative deepening and move ordering optimisations live PV search
2. A BFS that calculates the degree of separation between actors
3. An ANN using TensorFlow that classifies the MNIST dataset
4. A CNN using TensorFlow for cancer recognition
5. Perform K means clustering on a dataset
6. Implement MCTS on noughts and crosses
7. Attempt your own project

NEXT STEPS

- If you wish to brush up on AI, this is a brilliant resource: https://www.youtube.com/watch?v=LucW-p6zC5c&list=PLBw9d_OueVJS_084gYQexJ38LC2LEhpR4
- Otherwise, simply read up on area of interest, problem online. There's lots of documentation and discussion easily available for free on the web
- This is perhaps the best single place to start your future journey if you wish: https://www.tensorflow.org/resources/learn-ml
- I would be open to doing custom work / future teaching / whatever you wish if that's something that interests you
- You can contact me on Fiverr at https://www.fiverr.com/philiplearning
- Alternatively, feel free to email me at "philipmortimer99@aol.com"

THANK YOU

- Thank you so much for taking this course!
- All the best with the future!!