- ➔ Our group is going to talk about the Monte Carlo Tree Search.
- ➔ First, I am going to talk about the overview of the Monte Carlo Tree Search and basic algorithm of the implementation, and then
- ➔ Subaru is going to talk about some variations, and
- ➔ Yuki is going to talk about the enhancement on algorithm, and then
- ➔ Finally pass it to Justin, and he is going to talk about some applications of the Monte Carlo Tree Search.

Overview of Monte Carlo Methods

- ➔ First of all, Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- ➔ The underlying concept is to use randomness to solve problems that might be deterministic in principle.
- ➔ They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches.
- ➔ The common example of the Monte Carlo Methods is determining the value of π. If you look at this diagram, the area of the square is 4.
- ➔ and the area of the circle is π.
- ➔ In order to determine the value of π using Monte Carlo Methods, we randomly plot the dots inside the square.
- ➔ In this diagram, there are 15 dots inside the square and 11 dots inside the circle.
- ➔ So the ratio of this can be expressed as 4 to π equals 15 to 11.
- ➔ Baced on this expression, the π values is approximately 2.93.
- ➔ In this example, we only plotted 15 dots, but when the number of dots increases, the π values will be more accurate.

Basic Monte Carlo Tree Search Algorithm

- ➔ When this method is applied to search Tree, each node represents the state and the edge represents the possible actions.
- ➔ A tree for the MCTs is built in an incremental and asymmetric manner.
- ➔ For each iteration of the algorithm, a tree policy is used to find the most urgent node of the current tree.

- ➔ The tree policy attempts to balance considerations of exploration which you look in areas that have not been well sampled yet and exploitation, which you look in areas that is appear to be promising.
- ➔ A simulation is then run from the selected node and the search tree updated according to the result.
- ➔ This involves the addtion of a child node corresponding to the action taken from the selected node, and an update of the information of its ancestors.
- ➔ Moves are made during this simulation according to some default policy. The simplest case is to make uniform random moves.
- ➔ A great benefit of MCTs is that the values of intermediate states do not have to be evaluated, as for depth-limited minimax search, which greatly reduces the amount of domain knowledge required.
- ➔ Only the value of the terminal state at the end of each simulation is required. Lastly, when the simulation reaches the terminal states,
- ➔ The result is backpropagated, which means it is “backed up” through the selected nodes to update their information.

Now I’m going to pass it to Subaru to explain the variations

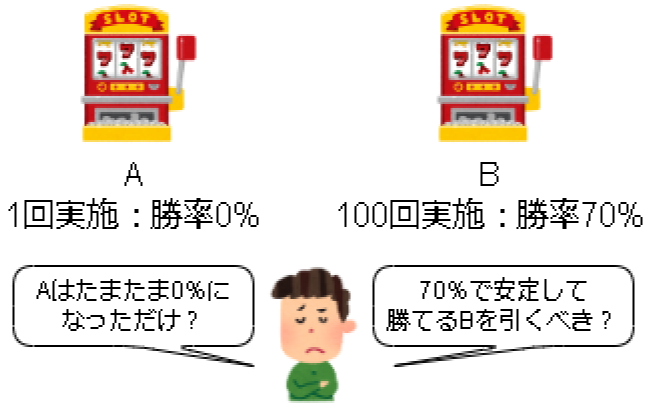
Monte Carlo Tree Search Methods

1. Introduction

Overview of Monte Carlo Tree Search
- A tree is built in an incremental and asymmetric manner. → Tree が左右非対称に構築される
- a *tree policy* is used to find the most urgent node of the current tree.
- The tree policy attempts to **balance** considerations of 知識の獲得 exploration:探索 (look in areas that have not been well sampled yet) and 知識の利用 exploitation:活用 (look in areas which appear to be promising(有望))

勝率の精度を上げる → 試行回数あげる → 負ける手も多く指してしまう(random)実際の勝率は下がる

A: たまたま0%になったのか本当に0%になったのかわからない
B:



- A random or statistically biased sequence of actions applied to the given state until a terminal condition is reached.
- A simulation is then run from the selected node and the search tree is updated according to the result.
- This involves the addition of a child node corresponding to the action taken from the selected node
- default policy, which in the simplest case is to make uniform random moves.
- Only the value of the terminal state at the end of each simulation is required.

Importance
- MCTS has gained interest due to its success in computer Go, which is one of the few games where human players are far ahead of computers.
- MCTS has also achieved great success in many specific games, general games, and complex real-world problems, making it an important tool for AI researchers with selective sampling approaches that can potentially provide insights into hybridizing (交配) and improving other algorithms.

Aim
- a comprehensive survey of known MCTS research
- underlying mathematics behind MCTS, the algorithm itself, its variations and enhancements, and its performance in a variety of domains

2. Background (Decision theory → Game theory → Monte Carlo → Bandit-based methods)

Decision theory
- Decision theory combines probability theory with utility theory to provide a formal and complete framework for decisions made under uncertainty

Markov Decision Processes (MDPs):

S: A set of states, with s0 being the initial state.
A: A set of actions.
T(s, a, s0): A transition model that determines the probability of reaching state s 0 if action a is applied to state s.
R(s): A reward function

- Overall decisions are modelled as sequences of (state, action) pairs
- next state s' 0 is decided by a probability distribution which depends on the current state s and the chosen action a

A *policy* is a mapping from states → actions, specifying which action will be chosen from each state in S. The aim is to find the policy π that yields the **highest expected reward**.

Partially Observable Markov Decision Processes (POMDP):
O(s, o): An observation model that specifies the probability of perceiving observation o in state s.
- all cases the optimal policy π is deterministic, in that each state is mapped to a single action rather than a probability distribution over actions

Game Theory extends decision theory to situations in which multiple agents interact.
S: The set of states, where s0 is the initial state.
ST ⊆ S: The set of terminal states.
n ∈ N: The number of players. • A: The set of actions.
f : S × A → S: The state transition function.
R : S → R k : The utility function.
ρ : S → (0, 1, . . . , n): Player about to act in each state.
t = 1, 2, . . . until some terminal state is reached

- Each player receives a reward. These values may be arbitrary (e.g. positive values for numbers of points accumulated or monetary gains, negative values for costs incurred) but in many games it is typical to assign nonterminal states a reward of 0 and terminal states a value of +1, 0 or −1 (or +1, + 1 2 and 0) for a win, draw or loss, respectively. These are the *game-theoretic* values of a terminal state.
- Each player's strategy (policy) determines the probability of selecting action a given state s。

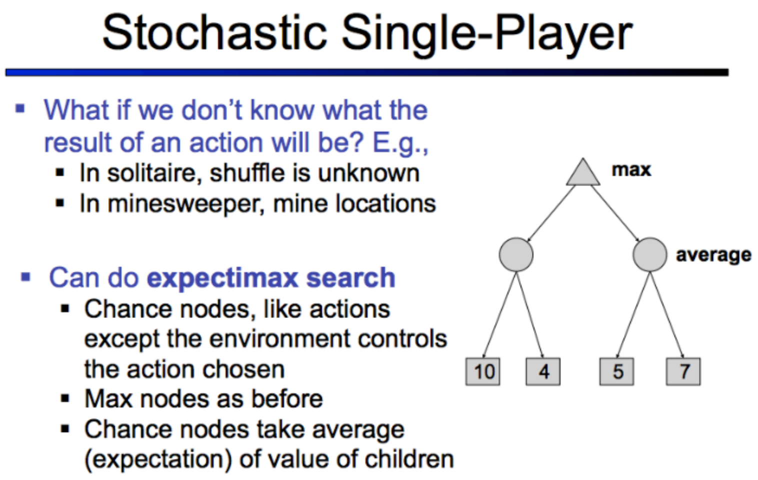
Combinatorial Games:
• Zero-sum: Whether the reward to all players sums to zero (in the two-player case, whether players are in strict competition with each other).
• Information: Whether the state of the game is fully or partially observable to the players.
• Determinism: Whether chance factors play a part (also known as completeness, i.e. uncertainty over rewards).
• Sequential: Whether actions are applied sequentially or simultaneously.
• Discrete: Whether actions are discrete or applied in real-time.

- Games with TWO players that are zero-sum, perfect information, deterministic, discrete and sequential are described as *combinatorial games*.

- Combinatorial games make excellent test beds for AI experiments as they are controlled environments defined by simple rules → exhibit deep and complex play that can present significant research challenges

AI in Real Games:
- Involve a delayed reward structure in which only those rewards achieved in the terminal states accurately describe how well each player is performing
Games are therefore typically modelled as trees of decisions as follows:

• Minimax attempts to minimise the opponent's maximum reward at each state, and is the traditional search approach for two-player combinatorial games. the α-β heuristic is typically used to prune the tree.
• Expectimax generalises minimax to stochastic (確率的) games in which the transitions from state to state are probabilistic. The value of a chance node is the average of its child nodes (expected values = sum(x*P(x))).


Stochastic Single-Player

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children

max
average
10
4
5
7

• Miximax is similar to single-player expectimax and is used primarily in games of imperfect information.

Monte Carlo Methods

Qnew (s) = Num. of Wins/Num. of Plays

flat Monte Carlo: Monte Carlo approaches in which the actions of a given state are uniformly sampled → it is simple to construct degenerate cases in which flat Monte Carlo fails, as it does not allow for an opponent model → improve the reliability of gametheoretic estimates by biasing action selection based on past experience. move selection towards those moves that have a higher intermediate reward.

Bandit-Based Methods
- one needs to choose amongst K actions (e.g. the K arms of a multi-armed bandit slot machine) in order to maximise the cumulative reward by consistently taking the optimal action.
- This leads to the exploitation-exploration dilemma

Regret:
1ステップあたりの機会損失 (そのステップで取得可能と予測される報酬の中で最大の報酬と実際に取得した報酬の差) → 累積後悔の最小化 = 累積報酬の最大化
The policy should aim to minimise the player's regret, which is defined after n plays as:

$$R_N = \mu^\star n - \mu_j \sum_{j=1}^{K} \mathbb{E}[T_j(n)]$$

μ* = best possible expected reward
E[Tj(n)] = the expected number of plays for arm j in the first n trials.
→ regret is the expected loss due to not playing the best bandit

- Bandits問題の難しさは、最適なアクションとその他のアクションの価値の差の大きさによって決まる。
差が大きければ簡単に学習ができ、差が小さければ学習は難しくなる。
- Optimism in the Face of Uncertaintyの主要な考え方は、現在分かっている最適なアクションではなく、最適となる可能性があり(optimism)、価値についての不確かさ(uncertainty)が大きいアクションを次に選択するということである。
- to ensure that the optimal arm is not missed due to temporarily promising rewards from a sub-optimal arm
- It is thus important to place an upper confidence bound on the rewards observed so far that ensures this.
- made use of upper confidence indice (difficult to compute)

Upper Confidence Bounds (UCB):

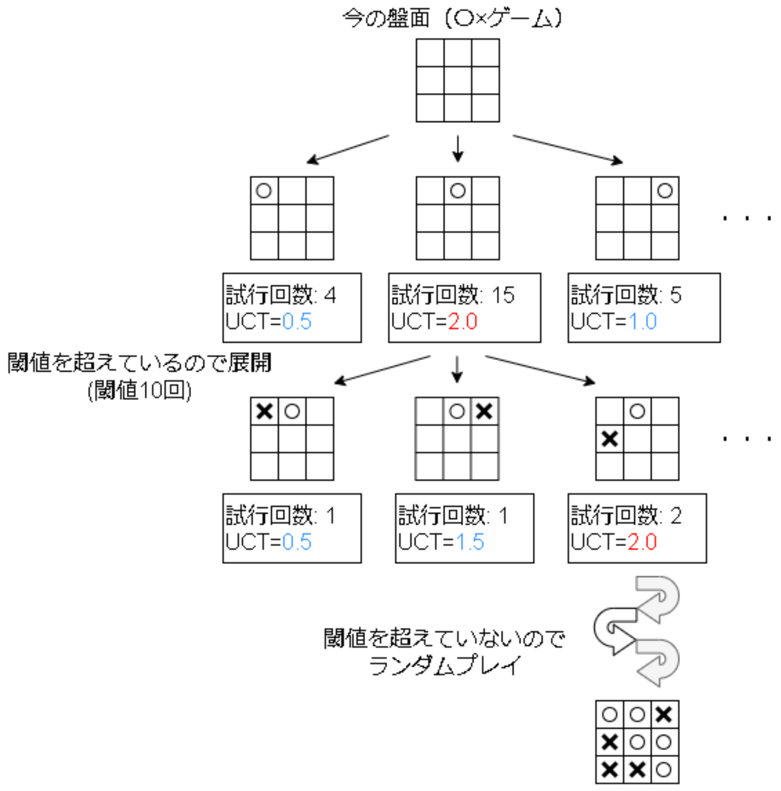
$$\text{UCB1} = \overline{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

The reward term Xj encourages the exploitation of higher-reward choices,
while the right-hand term q2 ln n nj encourages the exploration of less visited choices

3. Monte Carlo Tree Search

Algorithm

- involves iteratively building a search tree until some predefined computational budget () – typically a time, memory or iteration constraint – is reached,



1) Selection: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unvisited (i.e. unexpanded) children.
2) Expansion: One (or more) child nodes are added to expand the tree, according to the available actions.
3) Simulation: A simulation is run from the new node(s) according to the default policy to produce an outcome.
4) The simulation result is “backed up” (i.e. backpropagated) through the selected nodes to update their statistics.

1) Tree Policy: Select or create a leaf node from the nodes already contained within the search tree (selection and expansion).
2) 2) Default Policy: Play out the domain from a given non-terminal state to produce a value estimate (simulation).

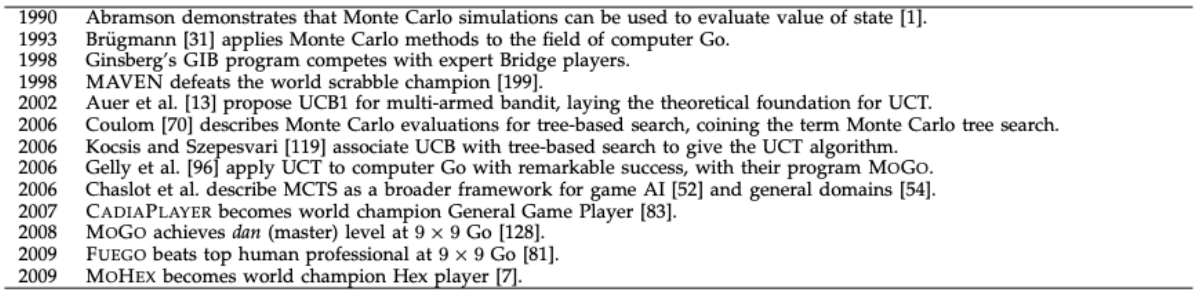
Starting at the root node t0, child nodes are recursively selected according to some utility function until a node tn is reached that either describes a terminal state or is not fully expanded (note that this is not necessarily a leaf node of the tree). An unvisited action a from this state s is selected and a new leaf node tl is added to the tree, which describes the state s 0 reached from applying action a to state s. This completes the tree policy component for this iteration. A simulation is then run from the newly expanded leaf node tl to produce a reward value ∆, which is then backpropagated up the sequence of nodes selected for this iteration to update the node statistics; each node's visit count is incremented and its average reward or Q value updated according to ∆. As soon as the search is interrupted or the computation budget is reached, the search terminates and an action a of the root node t0 is selected by some mechanism.

Four criteria for selecting the winning action, based on the work of Chaslot et al

1) Max child: Select the root child with the highest reward.
2) 2) Robust child: Select the most visited root child.
3) 3) Max-Robust child: Select the root child with both the highest visit count and the highest reward. If none exist, then continue searching until an acceptable visit count is achieved.
4) 4) Secure child: Select the child which maximises a lower confidence bound.

Development

Monte Carlo methods used in games with randomness and partial observability → can be used in deterministic games of perfect information



1990	Abramson demonstrates that Monte Carlo simulations can be used to evaluate value of state [1].
1993	Brügmann [31] applies Monte Carlo methods to the field of computer Go.
1998	Ginsberg's GIB program competes with expert Bridge players.
1998	MAVEN defeats the world scrabble champion [199].
2002	Auer et al. [13] propose UCB1 for multi-armed bandit, laying the theoretical foundation for UCT.
2006	Coulom [70] describes Monte Carlo evaluations for tree-based search, coining the term Monte Carlo tree search.
2006	Kocsis and Szepesvari [119] associate UCB with tree-based search to give the UCT algorithm.
2006	Gelly et al. [96] apply UCT to computer Go with remarkable success, with their program MoGo.
2006	Chaslot et al. describe MCTS as a broader framework for game AI [52] and general domains [54].
2007	CadiaPlayer becomes world champion General Game Player [83].
2008	MoGo achieves *dan* (master) level at 9×9 Go [128].
2009	Fuego beats top human professional at 9×9 Go [81].
2009	MoHex becomes world champion Hex player [7].

TABLE 1
Timeline of events leading to the widespread popularity of MCTS.

Upper Confidence Bounds for Trees (UCT)

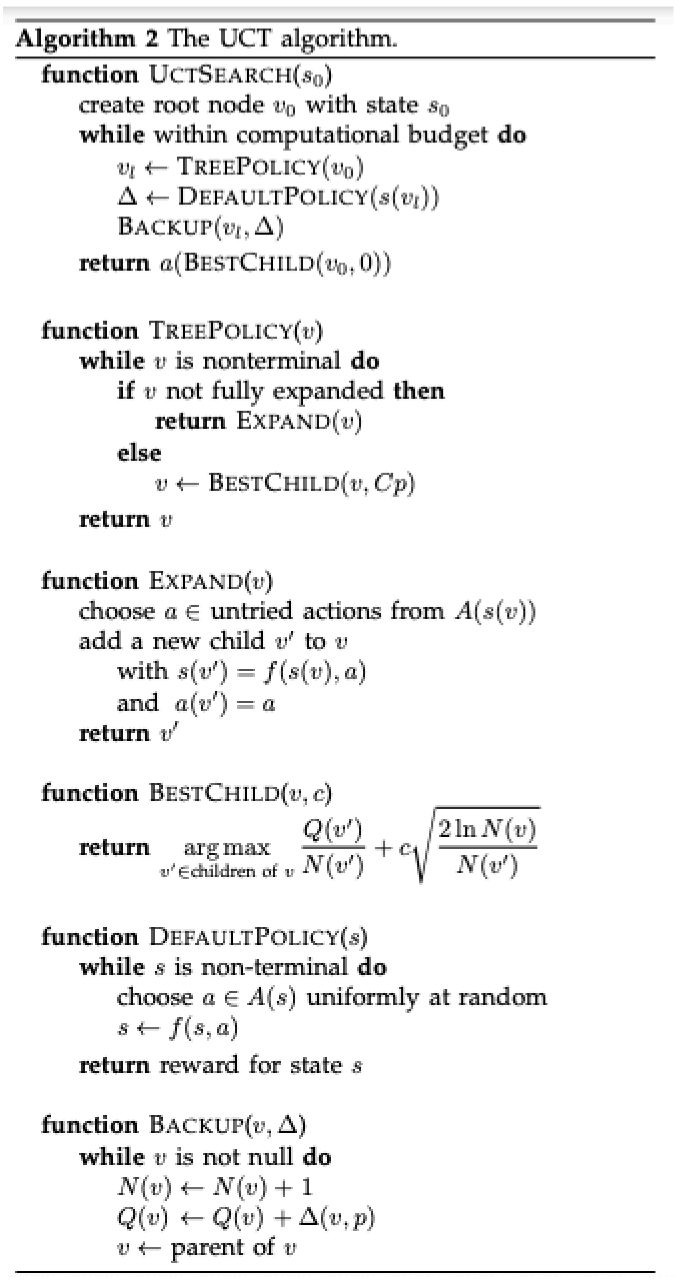
The UCT algorithm:
How the tree is built depends on how nodes in the tree are selected. The success of MCTS, especially in Go, is primarily due to this tree policy
- UCB1 has some promising properties: it is very simple and efficient and guaranteed to be within a constant factor of the best possible bound on the growth of regret.

The exploration term ensures that each child has a nonzero probability of selection,
The value Cp = 1/ √ 2 was shown by Kocsis and Szepesvari [120] to satisfy the Hoeffding ineqality with rewards in the range [0, 1].

if the node selected by UCB descent has children that are not yet part of the tree, one of those is chosen randomly and added to the tree. The default policy is then used until a terminal state has been reached.

return child with the highest reward or action that leads to the most visited child. This potential discrepancy is addressed in the Go program ERICA by continuing the search if the most visited root action is not also the one with the highest reward. This improved ERICA's winning rate against GNU GO from 47% to 55%.

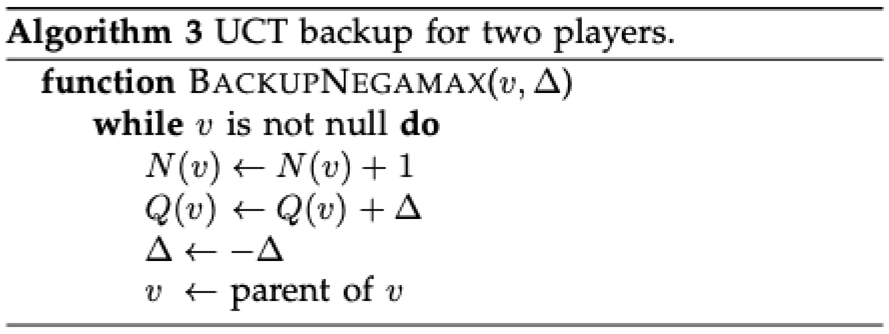
Algorithm 2 shows the UCT algorithm in pseudocode.



Algorithm 2 The UCT algorithm.

```
function UCTSEARCH($s_0$)
    create root node $v_0$ with state $s_0$
    while within computational budget do
        $v_l \leftarrow$ TREEPOLICY($v_0$)
        $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
        BACKUP($v_l, \Delta$)
    return $a$(BESTCHILD($v_0, 0$))

function TREEPOLICY($v$)
    while $v$ is nonterminal do
        if $v$ not fully expanded then
            return EXPAND($v$)
        else
            $v \leftarrow$ BESTCHILD($v, Cp$)
    return $v$

function EXPAND($v$)
    choose $a \in$ untried actions from $A(s(v))$
    add a new child $v'$ to $v$
        with $s(v') = f(s(v), a)$
        and $a(v') = a$
    return $v'$

function BESTCHILD($v, c$)
    return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function DEFAULTPOLICY($s$)
    while $s$ is non-terminal do
        choose $a \in A(s)$ uniformly at random
        $s \leftarrow f(s, a)$
    return reward for state $s$

function BACKUP($v, \Delta$)
    while $v$ is not null do
        $N(v) \leftarrow N(v) + 1$
        $Q(v) \leftarrow Q(v) + \Delta(v, p)$
        $v \leftarrow$ parent of $v$
```

Algorithm 3 shows an alternative and more efficient backup method for two-player, zero-sum games with alternating moves.



Algorithm 3 UCT backup for two players.

```
function BACKUPNEGAMAX($v, \Delta$)
    while $v$ is not null do
        $N(v) \leftarrow N(v) + 1$
        $Q(v) \leftarrow Q(v) + \Delta$
        $\Delta \leftarrow -\Delta$
        $v \leftarrow$ parent of $v$
```

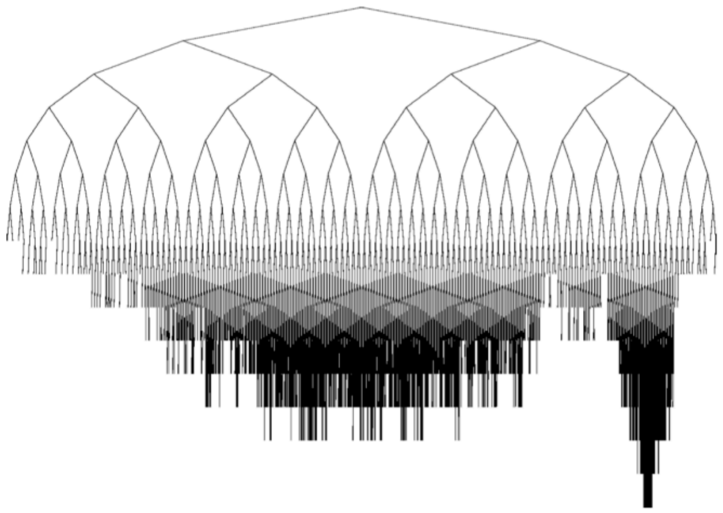
Convergence to Minimax:

Characteristics
characteristics that make MCTS a popular choice of algorithm

Aheuristic:
- lack of need for domain-specific knowledge → readily applicable to any domain that may be modelled using a tree.
- Although full-depth minimax is optimal, quality of play for depthlimited minimax depends significantly on the heuristic used to evaluate leaf nodes
- useful heuristics are much more difficult to formulate, the performance of minimax degrades significantly.
- Although MCTS can be applied in its absence, significant improvements in performance may often be achieved using domain-specific knowledge.
- There are trade-offs to consider when biasing move selection using domain-specific knowledge: one of the advantages of uniform random move selection is speed

Anytime:
- MCTS backpropagates the outcome of each game immediately (the tree is built using playouts as opposed to stages [119]) which ensures all values are always upto-date following every iteration of the algorithm → allows the algorithm to return an action from the root at any moment in time → allowing the algorithm to run for additional iterations often improves the result
- It is possible to approximate an anytime version of minimax using iterative deepening
- However, the granularity of progress is much coarser as an entire ply is added to the tree on each iteration.

Asymmetric:
- The tree selection allows the algorithm to favour more promising nodes → leading to an asymmetric tree over time.
→ the building of the partial tree is skewed towards more promising and thus more important regions



- The tree shape that emerges can even be used to gain a better understanding about the game itself. (demonstrates that shape analysis applied to trees generated during UCT search can be used to distinguish between playable and unplayable games.)

Comparison with Other Algorithms
- If the game tree is of nontrivial size and no reliable heuristic exists for the game of interest, minimax is unsuitable but MCTS is applicable.
- If domain-specific knowledge is readily available, on the other hand, both algorithms may be viable approaches.

- MCTS approaches to games such as Chess are not as successful as for games such as Go.
→ synthetic spaces in which UCT significantly outperforms minimax, the model produces bounded trees where there is exactly one optimal action per state. sub-optimal choices are penalised with a fixed additive cost.

- systematic construction of the tree → ensures that the true minimax values are known.12 In this domain, UCT clearly outperforms minimax and the gap in performance increases with tree depth

- UCT performs poorly in domains with many trap states (states that lead to losses within a small number of moves)
- iterative deepening minimax performs relatively well.
- Trap states → common in Chess but uncommon in Go, which may go some way towards explaining the algorithms' relative performance in those games.