Games and the Minimax algorithm

April 1, 2021

Games and the Minimax algorithm

- We will discuss some aspects of Game Theory

Games and the Minimax algorithm

- We will discuss some aspects of Game Theory
- In particular we will discuss the Minimax Algorithm

Games and the Minimax algorithm

- We will discuss some aspects of Game Theory
- In particular we will discuss the Minimax Algorithm
- The concepts will be applied to simple examples.

Overview

Formalization of games

- Games can be formalized mathematiclally

Zero sum normal form games

- We will consider games with two players.

Zero sum normal form games

- We will consider games with two players.
- Both players play simultaneously. Player A does action a, players B does action b.

Zero sum normal form games

- We will consider games with two players.
- Both players play simultaneously. Player A does action a, players B does action b.
- Their action results in a gain $g(a, b)$.

Zero sum normal form games

- We will consider games with two players.
- Both players play simultaneously. Player A does action a, players B does action b.
- Their action results in a gain $g(a, b)$.
- We assume that what A wins is what B looses : hence the term **zero-sum** game.
- For instance A wins $g(a, b)$ and B "wins" $-g(a, b)$.

Examples

- Paper, Scissors, Stone
- football penalty

Normal form description

- These games can be represented by a **payoff matrix**.

Zero sum normal form games

		Player B action	
		a	b
Player A action	a	$g(a, b)$	$g(a, b)$
	b	$g(a, b)$	$g(a, b)$

Table: In this game, the players can perform two possible actions. Since the game is zero-sum, it is sufficient to represent the gain of player A.

Example

		Player B action	
		a	b
Player A action	a	2	3
	b	1	8

Table: Example gains.

Concept of *supinf*

		Player B		
		a	b	c
	a	2	0	9
Player A	b	4	4	7
	c	10	1	3

Table: What is the gain the A can be **sure** of obtaining ?

Concept of *supinf*

		Player B		
		a	b	c
	a	2	0	9
Player A	b	4	4	7
	c	10	1	3

Table: What is the gain the A can be sure of obtaining ? Reminder : B wants to **minimize the gain** and acts rationally.

Pure strategy and mixed strategy

- A **pure strategy** is completely deterministic
- A **mixed strategy** assigns a probability distirbution to the set of actions.

Rock Scissors Paper

- We will study mixed strategies in the Rock Scissors Paper game.

Rock Scissors Paper

- How many action tuples are possible ?

Rock Scissors Paper

Exercice 1 : Action probabilities

- **cd minimax_and_games/zero_sum** folder.
- Modify the file **paper_scissors_rock.py** so that the actions performed by the two players are drawn from the relevant distributions (= strategies) **player_X_strategy**.
- Use can use the function **choice** from **numpy** (look for its documentation)

Rock Scissors Paper

Exercice 1 : **Action probabilities**

- Modify the file so that the correct statistics about the victory rate are computed.

Rock Scissors Paper

Exercice 2 : **Action probabilities**

- What happens if you change the strategy of player B ?

Rock Scissors Paper

Exercice 2 : Action probabilities

- What happens if you change the strategy of player B ?
- And the stategy of player A ?
- How can we interpret that result ?

Rock Scissors Paper

Let us define probabilistic **events :**

- V : A wins the game
- Br : B plays rock.
- Bs : B plays scissors.
- Bp : B plays paper.

Rock Scissors Paper

- V : A wins the game
- Br : B plays rock.
- Bs : B plays scissors.
- Bp : B plays paper.

$$\begin{aligned} P(V) &= P(V \cap (Br \cup Bs \cup Bp)) \\ &= P((V \cap Br) \cup (V \cap Bs) \cup (V \cap Bp)) \end{aligned} \tag{1}$$

Symbols :

- P means "probability of".
- $\cup$ means "or"
- $\cap$ means "and"

Rock Scissors Paper

$$
\begin{aligned}
P(V) &= P(V \cap (Br \cup Bs \cup Bp)) \\
&= P((V \cap Br) \cup (V \cap Bs) \cup (V \cap Bp)) \\
&= P(V \cap Br) + P(V \cap Bs) + P(\cap Bp) \text{ (Incompatibility of events)}
\end{aligned} \tag{2}
$$

Rock Scissors Paper : conditional probabilities

$$
\begin{aligned}
P(V) &= P(V \cap (Br \cup Bs \cup Bp)) \\
&= P((V \cap Br) \cup (V \cap Bs) \cup (V \cap Bp)) \\
&= P(V \cap Br) + P(V \cap Bs) + P(V \cap Bp) \\
&= P(V|Br)P(Br) + P(V|Bs)P(Bs) + P(V|Bp)P(Bp)
\end{aligned} \tag{3}
$$

$P(V|Br)$ means "Probability that A wins, given that B plays rock".

Rock Scissors Paper : conditional probabilities

$P(V|Br)$ means "Probability that A wins, given that B plays rock".
If the strategy of A is $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, then

- $P(V|Br) = ?$
- $P(V|Bs) = ?$
- $P(V|Bp) = ?$

Rock Scissors Paper : conditional probabilities

$P(V|Br)$ means "Probability that A wins, given that B plays rock".
If the strategy of A is $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, then

- $P(V|Br) = \frac{1}{3}$
- $P(V|Bs) = \frac{1}{3}$
- $P(V|Bp) = \frac{1}{3}$

Rock Scissors Paper : conditional probabilities

$P(V|Br)$ means "Probability that A wins, given that B plays rock".
If the strategy of A is $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, then
$P(V|Br) = P(V|Bs) = P(V|Bp) = \frac{1}{3}$.
Hence :

$$\begin{aligned} P(V) &= P(V|Br)P(Br) + P(V|Bs)P(Bs) + P(V|Bp)P(Bp) \\ &= \frac{1}{3}P(Br) + \frac{1}{3}P(Bs) + \frac{1}{3}P(Bp) \\ &= \frac{1}{3}(P(Br) + P(Bs) + P(Bp)) \end{aligned} \tag{4}$$

Rock Scissors Paper : conditional probabilities

$P(V|Br)$ means "Probability that A wins, given that B plays rock".
If the strategy of A is $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, then
$P(V|Br) = P(V|Bs) = P(V|Bp) = \frac{1}{3}$.
Hence :

$$\begin{aligned} P(V) &= P(V|Br)P(Br) + P(V|Bs)P(Bs) + P(V|Bp)P(Bp) \\ &= \frac{1}{3}P(Br) + \frac{1}{3}P(Bs) + \frac{1}{3}P(Bp) \\ &= \frac{1}{3}(P(Br) + P(Bs) + P(Bp)) \\ &= \frac{1}{3} \end{aligned} \tag{5}$$

Biased game

Exercice 3 : **Alternative game with the well.**

- Modify the file **paper_scissors_rock_well.py** so that the actions performed by the two players are drawn from the relevant distributions (= strategies) **player_X_strategy**, and so that the statistics are correctly computed.
- find a strategy that gives a better victory rate for A.

Statistics on strategies

Exercice 4 : **Learning a strategy**

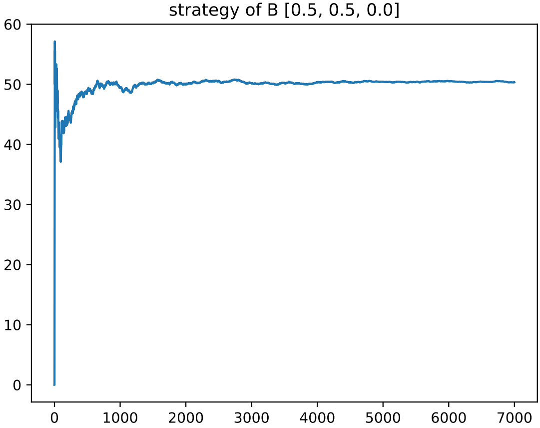
- Modify the file **paper_rock_scissors_learn.py** in order to **learn the strategy of B**, and adapt the strategy of player A in order to have a better victory rate.

Statistics on strategies

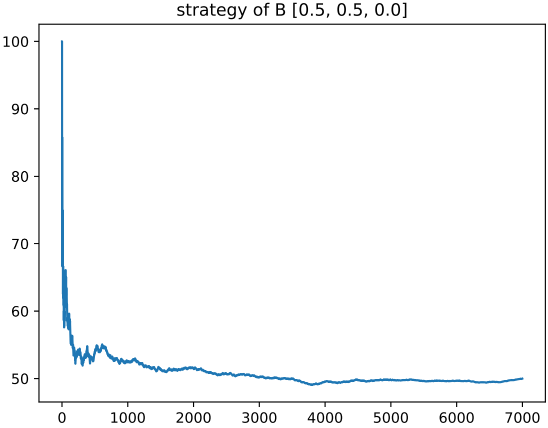
Exercice 5 : **Learning a strategy**

- Modify the file **paper_rock_scissors_learn.py** in order to **learn the strategy of B**, and adapt the strategy of player A in order to have a better victory rate for A.
- Several solutions are possible.
- You can work in groups.

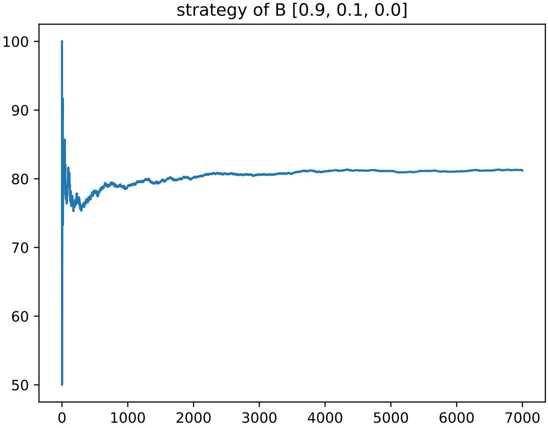
Percentage of victory



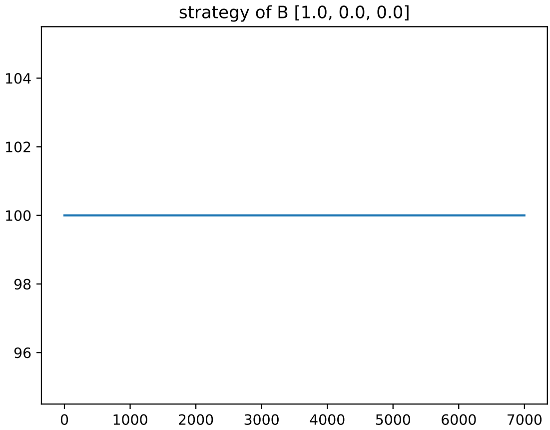
Percentage of victory



Percentage of victory



Percentage of victory



Sequential games

- We will now change the type of games studied
- We still have two players but the game consists in a **sequence of actions**, instead of a **single action**.

Sequential games

- We will now change the type of games studied
- We still have two players but the game consists in a **sequence of actions**, instead of a **single action**.
- The two players play successively, taking into acount the previous actions, and also the following actions from their opponent.

Sequential games

- We will now change the type of games studied
- We still have two plkayers but the game consists in a **sequence of actions**, instead of a **single action**.
- The two players play successively, taking into acount the previous actions, and also the following actions from their opponent.
- until the game reaches a **final state**. When the game is in its final state, the players receive a score.

Sequential games

- One player is called the **maximiser**, the other player the **minimiser**.

Sequential games

- One player is called the **maximiser**, the other player the **minimiser**.
- The maximiser tries to get the **highest score**, while the minimiser tries to get the **lowest score**.

The Minimax algorithm

- We will study and implement an algorithm that computes the values of the actions of the two players.

The Minimax algorithm

- We will study and implement an algorithm that computes the values of the actions of the two players.
- **Very important hypotheses :** the agents are assumed to behave **rationally**

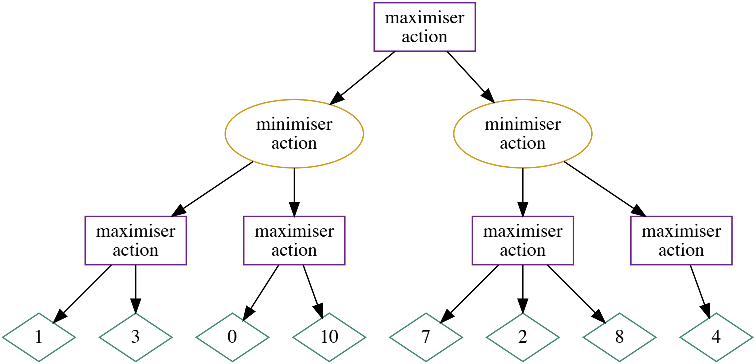


Figure: Representation of the game

Recursion

- The Minimax algorithm is based on a concept called **recursion**.

Recursion

- **Proposed definition** : a method to solve a problem based on smaller instances of the same problem.

First Recursion example

- **cd recursion**
- Please modify **factorial_rec.py** so that it computes the factorial
- $n! = 1 \times 2 \times \ldots \times n$

Recursion

A recursive function always has :

- a base case
- a recursive case

Warning

- Decrease does not mean terminate !
- What happens with the example **bad_recursion** ?
- In python, you can see the recursion limit with **sys.getrecursionlimit()**

Minimax algorithm

- The **Minimax algorithm** is a recursive algorithm that computes the values of all the nodes.

Minimax algorithm

- The **Minimax algorithm** is a recursive algorithm that computes the values of all the nodes.
- It does so by propagating the information from the **leaf nodes** (the **final states of the game**) to the parent nodes.

Minimax algorithm

- The **Minimax algorithm** is a recursive algorithm that computes the values of all the nodes.
- It does so by propagating the information from the leaf nodes (the **final states of the game**) to the parent nodes.
- Let us apply it on an example.

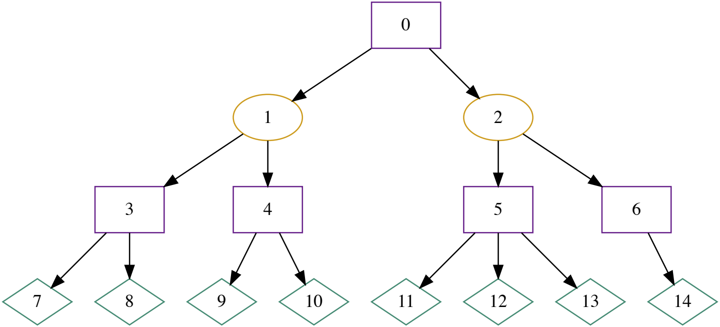
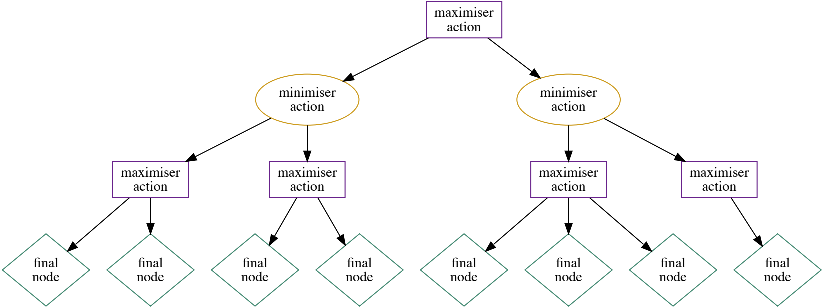
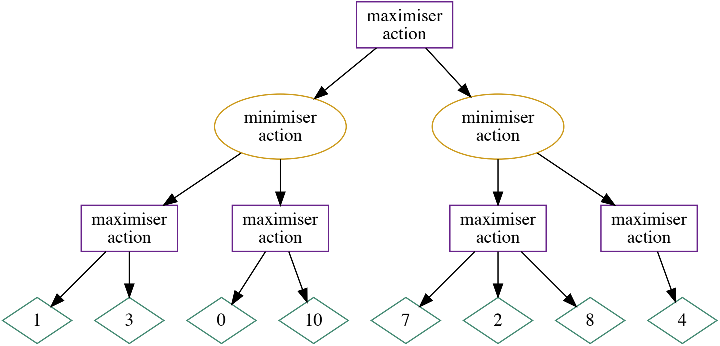


Figure: The numbers do not represent the values here : they represent the index of the node.


maximiser action
minimiser action
minimiser action
maximiser action
maximiser action
maximiser action
maximiser action
final node
final node
final node
final node
final node
final node
final node
final node


maximiser action
minimiser action
minimiser action
maximiser action
maximiser action
maximiser action
maximiser action
1
3
0
10
7
2
8
4

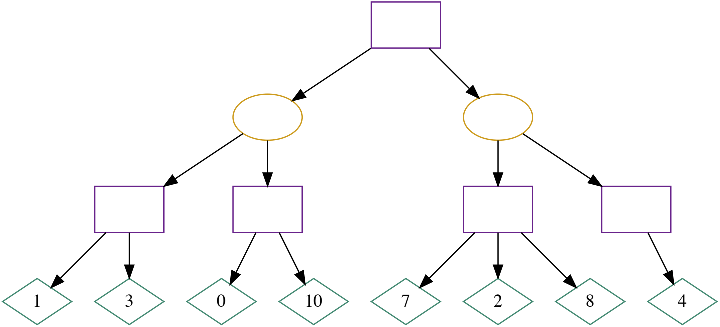
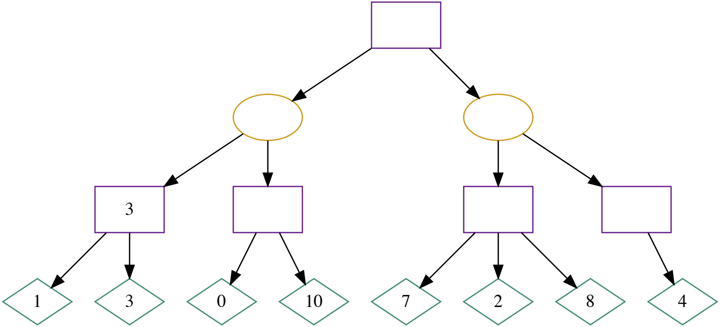
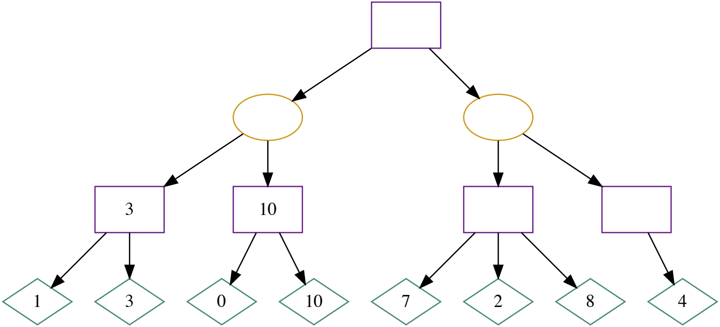
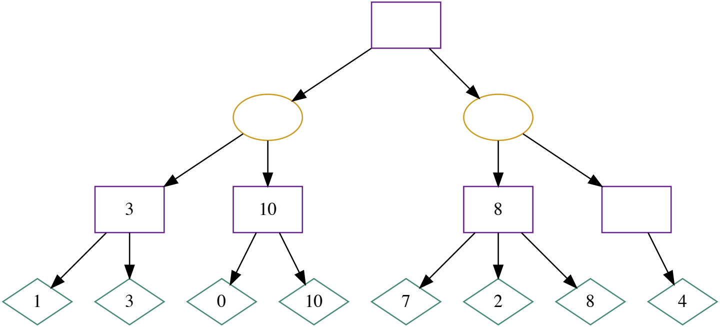
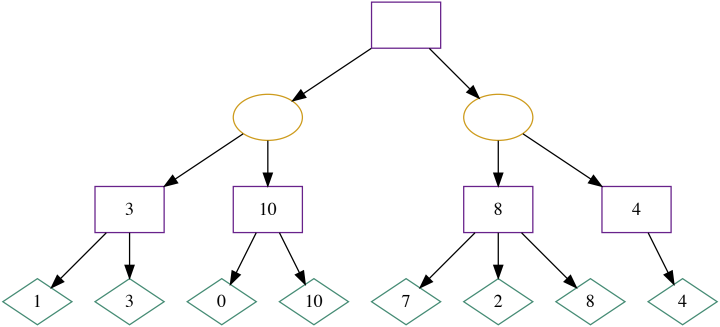


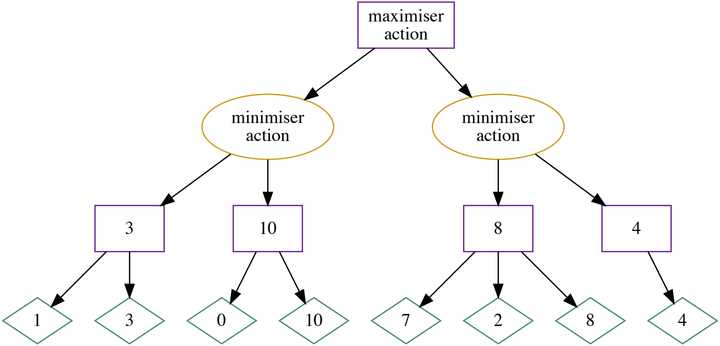
Figure: Values of the final states

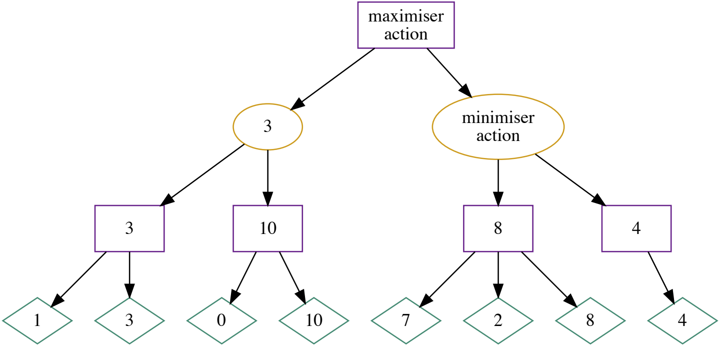

3
1
3
0
10
7
2
8
4

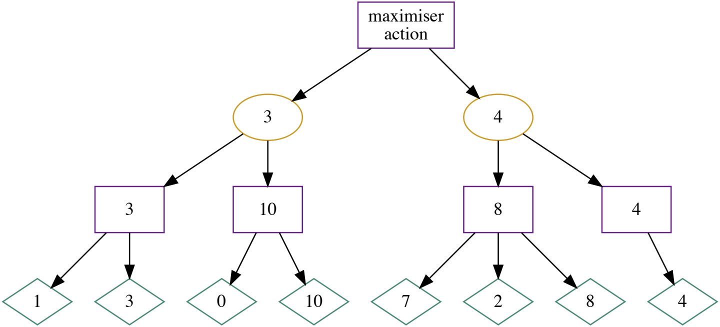

3
10
1
3
0
10
7
2
8
4

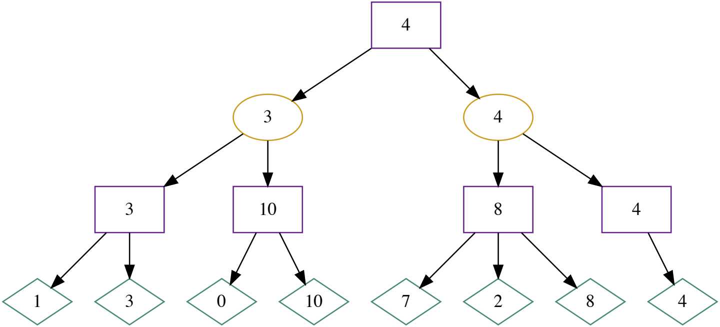

3
10
8
1
3
0
10
7
2
8
4


3
10
8
4
1
3
0
10
7
2
8
4


maximiser action
minimiser action
minimiser action
3
10
8
4
1
3
0
10
7
2
8
4


maximiser action
3
minimiser action
3
10
8
4
1
3
0
10
7
2
8
4


maximiser
action
3
4
3
10
8
4
1
3
0
10
7
2
8
4


4
3
4
3
10
8
4
1
3
0
10
7
2
8
4

Python dictionaries

- **dictionaries** are a useful data structure.
- Demo with **ipython**

Minimax algorithm

Exercice 6 : **Implementing the algorithm**

- Please use the file **minimax.py** in order to implement the algorithm.
- I inserted 2 errors in the **minimax** function.
- you can also try with different values for the final states.

Limitations of the method

- What could be the problems with the Minimax algorithm ?

Limitations of the method

- Let p be the **branching factor** of the tree. (Here, the average number of children at each node)
- Let d be the **depth** of the tree.

Limitations of the method

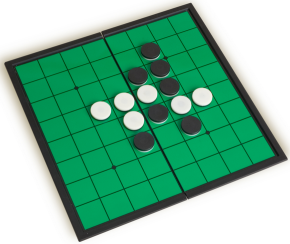
- Let p be the **branching factor** of the tree of actions. (Here, the average number of children at each node)
- Let d be the **depth** of the tree.
- What is the order of magnitude of the number of nodes in the tree ?

Limitations of the method

- Let p be the **branching factor** of the tree of actions. (Here, the average number of children at each node)
- Let d be the **depth** of the tree.
- What is the order of magnitude of the number of nodes in the tree ?
- So in order to run the minimax algorithm needs to perform p^d evaluations.

Exercise

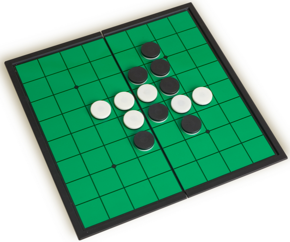
- In the **Othello game**, the average number of actions in each state is around 8.
- We assume that the evaluation for one leaf node takes 1×10^{-6} seconds.



-

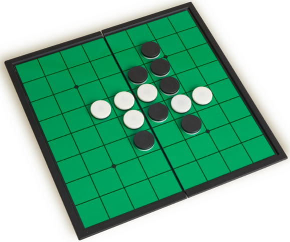
Exercise

- In the **Othello game**, the average number of actions in each state is around 8.
- We assume that the evaluation for one leaf node takes 1×10^{-6} seconds.
- How long would be the search of the minimax if we look 10 actions ahead ?



Exercise

- In the **Othello game**, the average number of actions in each state is around 8.
- We assume that the evaluation for one leaf node takes 1×10^{-6} seconds.
- This duration is too long to be used.



-

Conclusion

For real games, we need to use either :

- a smaller tree (look not too many steps ahead)
- a faster algorithm

Alpha-Beta pruning

- We will study a method that opitmizes the Minimax algotihm.

Alpha-Beta pruning

- We will study a method that opitmizes the Minimax algotihm.
- It does so by preventing us from computing **useless nodes**

Alpha-Beta pruning

- We will study a method that opitmizes the Minimax algotihm.
- It does so by preventing us from computing **useless nodes**
- Let us do it on an example.

Alpha-Beta pruning

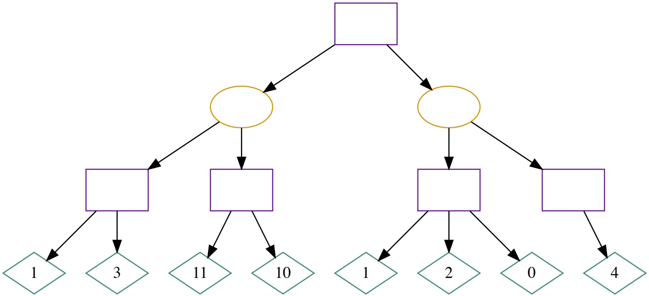
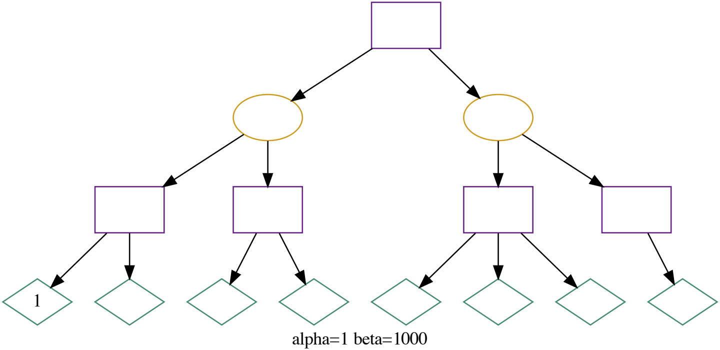
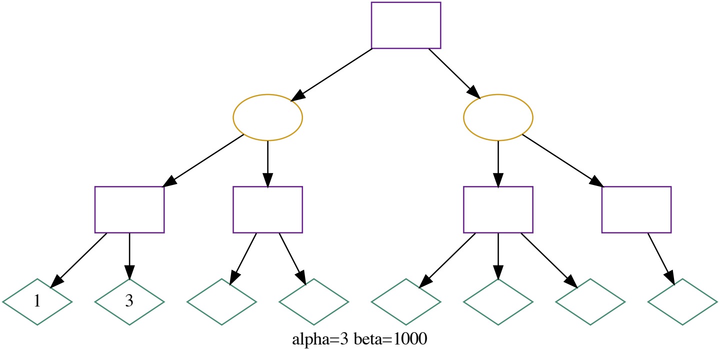


Figure: We use different final state values

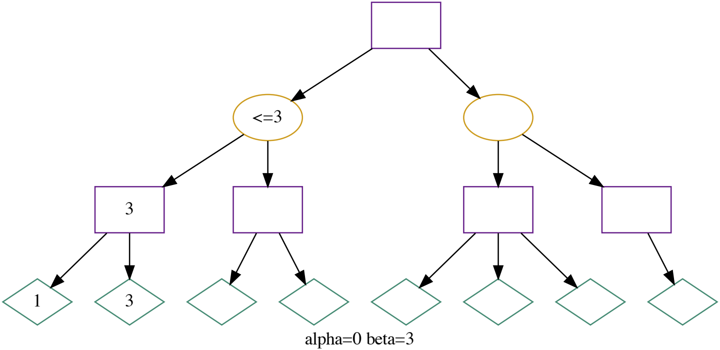
Alpha-Beta pruning



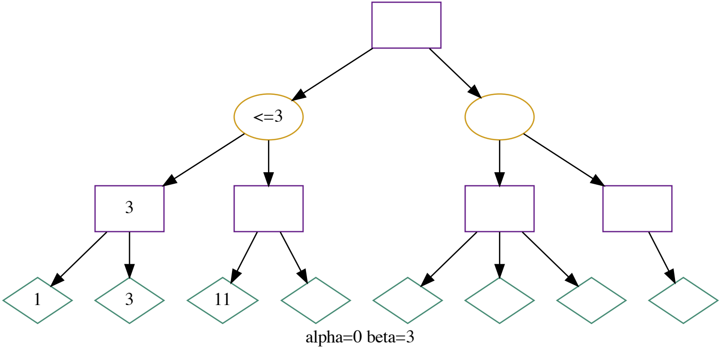
Alpha-Beta pruning



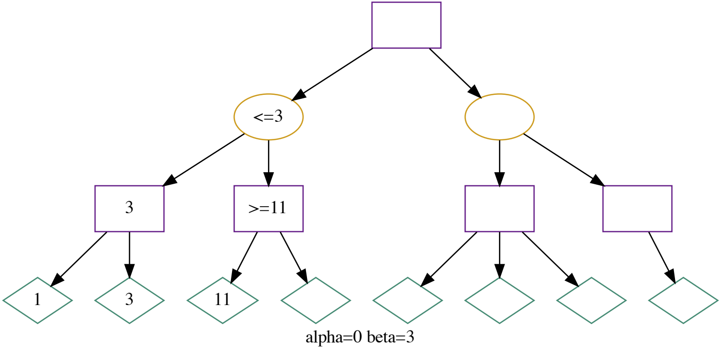
Alpha-Beta pruning



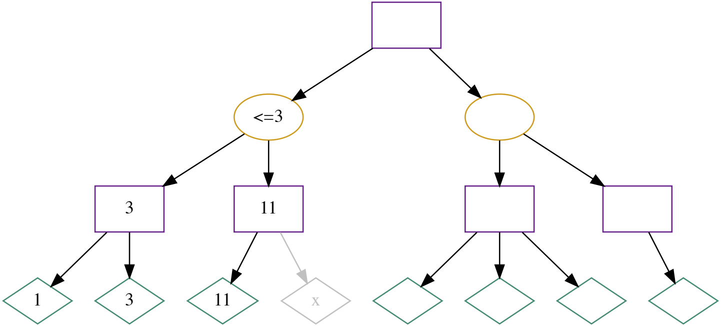
Alpha-Beta pruning



Alpha-Beta pruning

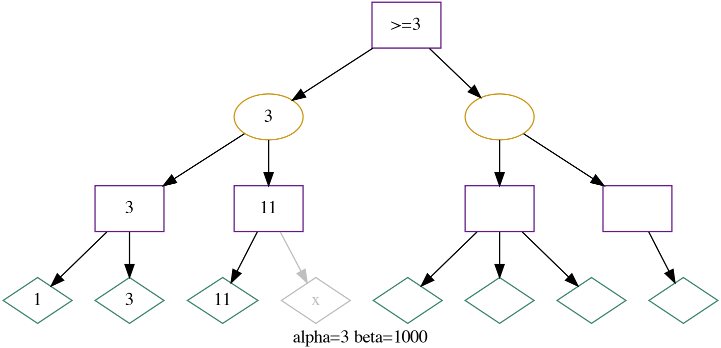


Alpha-Beta pruning

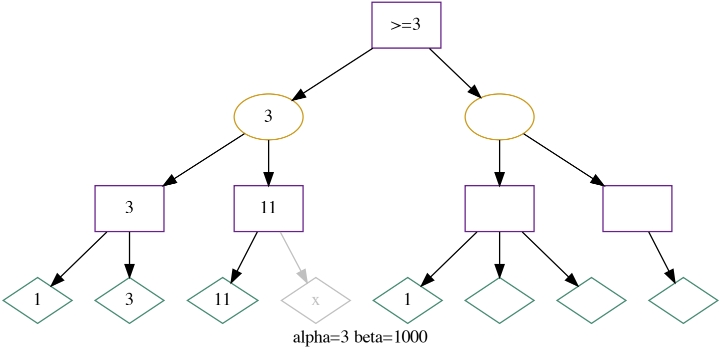


alpha=0 beta=3

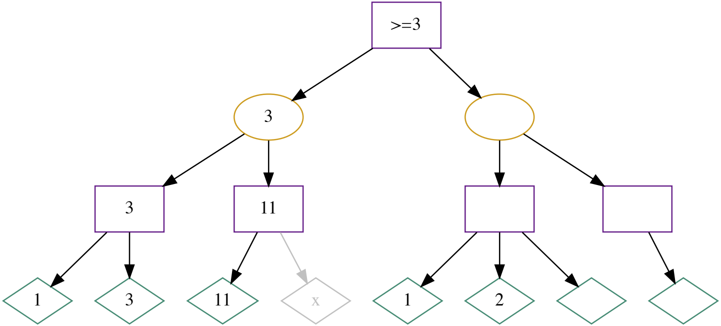
Alpha-Beta pruning



Alpha-Beta pruning

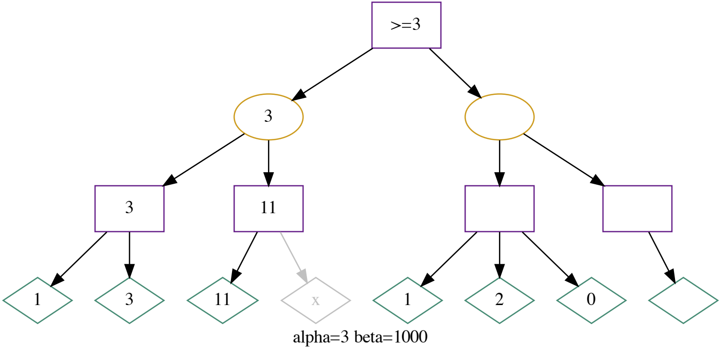


Alpha-Beta pruning

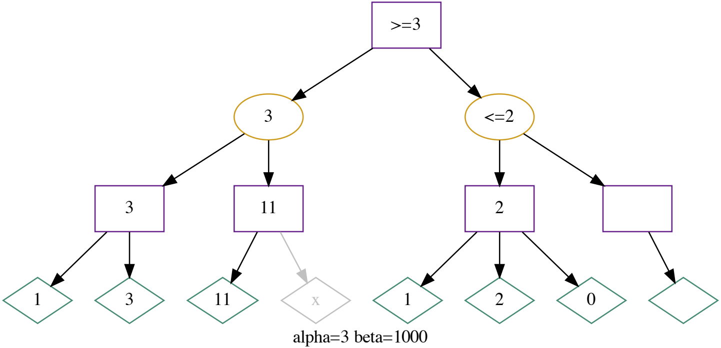


alpha=3 beta=1000

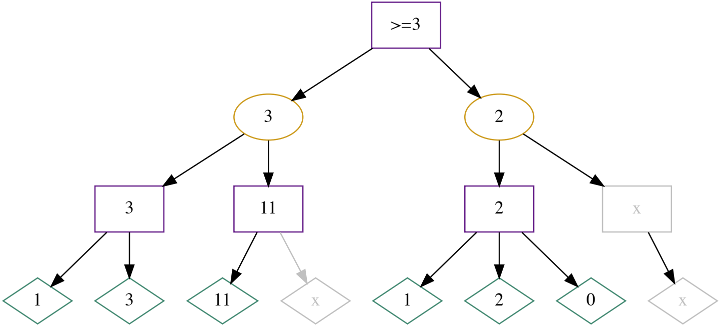
Alpha-Beta pruning



Alpha-Beta pruning

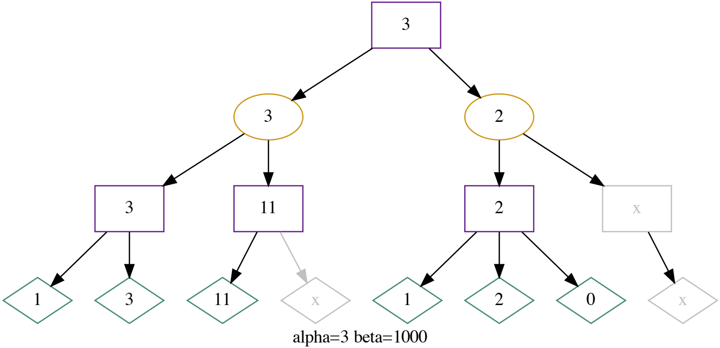


Alpha-Beta pruning



alpha=3 beta=1000

Alpha-Beta pruning



Implementation

Exercice 7 : **Alpha beta pruning**

- Let us now implement the algrorithm
- use the file **alpha_beta.py** in order to implement the algorithm.
- There are several mistakes in the code.

Verification

- Please verify that the AlphaBeta algorithm gives the same result as the Minimax algorithm !

Other values

- Please try to modify the initial values to change the behavior of the algorithm.

Orders of magnitude

- If N is the number of nodes explored by the normal minimax algorithm, the number of nodes explored by Alphabeta can be of order of magnitude $\sqrt{N}$
- This is a great improvement.
- **However**, please note that the improvement depends on the tree. Sometimes the pruning will not accelerate the algorithm that much.

Intermediate scoring

- Sometimes it is not possible to explore the entire tree, if it is too large, even with alpha beta pruning.

Intermediate scoring

- Sometimes it is not possible to explore the entire tree, if it is too large, even with alpha beta pruning.
- In this situation, it is necessary to use **intermediate scoring**

Intermediate scoring

- Sometimes it is not possible to explore the entire tree, if it is too large, even with alpha beta pruning.
- In this situation, it is possible to use **intermediate scoring**
- It is a heuristic : there is no theoretical proof that it yields the best solution, but it permits computation

Intermediate scoring and phantom of the opera

- Can you think of an intermediate scoring ?

Intermediate scoring and phantom of the opera

- Can you think of an intermediate scoring ?
- What are the depth and the width of the tree ?