GOMOKU

BOOTSTRAP

GOMOKU

Language: Anything working on “the dump”

- ✓ The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- ✓ All the bonus files (including a potential specific Makefile) should be in a directory named bonus.
- ✓ Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

At the end of this Bootstrap, you should have a sufficient global knowledge to start working on your algorithms.
2 themes must be introduced before you are able to do this:

1. the game interface and rules
2. basic AI algorithms

Game Interface

At the end of the project, your program should be able to play against AIs of various difficulty levels. The best ones may even be selected for a competitive tournament. For that reason, it is necessary your program complies with the same rules as everyone else:

1. Game rules. We use a simplified version here. Every player plays a stone at his/her turn, and the game ends as soon as one has 5 stones in a row (vertical, horizontal or diagonal)
2. Game protocol. We decided to play with the Gomoku AI protocol

Before starting to develop your AI, **implement the mandatory part of the Gomoku AI protocol**. We actually recommend you quickly write and test a dumb AI (say, random) to make sure you comply with the interface.

Once you have a working dumb AI (once again something that is random is good enough) you can test it directly using **liskvork** (the game arbitrator that you will be using for the whole project).

You can get pre-compiled binaries of **liskvork** here.

Or if you are feeling adventurous you can compile the code yourself and perhaps modify its code to test your AI in the best way possible from the
Epitech Mirror.

Only the mandatory commands are required for this project.

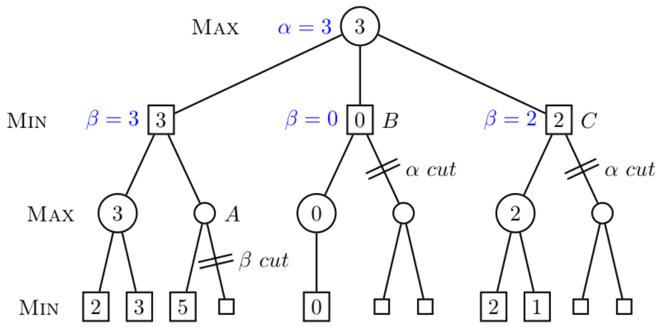
Algorithms

Once you have implemented the Gomoku AI protocol, you need to think about the proper AI algorithm for your bot.
Several choices seem relevant, and you have to study, understand and try.

The most obvious choice would be the **MinMax** algorithm (along with alpha-beta pruning and a couple of optimizations). Here you mostly have to design smart rules and control depth.

You could also try a statistical algorithm, based on **Monte-Carlo** method for instance. These techniques mix up pretty well with the former ones.

Finally, you could go for something completely different but very powerful, and dig for **Machine Learning** based algorithms. Be careful to use only methods you are able to implement yourself, since scientific libraries are not allowed for that project (no tensorflow, scikit-learn or scipy for instance). If you have mastered the functional programming projects, you already know some of them.



Google all the terms you don't perfectly master, write and execute bits of code, read some papers, discuss with your mates, search for optimizations,...

By the way, any idea on how to optimize your **evaluation function**?

V 4

{EPITECH}