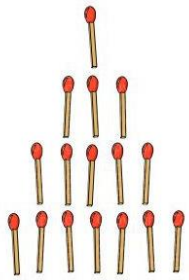
GPU Programming – Final Project Abstract

Policy search on Nim



Nim is a mathematical game of strategy in which two players take turns removing (or "nimming") objects from distinct heaps or piles. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap or pile. The goal of the game is to take the last object.

During the Computational Intelligence's course, I developed some agents that were able to play Nim with different strategies. Those agents were:

- An agent using some hardcoded rules (like random strategy, nim sum strategy…).
- An agent using some evolved rules: several human-like moves were evaluated, and the best was selected. Those rules were shortest-row, longest-row, first-row, last-row, middle-row, and for each of them a different number of taken object was selected.
- An agent using minmax: this strategy was able to calculate solutions on CPU only for a small number of rows (not more then 3).
- An agent using reinforcement learning.

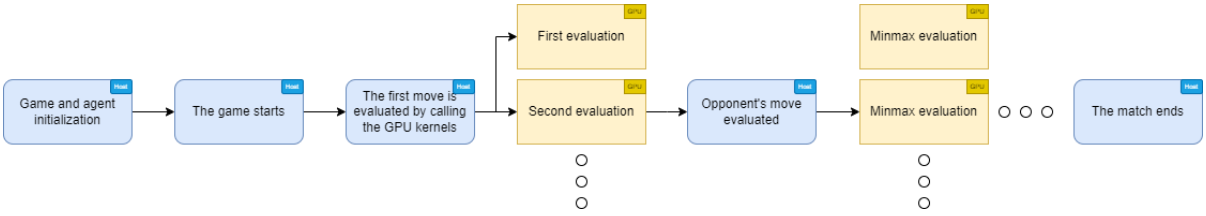
Here it is the course repository on Github: the third lab contains the implementation of the Nim strategies, along with a readme.

The Jetson Nano implementation

The idea is to adapt some of these strategies in CUDA. In particular, the minmax strategy can be improved a lot by parallelizing the evaluation of all the possible moves; as a matter of fact, right now it is the slower strategy that can be used (even if it's the more accurate).

During a match, the minmax is called at every move of the player, so each move can be evaluated by several kernels and the result is returned to the host, which actuate the move and continue the game.

Here it is a simple schema of the expected execution flow:



Some difficulties

The current implementation of minmax is recursive, so each kernel will contain a lot of for cycles: a solution (like assigning more then one kernel to each execution) should be found.

Several data structures like lists or dictionary are used; it will be a challenge to implement them in C or C++ (since I'm not so experienced with those programming languages).

Profiling in python could be difficult since I'm not so experienced with python either.