Project Arimaa – CR8 : Wednesday, November, 5th

Presents : Everyone, with Nikos Parlavantzas

Order of business :

1. Parallelization method implementation (Mikail)
2. Packages with rules implementation (Baptiste)
3. MCTS simple implementation + missing presentation (Benoit)
4. General Software architecture : diagrams (Gabriel)
5. API & I/O diagrams (Prateek & Dan)
6. Plan of the second report

Information :

- Parallelization methods by Mikail : It's about the parallelization of a cluster of values, there is 2 choices root parallelization with a cluster already implemented, more efficient but redundant (already executed before) and UCD Tree Split. The last one seems complicated toimplement, it only works for HPC (High Performance Computers). The question asked to Nikos was about the performance of Grid'5000 computer. Actually, we don't know what all computers we will use will be, but we sure know that a majority of these computers should have a great performance. For now, we will focus on the simple method (root parallelization), to start, and then go further with Grid'5000 computers. Mikail will do the implementation later (It's not for the next report anyway). Nikos prefers the architecture sharing messages with the same memory instead of MPI (but the decision has already been made), because the communication would have been easy with this messages. Thread are better than MPI for him because of the weight of the computation
- MCTS class diagram of Benoît. The diagram represents the MCTS algorithm. An implementation has been done, but without any demonstration in front of us (no test presented). Description : Bitboard is an implementation of the board. Move (class) is the output (it will evolves with the API, because an abstract class (with string) is better). For now, the main MCTS fonction uses a certain number of simulations, it can evolve towards a simulation time instead. We have to complete this algorithm with root parallelization. There is a possibility of the reuse of the old results, good but limited by the way we will stock it. The class state can be hidden or changed with an interface. The first fight point is "do we have to use Boost Library ?". For now, Benoit has begun without, but we will have to test it to compare execution times (we have time). The second fight point is "Do the MCTS Tree algorithm is a good solution near the end of the game ?" Benoit thought that the algorithm is not able to do it, because it will evaluate nodes randomly, based on a serious paper. He wanted to change the algorithm near the end of the game, forcing the algorithm to avoid a defeat for instance. The problem is the paper is about MCTS search algorithm and not MCTS tree search algorithm, as written in the same paper *"This note presents a simple counterexample to dispel the illusion that increasing the number of random playouts necessarily implies better estimates for simulation-based move planners. Incorporating a tree structure into the search helps redress the problem. "*
  So this is not a good solution. It will make the treatment heavier if we test everytime at the end of the game what to do, with the tree working this out for us. We will keep using MCTS, but still we need to see with Christian what he thinks (Nikos agrees with the MCTS theory). We want to reduce the number of moves calculated is 20000 at depth one (too big).
  We will keep using the idea of deleting the nodes with the less probability (or the more for the AI) to reduce the number of possibilities.
- About the set of rules/C++ library foreseen, according to Pascal Garcia, it is useless to do it in that kind of game because it is too easy to understand. We will use a simple interface and his implementation to design the rules class. This was Baptiste part, and the implementation of the interface has been done by Benoit. No more work for him to do here.

- About the general architecture, Gabriel has prepared a simple communication diagram between the Model and the UI, then a use case diagram of our game, and a general UML diagram for his application, (the first one), using the MVC mode, detailed (We don't need UML for know, except in the case we want to use his code in or algorithms)
- About the API part (Prateek and Dan), they have done a diagram describing all states of the game, from the start menu to the end of the game, describing initialisation (MCTS then user) and moves. The main problem was the relationship between the MCTS program and the original application program (by Gabriel, handling the view). We need to create a program that converts one format to another. We have decided to put it close to the MCTS program, reducing the sending of messages (1 instead of 2).
  We have chosen active waiting for our program because we will do parallel calculs in different clusters of processors that will finished almost together at the same time. So we won't use interruptions, just simple functions calls.
  We will represent pieces by binary codes to go faster analyzing it (a map is a good beginning). See more on the next part : detailed todo list (input/output part)
- add the creation of a file to create the game, listing all moves

Detailed todo list (mostly to AVOID next fights)

- Parallelization implementation (Mikail) :
  - Look for a framework that will talk with messages between threads.
  - Does MPI support asynchronous communication (blocking and none blocking) ?
- MCTS algorithm implementation (Benoit) :
  - Transform Bitboard and moves into an interface (The API part will take care of the move implementation of the interface)
  - Complete the algorithm with root parallelization
  - Keep the same algorithm regarding the end of the game (1st Fight Point)
  - Test it (or in front of us)
  - Use Boost library to do the same thing and compare performance (Boost described as used in the first report) (2nd Fight Point)
- General architecture
  - Not only diagrams (explanations with it), based on the presentation of the project
  - talked about secondly architecture, user interfaces
  - Structures and behaviour of the system,
  - put details if necessary (and if we have them), in another diagram
  - Write the input/output of the diagram for the competition between two computers (user of not), using network (find the protocole). Look on Arimaa website. Only boxes for now, but we can add things we know. Add what we will do, not how (avoid), no low level for the program

Planification

Task	Responsible	Deadline
OpenMP report part	Benoit and Baptiste	11/19
MCTS report part	Benoit	11/19
MPI report part	Mikail	11/19
Parallelization method report part	Mikail	11/19
OpenAcc report part	Baptiste	11/19
Behaviour of the game report part	Gabriel	11/19
General Architecture review	Gabriel	11/19
Conclusion report part	Gabriel	11/19
Input/Output report part	Dan	11/19

Introduction report part	Dan	11/19
Abstract report part	Prateek	11/19
API report part	Prateek	11/19
First draft of the report	Everyone	11/19
Last draft of the report	Everyone	11/26
Due date for the report	Dan	11/27
OpenMP implementation	Not decided	Begin after 11/19
Finish the game application	Gabriel	12/18

Next meeting : 11/12