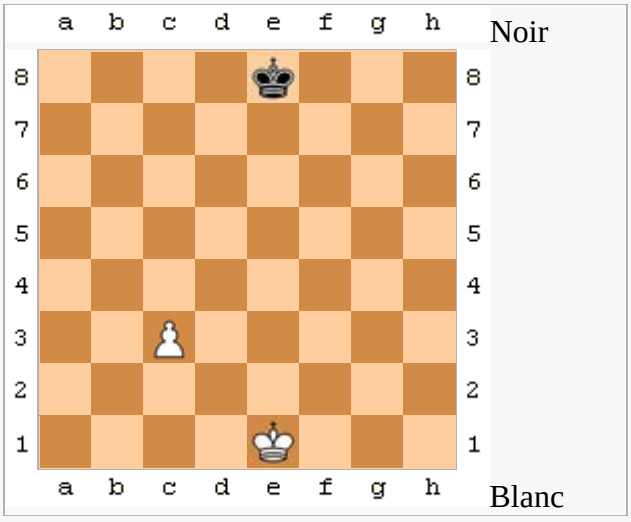
M1 SIA : Lab Session 3 2008/09

ChessFighter

A company offering a Web site dedicated to the game of Chess (articles, tutorials, …) wishes to build an application to compare computer chess artificial intelligences (AI). The AI will be connected to the ChessFighter application through a specific API. ChessFighter determines the winner of each game and computes the ratings of the Ais, using ELO rules not detailed here.



A game of chess : Black King e8 ; White King e1 ; White Pawn c3

Chess is a game for two players, designated as White and Black. The game is played on a square chequered chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") and there is no way to remove it from attack on the next move.

Each piece has specific rules governing its possible movements which will not be detailed here. The game is finished when one player checkmates his opponent (victory), when a draw situation is reached (draw), or when one of the players abandons (opponent victory). A draw can also be proposed to the opponent, who can accept it or continue playing.

A game is started by configuring two virtual players that should implement a software interface that needs to be designed. Players then alternate playing by notifying Chessfighter of their moves. A move is described through two coordinates giving the start and end position, e.g. “e2-e4”. The start cell should contain a piece of the player’s colour. The move should be legal according to specific movement rules that depend on the piece (see question 4).

Question 1 (5 point)

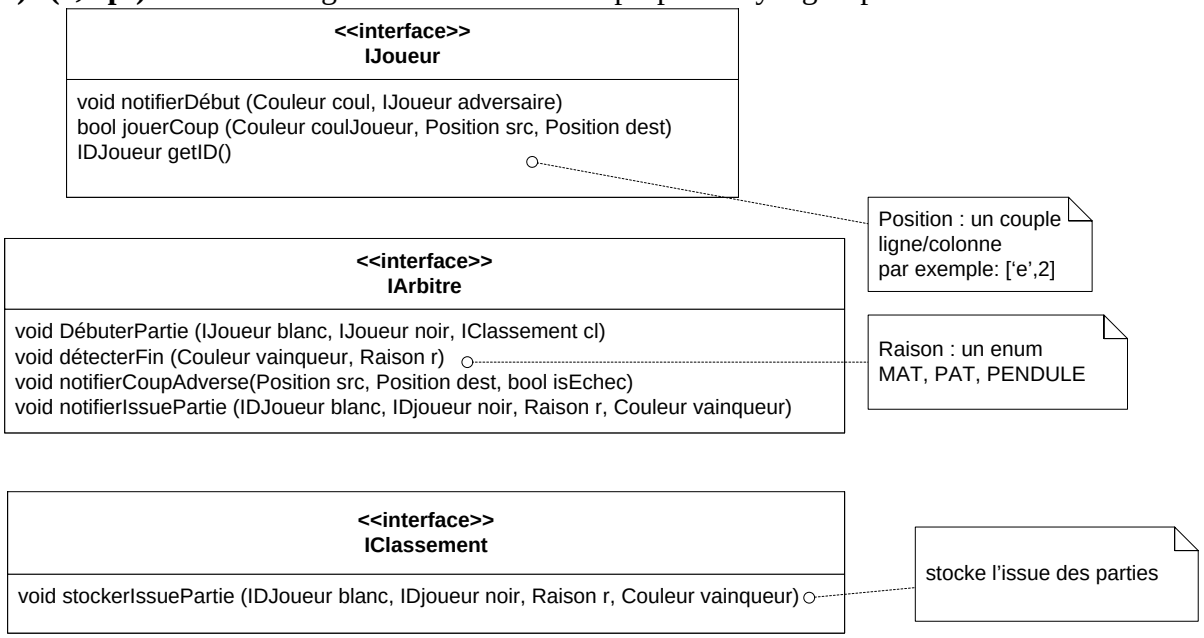
Preliminary design has identified three main components in the application :

- A component « rating » whose role is to rate players according to their victories, defeats and tied games. It records in a persistent manner the results of past games. It

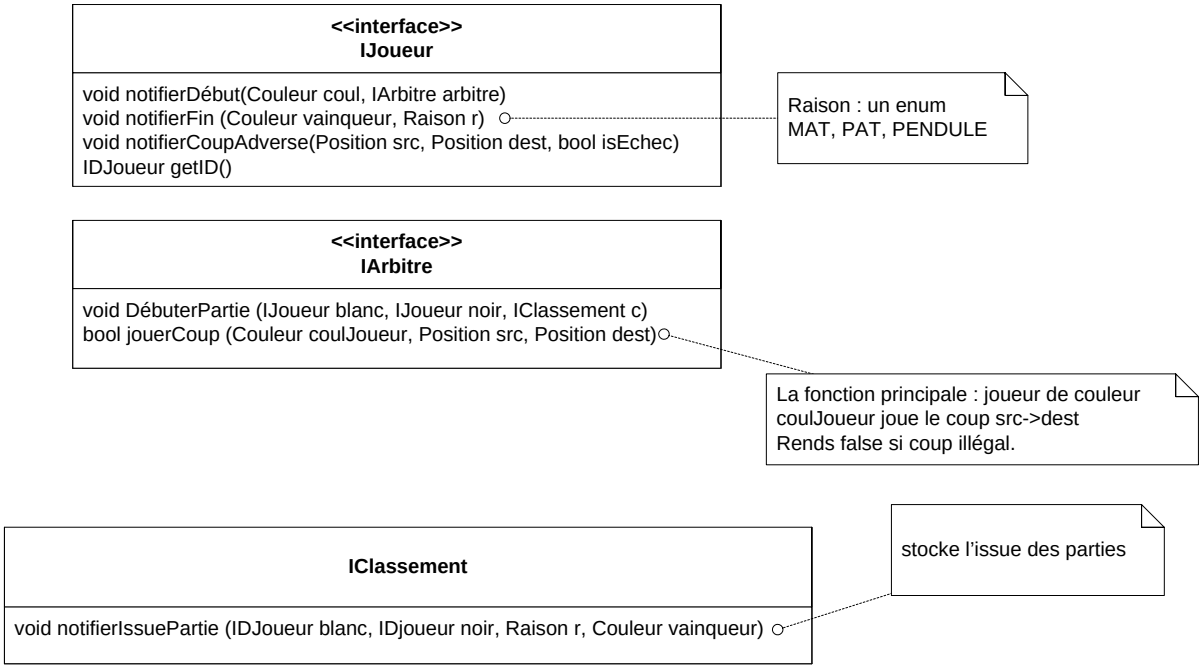
can be queried to know the rating of a player, his statistics… But this query interface will not be detailed in this subject.

- A component « player » that represents an AI playing games. It can be notified of the start or end of a game. During a game it is notified each time the opponent plays, and provides in return his own move.
- A « referee » component, instantiated once per game. It checks that the moves proposed by the « player » components are legal, otherwise moves are not validated. The same player should then play again. When the move is valid, the opposing player is informed of the move. It also controls the position of the pieces, and announces any « check », « check mate » or particular win situation (End of clock time, tie situations…) when they happen. Finally it informs the rating component of the game outcome.

a) (2,5 pt) The following interfaces have been proposed by a group of students :



Another group of students propose instead this set of interfaces :



Unfortunately, some students missed the course and have gotten things severely upside down.
Which design is correct? Explain why and criticize the incorrect design.

b) Build a sequence diagram that shows these components interacting in a typical nominal game situation. Try to exhibit all the operations defined.
c) Give a component diagram showing the required and provided interfaces of the three components.
c) Represent a possible component instanciation in a game opposing « Deep Blue » to « Simply Red ».

Question 2 (3 point)

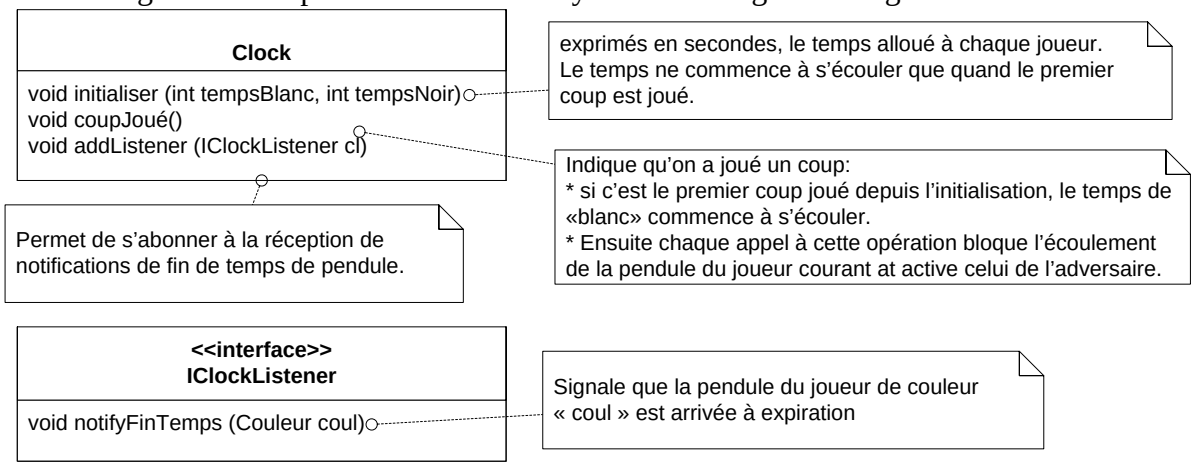
a) (2 pt)
Define a class diagram allowing to represent the state of a game in the referee component : game situation, player turn…
Detail all attributes (precise types) and relationships (navigability, cardinality, composition/aggregations…).
Operations of these classes need not be represented in this question.
b) (1 pt)
Give the links that exist between these classes and the interfaces of question 1.

Question 3 (4 point)

The game of chess, particularly online, is often limited in length by a game clock. The clock limits the amount of time available to play moves. A **game clock** consists of two adjacent decrementing clocks and allows to stop one clock while starting the other, such that the two component clocks never run simultaneously. The purpose is to keep track of the total time each player takes for his own moves. When a player's clock reaches 0, he loses the game. The game duration can be configured when setting up the game, and is usually less than 10 minutes per player. Of course, the game can also finish normally before the clock runs out.

This behaviour will be integrated in the referee component. The "StartGame" (DébuterPartie) operation will now take as additional parameters the initial clock values (cf. Q1)
We provide a Clock and an interface IClockListener, already implemented in a separate project. The Clock manages a game clock, i.e. with separate time accounting for each player.

The existing Clock component is described by the following class diagram.



a) (1 pt) Which Design Pattern do you recognize here ? Describe briefly its usefulness.
b) (1 pt) Explain how to use this existing Clock by integrating it into the diagram of Question 2.
c) (2 pts) Show through some sequence diagrams how the interaction between the Clock and the ClockListener happens. Represent one lifeline for the Clock and one for the IClockListener. Represent:

c.1) The startup phase, with 5 minutes per player.
c.2) The interaction when a player has played a valid move.
c.3) What happens when a clock reaches 0.

Question 4 (4 point)

Analysis phase has identified the following Validation test :

Test TV2 : End of game on Clock timeout.
Context : A game has been started with 1 minute per player. Cf. TV1, to be executed before this test.
Input : white's move :« e2-e4 »
Scenario :
1. White plays the move « e2-e4 ».
2. We wait for one minute, without touching the GUI.
Expected result : ChessFighter announces White victory
Verification Means : use a manual chronometer outside the application.

We wish to set up some integration level tests to ensure the good behavior of the referee.
To this end, we wish to implement a minimal « cork » class, allowing to run an integration test on the referee component that corresponds to this validation test.

a) (1 pt) Which interfaces must the cork class implement to represent the environment as seen from the referee's point of view ?
b) (3 pts) Explain how to implement this integration test by proposing an implementation of the cork class operations. Use java or pseudo-code.