

SWS: Final Assignment

November 2, 2006

The report may be in Dutch or English.

Acceptable formats are: plain text (ASCII) and pdf.

E-mail it, at the latest on Monday 13 November 2006, to `lambert@cs.uu.nl`.

Part I

Games like checkers, chess and go are two-person games with complete information. In such games, two players take turns, starting from some initial *state of the game*, in performing *moves* that change the state, until one of the players *wins* (in which case the other *loses*), or the game ends in a *draw*, in which neither player wins or loses. Either player knows the complete state of the game when it is their turn to move. Chance plays no role, except perhaps in determining who has the first move.

Two remarks. (1) The state can be more than just what you see on the board. For example, it also covers whose turn it is, and for chess furthermore whether castling has occurred, and in fact the whole history of the game because of the rules concerning repetition of moves. (2) A move can be more than a move on the board. For example, in chess a player can resign, which we also consider a move.

The company BrainGames Inc. develops and markets game-playing programs, in particular such two-person games. Instead of specifying the logic of each next game from scratch, they want to have a suite of modular reusable specifications from which the specification for a new game can be assembled. A central role is taken by the “game shell”, a generic module that needs to be parameterized only by one game-specific component. The game shell by itself must be usable across the whole gamut of games: chess, go, etc.

The game shell (GS) communicates with two parties: the Player (P), and the game rules (GR). Although different for each game, the protocol remains the same.

Because these games can be difficult, GS may be in a “thinking” mode: it is the turn of GS to move, but no move has been computed yet. During this time it will only accept an “I resign” message from P.

In communicating with P, the typical interaction is as follows. GS initiates the game by transmitting the initial state to P (thereby implying the question whether P wants to have the first move or wants the computer to start). If P responds by indicating “I start”, GS prompts P for the first move. Otherwise, on receipt of “You start”, GS goes into thinking mode, until a first move has been computed. GS then transmits the first move and the new game state, implying that it is now P’s turn. In either case, P has to respond with a move. Upon receipt of P’s move, GS reacts with one of several possibilities:

- You win / You lose / It is a draw; game is over.
- Illegal move; try again.
- OK, here is the new state.

In the first case (game over), no further communication takes place. In the second case, GS waits for another move from P. In the third case, GS goes into thinking mode. Once a move has been computed, it is transmitted to P, together with the new state of the game, again with several possibilities:

- You win / You lose / It is a draw; game is over.
- Game continues.

If the game is not over, GS awaits a next move from P as before, and so on.

In communicating with GR, the typical interaction is as follows: GS can send a message “New game” to GR, which is responded to by transmission of the initial state. GS can further send a message to GR containing a game state and a move. The response contains the new game state after the move, unless the move was illegal. For the rest, the possible responses are further:

- P wins / P loses / It is a draw

- Illegal move
- Normal (legal move, game not yet over)

GR may take its time to respond, which is why GS needs a “thinking mode”. But note that it is actually the GR component that does the “thinking”. GS itself does not know the rules of the game. It always has to consult GR.

Your task is to write the specs for this game shell. You do not need to pay attention to the user-interface aspects (which are game-dependent). If convenient or necessary, you are allowed to make design changes to the above approach *provided that the generality is not lost*, and if you do, make sure you explain the changes.

Part II

In order to test game software to be brought on the market, BrainGames Inc. lets the software play games against itself, or different versions for the same game against each other. So assume we have a GR1 and a GR2. By composing these with GS we get two game versions: $GS+GR1 = GV1$ and $GS+GR2 = GV2$. These should play against each other.

This will be done by putting a module between the two: the game master (GM). GM accepts moves from the GV’s and transmits them to the other GV according to the protocols of Part I. So GV “looks like” P to each of the GS components of the GV’s; they don’t know they are playing against a computer.

Instead of a single game, the whole system $GV1+GM+GV2$ should play a continuing stream of games, logging the games in a database, until some external operator transmits a HALT signal.

Your task is to write the spec for GM. As before, you may make necessary changes, but describe these well.

Part III

Write a brief essay in which you summarize, in your own words, what you learned in this course. Please don’t use more than one page.