Distribuerte systemer, øving 1

Markus Pike, Kristine Steine

Vi har valgt å implementere nettverksdelen av oppgaven ved å lage:

1. Et interface som definerer alle metodene som skal brukes av `Remote`-objekter
2. En klasse som arver fra dette interfacet og håndterer alt av enkeltspillerens funksjonalitet, inkludert sending og mottak av beskjeder om at en spiller har gjort en endring på sitt spill.

Interfacet, `Player`, utvider (`extends`) `Remote`, og setter standarden for objektene. Her har vi definert metoder for blant annet `lookup, connect, disconnect.`

Objektklassen, `TicTacToePlayer`, implementerer `Player` og utvider `UnicastRemoteObject`. En `TicTacToePlayer` opprettes med en gitt url og et portnummer som angir hvor objektet skal være tilgjengelig.

Når en `TTTP` opprettes vil den gjøre en `lookup` for å se om url-en er i bruk, og ut fra resultatet vil den sette seg opp som en server eller en klient. Om `lookup` ikke returnerer en motstander, vil `TTTP` opprette et `Registry` og binde seg til url-en. Om `lookup` derimot returnerer en motstander, vil `TTTP` utføre fjernkallet `opponent.connect(this)` som sender motstanderen en referanse til det kallende objektet slik at de begge kan kjenne hverandre.

Opprettelsen av `TTTPlayer` skjer i `TicTacToe`-objektet som en del av konstruktøren. Hvert spillobjekt oppretter kun én `TTTP`.

Spillobjektet `TicTacToe` håndterer endringer i `BoardModel` ved å lytte etter endringer og sjekke om disse er gyldige. Om de er det, vil spillet kalle `TicTacToePlayer.notifyOpponent(int x, int y)`, som gjør et fjernkall til motstanderen om at den må endre sitt spill til å inkludere endringen. Samtidig settes en verdi om at det er motstanderens tur til å gjøre endringer. Slik oppdateres spillet hos begge spillerne samtidig.