CS1006
P2 – backgammon

Saleem Bhatti

with material originally from

John Thomson

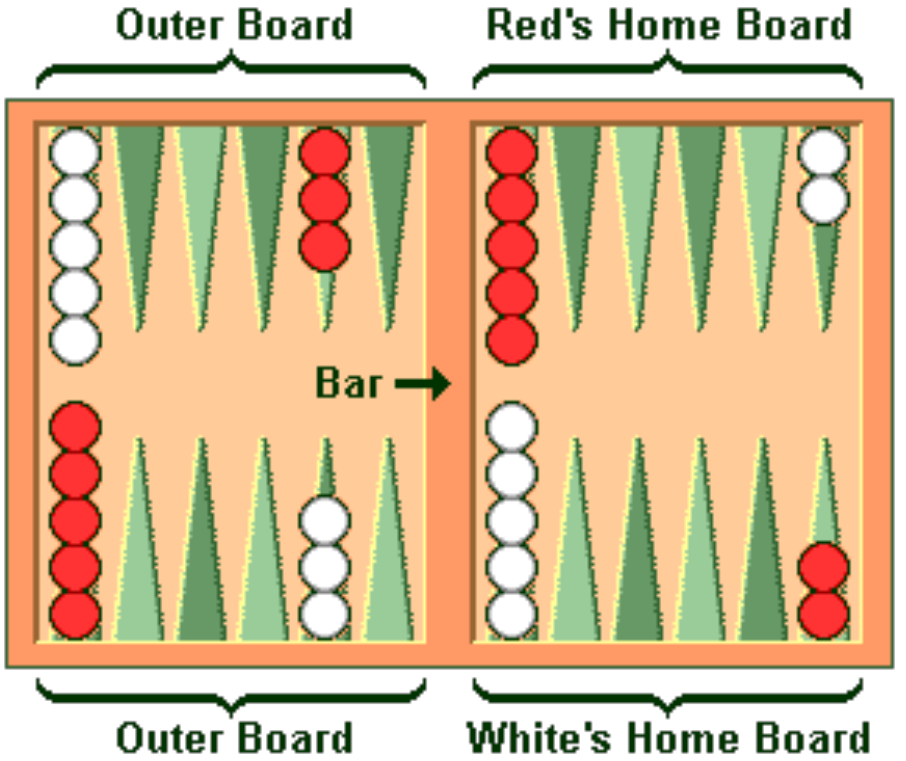
What is backgammon?

University of St Andrews

- Backgammon is a table/board game (probably) originating in Persia.
- The objective is fairly simple – get all your counters from one side of the board to the other and out the other side.
  - counters aka checkers aka draughts
- Rules are also straight-forward, but that does not mean that game play is simple!

Backgammon board

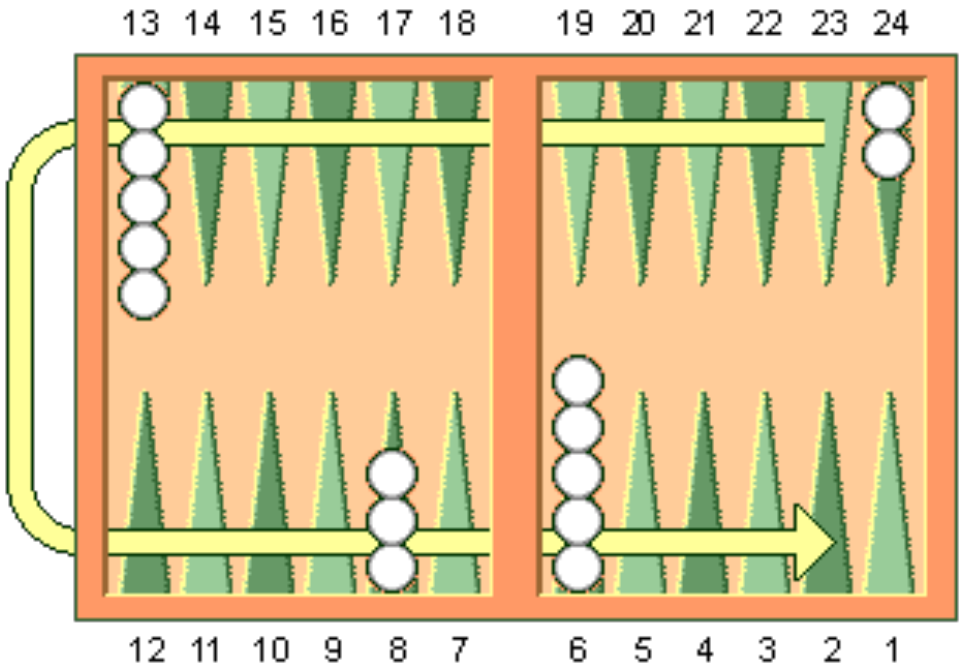
- Game for two players
- Red / White
- Two dice
- Board has four areas:
  - Red's home board
  - White's home board
  - Outer board
  - Bar



http://www.bkgm.com/rules.html

Backgammon play

- Move all your counters to your home board
- Cast them off (remove them) from the board, e.g. direction of White's moves are shown
- Dice values control moves



http://www.bkgm.com/rules.html

Rules [1]

University of St Andrews

- Twelve triangles on each side of the board, connected in a horseshoe.
- Get all your pieces into the home board, then ‘bear off’ (remove them).
- Roll two dice and move a piece for each dice (can be the same one).
- Roll a double, and move four times!
- Moves are compulsory unless impossible.

Rules [2]

University of St Andrews

- Safety in numbers
  - single counters on a triangle can be ‘captured’ by the opponent landing on them.
  - captured counters are sent back to the start.
  - a player cannot move into a triangle occupied by two or more of their opponent’s draughts.
- Need to think about (relative) placement of counters.

Rules [3]

University of St Andrews

- When all counters have reached a player's home board (last quarter) then bearing off occurs.
- Counters exit the board by moving off the end of the board.
- Rolls need to be exact, e.g.: if a counter is in triangle 1 then a 1 must be rolled; if in triangle 2, roll a 2 etc.
- If no move is possible, then nothing happens (turn passes to other player).
- First person to bear off all counters **wins**!

Rules [4]

University of St Andrews

- Some rules at:
  - http://www.bkgm.com/rules.html
- We will use a simplified version:
  - we do not use the Bar (a caught player goes to the start, i.e. the last triangle)
  - no 'stake' for each point, just first to bear-off all counters
  - we do not use a "doubling dice"
  - no "optional rules"
  - single / individual games only

Strategy

University of St Andrews

- Beyond the fairly simple rules comes strategy:
  - what is the best move to make?
- Think about:
  - how you might maximise your chance of getting your counters to you home board
  - 'protect' your counters from capture as they go round the board

Requirements

University of St Andrews

- Should have a well-designed interface (text-based or graphical)
- Should be playable by:
  - two humans
  - one human vs. “the computer”
  - two computers
  - over the network (in the lab)

Network communication [1]

University of St Andrews

- Backgammon program starts. Begins listening on a specified port, ready to receive requests.
- Interface option to initiate a game on a specified machine name. Request a game.
- Accept or reject a new game. Negotiate who starts first.
- Exchange of text messages (protocol specification is provided).

Network communication [2]

University of St Andrews

- A **protocol** is required for communication:
  - a well-defined set of messages and rules for the exchange of messages.
- Think about:
  - what needs to be communicated
  - how it should be represented
- A common protocol will allow interoperability between programs.

Network communication [3]

University of St Andrews

- You will use the Transmission Control Protocol (TCP) via the Java Sockets API.
- A TCP socket will be used to transport your own "backgammon protocol" messages
- A tutorial on using Java Sockets is here: http://docs.oracle.com/javase/tutorial/networking/sockets/
  - (you do not need Datagram Sockets or Multicast Sockets.)
- You will need to make the sockets read operations non-blocking by using the method: http://docs.oracle.com/javase/8/docs/api/java/net/Socket.html#setSoTimeout-int-
- I will also give you some example code for Sockets.

Network communication [4]

University of St Andrews

- To use a Socket, you need to use:
  - a hostname, e.g. (or IP address)
  - a port number
- The hostname will be the one for the lab workstation, e.g.: *pc3-045-l.cs.st-andrews.ac.uk* (138.251.212.107)
- For the port number (from a linux login):

  $ id -u
  10784

  add 30000 to the number that is reported.
- Connect to other user's code by using the hostname and portnumber for their program (your IP address and port number will be passed to them).

Testing

University of St Andrews

- Use text messages for your protocol:
  - easy to debug – just print them!
- You can print your protocol messages to debug the message flow between your programs.
- You can print to System.out, System.err, or to a separate log file – the latter is particularly useful.
- Information on working with files in Java:
  - http://docs.oracle.com/javase/tutorial/essential/io/streams.html
  - (you do not need NIO or NIO2)

Artificial 'intelligence' (AI) [1]

University of St Andrews

- The computer as a player - a very basic AI:
  - find a legal move and play it!
  - this is a good starting point, but unlikely to win.
- A good AI performs some sort of **evaluation** of the current board.
- Need to decide an **evaluation function**:
  - may still not make a very good player! Why?
- Can use **heuristics**:
  - some things that a 'good' player might do

Artificial ‘intelligence’ (AI) [2]

University of St Andrews

- More sophisticated AI - look-ahead:
  - a good player will ‘look-ahead’ more than one move
  - anticipate other player’s possible moves
- Example – the **minimax** approach:
  - minimise your possible loss
  - maximise your possible gain
  - (may need to prioritise one of these)

minimax

University of St Andrews

- We need a **metric** to decide how good a position is **from the viewpoint of a particular player**.
- Given the metric we evaluate possible moves:
  - if I make this move, what is my **opponent's best** move?
  - and if he/she makes **that** move what is **my** best move then?
  - and so on …
  - trade-off more information against compute time.
- Watch out for more on this in **tutorials**.

Comparison / analysis [1]

University of St Andrews

- You may want to try more than one AI
- Modular programming will allow you to choose and/or ‘plug-in’ different AI ‘modules’, e.g. different AI classes
- Develop code incrementally:
  - do not throw away simple AI code that works!
  - you can use it as a test if your newer AI
  - older code might be better then new code!

Comparison / analysis [2]

University of St Andrews

- In your report, document your different AI attempts and document how:
  - you **tested** them
  - **analysed** their performance
  - **compared** them against one another
- Does one AI *always* beat another one?
- Is it easy to work out which one is '*better*'?
- How often does the AI beat a human?

Possible extensions [1]

University of St Andrews

- Extensions for this practical are really about the sophistication of your AI modules, and your analysis of the results & comparison of your AI modules.
- Think about different constraints. How do the AI modules perform under time pressure? Does it make a difference as to which is the '*best*'?
- You can even compare against each others, i.e. from other teams:
  - 'blackbox' comparison – do not tell the other teams how your AI works!

Possible extensions [2]

University of St Andrews

- Neither of these are necessary to achieve a 20!
- Threaded server:
  - allow your implementation to play more than one game at once!
- A graphical interface:
  - start with a text-based interface so you can concentrate on the core game and AI module(s)
  - move onto a graphical user interface (GUI) once you have basic game play and an AI module that is playable.

Deliverables for P2

University of St Andrews

- Two parts:
  1. Source code.
  2. Report.
- Both of these are **required**.
- You should not use any third party libraries, code fragments etc. (only things found in the main JDK and your own code)