**CS1006 – P2 - backgammon**

Set by Saleem Bhatti, School of Computer Science, University of St Andrews
Tue 10 Feb 2015
*Originally created by John Thomson*
**Deadline: 2100 on 03 March 2015**

## Specification

(to be read in conjunction with the notes from the lecture)

Backgammon an ancient is a two-player game, whose provenance is more or less unknown, but may have originated in Persia around 5000 years ago. The goal of the game is for a player to move all of his or her counters round the board to his or her home quadrant. From there the player can 'bear off' the counters (removing them from the board). The first player to bear off all of their counters, **wins.**

An overview of the rules of Backgammon can be found at:

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

The goal of this practical is to produce an implementation of Backgammon that can be played by human players and/or AI players over a network. You should produce a user interface for Backgammon so a human player can play, or observe an AI player. You should start with a textual console-based interface, but you can move on to a graphical interface as an extension.

The implementation should allow a single player (human or AI) to observe the board and make moves. The opponent player should be assumed to running the same, or a compatible implementation. In this way, you can achieve a simulation of a game by running two instances of your software, one for each player, either on the same computer or different computers.

Your game should support:

- Human vs. human (Basic: humans at the same keyboard)
- Human vs. AI (Intermediate: Basic + one player can be networked and one player can be AI)
- AI vs. AI (Advanced: Intermediate + either player can be networked and either player be AI, plus more sophisticated AI)

However, it should be transparent to your software as to who your opponent is (a human player running your software, an AI player running your software, or a player running interoperable software from another team).

You should allow communication between instances of the software using ***sockets***. The expected format for communication is defined below. You must adhere to that description if you intend your software to be interoperable with other students' implementations.

## Artificial Intelligence (AI)

A major part of this project is to produce one (or more) AI modules which play the game. You should produce at least two different AIs, and compare them in your report. You will need to experiment by running your AIs against each other, recording the results and analysing the statistics. For an extension, not only analyse the performance of the AIs in general, but also in specific games, explaining why a particular AI does well.

## Networked player

You should design your code so that one or both of your players can be remote, from the network streams. If you model each player as an input stream of commands to be executed, then that stream of

commands could come from the keyboard (a local user) or from the network (a remote user). Also, if you get this right, either of the input streams could also be an AI module rather than a human player.

## Deliverables

As explained in the lecture, the deliverable for this project comprises two parts:

1. Your source code, including a README with any execution instructions.
2. Your report.

Your source code should be well structured and well commented.
Your report should give a detailed account of your solution. It should contain a justification of all of the design decisions you made and some sample out output of your program (e.g. as part of the testing). It should contain description of your implementation and testing. It should also include a comparison of your AI players as stated above.
The following describes three levels of deliverable, intended as a guide to the grade you can expect.

## Basic Deliverable

A game that allows human vs. human, both sitting and playing at the same keyboard. The game should make sure that a player's moves conform to the rules.

A report explaining design decisions made, the implementation and testing.

## Intermediate Deliverable

In addition to the Basic Deliverable, your game should be able to communicate and play against another player – human or AI – from other teams across the network. You should produce at least one sophisticated AI, and include more advanced analysis.

## Advanced Deliverable

For a mark of 20 in this practical, I do not necessarily expect more features (extensions), but a more sophisticated attempt at producing interesting and high-performance AIs, and a thorough statistical analysis of their performance. A good user interface is also important, but not necessarily a GUI – a well thought-out, text-based interface with good feedback to the user is a good result! You may attempt any ***relevant*** extension, e.g. the full rule set rather than the simplified one we have used, or a GUI to improve game play.

## Marking

Up to 10 for Basic Deliverable.
Up to 17 for Intermediate Deliverable.
Up to 20 for Advanced Deliverable.

Please note that the final mark you get will be based on the overall quality of your submission – attempting intermediate and/or advanced features is not a guarantee of a higher mark.

## Administrative information

This assignment is worth 25% of your overall coursework mark for CS1006.

Normal lateness penalties apply:
https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/lateness-penalties

You are reminded to follow Good Academic Practice:
https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/Good_Academic_Practice

Marking will follow the University's 20-point scale:
https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#/Mark_Descriptors

## Backgammon protocol description

Assuming that there is an initiator – a client – for a network connection and a responder – a server – for the connection, then the interaction would be as detailed below. Note that client and server are logical roles – it is possible that your program act as a client in one interaction and a server in another.

Your communication protocol will consist of text-based messages that allow a communication **session** to be established which will have the following phases:

- open a connection
  - as a server, wait for incoming connection requests and then respond
  - as a client, initiate connections to a remote server
- communicate information about moves being made between client and server once the connection is established
- close a connection

In the description below, protocol messages are shown in a `fixed font typeface`.

## Establishing a game

For establishing a session, the protocols :

Client: `hello`
Server: `hello`
Client: `newgame`
Server: `ready` if the server wants to play, or
Server: `reject` if the server cannot play.

After sending `reject`, server should close the connection. The `reject` might be sent because a session is already in progress.

For terminating a session:
A: `bye`
B: `bye`

Note here that either side, server or client, could terminate the session. So, if A is server, then B is client; and if A is client, then B is the server. If A or B send bye, they can then just close the connection unilaterally – they do not have to wait for the other side to 'agree'.

### The Board

The board slots should be numbered from 1-24. Server always plays from the lower numbers to the higher numbers.

### Expressing a turn

A turn should be expressed in the following way:

```
dice1-dice2:(StartPosition1 | EndPosition1), (StartPosition2 | EndPosition2);
```

Whitespace should be ignored. If a double is rolled, then two extra moves are added before the ';':

```
dice1-dice2:(StartPosition1 | EndPosition1), (StartPosition2 | EndPosition2),
(StartPosition3 | EndPosition3), (StartPosition4 | EndPosition4);
```

Two examples of a `{turn}`:

```
6-2:(1|7),(3|5);
3-3:(1|4),{1,4),(5|8),(10|13);
```

Hereafter, I use `{turn}` to represent this.

**Bearing off**
Bearing off can be represented by using the numbers '0' or '25'. For example:

```
2:3:(2|0),(3,0);
```

Similarly, pieces that have been captured should have a starting position of '0' or '25'.

**No move possible**
If no move can be made, then (-1,|-1) is used to represent this. For example:

```
3-1:(24|25),(-1|-1);
```

In this case, it could be that bearing off is in progress. The '1' roll is used to bear off one counter, but no move can be made with the '3' roll.

**Starting the game**
Client chooses who plays first. Flip a (logical) coin. If client plays first:

Client: `{turn}`

If client wants the server to play first:

Client: `pass`

**Continuing / playing the game**

Server: `{turn}`
Client: `{turn}`
Server: `{turn}`
Client: `{turn}`
Server: `{turn}`
Client: `{turn}`
(and so on …)

**Ending the game**

If either side quits before the end of the game:

Server: `bye` (close connection), or
Client: `bye`

(this might just be because the user no longer wants to play), or if either side detects the game is over:

Server: `you-win; bye` (close connection)
Client: `you-win; bye` (close connection)
Any other communication should be regarded as an error. You may extend this specification if you wish, but you must also provide a 'compatibility' mode that conforms to exactly this specification for

interoperability with other teams.

**Examples of protocol message exchanges during a session**

| Server | Client | Comment |
|---|---|---|
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |
| | `hello` | |
| `reject` | | Server does not want to play (perhaps it is already involved in another game), client closes connection. |
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |

| Server | Client | Comment |
|---|---|---|
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |
| | `hello` | |
| `hello` | | |
| | `New-game` | Client request a new game |
| `ready` | | |
| | `4-3:(24\|20),(19\|16);` | Client makes first move. |
| `1-1:(4\|5),(4\|5),(7\|8),(8\|9);` | | |
| | | … game continues … |
| `6-2:(23\|25),(-1\|-1);` | | |
| | `you-win; bye` | Client detects that server has won, declares the win, and closes the connection. |
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |

| Server | Client | Comment |
|---|---|---|
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |
| | `hello` | |
| `hello` | | |
| | `new-game` | Client request a new game |
| `ready` | | |
| | `4-3:(24\|20),(19\|16);` | Client makes first move. |
| `1-1:(4\|5),(4\|5),(7\|8),(8\|9);` | | |
| `bye` | | User at the server has decided that he no longer wishes to play so the server quits and closes the connection. |
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |

| Server | Client | Comment |
|---|---|---|
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |
| | `hello` | |
| `hello` | | |
| | `new-game` | Client request a new game |
| `ready` | | |
| | `pass` | Client askes server to make first move. |
| `1-1:(4\|5),(4\|5),(7\|8),(8\|9);` | | |
| | `4-3:(24\|20),(19\|16);` | |
| | | … game continues … |
| `you-win; bye` | | Server detects that client has won, declares the win, and closes the connection. |
| | | Server is running, waiting for incoming connection requests, or for local (keyboard) user to request a connection to a remote server. |