The *connect 4* game

1 Context

Connect 4 is a classical video game in which the players choose a color and then take turns dropping colored discs into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs.

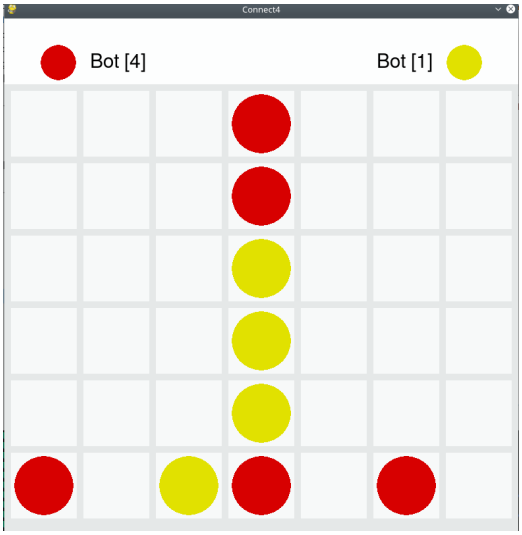

Figure 1: Game interface where the first player has red tokens and the second player has yellow tokens.

2 Setting up the game

The game is already installed in the Virtual Machine and on the lab computers.
To get the code template, go to your favorite coding folder and type:

```
cp -r /opt/duels/games/connect4 .
```

The files are then ready to use in the `connect4` folder.

3 Game description

The initial code is an infinite loop where the *feedback* variable is sent by the game. You have one second to compute your *input* and send it to the game. You can fight various AI levels (from 0 to infinity) depending on your confidence in your own AI. The code should be compiled according to the included `CMakeLists.txt` file.

3.1 Input rules

The game is played on a 6 rows × 7 columns grid. The feedback you get from the game is a list of all existing tokens in the game.

The input to send to the game is simply in which column you want to play the next move. From a programming point of view:

- `feedback.cells` is a `std::vector<Cell>` where `Cell` is a structure composed of:
  - `cell.row (int)`: the row of this token (0-5, 0 for bottom row)
  - `cell.column (int)`: the column of this token (0-6, 0 for leftmost column)
  - `cell.token (Token)`: which token it is (`Token::PLAYER1` or `Token::PLAYER2`)
- `input.column (int)`: which column you want to play (0-6, 0 for leftmost column)

Note that you can easily guess if you play Player1 or Player2 by checking the number of elements in `feedback.cells`.

3.2 Game rules

Sending an invalid move makes you lose the game:

- Playing outside of columns 0-6
- Playing an already full column

You or your opponent win the game if they can align 4 tokens horizontally, vertically or diagonally. As a consequence, games may also end with a draw if the grid is full but no player could align 4 tokens.

4 Expected work

4.1 A class for your AI

In order to design your own AI, you have to:

- Create a class (named e.g. `Connect4AI`) that manages your AI
- The class can have any member variable / function to design your AI
- The class should have at least the following method:

```
// compute next game input from current feedback from the game
Input compute(const Feedback &feedback);
```

where `Input` and `Feedback` are defined in:

- the `<duels/connect4/msg.h>` file
- the `duels::connect4` namespace

You should of course `include` the file and use either the explicit namespace, or the `using` keyword.

Besides, here are some useful tips:

- in difficulty 0, the game AI will not even try to win (it will randomly play a valid move) - you can test your code here
- to check a winning situation, it is only necessary to check if the last token is aligned with 3 other ones. Indeed if 4 previous tokens had already been aligned then the game would have ended before.
- besides trying to align 4 tokens, also try to prevent your opponent from doing so
- a useful algorithm here is called MinMax (with alpha-beta pruning for an optimized version)