Template-Based Connect 4 (Playfield)

Implement a template-based version of the Connect 4 game. Connect 4 builds on a playing field composed out of 7 columns each having 6 rows. When a player puts a stone into a column, gravity pulls the stone towards the lowest unoccupied row. The player who first has 4 stones in a row (horizontally, vertically, or diagonally) wins.

After each turn display the game field using simple ASCII graphics. Implement the game in such a way that players can be exchanged easily using templates. To facilitate this, implement a playfield class that is based on the following playfield skeleton. You may add to these classes, but you are not allowed to modify any element that has been provided by these templates.

```
struct playfield {
  // the size of the field
  const static int width=7;
  const static int height=6;

  // these elements are used to indicate whether a given position
  // in the playing field is taken by a given player
  const static char none=0;
  const static char player1=1;
  const static char player2=2;

  // the internal representation of the field
  char rep[playfield::width][playfield::height];

  // return the stone (none/player1/player2) at position(x, y)
  // 0 <= x <= width
  // 0 <= y <= height
  // stoneat(0, 0) ................ top left
  // stoneat(width-1, height-1) ... bottom right
  // if we insert a stone in a new game in column i,
  // it lands at (i, height-1)
  // implementation may be changed, interface not
  int stoneat(int x, int y) const { return rep[x][y]; }
};
```

In the next exercise, we will add a computer player and let different players compete against each other. To make it easier to implement the computer player, you may want to use the interface required for the computer player for the human player as well.

Template-Based Connect 4 (Computer Player)

Implement a computer player. The computer player does not have to be intelligent. At a minimum, however, the computer player should be able to identify whether he can win the game by placing a stone. Let your computer player compete against computer players from your colleagues. Think of at least one more optimization and add it as well.

In order to be able to let computer players from different students, use the following templates to implement your player:

```
template<typename F>
struct player {
  // returns the column where the player decides to insert his
  // stone
  // F is the playfield which may be any playfield implementing
  // the stoneat method, if you expect a different class because
  // you need methods to verify whether the opponent can win,
  // copy the field into the class that you expect.
  int play(const F &field);
};
```

Make sure that your player class is self-contained and does not require functions from your playfield, etc. If you need functions shared with the playfield, you can instead put them into an extra class that you can be shipped with the player.

Collect at least 2 more computer players from your colleagues and let them play against each other.