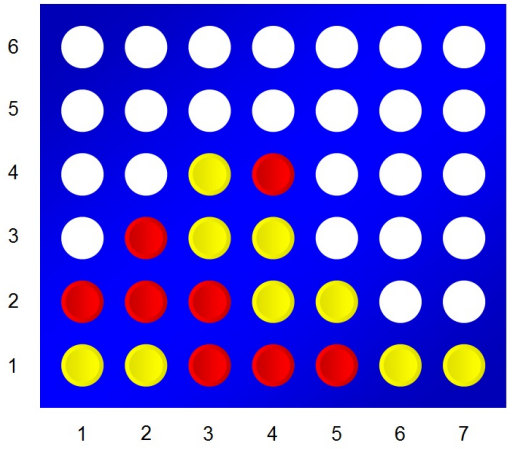
Cairo University
Faculty of Computers and Artificial Intelligence
Computer Science Department

Programming-1 CS112 **Application: Connect game** Dr. Amin Allam

1 Connect four game

Connect four game is a two player game. The game contains a 6x7 board where the height of this board is 6 rows and the width is 7 columns. Rows are numbered from 1 to 6 while columns are numbered from 1 to 7. Player one starts by selecting a column and inserting a yellow tile into the top of that column. The yellow tile will move down until it reaches the bottom of the board, which is row 1. Player two proceeds by selecting a column and inserting a red tile into the top of that column. The red tile will move down until it reaches row 1 or until it reaches above existing tiles.



That is, if player two selects the same column chosen by player one, his tile will go to row 2, otherwise it will reach row 1. Therefore, each player selects a column, but the row is determined automatically based on the number of existing tiles in that column. Players should alternate turns until one of them is able to construct 4 adjacent tiles horizontally, vertically, or diagonally. The above figure is a winning situation for player one because there exist 4 yellow diagonally adjacent tiles. If the board becomes full without having any winner, it is considered a draw situation.

Our target is to write a C++ program for a generalized connect four game. The generalized game follows the same rules of the original game, except that the height `n`, width `m`, and the number of required adjacent tiles `r` will be determined by the user before starting playing any game. That is, the original connect four game will be a specific instance of our game where `n`=6, `m`=7, and `r`=4.

2 Connect game version 1.0

We will implement a general connect game. Since we will work on a non-colored console, we will use `'x'` and `'o'` characters instead of yellow and red tiles. We call the players player x and player o instead of player one and player two for simplification. Since C++ arrays are 0-indexed, we will internally number rows from 0 to `n`-1 and columns from 0 to `m`-1. However, since 1-indexed numbers are more user friendly, we will ask the players to select a column number from 1 to `m` and we will decrease it by one before we proceed. We will keep numbering rows from bottom to top as in the figure. This ordering will simplify the game logic when we automatically determine the row number when a tile is inserted, but it may slightly complicate printing the board. Our initial version will be as follows:

```
#define EMPTY_CHAR '-'
#define PLAY1_CHAR 'x'
#define PLAY2_CHAR 'o'
```

```
struct ConnectGame
{
   char** board;     // n x m cells, each cell value is 'x', 'o', or '-'
   int n, m, r;      // Number of rows, columns, and required adjacent tiles
};
```

```
void InitializeGame(ConnectGame* g, int cn, int cm, int cr)
{
   int i, j;
   g->n=cn; g->m=cm; g->r=cr;

   g->board=new char*[g->n];
   for(i=0;i<g->n;i++) g->board[i]=new char[g->m];

   for(i=0;i<g->n;i++) for(j=0;j<g->m;j++)
      g->board[i][j]=EMPTY_CHAR;    // All cells are initiallly empty
}
```

```
void DestroyGame(ConnectGame* g)
{
   int i;   // De-allocate all arrays
   for(i=0;i<g->n;i++) delete[] g->board[i];
   delete[] g->board;
}
```

```
bool IsFull(ConnectGame* g)
{
   int i, j;

   for(i=0;i<g->n;i++)
   {
      for(j=0;j<g->m;j++)
      {  // If there exist at least one empty cell, it is not full yet
         if(g->board[i][j]=='-') return false;
      }
   }
   return true;  // No empty cell is found
}
```

```
1  char IsWinning(ConnectGame* g)
2  {
3     // Return the character of the winning player or '-' of no winner
4     int i, j, k;
5     char** b=g->board;
6     int n=g->n, m=g->m, r=g->r;
7
8     for(i=0;i<n;i++)
9     {
10       for(j=0;j<m;j++)
11       {
12          // Check possible winning situations starting from b[i][j]
13          char c=b[i][j];
14          if(c==EMPTY_CHAR) continue;
15
16          if(j+r<=m)   // Start from b[i][j] and go right
17          {
18             for(k=1;k<r;k++) if(b[i][j+k]!=c) break;
19             if(k==r) return c;
20          }
21          if(i+r<=n)   // Start from b[i][j] and go top
22          {
23             for(k=1;k<r;k++) if(b[i+k][j]!=c) break;
24             if(k==r) return c;
25          }
26          if(i+r<=n && j+r<=m) // Start from b[i][j] and go top right
27          {
28             for(k=1;k<r;k++) if(b[i+k][j+k]!=c) break;
29             if(k==r) return c;
30          }
31          if(i-r+1>=0 && j+r<=m) // Start from b[i][j] and go bottom right
32          {
33             for(k=1;k<r;k++) if(b[i-k][j+k]!=c) break;
34             if(k==r) return c;
35          }
36       }
37    }
38    return EMPTY_CHAR;
39 }
```

```
bool PlayTurn(ConnectGame* g, char play, int col)
{
   // Return false if this column is full , otherwise modify the correct cell
   int i;
   if(col<0 || col>=g->m) return false;

   for(i=0;i<g->n;i++)
   {
      if(g->board[i][col]=='-')
      {
         // Here board[i][col] is the most bottom empty cell in that column
         g->board[i][col]=play;
         return true;
      }
   }
   return false; // This column is full
}
```

```
void PrintGame(ConnectGame* g)
{
   int i, j;

   cout<<"  "; for(j=1;j<=g->m;j++) cout<<(j%10); cout<<endl;
   cout<<" +"; for(j=0;j<g->m;j++) cout<<"="; cout<<endl;

   for(i=g->n-1;i>=0;i--)
   {
      cout<<((i+1)%10)<<"|";
      for(j=0;j<g->m;j++) cout<<g->board[i][j];
      cout<<endl;
   }
   cout<<endl;
}
```

```
void StartGame()
{
   int cn, cm, cr;
   cout<<"Enter NumRows, NumCols, NumRequiredTiles:"<<endl;
   cin>>cn>>cm>>cr;

   ConnectGame game;
   InitializeGame(&game, cn, cm, cr);

   char cur_player=PLAY1_CHAR;   // The player who has the turn

   while(true)
   {
      PrintGame(&game);

      while(true) // Exit loop only when the player selects a valid non-full column
      {
         cout<<"Enter column (1 to "<<game.m<<")"<<endl;
         int cur_col; cin>>cur_col; cur_col--;   // Make it 0-indexed
         if(PlayTurn(&game, cur_player, cur_col)) break;
      }

      if(IsWinning(&game)==cur_player)
      {
         cout<<"Player "<<cur_player<<" wins!\n"<<endl;
         break;
      }

      if(IsFull(&game)) {cout<<"Game draw!\n"<<endl; break;}

      if(cur_player==PLAY1_CHAR) cur_player=PLAY2_CHAR;
      else cur_player=PLAY1_CHAR;     // Give the turn to the other player
   }

   PrintGame(&game);
   DestroyGame(&game);
}
```

```
int main()
{
   StartGame();
   return 0;
}
```

3 Connect game version 2.0

The following version contains several improvements:
• We save the total number of inserted tiles, so that the implementation of `IsFull()` becomes simpler and more efficient.
• We save the number of inserted tiles in each column, so that the implementation of `PlayTurn()` becomes simpler and more efficient.
• We introduce the `GetCell()` function, which acts as a functional sentinel which simplifies writing the `IsWinning()` but makes it slower.
The modified functions in this version will be as follows:

```
#define BORDER_CHAR '#'
```

```
struct ConnectGame
{
    char** board;     // A board will be logically surrounded by '#' characters
    int n, m, r;

    int num;          // The total number of tiles inserted in the board
    int* num_col;     // num_col[i]= number of tiles inserted in column i
};   // num_col[i] is also the row index of the next tile to be inserted in column i
```

```
void InitializeGame(ConnectGame* g, int cn, int cm, int cr)
{
    int i, j;
    g->n=cn; g->m=cm; g->r=cr;

    g->board=new char*[g->n];
    for(i=0;i<g->n;i++) g->board[i]=new char[g->m];

    for(i=0;i<g->n;i++) for(j=0;j<g->m;j++)
        g->board[i][j]=EMPTY_CHAR;

    g->num=0;     // No tiles inserted yet
    g->num_col=new int[g->m];
    for(i=0;i<g->m;i++) g->num_col[i]=0;     // No tiles inserted yet
}
```

```
void DestroyGame(ConnectGame* g)
{
    for(int i=0;i<g->n;i++) delete[] g->board[i];
    delete[] g->board;
    delete[] g->num_col;
}
```

```
bool IsFull(ConnectGame* g)
{
   return g->num==g->n*g->m;
}
```

```
bool PlayTurn(ConnectGame* g, char play, int col)
{
   if(col<0 || col>=g->m || g->num_col[col]==g->n)
      return false;
   g->board[g->num_col[col]][col]=play;
   g->num_col[col]++;   g->num++;    // A tile is inserted
   return true;
}
```

```
char GetCell(ConnectGame* g, int i, int j)
{  // Return board[i][j] only if i and j are valid indexes
   if(i<0 || j<0 || i>=g->n || j>=g->m) return BORDER_CHAR;
   return g->board[i][j];
}
```

```
char IsWinning(ConnectGame* g)
{
   int i, j, k, r=g->r;

   for(i=0;i<g->n;i++)
   {
      for(j=0;j<g->m;j++)
      {
         char c=g->board[i][j];   if(c==EMPTY_CHAR) continue;

         for(k=1;k<r;k++) if(GetCell(g, i, j+k)!=c) break;
         if(k==r) return c;
         for(k=1;k<r;k++) if(GetCell(g, i+k, j)!=c) break;
         if(k==r) return c;
         for(k=1;k<r;k++) if(GetCell(g, i+k, j+k)!=c) break;
         if(k==r) return c;
         for(k=1;k<r;k++) if(GetCell(g, i-k, j+k)!=c) break;
         if(k==r) return c;
      }
   }
   return EMPTY_CHAR;
}
```

4 Connect game version 3.0

In the following version, we replace the functional sentinel `GetCell()` by actual sentinel, which is a border of `'#'` characters around the matrix. The modified functions in this version will be as follows:

```
void InitializeGame(ConnectGame* g, int cn, int cm, int cr)
{
   int i, j;
   g->n=cn; g->m=cm; g->r=cr;

   g->board=new char*[g->n+2];
   for(i=0;i<g->n+2;i++) g->board[i]=new char[g->m+2];

   for(i=0;i<g->n+2;i++) for(j=0;j<g->m+2;j++)
      g->board[i][j]=BORDER_CHAR;
   for(i=1;i<=g->n;i++) for(j=1;j<=g->m;j++)
      g->board[i][j]=EMPTY_CHAR;

   g->num=0;
   g->num_col=new int[g->m];
   for(i=0;i<g->m;i++) g->num_col[i]=0;
}
```

```
void DestroyGame(ConnectGame* g)
{
   int i;
   for(i=0;i<g->n+2;i++) delete[] g->board[i];
   delete[] g->board;
   delete[] g->num_col;
}
```

```
bool PlayTurn(ConnectGame* g, char play, int col)
{
   if(col<0 || col>=g->m || g->num_col[col]==g->n)
      return false;
   g->board[g->num_col[col]+1][col+1]=play;
   g->num_col[col]++;
   g->num++;
   return true;
}
```

```
char IsWinning(ConnectGame* g)
{
   int i, j, k;
   char** b=g->board;
   int r=g->r;

   for(i=1;i<=g->n;i++)
   {
      for(j=1;j<=g->m;j++)
      {
         char c=b[i][j];
         if(c==EMPTY_CHAR) continue;

         for(k=1;k<r;k++) if(b[i][j+k]!=c) break;
         if(k==r) return c;

         for(k=1;k<r;k++) if(b[i+k][j]!=c) break;
         if(k==r) return c;

         for(k=1;k<r;k++) if(b[i+k][j+k]!=c) break;
         if(k==r) return c;

         for(k=1;k<r;k++) if(b[i-k][j+k]!=c) break;
         if(k==r) return c;
      }
   }
   return EMPTY_CHAR;
}
```

```
void PrintGame(ConnectGame* g)
{
   int i, j;

   cout<<"  "; for(j=1;j<=g->m;j++) cout<<(j%10); cout<<endl;

   for(i=g->n+1;i>=0;i--)
   {
      if(i>=1 && i<=g->n) cout<<(i%10); else cout<<" ";
      for(j=0;j<g->m+2;j++) cout<<g->board[i][j];
      cout<<endl;
   }
   cout<<endl;
}
```

5 Connect game version 4.0

In the following version, we introduce helper arrays to simplify writing `IsWinning()` and avoid duplicating code:

```
1  char IsWinning(ConnectGame* g)
2  {
3     int i, j, k, p;
4
5     int di[]={0,1,1,-1};
6     int dj[]={1,0,1,1};
7
8     for(i=1;i<=g->n;i++)
9     {
10       for(j=1;j<=g->m;j++)
11       {
12          char c=g->board[i][j];
13          if(c==EMPTY_CHAR) continue;
14
15          for(p=0;p<4;p++)
16          {
17             // Check adjacent cells starting from board[i][j] and going in the
18             // direction such that i increases by di[p] and j increases by dj[p]
19
20             int ci=i, cj=j;
21             for(k=1;k<g->r;k++)
22             {
23                ci+=di[p]; cj+=dj[p];
24                if(g->board[ci][cj]!=c) break;
25             }
26             if(k==g->r) return c;
27          }
28       }
29    }
30
31    return EMPTY_CHAR;
32 }
```

6 Connect game version 5.0

In the following version, we modify `IsWinning()` to make it much more efficient. The modified functions in this version will be as follows:

```
struct ConnectGame
{
   char** board;
   int n, m, r;

   int num;
   int* num_col;
   int last_play_col;     // Column index of the last inserted tile
};
```

```
void InitializeGame(ConnectGame* g, int cn, int cm, int cr)
{
   int i,j;
   g->n=cn; g->m=cm; g->r=cr;

   g->board=new char*[g->n+2];
   for(i=0;i<g->n+2;i++) g->board[i]=new char[g->m+2];

   for(i=0;i<g->n+2;i++) for(j=0;j<g->m+2;j++)
      g->board[i][j]=BORDER_CHAR;
   for(i=1;i<=g->n;i++) for(j=1;j<=g->m;j++)
      g->board[i][j]=EMPTY_CHAR;

   g->num=0;
   g->num_col=new int[g->m];
   for(i=0;i<g->m;i++) g->num_col[i]=0;

   g->last_play_col=-1;
}
```

```
bool PlayTurn(ConnectGame* g, char play, int col)
{
   if(col<0 || col>=g->m || g->num_col[col]==g->n)
      return false;
   g->board[g->num_col[col]+1][col+1]=play;
   g->num_col[col]++;   g->num++;
   g->last_play_col=col;
   return true;
}
```

```
1  char IsWinning(ConnectGame* g)
2  {
3     int k,p,q;
4     if(g->last_play_col<0) return EMPTY_CHAR;
5
6     int di[]={0,1,1,-1};
7     int dj[]={1,0,1,1};
8
9     int sti=g->num_col[g->last_play_col];
10    int stj=g->last_play_col+1;
11    char c=g->board[sti][stj];
12
13    for(p=0;p<4;p++)
14    {
15       int cnt=1;
16
17       // Count number of adjacent cells starting from the cell of the last
18       // inserted tile and going in the direction such that i increases
19       // by di[p] and j increases by dj[p] and also in the inverse of this
20       // direction where i decreases by di[p] and j decreases by dj[p]
21
22       for(q=1;q>=-1;q-=2)
23       {
24          int ci=sti, cj=stj;
25          for(k=1;k<g->r;k++)
26          {
27             ci+=q*di[p]; cj+=q*dj[p];
28             if(g->board[ci][cj]!=c) break;
29             cnt++;
30          }
31       }
32
33       if(cnt>=g->r) return c;
34    }
35
36    return EMPTY_CHAR;
37 }
```