



27 MEI, 2016

KUNSTMATIGE INTELLIGENTIE TOEGEPAST OP VIER OP EEN RIJ

IN HET KADER VAN HET VAK WETENSCHAPPELIJKE TOEPASSINGEN

LAURENS DE COCK
ODISEE



1 Het gieten van vier op een rij in een programmeertaal

1.1 Het basisspel vier op een rij

Allereerst dient de logica van het spel vier op een rij in programmeerregels gegoten te worden.

Een basisklasse zal voorzien worden om dit te verwerken.

In de essentie moet deze logica enkel de absolute basis van het spel bevatten: Het doen van een zet, het definiëren van een spelbord en het controleren van de win –en gelijkspeltoestanden.

1.2 Het definiëren van spelers

Wanneer er verder gedacht wordt over een normaal spelverloop, zal er bovenop deze basislogica de ‘flow’ van het programma benodigd zijn.

In het einddoel van het programma zullen er verschillende soorten spelers zijn: Menselijke spelers en computerspelers (al dan niet intelligent). Er dient een logica voorzien te worden die wacht op de zet van een speler, of een zet vraagt aan de computer.

1.3 De grafische laag

Verder dient de status van het bord te worden getekend na elke zet. Een grafische laag is dus verantwoordelijk voor het presenteren van het spelbord. Deze laag tekent zonder enige logica een meegeleverd spelbord.

2 Uitwerking van de spellogica

2.1 Spellogica

De spellogica bevat zich in de klasse Game. Deze klasse definieert allereerst het spelbord in een 2-dimensionale array op de manier weergegeven in Figuur 2.1.1.

```
public const int NROWS = 6;  
public const int NCOLS = 7;  
  
public int[,] Board { get; set; }
```

Figuur 2.1.1. Spelbord

De beurt wordt bijgehouden met behulp van een Enum (Figuur 2.1.2.). Er zijn in het spel vier op een rij slechts 2 mogelijke waarden voor een beurt: Het is aan speler 1 of aan speler 2.

In de klasse Game wordt de Turn bijgehouden.

Op het spelbord wordt een leeg vak voorgesteld als -1, en een ingevuld vak door de overeenkomstige integer van de speler (1 of 2 respectievelijk voor speler 1, speler 2).

```
public enum Turn { P1 = 1, P2 = 2 }
```

Figuur 2.1.2. De enum Turn

```
public Turn Turn { get; set; }
```

Figuur 2.1.3. Implementatie van de enum Turn

Verder zijn er 2 belangrijke methodes aanwezig: De methode **MakeMove(int col)** en de methode **CheckWinner()**.

De methode `MakeMove(int col)` zet een schijf in de kolom die meegegeven wordt voor de speler aan wiens beurt het is.

Deze methode is neutraal en zal in een verder stadium door ofwel een menselijke speler ofwel door een al dan niet intelligente computer worden aangeroepen.

Als bewuste designkeuze wordt het wisselen van beurten en het kijken naar een volle kolom niet in deze basislogica weergegeven. Het idee achter deze keuze is dat er hier variabel mee omgegaan kan worden door een bovenliggende class die alleszins geïmplementeerd zal dienen te worden om het spel te kunnen verwezenlijken. Dit zal in dit project verwezenlijkt worden door de `GameFlow` klasse.

Er zijn echter wel publieke hulpfuncties voorzien om deze logica te implementeren. Enkele hulpfuncties zijn bijvoorbeeld: **`ResetGame()`** – reset het spelbord naar initiële staat, **`bool ColHasSpace(int col)`** – retournt true als een kolom niet vol is, **`bool DetectDraw()`** – retournt true als het spel in een gelijkstand terechtkomt en de methode **`bool DetectWinner()`** – retournt true indien er 4 op een rij op het spelbord bevindt (De huidige Turn is dan de winnaar).

Verder is er een essentiële methode voor de flow van het spel: de methode **`ShiftTurns()`**, die de beurt instelt op de volgende speler.

2.2 Flowlogica

Een bovenliggende laag die het spelverloop controleert bevindt zich in de klasse `GameFlow`.

De publieke properties van de klasse zien er als volgt uit (Figuur 2.2.1).

```
public class GameFlow {  
    public Game Game { get; }  
  
    public PlayerType P1;  
    public PlayerType P2;  
  
    public event Action<Turn> ThereIsAWinner;  
    public event Action Draw;  
}
```

Figuur 2.2.1. Publieke properties van de klasse `GameFlow`

Deze klasse bevat een `Game`-object, en bovenliggende logica.

Er worden events opgegooid indien een speler wint of het spel zich in gelijkstand bevindt.

Verder wordt er gedefinieerd wat voor spelers P1 en P2 zullen zijn. Dit met behulp van een Enum `PlayerType` (Figuur 2.2.2).

```
public enum PlayerType { Player, CPURandom, CPUSmart }
```

Figuur 2.2.2. De Enum `PlayerType`

Een speler kan hier menselijk zijn, een computer die willekeurige zetten doet, of een computer die slimme zetten doet.

Indien het aan een menselijke speler is, dient de methode **`PlayerMove(int col)`** aangeroepen te worden om de flow van het spel naar de volgende zet te doen gaan. De methode verifieert dat de huidige beurt door een menselijke speler dient te worden gemaakt, doet dan de zet, controleert op winconditie of

gelijkspel en stelt de beurt op de volgende speler in.
De volledige methode is te zien in figuur 2.2.3.

```
public void PlayerMove(int col) {
    if( (Game.Turn == Turn.P1 && P1 == PlayerType.Player) || (Game.Turn == Turn.P2 && P2 == PlayerType.Player) ) {
        Game.MakeMove(col);

        if (Game.DetectWinner()) {
            if (ThereIsAWinner != null) ThereIsAWinner(Game.Turn);
        }
        else if (Game.DetectDraw()) {
            if (Draw != null)
                Draw();
            return;
        }
        else Game.ShiftTurns();
    }
}
```

Figuur 2.2.3. De methode PlayerMove

Gelijk wanneer kan de functie **Continue()** worden aangeroepen, waarbij de computer een zet zal doen indien het zijn beurt is.

In deze klasse GameFlow wordt bepaald wat voor zet de computer zal doen. Indien hij een Random computer is wordt hier een random kolom berekend, en indien hij een intelligente computer is, wordt het spelbord doorgegeven aan de klasse **MasterMindMoveMaker**, waarin een methode aangeroepen wordt die een intelligente zet zal berekenen.

Een overzicht van de functie **Continue()** is te zien in figuur 2.2.4.

```
public void Continue() {
    if( (Game.Turn == Turn.P1 && P1 == PlayerType.CPURandom) || (Game.Turn == Turn.P2 && P2 == PlayerType.CPURandom) ) {
        Game.MakeMove(random.Next(0, Game.NCOLS));

        if (Game.DetectWinner()) {
            if (ThereIsAWinner != null) ThereIsAWinner(Game.Turn);
            return;
        }
        else if (Game.DetectDraw()) {
            if (Draw != null) Draw();
            return;
        }
        else Game.ShiftTurns();
    }

    else if( (Game.Turn == Turn.P1 && P1 == PlayerType.CPUSmart) || (Game.Turn == Turn.P2 && P2 == PlayerType.CPUSmart)) {
        MasterMindMoveMaker m = new MasterMindMoveMaker();
        Game.MakeMove(m.GetSmartMove(Game));

        if (Game.DetectWinner()) {
            if (ThereIsAWinner != null) ThereIsAWinner(Game.Turn);
            return;
        }
        else if (Game.DetectDraw()) {
            if (Draw != null)
                Draw();
            return;
        }
        else Game.ShiftTurns();
    }
}
```

Figuur 2.2.4. De methode Continue()

3 Een intelligente zet

3.1 De logica en beginsituatie

De klasse **MasterMindMoveMaker** heeft enkel tot doel om een intelligente zet te berekenen vanuit de staat van een meegegeven spelbord.

```
public int GetSmartMove(Game game) {
    Dictionary<int, List<int>> MoveAndItsEvaluation = new Dictionary<int, List<int>>(); // Attempted move (column) and its value result

    for (int n1 = 0; n1 < Game.NCOLS; n1++) {
        MoveAndItsEvaluation.Add(n1, new List<int>());

        // Reset state to game.Board
        Game attempt = new Game();
        Array.Copy(game.Board, attempt.Board, game.Board.Length);
        attempt.Turn = game.Turn;

        int cpuValue = 100; // Base value for CPU

        // A full column would be the worst move
        if (!attempt.ColHasSpace(n1)) {
            cpuValue = -1000000;
        }

        attempt.MakeMove(n1);

        // Evaluate it
        if (attempt.DetectWinner()) cpuValue += 100000; // We will win with this move
        cpuValue += 50 * Detect3(attempt);
        cpuValue += 15 * Detect2(attempt);

        attempt.ShiftTurns();

        for(int n2 = 0; n2 < Game.NCOLS; n2++) {
            // From this situation, calculate the best move player can make, on top of the attempt.MakeMove
            // Reset state to attempt.Board
            Game playerAttempt = new Game();
            Array.Copy(attempt.Board, playerAttempt.Board, attempt.Board.Length);
            playerAttempt.Turn = attempt.Turn;

            int playerValue = 100; // Same base value for Player
            playerAttempt.MakeMove(n2);

            // Evaluate it
            if (playerAttempt.DetectWinner()) playerValue += 5000; // He will win this one, worth less than CPU win because it's his turn
            playerValue += 50 * Detect3(playerAttempt);
            playerValue += 15 * Detect2(playerAttempt);

            // Compare cpu value to player value and store it in Dict
            int realValue = cpuValue - playerValue;
            MoveAndItsEvaluation[n1].Add(realValue);
        }
    }

    int[] sums = new int[Game.NCOLS];

    // Make sum of all evaluations for moves - the highest result is the best move
    foreach(KeyValuePair<int, List<int>> kvp in MoveAndItsEvaluation) {
        for(int i = 0 ; i < kvp.Value.Count; i++) {
            sums[kvp.Key] += kvp.Value.ElementAt(i);
        }
    }

    int biggest = sums.Max();

    // Get all equally good moves
    List<int> bestMoves = new List<int>();
    for(int i=0; i < sums.Length; i++) {
        if(sums[i] == biggest) bestMoves.Add(i);
    }

    int aBestMove = bestMoves[new Random().Next(0, bestMoves.Count)]; // One random move of equally good moves, to have less stale AI
    return aBestMove;
}
```

Figuur 3.1.1. De methode GetSmartMove

De belangrijkste publieke methode is dan ook de methode **int GetSmartMove(Game game)** (figuur 3.1.1), waar een game (met dus spelbord als property) wordt aan meegegeven. De methode geeft een integer mee die overeenkomt met de kolom waarin een gedane zet de best berekende zet zal zijn.

3.2 Waarde geven aan een zet

Een eerste stap om te bepalen of een zet een goede is, is onderkennen dat er een **waarde** aan een zet zal moeten worden toegekend.

Er wordt geredeneerd volgens 4 verschillende statussen van een schijf.

- De schijf ligt afgezonderd en alleen: Dit is de slechtste situatie.
- De schijf heeft 2 op een rij gevormd, met ruimte om een verdere combinatie te maken: Deze situatie is waardevoller.
- De schijf heeft 3 op een rij gevormd, met ruimte om 4 op een rij te maken: Deze situatie is waardevoller dan 2 op een rij.
- De schijf heeft 4 op een rij gevormd: Dit is de beste situatie, we winnen het spel.

Allereerst worden 2 hulpfuncties geschreven die tellen hoeveel keer een 'open 2 op een rij' en een 'open 3 op een rij' is gevormd: **int Detect2(Game game)** en **int Detect3(Game game)**. Aan de methodes wordt een spelbord meegegeven, waarin gecontroleerd wordt hoeveel keer 2 resp. 3 op een rij is gevormd. Het aantal keer dat de vereisten aanwezig zijn wordt gereturned.

Om nu een waarde aan een zet toe te kennen, wordt een spelbord eerst gekopieerd. Er wordt geen zet gedaan in het originele spelbord, maar in het gekopieerde. Dit spelbord kan dan meegegeven worden aan de hulpfuncties zodat de zet geëvalueerd wordt.

De waarde van de zet wordt nu als volgt bepaald:

- Afgezonderde schijf: 0 waarde
- 2 op een rij schijf: +15 voor elke keer 2 op een rij
- 3 op een rij schijf: +50 voor elke keer 3 op een rij
- 4 op een rij schijf: +100 000

In de spellogica wordt het toegestaan om een zet in een volle kolom te doen. Dit is in de realiteit ook mogelijk door een menselijke speler en wordt met die filosofie toegestaan. Een zet doen in een volle kolom is echter nog slechter dan een winnende zet doen, hieraan wordt de score -1 000 000 gegeven. Zo 'denkt' de computer na over een zet doen in een volle kolom, maar vermijdt het.

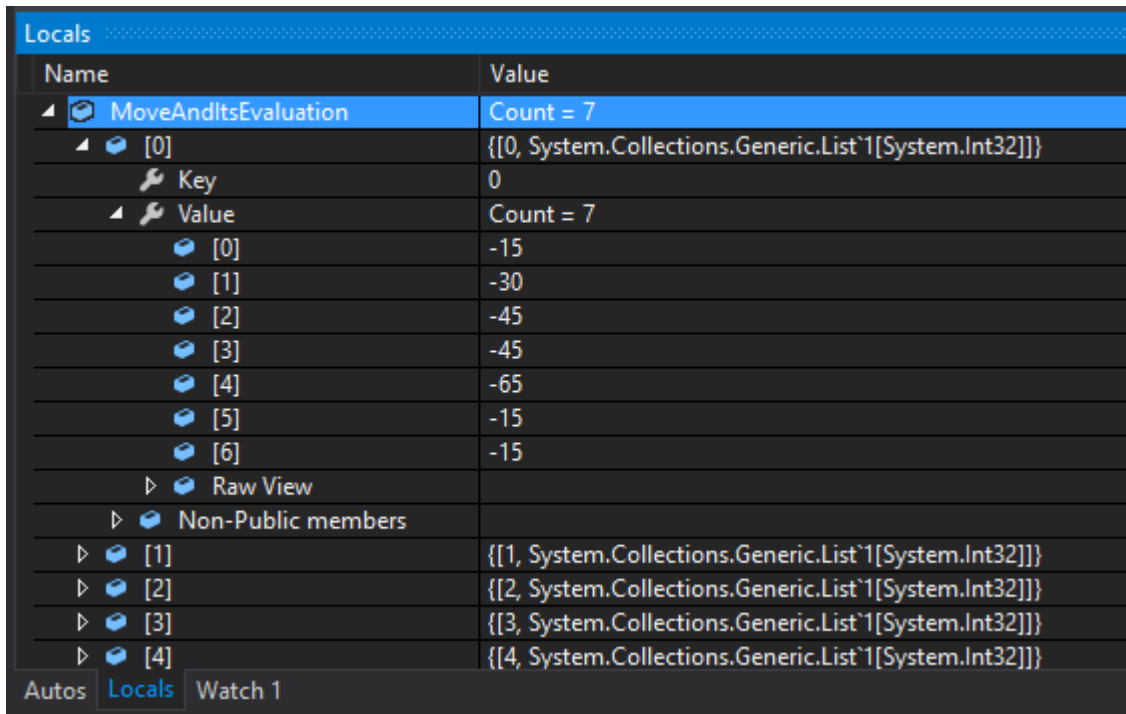
3.3 Verder denken

Nu hebben we een waardescore **cpuValue** voor de zet die een computer heeft gedaan en het overeenkomstige spelbord. Op een gelijkaardige manier wordt nu het spelbord weer gekopieerd, waarbij voor alle mogelijke zetten die kunnen volgen op gelijkaardige manier een score wordt berekend. Dit is de **playerValue**. De enige afwijking op de berekening van de score, is de score voor een winnende zet. Hier wordt een lagere waarde aan toegekend (+5000). Dit is voor een speler de beste zet. De 2 scores worden van elkaar afgetrokken, zodat de waarde van een zet van de computer een betekenisvolle waarde krijgt. Echter, wanneer de computer een winnende zet doet is dit meer waard dan dat een zet die daarop volgt een winnende zet is. Bij het maken van een winnende zet is het spel namelijk voorbij.

3.4 De waarden verwerken

Voor elke zet die de computer kan doen wordt een waarde bepaald en voor elke zet die daarop kan volgen wordt ook een waarde bepaald.

Om van dit geheel een zinnig resultaat te bekomen is de dictionary **MoveAndItsEvaluation** er: Dictionary<int, List<int>>. De eerste integer beschrijft de gesimuleerde zetten van de computer (rij 0 t.e.m. 6), en de List<int> is een lijst van alle waarden van de daarop volgende zetten die een speler zou kunnen doen. Een voorbeeld hiervan is te zien in figuur 3.4.1.



Name	Value
MoveAndItsEvaluation	Count = 7
[0]	{[0, System.Collections.Generic.List`1[System.Int32]]}
Key	0
Value	Count = 7
[0]	-15
[1]	-30
[2]	-45
[3]	-45
[4]	-65
[5]	-15
[6]	-15
Raw View	
Non-Public members	
[1]	{[1, System.Collections.Generic.List`1[System.Int32]]}
[2]	{[2, System.Collections.Generic.List`1[System.Int32]]}
[3]	{[3, System.Collections.Generic.List`1[System.Int32]]}
[4]	{[4, System.Collections.Generic.List`1[System.Int32]]}

Figuur 3.4.1.

In figuur 3.4.1 evalueert de computer een zet in kolom 0. De waarde van zijn zet in kolom 0 min de waarde van de 7 mogelijke zetten van een speler die erop volgen, zijn te zien in de bijhorende List. Hier wordt al duidelijk dat de zet in kolom 0 niet voordelig is voor de computer. *Elke mogelijke zet die de speler hier zou kunnen doen is voordeliger voor de speler dan voor de computer* (volgens de waardebeoordelingen die eerder gesteld werden).

Op deze manier is een kwantificering van elke zet bekomen in de MoveAndItsEvaluation dictionary.

Omdat er niet geweten is welke zet een speler zal doen, wordt er rekening gehouden met elke mogelijke zet die een speler *kan* doen. Als alle waarden voor deze zetten worden opgeteld en vergeleken met alle waarden bekomen uit alle zetten van de computer, is de meest waardevolle zet bekend. Dit gebeurt in het laatste stuk van de methode, zoals te zien is in figuur 3.4.2.

```
int[] sums = new int[Game.NCOLS];

// Make sum of all evaluations for moves - the highest result is the best move
foreach(KeyValuePair<int, List<int>> kvp in MoveAndItsEvaluation) {
    for(int i = 0 ; i < kvp.Value.Count; i++) {
        sums[kvp.Key] += kvp.Value.ElementAt(i);
    }
}

int biggest = sums.Max();
```

Figuur 3.4.2. Bepalen van de zet met de hoogste waarde

Het is zeer goed mogelijk dat meerdere zetten dezelfde waarde hebben. Om de computerzet minder voorspelbaar en meer menselijk te laten lijken, wordt random gekozen uit een zet met dezelfde waarde (Figuur 3.4.3).

```
// Get all equally good moves
List<int> bestMoves = new List<int>();
for(int i=0; i < sums.Length; i++) {
    if(sums[i] == biggest) bestMoves.Add(i);
}

int aBestMove = bestMoves[new Random().Next(0, bestMoves.Count)]; // One random move of equally good moves, to have less stale AI
return aBestMove;
```

Figuur 3.4.3 Random zet uit de beste zetten kiezen

Een van de meest waardevolle zetten is bepaald en returned aan de FlowLogica. Deze zet wordt dus gekozen door de slimme computer.

3.5 Bespreking van de intelligentie

De computer denkt op een vrij menselijke manier. Een alternatieve methode is om de computer alle zetten te doen berekenen en zo het beste ‘pad’ naar de overwinning te kiezen.

In dit project is echter geopteerd voor een meer natuurlijke, ‘menselijke’ manier van redeneren. Een zet wordt overwogen, en hiervoor wordt een waardeoordeel bepaald. Niet alleen voor de zet zelf, maar ook voor mogelijke opvolgingen van deze zet. De opvolgingen worden ook geschat naar waarde. Het is vanuit deze waardeoordelen dat de beste zet wordt bepaald.

Doordat de computer op deze manier geprogrammeerd is, heeft hij een menselijk karakter. Hij doet natuurlijke zetten, maakt soms fouten, maar doet steeds de duidelijk goede zetten en blokkeert ook goede zetten van de tegenstander af. De resultaten hiervan zijn zeer bevredigend en het deze heuristiek resulteert in een menselijke, uitdagende doch imperfecte kunstmatige intelligentie.

4. Het samenbrengen van de onderdelen

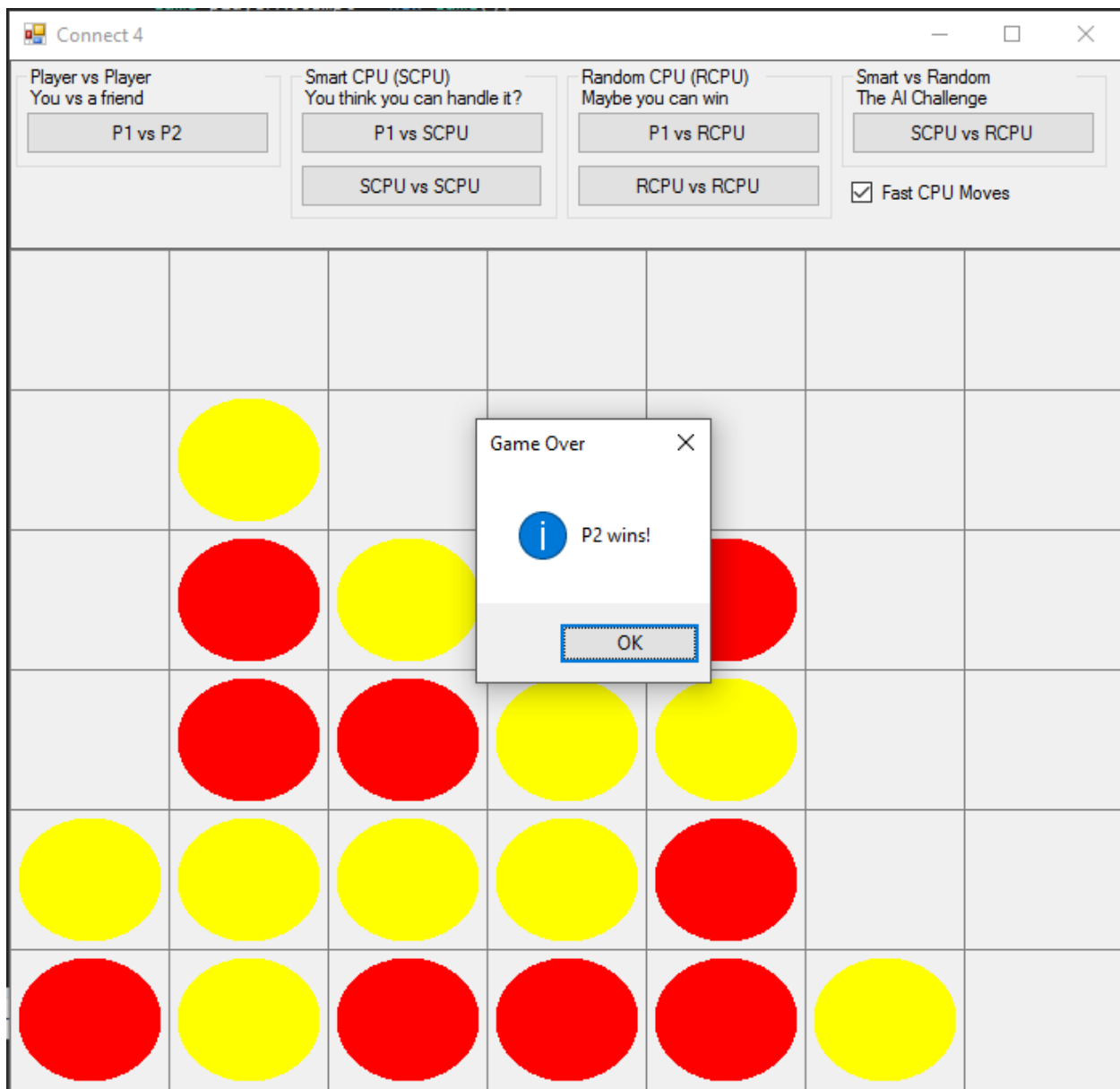
4.1 Het samenbrengen in de GUI

Door de het project op te delen in de besproken onderdelen, is het nu heel eenvoudig om enkele spelscenario's op te zetten.

Speler 1 en 2 kunnen gedefinieerd worden als mens, random computer of slimme computer. Het spelbord wordt gereset en het spel begint. Het enige wat dan nog moet gebeuren is de keuze invullen van wanneer de computer een zet dient te doen (methode Continue() aanroepen), en de events voor winstaat en gelijkspelstaat te koppelen aan een gewenst scenario.

In de GUI worden scenario's voorzien om spelers tegen elkaar te laten spelen, random computers of slimme computers tegen elkaar te laten spelen, de speler tegen een slimme of random computer te laten spelen of een random computer tegen een slimme computer te laten spelen.

Omdat de code zeer performant is en er geen animaties voor de zetten voorzien zijn, wordt een artificiële timer ingebouwd waarop de computer als het ware 'nadenkt'. Deze timer kan worden uitgeschakeld door een checkbox 'Fast CPU moves' aan te vinken.



Figuur 4.1. Overzicht van de GUI

4.2 Resultaten van enkele experimenten met de kunstmatige intelligentie

Bij het zelf spelen en andere mensen te laten spelen zijn de resultaten zeer bevredigend. Mensen kunnen de computer verslaan wanneer er goed opgelet en nagedacht wordt. De computer heeft in alle tests echter meer gewonnen dan verloren. Hij voorziet dus een uitdaging, maar is niet onverslaanbaar.

Wanneer de slimme computer tegen een andere slimme computer wordt opgezet, wint elke computer ongeveer de helft van de keren.

Wanneer de slimme computer tegen een random computer wordt opgezet, wint hij altijd.