SticksGame

Du skal lave et lille computerspil som kan anvende fem forskellige spillertyper: Human, Simple, Random, Smart og Intelligent.

Spillets regler:

a) Der er 15 pinde og to spillere
b) Spillerne skiftes til at tage fra 1 til 3 pinde af de tilbageværende.
c) Den der tager den sidste pind taber

Bemærk, der er en vindende strategi hvis der ikke er "4n+1" pinde tilbage, hvor n er et helt tal fra 0 til 3. Altså kan man altid vinde hvis der *ikke* er 1, 5, 9, eller 13 pinde – og dermed også hvis man kan bringe modstanderen i tur på en af disse værdier.

Del 1:

1) Opret et nyt C# Console Application projekt, kald det *SticksGame*.
2) Opret en ny class *Player* med følgende attributter og *public* metoder:
    a. public string Name { get; set; }
    b. private int _wins=0;
    c. public int Wins – property til at tilgå _wins med *private* setter, så der ikke kan snydes med wins:

```
{
    get { return _wins; }
    private set {  _wins = value; }
}
```

    d. public Player() til initialisering af spileren:
        i. sæt Name til "Simple"
        ii. sæt Wins til 0
    e. public String GetName() returnerer værdien af *name*.
    f. public void StartGame() til initialisering ved spilstart, tom funktion
    g. public int GetMove(int sticksInPlay) beregner næste træk, pt. 1 uanset parameterværdien.
    h. public void EndGame(bool youHaveWon) som laver eventuel opfølgning på det enkelte spil. youHaveWon == true, hvis den pågældende spiller har vundet. Tæl *wins* op hvis spilleren har vundet.
3) Opret en ny class *Game* med følgende attributter og (for nu tomme) *public* metoder:
    a. private Player[] players = new Player[2];
    b. public void Init() – som initialiserer denne spilsekvens data, inkl. Spillere
    c. public void Play() – som afvikler et spil ad gangen
    d. public void Play(int gameCount) – som afvikler *gameCount* spil ad gangen
    e. public string GetStat() – som returnerer spilstatistik som en string.
4) Implementer ovenstående metoder i Game.
    a. Init() skal oprette de to spillere (gemmes i Players[]) som skal Initialiseres ved at kalde deres Init metode.
    b. Play() skal initialisere spillet, efter tur bede om *move* fra spillerne og kalde EndGame på spillerne efter spillet. Desuden skal der under vejs udskrives info til konsollen.

   c. Play(gameCount) skal kalde Play() et fornuftigt antal gange.
   d. Endelig skal GetStat() returnere en string af formatet: ”xx spil, score xx/xx” hvor xx erstattes af den relevante værdier fra spillerene.
5) Test spillet – hver spiller skal tage én pind i hvert træk.

Del 2:

Du skal nu lave den næste spillertype *RandomPlayer*.

1) Opret nu class RandomPlayer, denne skal nedarve fra Player!
   class RandomPlayer : Player …
2) For at tilføje ny funktionalitet vil vi ”override” nogle af Player’s funktioner:
   a. Init() – skal sætte navnet til ”Random”
   b. GetMove(…) – skal returnere en værdi mellem 1 og (min af 3 og det resterende antal pinde)
3) For at den nye funktionalitet anvendes uanset om vi ser på en RandomPlayer som en Player eller som en RandomPlayer, skal de ”override”-ede metoder erklæres som ”virtuel” i vore Player class:
   a. public virtual void Init()
   b. public virtual int GetMove(…)
4) Tilret Game.Init(), så den, hver gang der skal initialiseres en spiller, udskriver en lille menu, læser brugerens valg og opretter en spiller af den pågældende type. Da RandomPlayer er nedarvet fra Player, kan vi *godt* udføre: Players[0] = new RandomPlayer(); - selvom Players er erklæret som af typen Player[].
5) Test RandomPlayer sammen med Player.

Del 3:

Spillertypen *HumanPlayer*:

1) Anvend samme metode som for RandomPlayer, med disse ændringer:
   a. Brugeren skal spørges om spillerens navn
   b. Brugeren skal indtaste hvert træk når *GetMove* kaldes, det indtastede skal testes for lovlighed og indtastningen genstartes ved ulovligt træk.
2) HumanPlayer skal tilføjes menuen og testes sammen med de andre spillertyper.

Del 4:

Spillertypen *SmartPlayer*:

1) Vi gentager metoden fra RandomPlayer, sæt navnet rigtigt. Anvend observationen fra opgavens indledning til altid at gøre det bedst mulige. Kald ”base.GetMove” hvis der ikke er et optimalt flyt.
2) Tilføj til menu og test.
3) Lav ændringer i RandomPlayer så Init() og GetMove er virtuelle
4) Lad SmartPlayer nedarve fra RandomPlayer i stedet for player.
5) Test og observer at SmartPlayer nu vælger tilfældigt når den ikke har et vindende træk.

Del 5:

Spillertypen IntelligentPlayer (laves ikke uden et vist personligt mod og overskud ☺)

1) Opret ny class som nedarver fra RandomPlayer
   a. Navn: ”EnSteen”

Vi vil lave en AI ved at huske slutresultatet af alle træk i alle spil vi har deltaget i ved at gemme +/-1 i en simpel tabel.

På den ene led har vi antallet af pinde tilbage [(1-15)-1] i spillet, på den anden led har vi de forskellige træk vi kan foretage [(1-3)-1].

Når vi spiller et spil, opdaterer vi ikke tabellen med det samme, men gemmer trækkene efterhånden som de udføres. Efter spillet (i EndGame) opdaterer vi så vores tabel med +1 for de træk der har været med til at vinde og med -1 for de træk som har deltaget i tabt spil.

I GetMove, vil vi vælge det relevante træk med den højeste score – og med flere lige store vælge et tilfældigt.

2) Tilføj de nødvendige datastrukturer til *IntelligentPlayer*
3) Tilføj kode til StartGame som initialiserer variablen vi bruger til at gemme dette spils træk – og andet der er behov for…
4) Lav kode der returnerer det hidtil mest lovende træk.
5) Tilføj kode til EndGame som opdaterer vores tabel
6) Test – det skal mellem 4 og 35 spil til før vores AI-spiller har lært lektien.