Uitwerkingen tweede deeltentamen Gameprogrammeren
Vrijdag 17 oktober 2014, 8.30-10.30 uur

---

1. Deze opgave bestaat uit een aantal deelvragen. Houd het antwoord kort: één of twee zinnen per onderdeel kan al genoeg zijn.

   (a) Gegeven is de volgende methode:

```
1  int Methode(List<int> x, int y)
2  {
3      int teller = 0, resultaat = 0;
4      while (teller < y) {
5          resultaat += x[teller];
6          teller++;
7      }
8      return resultaat;
9  }
```

   (1 punt) Omschrijf in woorden wat deze methode oplevert.

   Antwoord: Deze methode berekent de som van de eerste y elementen in de lijst x.

   (2 punten) Herschrijf de body van deze methode zodanig dat de methode hetzelfde resultaat oplevert, maar zonder een loop-constructie te gebruiken zoals **for**, **foreach**, **while**, of **do−while**. Je mag hierbij aannemen dat x nooit leeg is, en y altijd groter dan nul en kleiner of gelijk aan het aantal elementen in x.

   Antwoord: Dit kan gedaan worden met behulp van recursie:

```
1  int Methode(List<int> x, int y)
2  {
3      if (y == 1)
4          return x[0];
5      return Methode(x, y−1) + x[y−1];
6  }
```

   (b) *(2 punten)* Kruis aan welke van de volgende stellingen waar zijn:

   - X Een klasse mag meerdere interfaces implementeren.
   - X In de body van een statische methode mag **this** niet gebruikt worden.
   - X Een klasse heeft altijd maximaal één superklasse.
   - X Als een array eenmaal geïnitialiseerd is, dan mag het maximaal aantal elementen in de array niet meer gewijzigd worden.

   (c) *(1 punt)* Wat is het verschil tussen **base** en **this**?

   Antwoord: **base** en **this** in een methode wijzen allebei naar de huidige instantie waar de methode op wordt aangeroepen, maar **base** wijst naar het deel van de instantie dat is gedefinieerd in de superklasse.

   (d) *(4 punten)* Beschouw de volgende instructie:

```
1  A obj = new A();
```

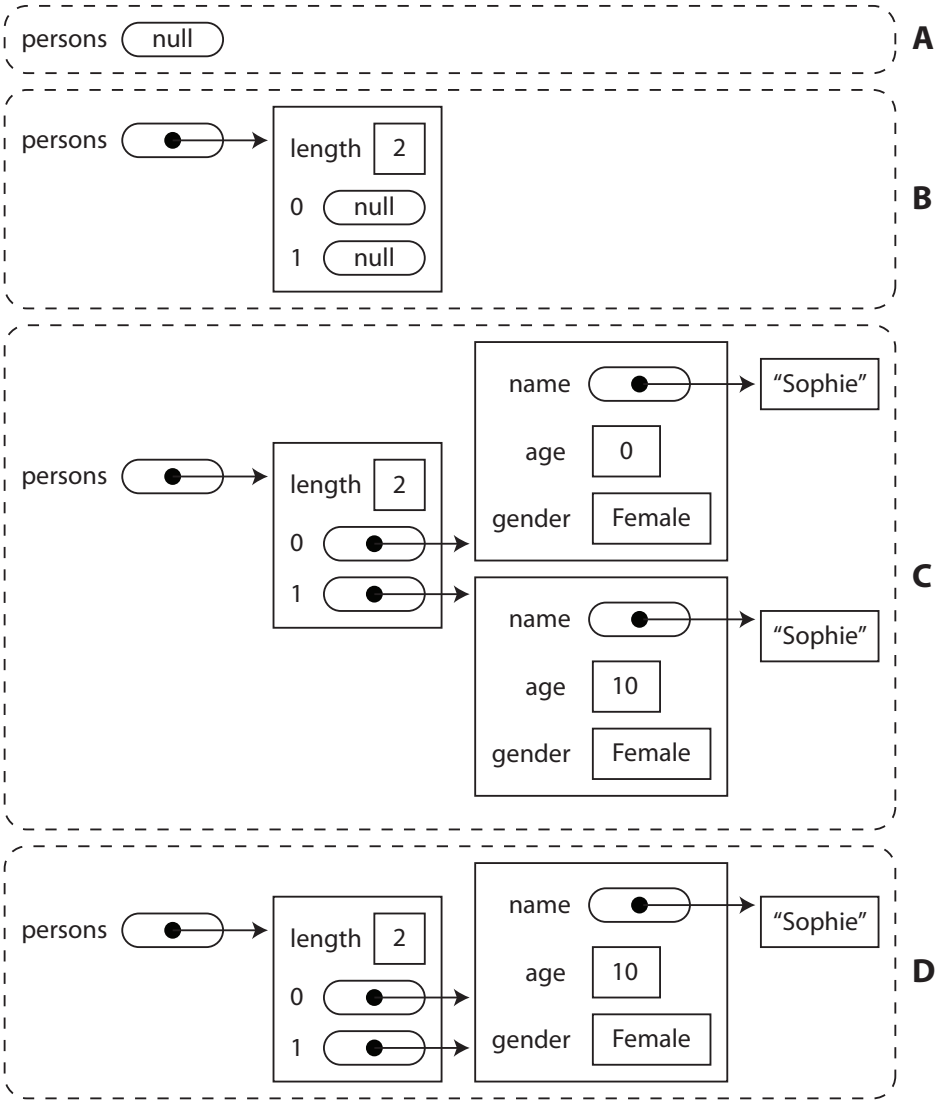
   Deze instructie bestaat uit een aantal onderdelen. Schrijf hieronder achter elk onderdeel wat het onderdeel precies doet:

   Antwoord:

   - declaratie: geeft het type en de naam van een variabele aan
   - **new** operator: reserveert het geheugen dat het object nodig heeft
   - constructor: kent waarden toe aan het geheugen gereserveerd door de **new** operator
   - toekenning: kent een verwijzing naar het geheugenadres van het object toe aan de variabele

2. *(2 + 3 + 3 + 2 = 10 punten)*

Antwoord:



3. (a) *(2 punten)* In de constructor van de GameWorld klasse mist nog een instructie. Welke instructie is dit? Tussen welke regelnummers hoort die instructie te staan?

Antwoord: de volgende instructie mist nog (initialisatie van grid):

```
1  foodgrid = new SpriteGameObject[cols, rows];
```

Deze instructie hoort tussen regelnummers 13 en 14.

(b) *(4 punten)* De code die het aantal rijen en kolommen uit de filenaam haalt hoort in de constructor van SpriteGameObject te staan. Schrijf de missende instructies op. Let op: SpriteGameObject moet met *alle* soorten sprites om kunnen gaan (enkel, strip, of sheet).

Antwoord:

```
1  string[] assetSplit = assetname.Split('@');
2  if (assetSplit.Length <= 1)
3      return;
4  string sheetNrData = assetSplit[assetSplit.Length - 1];
5  string[] colrow = sheetNrData.Split('x');
6  this.sheet_columns = int.Parse(colrow[0]);
```

```
7   if (colrow.Length == 2)
8       this.sheet_rows = int.Parse(colrow[1]);
```

(c) *(4 punten)* In de HandleInput methode van GameWorld wordt het klikken op een van de objecten in de grid afgehandeld. Als dat gebeurt, dan moet het object met zijn linkerbuurman verwisseld worden. Werk deze methode uit. Let op: we kijken nu enkel naar het verwisselen van twee objecten. Het vervangen van dezelfde objecten die naast elkaar staan door nieuwe objecten gebeurt in de Update methode (zie de laatste twee onderdelen van deze vraag).

Antwoord:

```
1   if (inputHelper.MouseLeftButtonPressed())
2   {
3       int xpos = (int)inputHelper.MousePosition.X / grid_width;
4       int ypos = (int)inputHelper.MousePosition.Y / grid_height;
5       if (xpos > 0)
6       {
7           SpriteGameObject tmp = foodgrid[xpos, ypos];
8           foodgrid[xpos, ypos] = foodgrid[xpos − 1, ypos];
9           foodgrid[xpos − 1, ypos] = tmp;
10      }
11  }
```

In de Update methode van GameWorld zorgen we ervoor dat groepen van dezelfde objecten verwijderd en weer vervangen worden door nieuwe (willekeurige) voedsel objecten. Dit gebeurt in twee stappen. In de eerste stap wordt het hele grid doorgelopen, en wordt voor elk object bewaard of het object vervangen moet worden of niet. Deze informatie wordt bewaard in de lokale variabele toreplace (zie de GameWorld klasse in de bijlage).

(d) *(6 punten)* Schrijf de instructies uit die door het grid lopen en voor elk object in de variabele toreplace markeren of het object vervangen moet worden of niet. Let op dat je rekening houdt met alle mogelijkheden (voor een paar voorbeelden, zie de vorige screenshot).

Antwoord:

```
1   for (int x = 0; x < cols; x++)
2       for (int y = 0; y < rows; y++)
3       {
4           if (y > 0 && foodgrid[x, y].SheetIndex == foodgrid[x, y − 1].SheetIndex)
5           {
6               toreplace[x, y] = true;
7               toreplace[x, y − 1] = true;
8           }
9           if (x > 0 && foodgrid[x, y].SheetIndex == foodgrid[x − 1, y].SheetIndex)
10          {
11              toreplace[x, y] = true;
12              toreplace[x − 1, y] = true;
13          }
14      }
```

(e) *(4 punten)* De laatste stap is het vervangen van alle gemarkeerde objecten door nieuwe (willekeurige) objecten. Schrijf de instructies op die dit bewerkstelligen.

Antwoord:

```
1   for (int x = 0; x < cols; x++)
2       for (int y = 0; y < rows; y++)
3           if (toreplace[x,y])
4               foodgrid[x, y] = new SpriteGameObject("food@7x2", Content, FoodFight.Random.Next(14));
```