Uitwerkingen Aanvullende toets **Gameprogrammeren**
Woensdag 5 januari 2011, 8.30-10.30 uur

1. (a) *(2 punten)* Gegeven zijn de volgende twee declaraties van variabelen:

```
string s;
double waarde;
```

Iemand schrijft de volgende opdracht om de wortel van de waarde te berekenen en aan de gebruiker te tonen:

```
s = "De wortel is " + Math.Sqrt(waarde);
```

Maar als `waarde` negatief is wordt het programma afgebroken met een foutmelding.

In plaats daarvan willen we liever dat de tekst `onmogelijk` in de string bewaard wordt. Je kunt dit op twee manieren voor elkaar krijgen:

- Vooraf controleren of de foutsituatie zich gaat voordoen
- De wortel gewoon maar uitrekenen en de foutsituatie opvangen

Geef voor beide aanpakken aan hoe de opdracht er dan uit komt te zien.

Antwoord:

```
// aanpak 1: vooraf controleren
if (waarde<0)
     s = "onmogelijk";
else s = "De wortel is " + Math.Sqrt(waarde);

// aanpak 2: foutsituatie afvangen
try
{   s = "De wortel is " + Math.Sqrt(waarde);
}
catch (Exception e)
{   s = "onmogelijk";
}
```

(b) *(1 punt)* Wat is een drie-dimensionale array? Hoe declareer je zo'n drie-dimensionale array, en hoe kun je in het programma aangeven hoeveel waarden de array kan bevatten?

Antwoord: Een drie-dimensionale array is een hoeveelheid variabelen van hetzelfde type waarvan de individuele elementen zijn op te vragen door drie int-waarden als index op te geven.

Declaratie: *type* `[,,]` *naam* `;`

Initialisatie: *naam* `= new` *type* `[` a `,` b `,` c `];`
geeft een array met $a \times b \times c$ elementen.

(c) *(2 punten)* Gegeven is een klasse `Data` met daarin een member-variabele `getallen`, en methoden die deze variabele een zinvolle waarde geeft:

```
class Data
{
    private double [] getallen;

    // hier staan de bestaande methoden
    // hier komt een nog te schrijven property
}
```

Schrijf een read-only property `Totaal` waarmee het totaal van de opgeslagen waarden bepaald kan worden.

Antwoord:

```
public double Totaal
{   get
    {  double res = 0.0;
       for (int t=0; t<getallen.Length; t++)
           res += getallen[t];
       return res;
    }
}
```

(d) *(2 punten)* Hieronder staan vijf programma-fragmenten. Geef in elk van de gevallen aan *hoe vaak* de methode `iets` wordt aangeroepen. Licht het antwoord kort toe.

1	`for (x=0; x<5; x++)`
	`    this.iets();`
	`for (y=0; y<5; y++)`
	`    this.iets();`
		2	`for (x=0; x<5; x++)`
			`for (y=0; y<x; y++)`
			`    this.iets();`

3	`for (x=0; x<5; x++)`
	`{   this.iets();`
	`    x=x+1;`
	`}`
		4	`x=0;`
			`while (x<0) ;`
			`    this.iets();`
			`    x=x+1;`

5	`for (x=0; x<x; x++)` `    this.iets();`		

Antwoord:
fragment 1: 10 keer (5 keer voor elke `x`, en dan nog eens 5 keer voor elke `y`)
fragment 2: 10 keer (0+1+2+3+4 keer)
fragment 3: 3 keer (voor `x` is 0, 2 en 4)
fragment 4: 1 keer (de while-opdracht heeft een lege body, daarna is `iets` aan de beurt)
fragment 5: 0 keer (de voorwaarde is meteen `false`)

(e) *(3 punten)* Gegeven de volgende klassen:

```
abstract class A
{
    public abstract void methode1();

    public void methode2()
    {
        return;
    }
}
class B : A
{
    public override void methode1()
    {
        return;
    }

    public void methode3(A a)
    {
        a.methode1();
    }
}
```

Geef voor elk van de volgende opdrachten aan of ze mogen:

Antwoord:

```
A obj; // ja, de referentie declareren mag altijd
obj = new A(); // nee, van een abstracte klasse kun je geen object aanmaken
obj = new B(); // ja, van een concrete subklasse wel
obj.methode1(); // ja, de compiler weet dat elke subklasse deze methode heeft
obj.methode2(); // ja, deze methode wordt geerfd
obj.methode3(obj); // nee, de compiler kan niet controleren dat het object deze methode heeft
B anderObject = (B)(new A()); // nee, van een abstracte klasse kun je geen object aanmaken
A nogEenObject = (A)obj; // ja, je mag altijd casten naar een "hoger" type
obj.methode3(nogEenObject); // nee, de compiler kan niet controleren dat het object deze methode heeft
A[] lijst; // ja
lijst = new A[10]; // ja, hier wordt een array referenties aangemaakt
lijst[0] = new A(); // nee, van een abstracte klasse kun je geen object aanmaken
lijst[1] = new B(); // ja, van een concrete subklasse wel
List<A> nogEenLijst = new List<A>(); // ja
```

2. (a) *(2 punten)* De methode `laadTiles` in de `Level` klasse roept een methode `laadTile` aan die een tile bouwt aan de hand van het gelezen karakter. Deze methode zorgt er ook voor dat de lijst met dozen gevuld wordt en dat het speler object gemaakt wordt. Werk deze methode uit met behulp van de `switch` opdracht.
Antwoord:

```
private Tile laadTile(char tileType, int x, int y)
{
    switch (tileType)
    {
        case '.':
            return new Tile(null, TileType.Achtergrond, x, y);
        case 'W':
            return new Tile(muurSprite, TileType.Muur, x, y);
        case 'B':
            dozen.Add(new Tile(doosSprite, TileType.Doos, x, y));
            return new Tile(null, TileType.Achtergrond, x, y);
        case 'G':
            return new Tile(doelSprite, TileType.Doel, x, y);
        case 'P':
            speler = new Tile(spelerSprite, TileType.Speler, x, y);
            return new Tile(null, TileType.Achtergrond, x, y);
        default:
            throw new NotSupportedException("Onbekend tile type: " + tileType);
    }
}
```

(b) *(2 punten)* Om makkelijk uit te kunnen vinden of een doos zich bevindt op een bepaalde (x, y)-positie willen we ook een methode `VindDoos` maken in de `Level` klasse. Deze methode krijgt als parameter twee `int`s (de x- en de y-positie) mee en die loopt door de lijst van dozen om te zien of een van die dozen op de gevraagde positie staat. Zo ja, dan wordt die doos als resultaat opgeleverd. Zo nee, dan levert de methode de waarde `null` op. Implementeer deze methode.
Let op: het gaat hier niet om de pixel-coördinaten, maar om de locatie in het tweedimensionale tileveld!
Antwoord:

```
Tile VindDoos(int x, int y)
{
    foreach (Tile t in dozen)
        if (t.X == x && t.Y == y)
            return t;
    return null;
}
```

(c) *(4 punten)* De belangrijkste methode is de methode `VerplaatsSpeler` in de `Level` klasse. Gegeven een toets die de speler heeft ingedrukt, verplaatst deze methode de speler naar de nieuwe positie, indien mogelijk. Als er een doos op de gewenste positie van de speler staat, dan moet de doos verplaatst worden. Echter, dat mag alleen als de nieuwe positie van de doos géén muur of een andere doos is. Werk deze method uit. Maak hierbij gebruik van de methode `VindDoos` die we in het vorige onderdeel hebben uitgewerkt.
Hint: begin met het bepalen van de doelposities van de speler en een eventuele doos. Kijk daarna of je de speler en de doos (als die er staat) mag verplaatsen!

Antwoord:

```
public void VerplaatsSpeler(Keys key)
{
    int xoffset = 0, yoffset = 0;
    if (key == Keys.Left)
        xoffset--;
    else if (key == Keys.Right)
        xoffset++;
    else if (key == Keys.Up)
        yoffset--;
    else if (key == Keys.Down)
        yoffset++;
    int spelerdoelx = speler.X + xoffset;
    int spelerdoely = speler.Y + yoffset;
    int doosdoelx = spelerdoelx + xoffset;
    int doosdoely = spelerdoely + yoffset;

    Tile doos = VindDoos(spelerdoelx, spelerdoely);
    if (doos != null && IsGeldigePlaats(doosdoelx, doosdoely))
    {
        doos.X = doosdoelx;
        doos.Y = doosdoely;
    }
    if (IsGeldigePlaats(spelerdoelx, spelerdoely))
    {
        speler.X = spelerdoelx;
        speler.Y = spelerdoely;
    }
}

private bool IsGeldigePlaats(int x, int y)
{
    if (x < 0 || x >= tiles.GetLength(0) || y < 0 || y >= tiles.GetLength(1))
        return false;
    if (tiles[x, y].TileType == TileType.Muur || VindDoos(x,y) != null)
        return false;
    return true;
}
```

(d) *(1 punt)* Een level is afgemaakt als alle dozen op een doelpositie staan. Implementeer de methode `LevelAfgemaakt`. Deze methode geeft waarheidswaarde `true` als resultaat als het level is afgemaakt, en in alle andere gevallen `false`.

Antwoord:

```
bool LevelAfgemaakt()
{
    foreach (Tile b in dozen)
        if (tiles[b.X, b.Y].TileType != TileType.Doel)
            return false;
    return true;
}
```

(e) *(1 punt)* Is het een goede ontwerpkeuze geweest om de `Tile` klasse ook te gebruiken voor de speler en de dozen? Zo ja, wat zijn de nadelen om het anders te doen? Zo nee, wat zou een betere keuze zijn geweest?

Antwoord:
Het is eigenlijk geen goede ontwerpkeuze omdat op deze manier de x en y-posities dubbel worden opgeslagen van de achtergrond, muur en doeltiles. Ook zijn de speler en de dozen in principe geen tiles. Een betere keuze zou zijn geweest om een algemene `GameObject` klasse te maken, waarvan een `Tile` erft. We kunnen dan vervolgens subklassen maken van de `GameObject` klasse.