

AANVULLENDE TOETS GAMEPROGRAMMEREN (INFOB1GP)  
VRIJDAG 4 JANUARI 2013, 11.00 - 13.00 UUR

<b>Naam:</b>	
<b>Studentnummer:</b>	

- Het tentamen bestaat uit 2 opgaven. De eerste opgave levert 15 punten op, de tweede opgave 25 punten. Je cijfer is het totaal aantal punten gedeeld door 4. Als je een deel van een opgave niet weet, probeer dan toch zo veel mogelijk op te schrijven!
- De lijst met standaardfuncties na afloop graag weer inleveren.

*Veel succes!*

---

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

- (a) (3 punten) De methode `int.Parse` kan een `string`-waarde uitlezen en omzetten naar een `int`-waarde. Een voorbeeld:

```
string getal = LeesTekst();  
int getalAlsInt = int.Parse(getal);
```

De `LeesTekst` methode vraagt de gebruiker om tekst in te voeren, en de methode levert deze tekst op als resultaatwaarde. In de tweede regel wordt die waarde omgezet naar een `int` met behulp van de `int.Parse` methode. Het probleem is dat `int.Parse` excepties kan werpen, zoals *onder andere* een `FormatException` (indien de `string` geen getal bevat) of een `OverflowException` (als de string een groter getal bevat dan de maximum of minimum `int`-waarde). Schrijf een methode `LeesGetal` die een `string` inleest (gebruik hiervoor de `LeesTekst` methode), en die een `int` oplevert als resultaat. Als de ingevoerde `string` geen `int`-waarde bevat dan geeft de methode de waarde 0 terug. Als de gelezen `string` een `int`-waarde heeft die groter is dan de toegestane maximum- of minimumwaarde, dan moet de methode -1 opleveren. In het geval van een andere exceptie geeft de methode de waarde 99 terug.

- (b) (2 punten) Wat is het verschil tussen een *expressie* en een *opdracht/instructie*? Is `return x;` een opdracht/instructie of een expressie?

- (c) (4 punten) Kruis aan welke van de volgende stellingen waar zijn:
- Elke **if**-opdracht kan worden omgeschreven naar een **switch** opdracht en vice versa.
  - Excepties gebruik je vooral om het programma te stoppen in het geval dat er een fout optreedt die niet door de programmeur voorzien had kunnen worden, zoals een missend tekstbestand, of een verstoorde netwerkverbinding.
  - Arrays zijn nuttig want je kunt er tweedimensionale structuren zoals een 2D speelveld mee modeleren. Dit kan niet met een Collection (sub)klasse.
  - Botsingen tussen game objecten worden afgehandeld in de Draw methode, want we weten pas dat object botsen nadat ze getekend zijn.
- (d) (3 punten) Schrijf de volgende **if**-instructie om naar een **switch**-instructie. De variabele `x` is van het type **int**.

```
if (x == 1)
    Method1();
else if (x == 2 || x == 3)
{
    Method2();
    Method3();
    return;
}
else if (x > 0 && x < 7)
    Method4();
else
    Method5();
```

(e) (3 punten) Beschouw de volgende twee klassen:

```
class GameObject
{
    protected Vector2 position, velocity;
    protected string id;

    public GameObject(string id)
    {
        this.id = id;
    }

    public virtual void HandleInput(InputHelper inputHelper, GameWorld gameWorld)
    {
    }

    public abstract void Update(GameTime gameTime, GameWorld gameWorld);

    public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
    }
}
class SpriteGameObject : GameObject
{
    protected Texture2D sprite;

    public SpriteGameObject(string id, Texture2D sprite)
    {
        this.id = id;
        this.sprite = sprite;
    }

    public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(sprite, position, Color.White);
    }
}
```

Er zitten *drie* fouten in deze klassendefinities, waardoor een programma dat de klassen wil gebruiken niet compileert. Welke drie fouten zijn dit?

1:

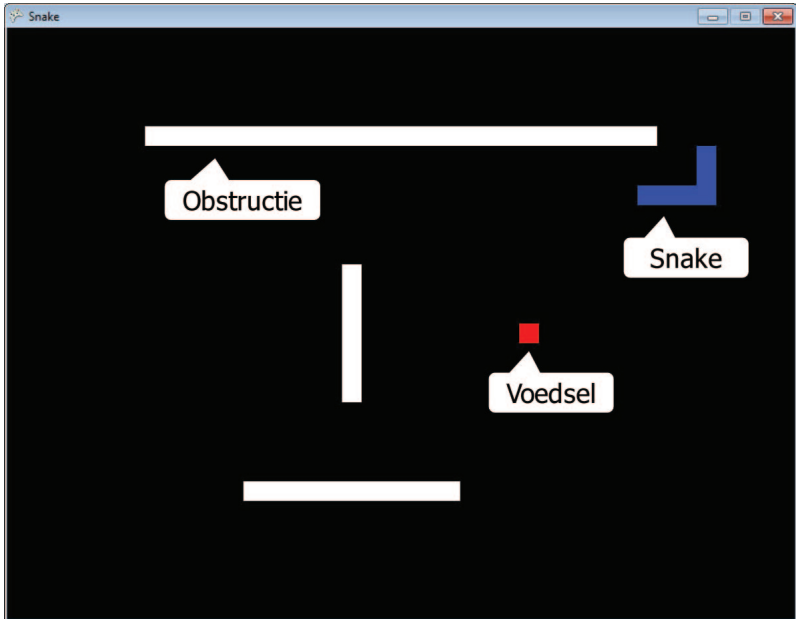
2:

3:

2. In deze opgave gaan we een klassiek spel, Snakes, uitwerken. In het spel Snakes bestuurt de speler een slang met de pijltjestoetsen. In het spel kun je voedsel verzamelen, maar bij iedere consumptie wordt de slang langer. Als de slang in botsing komt met een obstructie in het level, of met de rand van het scherm, of met zichzelf, dan is het spel afgelopen. In deze versie van het spel kunnen we een level laden uit een tekstbestand. Zo'n tekstbestand kan er bijvoorbeeld als volgt uitzien:

```
.....  
.....  
.....  
.....  
.....XXXXXXXXXXXXXXXXXXXXXXXXXXXX.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....X.....  
.....X.....  
.....X.....  
.....X.....  
.....X.....  
.....X.....  
.....X.....  
.....  
.....  
.....XXXXXXXXXX.....  
.....  
.....  
.....  
.....
```

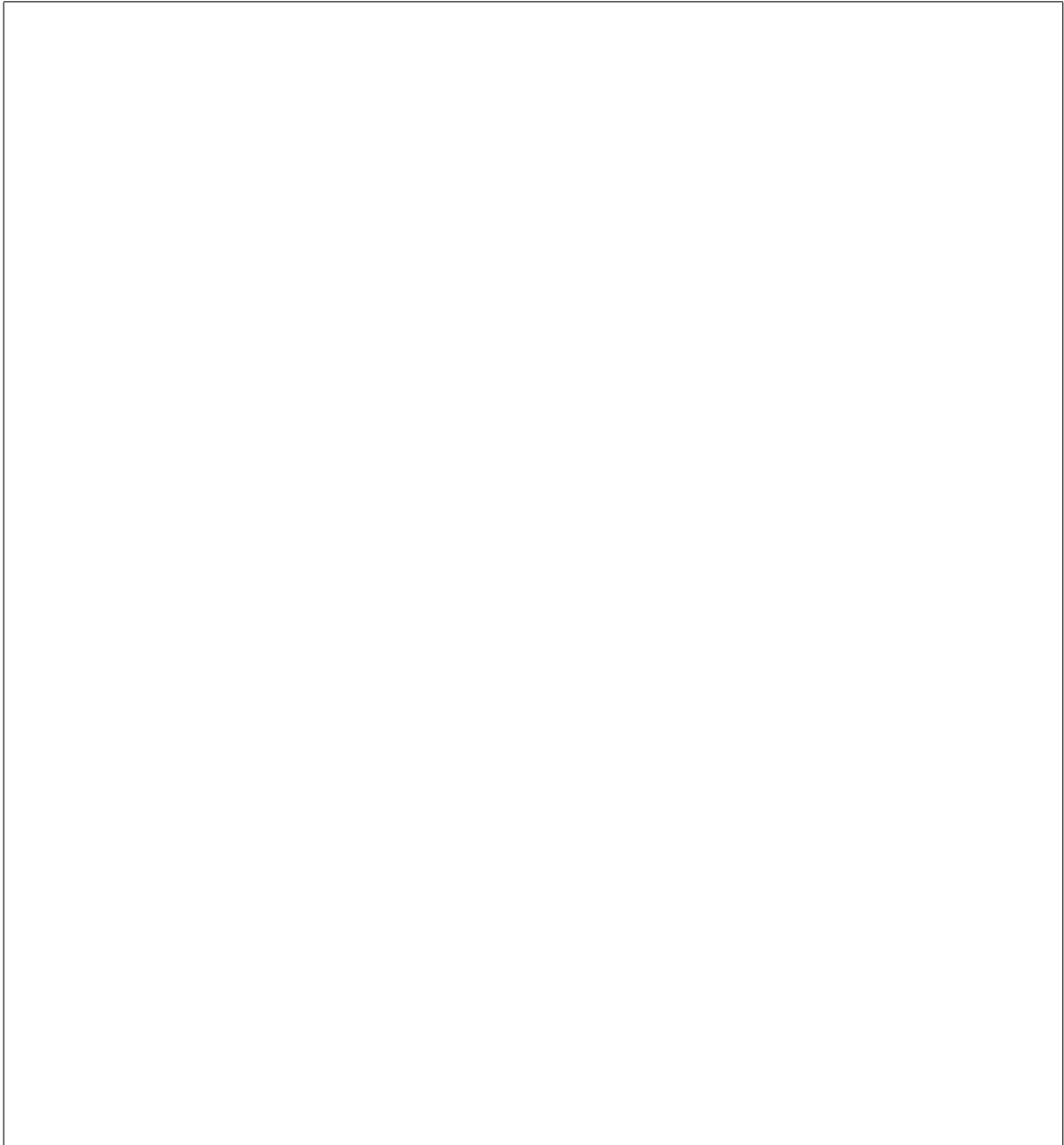
Een screenshot van het Snakes spel is als volgt gegeven:



In deze screenshot is met een paar tekstballonnen aangegeven wat welke elementen zijn (de slang, voedsel en obstructies). Een aantal klassen (`SnakeGame`, `Level` en `Snake`) zijn reeds (deels) uitgewerkt, en ze zijn te vinden in de bijlage van deze toets.

In de `Level` klasse laden we het level uit het tekstbestand. Dit level bestaat uit een aantal 'tiles'. Als basisbouwblok voor elk level gebruiken we de sprite `blockSprite`. Dit is een sprite van  $20 \times 20$  pixels. Deze sprite gebruiken we om de obstructies te tekenen (in het wit), het voedsel te tekenen (in het rood), en de slang te tekenen (in het blauw). In het tekstbestand staan alleen de obstructies. We bepalen de positie van het voedsel en de initiële positie en bewegingsrichting van de slang willekeurig.

- (a) (*6 punten*) De methode `loadLevel` is verantwoordelijk voor het laden van een level uit een tekstbestand. Dit level wordt opgeslagen in een 2D array van `bool` waarden, waarbij de waarde `true` inhoudt dat de betreffende tile een obstructie is. Normaalgesproken bevat een tekstbestand alleen regels met hetzelfde aantal karakters. Als dat niet het geval is, dan moet de methode `loadLevel` een exceptie van het type `IOException` werpen. Schrijf de body van de `loadLevel` methode uit.



- (b) (*4 punten*) In de `Level` klasse willen we ook een property `RandomFreePosition` toevoegen. Deze property moet een willekeurige positie in het level teruggeven, uitgedrukt in indices van de tile-array, niet in pixel-coördinaten. De conditie is wel dat die positie geldig is, oftewel: de positie valt binnen de scherm breedte en de positie is niet al bezet door een obstructie. Voor het gebruik van deze property, zie bijvoorbeeld de constructormethode van de `Level` klasse. Werk de property `RandomFreePosition` uit.

- (c) (5 punten) De `Snake` klasse wordt gebruikt om de slang in het level voor te stellen. De slang wordt voorgesteld door een lijst van posities in de variabele `snakePositions`. Verder beweegt de slang zich voort in een bepaalde richting die wordt bewaard in de variabele `direction`. Zoals je kunt zien in de `Reset` methode kennen we een willekeurige richting toe aan de slang bij het starten van het spel. Er zijn in principe *vier* verschillende richtingen mogelijk: naar links, naar rechts, naar boven en naar onder. De slang bestaat in het begin uit een enkel element, en de positie van dat element wordt bepaald aan de hand van de `RandomFreePosition` methode uit de `Level` klasse, zie ook de `Reset` methode.

In de `HandleInput` methode kan de speler de slang van richting laten veranderen met behulp van de pijltjestoetsen. Er is één beperking: de slang mag niet in tegengestelde richting gaan bewegen, dus als de slang momenteel naar links beweegt, dan heeft het indrukken van de rechterpijltjestoets geen effect (de richting mag alleen veranderd worden naar boven of naar onderen). Schrijf de body van de `HandleInput` methode op. Je mag hierbij gebruik maken van de methode `bool KeyPressed(Keys k)` in de `InputHelper` klasse.

- (d) (*7 punten*) In de `Update` methode van de `Snake` klasse moet een aantal dingen gebeuren. Ten eerste moet de slang iedere 0.2 seconden verplaatst worden. Een verplaatsing van de slang kun je opvatten als het toevoegen van een blok aan de kop van de slang in de bewegingsrichting van de slang, en het verwijderen van een blok aan de staart. Hierbij geldt dat de staart van de slang het eerste element is in de lijst, en de kop het laatste element.

Ook moet in deze methode gekeken worden of het spel afgelopen is. Dit is het geval als de slang met een obstructie of zichzelf botst, of als de slang buiten het scherm raakt. Indien dat het geval is, moet de `GameOver` property gedefinieerd in de `Level` klasse op `true` gezet worden.

Tenslotte moet deze methode het consumeren van een voedsleenheid afhandelen. Als de slang op de positie van een voedselblok komt, dan moet de slang één blok langer worden, en een nieuw voedselblok moet geplaatst worden op een willekeurige (geldige!) positie in het level.

Schrijf de body van de `Update`-methode uit.



- (e) (*3 punten*) In de `Draw` methode van de `Snake` klasse moet de slang op het scherm getekend worden (in het blauw). Schrijf de body van de `Draw`-methode uit.

## Bijlage: klassen

### SnakeGame

```
class SnakeGame : Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Level currentLevel;
    InputHelper inputHelper;

    static void Main()
    {
        SnakeGame game = new SnakeGame();
        game.Run();
    }

    public SnakeGame()
    {
        Content.RootDirectory = "Content";
        graphics = new GraphicsDeviceManager(this);
        graphics.PreferredBackBufferWidth = 800;
        graphics.PreferredBackBufferHeight = 600;
        inputHelper = new InputHelper();
    }

    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        Texture2D blockSprite = Content.Load<Texture2D>("block");
        currentLevel = new Level(blockSprite, "Content/level1.txt");
    }

    protected override void Update(GameTime gameTime)
    {
        inputHelper.Update();
        currentLevel.HandleInput(inputHelper);
        currentLevel.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.Black);
        spriteBatch.Begin();
        currentLevel.Draw(gameTime, spriteBatch);
        spriteBatch.End();
    }
}
```

## Level

```
class Level
{
    protected bool[,] tiles;
    protected Texture2D blockSprite;
    protected Random random;
    protected Snake snake;
    protected Vector2 food;
    protected bool gameover;

    public Level(Texture2D blockSprite, string path)
    { this.blockSprite = blockSprite;
      random = new Random();
      loadLevel(path);
      snake = new Snake(this, blockSprite);
      food = RandomFreePosition;
      gameover = false;
    }

    public void loadLevel(string path)
    {
        // TODO
    }

    public void HandleInput(InputHelper inputHelper)
    { snake.HandleInput(inputHelper);
    }

    public void Update(GameTime gameTime)
    { snake.Update(gameTime);
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    { // Code weggelaten
    }

    public bool isAvailable(Vector2 p)
    {
        return p.X >= 0 && p.Y >= 0 && p.X < Width && p.Y < Height && !tiles[(int)p.X, (int)p.Y];
    }

    // TODO: RandomFreePosition property toevoegen

    public Random Random { get { return random; } }

    public Vector2 Food
    { get { return food; }
      set { food = value; }
    }

    public bool GameOver
    { get { return gameover; }
      set { gameover = value; }
    }

    public int Width { get { return tiles.GetLength(0); } }
    public int Height { get { return tiles.GetLength(1); } }
}
```

## Snake

```
class Snake
{
    protected List<Vector2> snakePositions;
    protected Vector2 direction;
    protected Level level;
    protected double time;
    protected Texture2D blockSprite;

    public Snake(Level l, Texture2D blockSprite)
    {
        level = l;
        this.blockSprite = blockSprite;
        snakePositions = new List<Vector2>();
        Reset();
    }

    public void Reset()
    {
        int r = level.Random.Next(4);
        if (r == 0)
            direction = new Vector2(-1, 0);
        else if (r == 1)
            direction = new Vector2(1, 0);
        else if (r == 2)
            direction = new Vector2(0, -1);
        else
            direction = new Vector2(0, 1);
        snakePositions.Clear();
        snakePositions.Add(level.RandomFreePosition);
    }

    public void HandleInput(InputHelper inputHelper)
    {
        // TODO
    }

    public void Update(GameTime gameTime)
    {
        // TODO
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
    {
        // TODO
    }
}
```

GAME OVER