Derde deeltentamen Gameprogrammeren
Vrijdag 9 november 2012, 8.30-10.30 uur

Naam:	
Studentnummer:	

- Het tentamen bestaat uit 2 opgaven. Opgaven 1 levert 15 punten op. Opgave 2 levert 25 punten op. 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 deelvragen. Houd het antwoord kort: één of twee zinnen per onderdeel kan al genoeg zijn.

   (a) *(3 punten)* De volgende klasse is onderdeel van de library GameManagement:

   ```
   public class InputHelper
   {
       // ...
   }
   ```

   Wat betekent het woord **public** voor de klasse? Wat is het gevolg als we dat woord weg zouden laten?

   (b) *(2 punten)* Eén van de volgende drie declaraties-met-toekenningen is correct. Welke is dat, en waarom zijn de andere twee niet correct?

   ```
   List<int> a = new IList<int>(); // versie 1
   IList<int> b = new List<int>(); // versie 2
   IList<int> c = new IList<int>(); // versie 3
   ```

(c) *(3 punten)* Beschouw de volgende twee klassen:

```
class A {
    public void func1() { Console.WriteLine("A::func1"); }
    public virtual void func2() { Console.WriteLine("A::func2"); }
}
class B : A {
    public void func1() { Console.WriteLine("B::func1"); }
    public override void func2() { Console.WriteLine("B::func2"); }
}
```

Wat is de uitvoer van de volgende serie opdrachten?

```
A x = new A();
A y = new B();
B z = new B();
x.func1();
x.func2();
y.func1();
y.func2();
z.func1();
z.func2();
```

(d) *(3 punten)* Beschouw de volgende klassen:

```
abstract class GameObject {
    protected Vector2 position;
    protected Vector2 velocity;
}
class SpriteGameObject : GameObject {
    protected Texture2D sprite;
}
```

Kruis aan welke van de volgende instructies uitgevoerd mogen worden in een methode van een andere, ongerelateerde klasse:

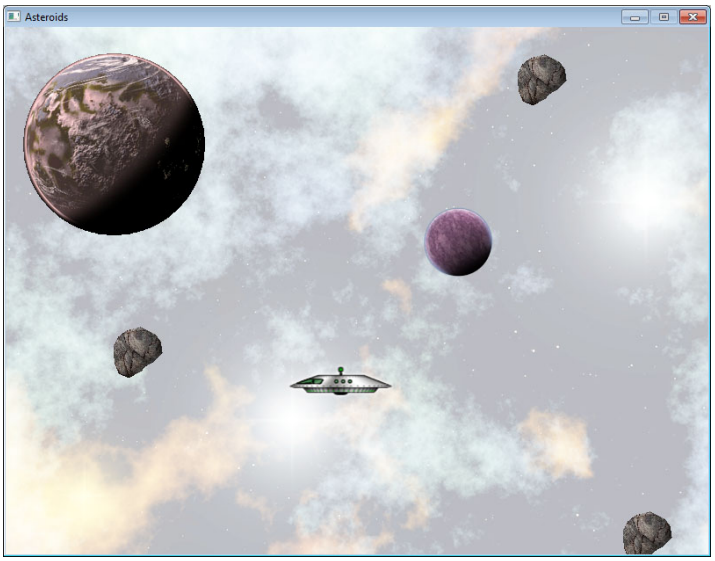
- ☐ `GameObject obj;`
- ☐ `GameObject obj = new GameObject();`
- ☐ `GameObject obj2 = new SpriteGameObject();`
- ☐ `GameObject obj3 = new SpriteGameObject() as GameObject;`
- ☐ `GameObject[] lijst;`
- ☐ `GameObject[] lijst = new GameObject[10];`

(e) *(2 punten)* Wat is een *partial class*? Wanneer is het nuttig om een partial class te gebruiken?

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

- ☐ Net zoals bij interfaces mag je ook van meerdere abstracte klassen erven.
- ☐ Nadat een object van het type **string** gemaakt is, mag dat object niet meer veranderd worden.
- ☐ De alpha-testfase is over het algemeen intern (developers), de beta-testfase is extern (grote groep spelers).
- ☐ Excepties gebruik je vooral om het programma te stoppen in het geval dat in de code een bug zit die geintroduceerd is door de programmeur.

2. In deze opgave gaan we de game "Asteroids" uitwerken. Het spel bestaat uit een scherm met rondvliegende asteroiden met op de achtergrond planeten. Een ufo wordt getekend op de muispositie van de speler. Het doel van de game is om de rondvliegende asteroiden te ontwijken. Als de ufo toch geraakt wordt door een asteroide, dan wordt dit aangegeven door de ufo rood te kleuren. Zo ziet het spel eruit:



De hoeveelheid en initiële positie van de planeten en asteroiden worden geladen uit een tekstbestand. Dit is een voorbeeld van de inhoud van zo'n tekstbestand:

```
asteroid 120 340
asteroid 700 550
asteroid 580 30
planet1 460 200
planet2 20 30
```

Voor elke asteroide of planeet wordt de x en y positie in het tekstbestand bewaard. Ook wordt aangegeven of het gaat om een asteroide ('asteroid') een planeet type 1 ('planet1') of een planeet van type 2 ('planet2'). Het lezen van het level uit het tekstbestand gebeurt in de GameWorld klasse door de methode readLevel aan te roepen. Deze methode leest het tekstbestand uit en creëert de verschillende game objecten (planeten en asteroiden). Als de methode een onbekend type game object tegenkomt (oftewel iets anders dan 'asteroid', 'planet1' of 'planet2') tijdens het lezen van de file, dan werpt de methode een IOException.

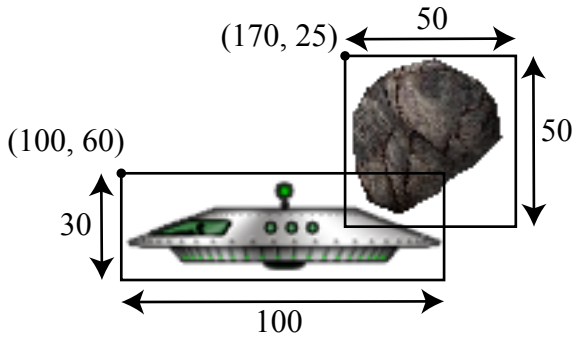
(a) *(4 punten)* Werk de header en body van de readLevel methode uit. Gebruik de **switch** opdracht om de verschillende soorten objecten af te handelen.

(b) *(2 punten)* Indien de readLevel methode een exceptie werpt, dan crasht de game. Hoe moeten we de GameWorld constructor aanpassen zodat dit niet meer gebeurt?

In het begin krijgen alle asteroiden een willekeurige snelheid en richting. In de Update methode van Asteroid wordt deze snelheid vermenigvuldigd. Hierdoor gaan de asteroiden steeds sneller vliegen. Als we gewoon de positie van de asteroide zouden aanpassen aan de hand van zijn snelheid, dan zouden alle asteroiden het scherm uitvliegen, waardoor de game niet zo interessant is. Wat we graag willen is dat als de asteroide uit het scherm vliegt, hij aan de tegenovergestelde kant weer tevoorschijn komt. Dit effect wordt ook wel *wrapping* genoemd.

(c) *(4 punten)* Werk de Update methode van Asteroid verder uit, zodat de positie wordt aangepast en wrapping plaatsvindt. Let hierbij op dat de wrapping netjes geïmplementeerd is, oftewel: geen asteroiden die al verplaatst worden als ze nog niet helemaal het scherm uitgevlogen zijn, of asteroiden die al deels in het scherm staan als ze weer tevoorschijn komen.

Normaalgesproken handelen we botsingen af door te kijken of de bounding boxes van twee game objecten elkaar snijden. Gegeven is het volgende voorbeeld:



Zoals je ziet snijden de bounding boxes elkaar wel, maar feitelijk botsen de ufo en de asteroide helemaal niet. Een betere manier om botsingen af te handelen is niet alleen kijken of de bounding boxes elkaar snijden, maar ook per pixel in de overlappende stukken van de twee sprites kijken of er een botsing is. Die botsing is er als voor twee overlappende pixels geldt dat beide pixels niet geheel transparant zijn, oftewel de alpha waarde van de kleur van beide pixels is niet gelijk aan nul.

Voordat we per pixel gaan berekenen of er een botsing is, is het handig om eerst uit te rekenen wat de overlappende rechthoek is. Bijvoorbeeld, in het plaatje hierboven is de overlappende rechthoek gegeven door de rechthoek op positie $(170, 60)$ met een breedte van 30 pixels en een hoogte van 15 pixels. We willen een *statische* methode intersection toevoegen aan de SpriteGameObject klasse die gegeven twee rechthoeken een rechthoek teruggeeft die het overlappende deel beslaat.

(d) *(5 punten)* Werk de header en body van de intersection methode uit. Je mag er vanuit gaan dat de rechthoeken die worden meegegeven als parameter altijd overlappen (dus je kunt altijd een overlappende rechthoek berekenen). Let op: deze methode moet voor alle mogelijke configuraties van overlapping werken!

(e) *(7 punten)* De volgende stap is het toevoegen van een methode collidesWith aan de SpriteGameObject klasse die bepaalt of het huidige sprite game object botst met een ander sprite game object dat meegegeven wordt als parameter. Deze methode kijkt per pixel in het overlappende gebied of er een botsing is. Werk de header en body van deze methode uit. Gebruik hierbij de intersection en getPixelColor methode. Zorg ervoor dat de methode efficient is. Oftewel: er wordt alleen per-pixel collision checking gedaan als de bounding boxes van de twee objecten overlappen.

(f) *(3 punten)* Tenslotte willen we onze per-pixel collision checking gebruiken in de game. We gaan dit doen in de `Update` methode van de `Ufo` klasse. Werk uit welke instructies aan die methode toegevoegd moeten worden zodat voor elke asteroide gekeken wordt of die botst met de ufo. Zo ja, dan moet de ufo rood worden. Zodra er geen botsing meer is, moet de ufo weer gewoon wit getekend worden. Let op dat *alleen* botsingen tussen de ufo en de asteroiden afgehandeld moeten worden. De andere game objecten (zoals de planeten en de achtergrondsprite) moeten genegeerd worden.

Appendix: klassen

Asteroids

```
1  public class Asteroids : Game
2  {
3      private GraphicsDeviceManager graphics;
4      private SpriteBatch spriteBatch;
5      private static GameWorld gameWorld;
6      private static Random random;
7      private static Vector2 screen;
8
9      static void Main() {
10         Asteroids game = new Asteroids();
11         game.Run();
12     }
13
14     public Asteroids() {
15         graphics = new GraphicsDeviceManager(this);
16         screen = new Vector2(800, 600);
17         graphics.PreferredBackBufferHeight = (int)screen.Y;
18         graphics.PreferredBackBufferWidth = (int)screen.X;
19         Content.RootDirectory = "Content";
20     }
21
22     protected override void LoadContent() {
23         spriteBatch = new SpriteBatch(GraphicsDevice);
24         random = new Random();
25         gameWorld = new GameWorld(Content);
26     }
27
28     protected override void Update(GameTime gameTime) {
29         gameWorld.Update(gameTime);
30     }
31
32     protected override void Draw(GameTime gameTime) {
33         spriteBatch.Begin();
34         gameWorld.Draw(gameTime, spriteBatch);
35         spriteBatch.End();
36     }
37
38     public static Random Random {
39         get { return random; }
40     }
41
42     public static Vector2 Screen {
43         get { return screen; }
44     }
45
46     public static GameWorld GameWorld {
47         get { return gameWorld; }
48     }
49 }
```

SpriteGameObject

```
public class SpriteGameObject
{
    protected Texture2D sprite;
    protected Vector2 position, velocity;
    protected Color color;

    public SpriteGameObject(Texture2D spr, Vector2 pos) {
        this.sprite = spr;
        this.position = pos;
        this.color = Color.White;
    }

    public virtual void Update(GameTime gameTime) {
        this.position += this.velocity;
    }

    public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch) {
        spriteBatch.Draw(this.sprite, this.position, this.color);
    }

    public Rectangle BoundingBox {
        get {
            return new Rectangle((int)position.X, (int)position.Y,
                                 sprite.Width, sprite.Height);
        }
    }

    public Vector2 Center {
        get { return new Vector2(this.sprite.Width, this.sprite.Height) / 2; }
    }

    public Color getPixelColor(int x, int y) {
        // code weggelaten
    }

    // TO DO: intersection methode (d)

    // TO DO: collidesWith methode (e)

}
```

Ufo

```
public class Ufo : SpriteGameObject
{
    public Ufo(Texture2D spr)
        : base(spr, Vector2.Zero) {
    }

    public override void Update(GameTime gameTime) {
        MouseState mouse = Mouse.GetState();
        this.position = new Vector2(mouse.X, mouse.Y) - this.Center;

        // TO DO: botsingen afhandelen (f)
    }
}
```

GameWorld

```
public class GameWorld
{
    private Texture2D planet1, planet2, asteroid;
    private List<SpriteGameObject> gameObjects;

    public GameWorld(ContentManager Content) {
        asteroid = Content.Load<Texture2D>("asteroid");
        planet1 = Content.Load<Texture2D>("planet1");
        planet2 = Content.Load<Texture2D>("planet2");

        gameObjects = new List<SpriteGameObject>();
        gameObjects.Add(new SpriteGameObject(Content.Load<Texture2D>("background"), Vector2.Zero));
        readLevel("Content/level.txt");

        // TO DO: constructor uitbreiden (b)

        gameObjects.Add(new Ufo(Content.Load<Texture2D>("ufo")));
    }

    // TO DO: readLevel methode (a)

    public void Update(GameTime gameTime) {
        foreach (SpriteGameObject obj in gameObjects)
            obj.Update(gameTime);
    }

    public void Draw(GameTime gameTime, SpriteBatch spriteBatch) {
        foreach (SpriteGameObject obj in gameObjects)
            obj.Draw(gameTime, spriteBatch);
    }

    public List<SpriteGameObject> GameObjects {
        get { return gameObjects; }
    }
}
```

Asteroid

```
class Asteroid : SpriteGameObject
{
    public Asteroid(Texture2D geladenSprite, Vector2 pos)
        : base(geladenSprite, pos) {
        this.velocity.X = (float)Asteroids.Random.NextDouble() - 0.5f;
        this.velocity.Y = (float)Asteroids.Random.NextDouble() - 0.5f;
        this.velocity *= (30.0f * (float)Asteroids.Random.NextDouble());
    }

    public override void Update(GameTime gameTime) {
        this.velocity *= 1.00001f;
        // TO DO: positie berekenen (c)
    }
}
```

GAME OVER