Derde deeltentamen Gameprogrammeren
Vrijdag 8 november 2013, 8.30-10.30 uur

Naam:	
Studentnummer:	

- Het tentamen bestaat uit 3 opgaven. Opgaven 1 levert 10 punten op, opgave 2 levert 10 punten op, en opgave 3 levert 20 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!
- Het is niet toegestaan om boeken, aantekeningen of ander materiaal te gebruiken, met uitzondering van de lijst met standaardklassen, -methoden, en -properties. Deze lijst 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) *(1.5 punt)* Wat is een edit state? Waarvoor wordt een edit state gebruikt in games?

   (b) *(3 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.
   - ☐ Een abstracte klasse is een klasse die geen membervariabelen, en implementaties van properties en methoden bevat.
   - ☐ Een klasse mag meerdere interfaces implementeren, maar mag niet erven van meerdere abstracte klassen.
   - ☐ Objecten van het type **string** zijn immutable.
   - ☐ Botsingen tussen game objecten worden afgehandeld in de Draw methode, want we weten pas of objecten botsen nadat ze getekend zijn.

   (c) *(1.5 punt)* Voor wat voor soort fouten worden excepties gebruikt in een programma? Geef een concreet voorbeeld binnen de context van games.

(d) *(4 punten)* Hieronder staan vier programmafragmenten. Geef aan wat de waarde van de variable teller is na het uitvoeren van elk programmafragment.

1	**int** teller = 0;
	for (**int** i = 0; i < 10; i++)
	{
	  if (i % 3 == 0)
	  continue;
	  teller++;
	}	

2	**int** teller = 0;
	for (**int** i = 0; i < 10; i++)
	{
	  if (i % 3 == 0)
	  break;
	  teller++;
	}	

3	**int** teller = 0;
	while (teller < 10)
	  for (**int** i = 1; i <= 10; i++)
	  {
	  if (i % 4 == 0)
	  break;
	  teller++;
	  }	

4	**int** teller = 0;
	for (**int** i = 0; i <= 5; i++)
	  switch (i)
	  {
	  case 1: teller++; **break**;
	  case 2:
	  case 3: teller += 2; **break**;
	  case 4: teller−−; **break**;
	  default: teller *= 2; **break**;
	  }	

2. In de klasse **string** zit een methode Replace. Deze methode levert een nieuwe string op, waarin elk voorkomen van het character dat als eerste parameter wordt meegegeven, is vervangen door het character dat als tweede parameter wordt meegegeven. Bijvoorbeeld:

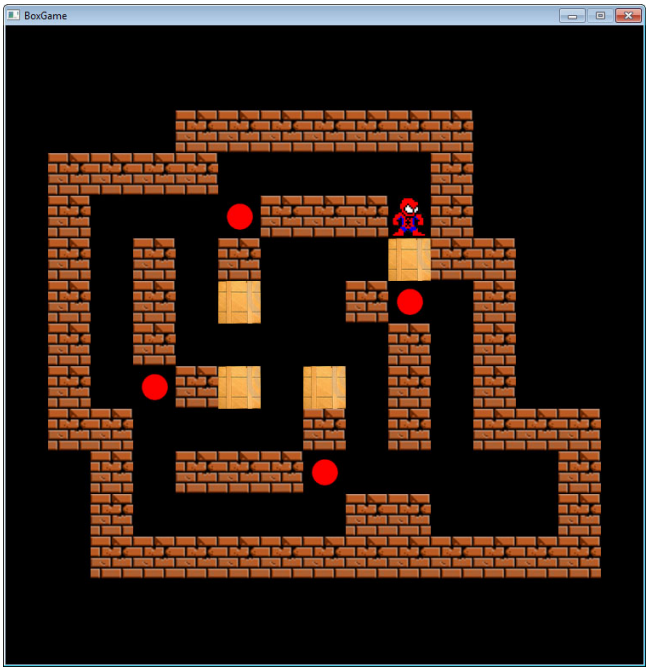
   ‘‘Utrecht”.Replace(t,x) geeft ‘‘Uxrechx”
   ‘‘A+2+#@?”.Replace(+,9) geeft ‘‘A929#@?”

   Stel dat je de auteur van de klasse **string** bent. Veel andere members van die klasse zijn al geschreven (die mag je dus gebruiken), maar nog niet de Substring en IndexOf methoden (die mag je dus niet gebruiken).

(a) *(5 punten)* Schrijf de methode Replace.

(b) *(5 punten)* Schrijf een statische methode met als parameter een array met strings, die de langste string van de array oplevert. Je mag aannemen dat de array minstens één string bevat. Als er meerdere strings zijn die even lang zijn mag je zelf kiezen welke je daarvan oplevert.

3. In deze opgave gaan we een 'box moving' puzzelspel maken. Het doel van het spel is om een aantal dozen te verschuiven zodanig dat ze op gegeven posities terechtkomen. Als speler kun je in een level bewegen met behulp van de vier pijltjestoetsen. Als je tegen een doos aanloopt, dan wordt deze één positie verschoven in dezelfde richting, behalve als aan de andere kant een muur staat. Als alle dozen op een doelpositie staan, dan is het level af. Een voorbeeld van een level is gegeven als volgt:



In dit level zijn vier dozen geplaatst die verschoven moeten worden naar de doelposities aangegeven met een stip. Om een voorbeeld te geven van hoe het spel reageert op spelersinvoer, laten we kijken wat er gebeurt als een speler de pijltjestoets omlaag indrukt. Onder de speler staat een doos, dus die wordt één positie naar beneden verschoven, oftewel, de doos komt op een doelpositie terecht. De speler komt terecht op de oude positie van de doos. Als de speler nu nog een keer op de pijltjestoets omlaag drukt, dan gebeurt er niks, want onder de doos staat nu een muur. Omdat aan de linkerkant van de doos nu ook een muur staan, kun je deze doos zelfs helemaal niet meer bewegen!

In ons spel worden levels geladen uit tekstbestanden. Het tekstbestand dat dit level omschrijft ziet er als volgt uit:

```
................
................
....WWWWWWWW....
.WWWW.....W....
.W...GWWWPW....
.W.W.W...BWW...
.W.W.B..WG.W...
.W.W.....W.W...
.W.GWB.B.W.W...
.WW....W.W.WWW.
..W.WWWG.....W.
..W.....WW...W.
..WWWWWWWWWWWW.
................
................
```

Hierbij staan de verschillende karakters voor verschillende soorten tiles in het spel:

- een 'W' (wall) staat voor een muurtile
- een 'G' (goal) staat voor ee doeltile
- een 'B' (box) staat voor een doostile
- een 'P' (player) staat voor de startpositie van de speler
- een '.' staat voor een lege tile

Een deel van de game is al uitgewerkt, zie de appendix. We gebruiken een klasse Tile om de verschillende tiles te representeren in het level. Deze klasse wordt voor het gemak ook gebruikt om de speler en de dozen voor te stellen. In de Level klasse laden we de tiles uit een bestand, en handelen we de verdere gameplay af. De klasse BoxGame is de klasse die erft van de XNA Game klasse, en die maakt een Level instantie, en roept de juiste methoden aan in de Update en Draw methoden.

(a) *(4 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. Daarnaast moet, als deze methode een tile karakter tegenkomt dat ongeldig is, een *exceptie* geworpen worden.

(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 een Vector2 object 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!

(c) *(4 punten)* Schrijf een methode IsGeldigePlaats die in de Level klasse staat. Deze methode kijkt of een bepaalde tile in het speelveld (positie aangegeven door een Vector2 object) leeg en toegankelijk is. Dit is het geval als er op die tile geen doos staat, het geen muur tile is, en de tile valt binnen het bereik van het speeldveld.

(d) *(6 punten)* De HandleInput methode in de Level klasse zorgt ervoor dat de speler verplaatst wordt als hij/zij op een van de pijltjestoetsen drukt. Natuurlijk gebeurt dit alleen als de speler ook daadwerkelijk op de nieuwe positie mag staan. Als er een doos op de doelpositie van de speler staat, dan moet eerst de doos verplaatst worden voordat de speler verplaatst wordt. 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 methoden VindDoos en IsGeldigePlaats die we in de vorige onderdelen 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!

(e) *(3 punten)* Een level is afgemaakt als alle dozen op een doelpositie staan. Implementeer een alleen-lezen property LevelAfgemaakt die aangeeft of een level afgemaakt is.

(f) *(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?

Appendix: klassen

BoxGame

```
class BoxGame : Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Level huidigLevel;
    InputHelper inputHelper;

    static void Main()
    {
        BoxGame spel = new BoxGame();
        spel.Run();
    }

    public BoxGame()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        graphics.PreferredBackBufferWidth = 750;
        graphics.PreferredBackBufferHeight = 750;
        inputHelper = new InputHelper();
    }

    protected override void LoadContent()
    {
        this.spriteBatch = new SpriteBatch(this.GraphicsDevice);
        huidigLevel = new Level(Content, "Content/level.txt");

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

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

Level

```
class Level
{
    Tile[,] tiles;
    Texture2D doosSprite, muurSprite, spelerSprite, doelSprite, levelAfSprite;
    List<Tile> dozen = new List<Tile>();
    Tile speler;
    private ContentManager Content;

    public Level(ContentManager c, string path)
    {
        this.Content = c;
        this.doosSprite = Content.Load<Texture2D>("doos");
        this.muurSprite = Content.Load<Texture2D>("muur");
        this.spelerSprite = Content.Load<Texture2D>("verhuizer");
        this.doelSprite = Content.Load<Texture2D>("doel");
        this.levelAfSprite = Content.Load<Texture2D>("levelAf");
        this.LaadTiles(path);
    }

    private void LaadTiles(string pad)
    {
        List<string> tekstregels = new List<string>();
        StreamReader fileLezer = new StreamReader(pad);
        string regel = fileLezer.ReadLine();
        int breedte = regel.Length;
        while (regel != null)
        {
            tekstregels.Add(regel);
            regel = fileLezer.ReadLine();
        }
        this.tiles = new Tile[breedte, tekstregels.Count];
        for (int y = 0; y < tekstregels.Count; ++y)
            for (int x = 0; x < breedte; ++x)
                this.tiles[x, y] = LaadTile(tekstregels[y][x], x, y);
    }

    // TODO (a): methode LaadTile

    // TODO (b): methode VindDoos

    // TODO (c): methode IsGeldigePlaats

    public void HandleInput(InputHelper inputHelper)
    {
                // TODO (d): body methode
    }

    // TODO (e): property LevelAfgemaakt

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

Tile

```
enum TileType
{
    Achtergrond, Muur, Doel, Doos, Speler
}

class Tile
{
    private Texture2D sprite;
    private TileType type;
    private Vector2 pos;
    private Vector2 size = new Vector2(50, 50);

    public Tile(Texture2D sprite, TileType type, int x, int y)
    {
        this.sprite = sprite;
        this.type = type;
        this.pos = new Vector2(x, y);
    }

    public TileType TileType
    {
        get { return this.type; }
    }

    public Vector2 Pos
    {
        get { return pos; }
        set { pos = value; }
    }


    public virtual void Draw(SpriteBatch spriteBatch)
    {
        if (sprite == null)
            return;
        spriteBatch.Draw(this.sprite, pos * size, Color.White);
    }
}
```

InputHelper

```
class InputHelper
{
    MouseState currentMouseState, previousMouseState;
    KeyboardState currentKeyboardState, previousKeyboardState;

    public void Update()
    {
        previousMouseState = currentMouseState;
        previousKeyboardState = currentKeyboardState;
        currentMouseState = Mouse.GetState();
        currentKeyboardState = Keyboard.GetState();
    }

    public Vector2 MousePosition
    {
        get { return new Vector2(currentMouseState.X, currentMouseState.Y); }
    }

    public bool MouseLeftButtonPressed()
    {
        return currentMouseState.LeftButton == ButtonState.Pressed &&
               previousMouseState.LeftButton == ButtonState.Released;
    }

    public bool KeyPressed(Keys k)
    {
        return currentKeyboardState.IsKeyDown(k) &&
               previousKeyboardState.IsKeyUp(k);
    }

    public bool IsKeyDown(Keys k)
    {
        return currentKeyboardState.IsKeyDown(k);
    }
}
```

GAME OVER