

1 Inleiding

Bozels hebben maar één doel. Nee, dat is niet waar, maar ze hebben wel maar één levensmissie en dat is om al hun doelen te vernietigen. Het is jouw taak om hen daarbij te helpen.

De bedoeling van het project dit jaar is om een programma te schrijven dat kan gebruikt worden om de parameters voor het spel Bozels in te stellen en te testen.

Voor dit project moet je een Java-applicatie schrijven. Het voornaamste werk zal draaien rond één hoofdvenster, waarin via tabbladeren de parameters voor het spel Bozels kunnen ingesteld worden en waarin het spel getest kan worden. De applicatie wordt gestart door de main methode van de klasse `bozels.Bozels` uit te voeren.

Hou steeds de website van dit vak (<http://twizz.ugent.be/student/prog2>) in de gaten. Er zullen hier tips, verduidelijkingen en eventuele verbeteringen van de opgave verschijnen. Op de site zijn er nu al demovideo's te vinden.

2 Spelbeschrijving

Een typisch Bozels level bevat drie soorten objecten:

- minstens één bozel,
- minstens één doel en
- een vrij aantal decorstukken.

Het doel van het spel is het vernietigen van de doelen door het afschieten van de bozels. De doelen bezwijken als ze geraakt worden door een bozel of als ze te zware klappen incasseren van decorstukken die omvallen. Als er na het sneuvelen van de laatste bozel nog doelen bestaan, dan verliest de speler het spel.

2.1 Decor

Er zijn verschillende soorten decor. Deze verschillen zowel qua voorkomen als fysische eigenschappen.

- **Beweegbare objecten:** Dit soort objecten staat aan het begin van een level stil maar kan beginnen bewegen bij botsing met een bozel of andere decorstukken. Een beweegbaar object bestaat uit een bepaald materiaal:
 - ijs,
 - hout,
 - steen of

- metaal.

Elk materiaal heeft verschillende eigenschappen:

- kleur,
- dichtheid (kg/m^3) (hoe zwaar een object is per grootte),
- restitutie $[0 \dots 1]$ (de elasticiteit bij botsingen),
- wrijving $[0 \dots 1]$ (hoe goed het materiaal schuift),
- sterkte (N/kg) en
- krachtdrempel (N).

De laatste twee eigenschappen zijn belangrijk om de breekbaarheid te modelleren. Bij een botsing oefenen twee objecten een (even grote, gemeten in Newton $[\text{N}]$) kracht op elkaar uit. Een object breekt als de som van alle krachten groter dan de krachtdrempel, groter is dan de materiaalsterkte maal de massa van het object. Voor elk object moet dus bijgehouden worden welke krachten het te verduren kreeg (behalve te kleine krachten); als de som ervan te groot wordt, dan breekt het object. Er zijn andere manieren om schade te modelleren, maar in dit project zullen we het daar bij houden.

- **Vaste objecten:** Deze objecten zijn stationair en onbreekbaar (bijv. het grondvlak). Vaste objecten bestaan ook uit een bepaald materiaal; de eigenschappen dichtheid, sterkte en krachtdrempel hebben evenwel geen betekenis.

Hieronder, in Tabel 1, staat per materiaal een overzicht van de standaardwaarden voor elke van die eigenschappen.

2.2 Doelen

De objectencategorie doelen is weinig anders dan die van de beweegbare decorstukken. Het verschil is dat

- doelen onmiddellijk breken als ze in contact komen met een bozel en
- het breken van alle doelen het spel beëindigt.







De eigenschappen van de twee soorten doelen zijn weergegeven in Tabel 1.

2.3 Bozels

Na het starten van een level zijn alle bozels zichtbaar aanwezig maar niet onderhevig aan de natuurwetten (gebruik `setActive`). Eén voor één en in volgorde wordt elke bozel naar de lanceerplaats gebracht. De speler schiet de bozel af in een bepaalde richting. Pas op dat moment wordt de bozel een actieve deelnemer van het spel. Zodra de bozel niet meer beweegt (gebruik `isAwake`), wordt ze verwijderd en begint de volgende bozel aan zijn avontuur. Klikte de speler tussen het lanceren en het verwijderen nogmaals op de muisknop, dan doet de bozel iets speciaals. Dit kan slechts één keer: kliks die volgen, hebben geen effect. Er zijn verschillende soorten bozels, elk met hun eigen materiaaleigenschappen en specialiteit:

- **rood:** doet niets
- **blauw:** doet niets
- **geel:** de huidige snelheid verdriedubbelt (gebruik `{get,set}LinearVelocity`)
- **wit:** explodeert met een zekere kracht F_m

De kleur, vorm en fysische eigenschappen van de verschillende bozels staat hieronder in Tabel 1.

	Dichtheid	Restitutie	Wrijving	Kleur/Vorm	Krachtdrempel	Sterkte
Vast	-	0.1	1.0	zwart	-	-
Beton	4.0	0.0	0.9	grijs	20.0	50.0
Hout	0.75	0.3	0.8	bruin	10.0	12.0
Metaal	3.0	0.2	0.3	groenblauw	18.0	52.0
Ijs	1.0	0.0	0.1	blauw	15.0	10.0
Klein	1.0	0.1	0.9	 $z : 2.5$	7.0	10.0
Groot	1.0	0.1	0.9	 $z : 4$	5.0	10.0
Rood	10.0	0.3	0.9	 $z : 2$	-	-
Blauw	8.0	0.7	1.0	 $r : 1$	-	-
Wit	5.0	0.0	0.2	 $z : 2$	-	-
Geel	10.0	0.1	0.9	 $h : 1, b : 2$	-	-

Tabel 1: Een overzicht van de standaardwaarden, kleuren en vormen voor de verschillende materialen, doelen en bozels. Voor de verschillende vormen staat z voor zijde, r voor straal, h voor hoogte en b voor breedte.

2.4 Lanceerinrichting

Het lanceermechanisme werkt als een eenvoudige katapult. Een bozel begint op locatie $\vec{r}_0 = (x_0, y_0)$. De speler sleept de bozel weg naar locatie $\vec{r}_1 = (x_1, y_1)$. Tussen \vec{r}_0 en \vec{r}_1 bevindt er zich een denkbeeldige elastiek. De lengte van de elastiekvector \vec{e} en dus ook de uitwijking van de bozel, moet beperkt worden tot 7 meter.

$$\vec{r}_d = \vec{r}_0 - \vec{r}_1$$

$$\vec{e} = \min(|\vec{r}_d|, 7) \frac{\vec{r}_d}{|\vec{r}_d|}$$

Als de speler de muisknop loslaat, dan wordt de bozel gelanceerd. Concreet gebeurt dit door een kracht \vec{F}_l te laten inwerken op de bozel. De kracht heeft dezelfde richting als \vec{r}_d en de grootte hangt af van hoe ver de elastiek werd uitgerokken ($|\vec{e}|$). Als de elastiek tot zijn maximum werd uitgerokken, dan is de grootte van de kracht L Newton.

$$\vec{F}_l = L \frac{|\vec{e}|}{7} \frac{\vec{r}_d}{|\vec{r}_d|}$$

In bovenstaande formule is de laatste factor de genormaliseerde versie van \vec{r}_d , berekenbaar met de methode `normalize` van de klasse `Vec2`. De lengte van zo'n vector kan berekend worden met de methode `length`.

3 Bestandsformaat

We voorzien jullie van een aantal levels waarmee getest kan worden. Het bestandsformaat van die levels is gebaseerd op XML. Het wortelelement heet `level` en bevat kinderen van vier verschillende types:

- `block` met attributen
 - `material` $\in \{\text{wood, ice, stone, metal, solid}\}$

- `angle`
- `width`
- `height`
- ellipse met dezelfde attributen als `block`
- bozel met attributen
 - `type` $\in \{\text{red, blue, yellow, white}\}$
 - `id` rangnummer, begint op 1
- `target`
 - `type` $\in \{\text{small, big}\}$

Alle elementen hebben twee gedeelde attributen: `x` en `y`, die de locatie van het middelpunt van het object vastleggen. De positie van de bozel met rang 1 legt ook de positie van de lanceerinrichting vast.

Je moet verplicht gebruik maken van JDOM om deze levelbestanden in te lezen en te parsen.

Een grondobject is niet voorzien in de levelbestanden. Er wordt verondersteld dat er een grondvlak is waarvan de bovenrand samenvalt met $y = 0$. Zorg er voor dat dit grondvlak breed genoeg is: objecten die er van sukkelen, blijven eeuwig doorvallen.

4 JBox2D

Bij het implementeren van dit project gaan we gebruik maken van de bekende *physics engine* Box2D. In plaats van zelf de nodige natuurkunde te implementeren, maken we gebruik van de vrij verkrijgbare bibliotheek JBox2D, de Java port van het originele Box2D. Hieronder wordt beschreven hoe je moet gebruik maken van JBox2D. De bibliotheek is een getrouwe vertaling van het origineel: extra informatie kun je vinden in de manual van Box2D en de API van JBox2D is beschikbaar op de website van dit vak.

4.1 Wereld

Alle fysische objecten behoren tot een bepaalde wereld: een object van het type `World`. Zo'n wereld kan eenvoudig aangemaakt worden met twee parameters: een vector die de zwaartekracht bepaalt en een logische waarde die aangeeft of objecten in rust kunnen zijn. Om de prestaties te verbeteren, kies je best `true` voor de tweede parameter. De zwaartekracht kun je opgeven in de vorm van een `Vec2`, een klasse die een 2D vector voorstelt. Aangezien de eenheden in Box2D overeenkomen met de eenheden uit de realiteit (meter, kilogram, seconde), wordt een werkelijkheidsgetrouwe wereld aangemaakt als volgt:

```
World world = new World(new Vec2(0f, -9.81f), true);
```

4.2 Objecten

Een wereld bevat een aantal objecten die zich volgens de natuurwetten gedragen. Het aanmaken van zo'n object begint met het specificeren van de begin eigenschappen in een `BodyDef` object. Deze klasse bevat een aantal publieke velden die ingesteld kunnen worden.

```
BodyDef bodyDef = new BodyDef();
bodyDef.position.set(new Vec2(2f, 3f));
bodyDef.angle = (float) Math.PI / 2;
bodyDef.type = BodyType.DYNAMIC;
```

Bovenstaand fragment legt de positie van het middelpunt vast op (x=2m, y=3m), veroorzaakt een beginrotatie (rond het middelpunt) van 45° en geeft aan dat het object een beweegbaar object is. Om onbeweeglijke objecten te maken zoals de grond en ander vast decor, moet **BodyType.STATIC** gebruikt worden. Er zijn nog andere instellingen voorhanden in de klasse **BodyDef**, maar dit zijn de voornaamste.

Na het vastleggen van de basiseigenschappen kan het eigenlijke object aangemaakt worden. Dit wordt gedaan door de wereld zelf in de methode **createBody**:

```
Body body = world.createBody(bodyDef);
```

Tot nu toe heeft het object vorm noch massa. Dit wordt verholpen door aan een **Body** een of meerdere *fixtures* te hechten. Er moet via een **FixtureDef** object opgegeven worden welke vorm, dichtheid, wrijvingscoëfficiënt, etc. zo'n fixture heeft.

```
CircleShape circle = new CircleShape();
circle.m_radius = 2.5f;

FixtureDef fDef = new FixtureDef();
fDef.shape = circle;
fDef.density = 5;
fDef.friction = 1;

Fixture fixture = body.createFixture(fDef);
```

Bovenstaand fragment voegt een cirkelvormige fixture (met straal=2.5m) toe aan **body**. De dichtheid wordt ingesteld op 5 kg/m² en de wrijving op het maximum (minimum, i.e. geen wrijving: 0).

Om objecten met een veelhoekvorm te maken, kun je gebruik maken van **PolygonShape**. Deze klasse heeft een methode **set** waarmee je de coördinaten van de hoekpunten kunt opgeven. Let bij het gebruiken van deze methode op de volgorde van de hoekpunten: overloop de hoekpunten van het object in tegenwijzerzin. Onderstaand voorbeeld maakt een gelijkzijdige driehoek met hoogte **h**. De coördinaten zijn zodanig gekozen dat het zwaartepunt van de driehoek op (0,0) valt.

```
float z=(float) (h * Math.sqrt(4.0 / 3.0)); //lengte zijde
PolygonShape triangle = new PolygonShape();
triangle.set(new Vec2[]{
    new Vec2(0f, h * 2f / 3f),
    new Vec2(-z / 2f, -h / 3f),
    new Vec2(z / 2f, -h / 3f)
}, 3);
```

Wil je een rechthoekig object maken, gebruik dan de methode **setAsBox**. Onderstaand voorbeeld maakt een rechthoekvorm met breedte 2hw en hoogte 2hh:

```
PolygonShape rectangle = new PolygonShape();
rectangle.setAsBox(hw, hh);
```

4.2.1 Eigenschappen aanpassen

Deze structuur weerspiegelt zich ook in de manier waarop eigenschappen van een object aangepast kunnen worden tijdens het simuleren. Een kracht laten inwerken, een hoeksnelheid geven en gelijkaardige manipulaties kunnen rechtstreeks op een **Body** toegepast worden met de methoden **applyForce**, **setAngularVelocity**, etc. De setter die op het eerste zicht ontbreekt, is die waarmee de positie van een object kan veranderd worden. Daarvoor moet echter **setTransform** gebruikt worden: aan die methode moet een nieuwe positie (voor het middelpunt) en rotatiehoek gegeven worden. Het is ook mogelijk om objecten te deactiveren (d.w.z. zijn niet onderworpen aan de natuurwetten) met **setActive**.

Materiaaleigenschappen veranderen van een object gebeurt via de fixture(s) van de **Body**. De methode **getFixtureList** geeft de eerste fixture van een object terug en die kan aangepast worden met setter-methoden. De volgende fixture kan je opvragen via de methode **getNext** van het fixture-object. Bij het aanpassen van de dichtheid (met **setDensity**) moet het **Body**-object op de hoogte gebracht worden met **resetMassData**. Voor de andere eigenschappen is zo iets niet nodig.

4.3 Simuleren

Hieronder staat een voorbeeldwereld waarin een elastische bal botst op de grond totdat alle energie verloren is gegaan. Om de simulatie te laten lopen, wordt **wereld.step** herhaaldelijk opgeroepen. De parameters zijn de tijdstap en twee getallen die bepalen hoe nauwkeurig de simulatie moet zijn. In het voorbeeld staan de waarden die aangeraden worden door de makers van Box2D.

```
//valversnelling: 9.81m/s/s
World wereld = new World(new Vec2(0f, -9.81f), true);

//maak een grondvlak

BodyDef grondBD = new BodyDef();
//middelpunt van grond is (0,0)
grondBD.position.set(new Vec2(0f, 0f));
//grond is een onbeweegbaar object
grondBD.type = BodyType.STATIC;

Body grond = wereld.createBody(grondBD);

//maak rechthoek 100mx1m
PolygonShape grondShape = new PolygonShape();
grondShape.setAsBox(50, 0.5f);

FixtureDef grondFD = new FixtureDef();
grondFD.shape = grondShape;

//verbind bovenstaande vorm met object grond
grond.createFixture(grondFD);

//maak een bal

BodyDef balBD = new BodyDef();
//middelpunt van bal is (0, 10m), dus 9.5m boven grond
balBD.position.set(new Vec2(0f, 10f));
//bal is een beweegbaar object
balBD.type = BodyType.DYNAMIC;

Body bal = wereld.createBody(balBD);

CircleShape balShape = new CircleShape();
//cirkel met een straal van 1 meter
balShape.m_radius = 1;

FixtureDef balFD = new FixtureDef();
balFD.shape = balShape;
```

```

//bal weegt 1kg/m^2 (=3.14 kg)
balFD.density = 1;
//met redelijk hoge wrijving
balFD.friction = 0.7f;
//maar grote elasticiteit
balFD.restitution = 0.8f;

bal.createFixture(balFD);

//—simulatielus—
while (bal.isAwake()) {
    //zolang de bal nog beweegt

    //neem een tijdstap van 1/60 seconden
    wereld.step(1f / 60f, 8, 3);

    Vec2 pos = bal.getPosition();
    System.out.println(pos.x + ", " + pos.y);
}

```

4.3.1 JBox2D en Swing

Een belangrijke regel voor goede spelfysica is dat de simulatie onafhankelijk moet zijn van de *framerate*. Een bal die kaatst op de vloer moet zich altijd op dezelfde manier gedragen, ongeacht of de videokaart dat aan 50 of 10 frames per seconde kan weergeven. De simulatielus hoort m.a.w. thuis in de *hoofdprogrammadraad* en kan er als volgt uit zien:

```

float dt = 1f / 60f;           //seconden
long dtm = (long) dt * 1000;   //milliseconds

while (!stop) {
    //simuleer 1/60 seconden
    wereld.step(dt, 8, 3);

    //vraag het paneel zich te hertekenen
    panel.repaint();

    try {
        //slaap 1000/60 milliseconds
        Thread.sleep(dtm);
    } catch (InterruptedException ex) {
        //dit gebeurt in principe niet
    }
}

```

Na het simuleren wordt telkens gevraagd aan het paneel of het zich wil hertekenen. Dit zal gebeuren in de *gebeurtenisverwerkingsdraad* zodra daarvoor tijd is. Is het paneel niet in staat snel genoeg te tekenen, dan worden er frames overgeslagen; de simulatie blijft echter gewoon lopen.

4.4 Botsingen

De klasse `World` biedt de mogelijkheid om een `ContactListener` te registreren. Dit object wordt op de hoogte gebracht van botsingen tussen objecten. Stel dat we bovenstaand voorbeeld willen uitbreiden met een rapportering van de botskracht. We voegen het volgende fragment net voor de lus toe:

```

wereld.setContactListener(new ContactListener() {

    protected boolean show;

```

```

public void beginContact(Contact cnt) {
    show=true;
}

public void postSolve(Contact cnt, ContactImpulse imp) {
    if(show) {
        System.out.println("Impulse:␣" + imp.normalImpulses[0]);
        show=false;
    }
}
...
});

```

Telkens de bal in contact komt met de grond, wordt `beginContact` opgeroepen. Er zijn maar twee objecten, dus we moeten de inhoud van `cnt` hier niet controleren. De methode `postSolve` wordt na elke tijdstap opgeroepen, zolang de bal de grond raakt. Enkel de eerste keer wordt de *stoot* (=kracht x tijd, Engels: impulse) waarmee dit gebeurt, uitgeschreven. Aangezien we hier werken met een rond object, zal enkel de eerste waarde van `normalImpulses` verschillend zijn van nul. Bij een botsing van twee `PolygonShape`-objecten zijn er mogelijks twee contactpunten en evenveel stootwaarden.

4.4.1 Breken van objecten

Op basis van deze krachtinformatie kan er voor gekozen worden objecten te laten breken tijdens de simulatie. Uit de stootgrootte (kg·m/s) kan de krachtgrootte ($N := kg \cdot m/s^2$) berekend worden door te delen door de lengte van de tijdstap. Een object breekt onder invloed van krachten: ofwel enkel in één klap die te hevig is voor het materiaal of ook incrementeel na meerdere botsingen. Een eenvoudige manier om dat te bereiken is de geïncasseerde krachten (groter dan de krachtendrempel) optellen. Is de som groter dan de materiaalsterkte (N/kg) maal de massa (kg), dan breekt het object.

Een eenvoudige modellering van breken, is het laten verdwijnen van het object. Gebruik daarvoor de methode `destroyBody` van `World`. Probeer dit echter niet te doen tijdens de simulatie (bijv. in de `ContactListener`), enkel tussen twee `step`-oproepen mag je objecten verwijderen en toevoegen.

4.5 Ray casting

Een andere feature van Box2D die je zult nodig hebben, is *ray casting*. Stel dat je een realistische ontploffing wilt implementeren. Je moet kunnen achterhalen welke objecten in de buurt van de bom weggeslingerd moeten worden. De methode

```
void raycast(RayCastCallback callback, Vec2 p1, Vec2 p2);
```

van `World` zoekt alle objecten tussen `p1` en `p2` en rapporteert ze (in onbekende volgorde) via `callback`. Zo'n callback-object heeft maar één methode: `reportFixture`. De onderstaande implementatie ervan laat een kracht inwerken op het object dat dichtst bij de bom gevonden wordt. De richting van de kracht wordt overgenomen van de vector tussen de bom en het raakpunt `p`, de grootte is omgekeerd evenredig met de afstand ertussen.

```

public class ExplosionRayCastCallback implements RayCastCallback {
    private Vec2 bomb;
    private Vec2 hitpoint;
    private Fixture closest;
    private float minf = 1.0f;
}

```



```

public ExplosionRayCastCallback(Vec2 bomb) {
    //kopieer de locatie van de bom
    this.bomb = bomb.clone();
}

public float reportFixture(Fixture fix, Vec2 p, Vec2 n, float f) {

    // f is een getal tussen 0 en 1 dat aangeeft waar op de ray
    //deze hit gevonden werd. We willen enkel de dichtste hit.
    if (f < minf) {

        minf = f;
        closest = fix;
        //expliciet het contactpunt kopiëren!
        hitpoint = p.clone();
    }

    //beperk deze ray tot hits dichtere dan deze hit
    return f;
}

//deze methode moet opgeroepen worden nadat raycast() voltooid is
public void explode() {

    final float maxForce = 10000f;

    //richting van de kracht = vector tussen bom en raakpunt p
    Vec2 fDir = hitpoint.sub(bomb);

    //bepaal de afstand tussen bom en raakpunt
    //en maak fDir lengte 1 (dit gebeurt samen)
    float dist = fDir.normalize();

    //laat een kracht inwerken op het getroffen object
    //de richting is fDir, de grootte neemt af naarmate de afstand
    //groter wordt (maximaal voor dist=0)
    closest.getBody().applyForce(fDir.mul(maxForce / (dist + 1)), hitpoint);
}
}

```

Een geloofwaardige implementatie van een bom laat een voldoende groot aantal rays in alle mogelijke richtingen vanuit de bom vertrekken.

5 JBox2D en tekenen

De voorwerpen waarvan het fysisch gedrag gesimuleerd worden, verschijnen niet automagisch op het scherm. Het is aan de gebruiker van JBox2D om dat te implementeren in een raamwerk naar keuze. In het geval van dit project hebben we enkel de Java API voorhanden. Om eenvoudiger de brug tussen JBox2D en de klasse Graphics(2D) te kunnen slaan, geven we wat raad.

5.1 Schaalverschil

De makers van Box2D waarschuwen dat *iedereen die pixeleenheden gebruikt, neergeschoten zal worden*. Ze bedoelen daarmee dat een voorwerp dat op het scherm 500 bij 100 schermpunten groot moet zijn, in geen geval afmetingen (500, 100) mag hebben in JBox2D: de physics engine is gewoonweg niet geschikt voor voorwerpen van die grootte. Dit kan opgelost worden door een schaalverschil te creëren tussen het simuleren en het tekenen.

De voorbeeldlevels die jullie krijgen, bevatten objecten in *simulatieformaat*. Alle objecten passen netjes op een paneel van 1024x450 als ze 7 keer groter getekend worden. Met andere woorden: één meter in JBox2D komt overeen met 7 pixels op een Java canvas.

5.2 Tekenen van geroteerde objecten

Het tekenen van gedraaide objecten wordt eenvoudiger als je gebruik maakt van de methode `rotate` van `Graphics2D`. Met die methode kun je het tekenvlak draaien rond de oorsprong. Samen met de methode `translate` kun je draaien rond een arbitrair punt. Moet je bijv. twee identieke voorwerpen tekenen, de eerste met middelpunt (x_1, y_1) en rotatie a_1 en een tweede middelpunt (x_2, y_2) en rotatie a_2 , dan kun je het volgende doen:

```
//maak een duplicaat van het Graphics object, zodat we
//kunnen roteren etc. zonder het origineel te veranderen.
Graphics2D g2dp = (Graphics2D) g.create();
//verplaats tekencoordinaten (0, 0) naar waar ons object moet komen
g2dp.translate(x1, y1);
//draai het tekenvlak al radialen
g2dp.rotate(a1);
//teken de voorwerp alsof hij op (0,0) zou staat, met rotatie 0
teken_voorwerp(g2dp);

//teken de tweede voorwerp op dezelfde manier, maar met een andere translatie
//en rotatie van het tekenvlak
g2dp = (Graphics2D) g.create();
g2dp.translate(x2, y2);
g2dp.rotate(a2);
teken_voorwerp(g2dp);
```

6 Gebruikersinterface

De gebruikersinterface bestaat uit twee hoofddelen.

Bovenaan heb je het levelvenster. Dit deel van de interface toont het huidige level en kan gebruikt worden om het spel te spelen. Je zal dit deel van het venster volledig zelf moeten tekenen.

Het onderste gedeelte van de gebruikersinterface wordt gebruikt om de parameters in te stellen en om levels te (her)starten en over te slaan. Dit deel van de gebruikersinterface is opgebouwd met behulp van klassieke Swingcomponenten.

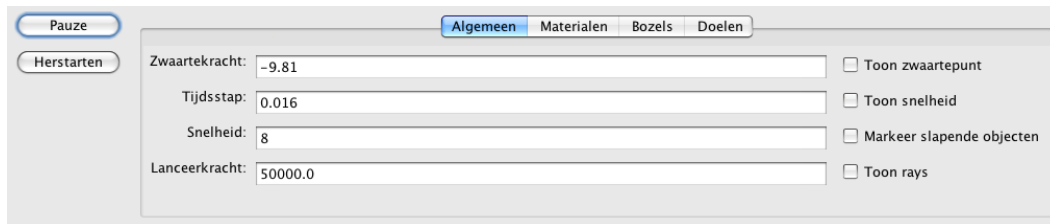
6.1 Het levelvenster

Hoe een level er moet uitzien werd reeds hierboven besproken.

6.2 De editor

In Figuur 1 zie je een afbeelding van de editor.

Aan de linkerkant bevinden zich twee knoppen boven elkaar. Deze knoppen controleren de simulatie in het bovenste gedeelte van het venster. De bovenste knop dient om het spel te pauzeren. In normale omstandigheden loopt het spel van zodra Bozels wordt opgestart. Met deze knop kan je dan vervolgens de simulatie pauzeren. De knop heeft twee toestanden waartussen geschakeld worden. Deze toestanden corresponderen met gepauzeerd en niet gepauzeerd. De onderste knop herstart het



Figuur 1: *De editor*

huidige level. Alle onderdelen van het spel worden terug in hun oorspronkelijke staat hersteld. De waarden die in de editor zijn gewijzigd, worden echter niet teruggezet.

Aan de rechterkant bevindt zich een paneel met verschillende tabbladeren. Het eerste tabblad correspondeert met enkele omgevingsinstellingen en de andere drie tabbladeren corresponderen met de drie verschillende objecttypen. In Figuur 2 zie je een overzicht van de verschillende tabbladeren. Naast veel gelijkenissen zijn er ook enkele verschillen. Hieronder leggen we je de betekenis van de verschillende instellingen uit. Voor meer details verwijzen we ook naar de bovenstaande uitleg over JBox2D en naar de video's op de website van dit vak.

Het eerste tabblad is heel verschillend van de andere drie. Het bevat vier tekstvelden die respectievelijk de zwaartekracht, de tijdstap voor de simulatie, de snelheid voor de simulatie en de kracht waarmee de bozels gelanceerd worden instellen. De tijdstap geeft aan hoeveel tijd (in seconden) er verlopen is in de simulatie als er één tijdstap wordt uitgevoerd. De snelheid geeft aan om de hoeveel tijd (in milliseconden) er één tijdstap wordt uitgevoerd. Daarnaast zijn er ook nog vier aankruisvakjes waarmee je de weergave van de scène kan aanpassen. In de uitleg over JBox2D werd reeds vermeld hoe je het zwaartepunt en de snelheid kan opvragen, hoe je de *rays* moet berekenen en kan te weten komen of een object slaapt. Het zwaartepunt toon je best als een punt of kruisje, de snelheid als een pijl en de *rays* als lijnen. De slapende objecten kan je bvb. in een vagere of zelfs transparante kleur tekenen.

Het derde tabblad lijkt goed op het tweede en het derde, maar het grote verschil is dat bozels nooit breekbaar zijn, dus deze eigenschappen kan je voor dit objecttype niet instellen. Helemaal links bevindt zich een lijst die een overzicht geeft van alle soorten van het corresponderende objecttype. Een gekleurd vierkant wordt getoond dat aangeeft welke kleur dit objecttype heeft in de scène. Wanneer je in deze lijst een andere soort selecteert, dan worden de tekstvelden aan de rechterkant bevolkt met de eigenschappen voor dit type. In de eerste kolom naast de lijst staan drie tekstvelden. Met deze tekstvelden kan je respectievelijk de dichtheid van het object, de restitutie en de wrijving aanpassen. Onder deze drie tekstvelden bevindt zich een knop waarmee een kleurdialoog geopend kan worden om de kleur van deze soort van objecten aan te passen. Behalve bij het tabblad bozels, kan je in een tweede kolom aangeven of objecten van deze soort breekbaar zijn. Bovenaan heb je een aankruisvakje, en daaronder heb je twee tekstvelden. Als het aankruisvakje geselecteerd is, dan telt elke kracht die groter is dan de krachtdrempel mee voor het totaal. Als de krachten die op een object inwerken dan groter zijn dan zijn sterkte, dan breekt het object.

De tekstvelden in de editor hebben een specifiek gedrag. Als er een waarde wordt ingevuld die verschillend is van de waarde die momenteel is ingesteld, dan wordt deze niet direct gewijzigd in het spel. Het tekstveld kleurt echter geel om aan te geven dat er een gewijzigde waarde in staat. Het tekstveld blijft ook geel als de cursor dit tekstveld verlaat. Als de gebruiker dan op Enter drukt terwijl de focus zich in dit tekstveld bevindt, wordt de waarde gewijzigd en krijgt het tekstveld terug de standaardachtergrond. Als er echter een foutieve waarde is ingevuld (i.e. geen kommagetal, behalve bij *Snelheid*, daar moet het een geheel getal zijn), dan kleurt het tekstveld rood en gebeurt er niets als de gebruiker op Enter drukt.

In Figuur 3 zie je hiervan een voorbeeld. In de bovenste afbeelding heeft de gebruiker de zwaartekracht verandert naar die van de maan, maar nog niet op Enter gedrukt. In de middenste afbeelding heeft de

Algemeen

Materialen

Bozels

Doelen

Zwaartekracht:
☐ Toon zwaartepunt

Tijdsstap:
☐ Toon snelheid

Snelheid:
☐ Markeer slapende objecten

Lanceerkracht:
☐ Toon rays

Vast...

Beton

Hout

Metaal

Ijs

Dichtheid:
☐ Breekbaar

Restitutie:
Krachtdrempel:

Wrijving:
Sterkte:

Kleur:

Rood...

Blauw

Wit

Geel

Dichtheid:

Restitutie:

Wrijving:

Kleur:

Groot...

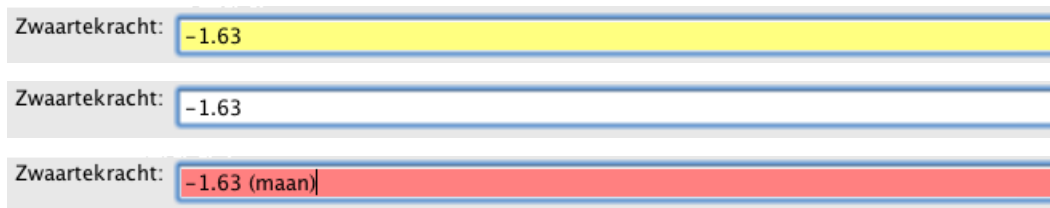
Dichtheid:
☒ Breekbaar

Restitutie:
Krachtdrempel:

Wrijving:
Sterkte:

Kleur:

Figuur 2: De verschillende tabbladeren van de editor.



Figuur 3: Een tekstveld als er een nieuwe waarde is ingevuld (boven), nadat er op enter is gedrukt (midden) en als er een foute waarde is ingevuld (onder).

<i>Bestand</i>	▷ <i>Level laden</i>	Toont een bestandsdialoog om een nieuw level te laden.
	▷ <i>Afsluiten</i>	Beëindigt het programma.
<i>Spel</i>	▷ <i>Pauzeren</i>	Pauzeert de simulatie net zoals de knop op de editor.
	▷ <i>Herstarten</i>	Herstart het huidige level net zoals de knop op de editor.

Tabel 2: Een overzicht van de verplichte menustructuur.

gebruiker dan op Enter gedrukt en is de waarde dus in het spel gewijzigd. In de onderste afbeelding probeert de gebruiker nog wat tekst toe te voegen aan dit veld, maar dit is dus niet toegelaten.

6.3 Menu

Tabel 2 geeft een overzicht van de verplichte menustructuur: er zijn twee menu's met elk twee menu-items. Het menu-item *Pauzeren* heeft twee toestanden waartussen geschakeld worden. Deze toestanden corresponderen met gepauzeerd en niet gepauzeerd.

7 Extra's

Natuurlijk zou een volledig afgewerkte versie van Bozels nog meer functionaliteit bevatten. Er zou allereerst ook de mogelijkheid zijn om de instellingen op te slaan naar en in te laden van een XML-bestand. De tekeningen zouden meer afgewerkt kunnen worden met zelfs de mogelijkheid om afbeeldingen te gebruiken om bepaalde objecten voor te stellen.

Dit zijn allemaal echter geen verplichte onderdelen van het project en je zal ook geen extra punten krijgen als je toch deze dingen aanbiedt. Zorg er dus eerst voor dat je project perfect werkt zoals hier beschreven staat en er gebruik gemaakt wordt van de technieken die in de lessen zijn aangeleerd en van de standaardconventies in Java vooraleer je deze extra's begint toe te voegen.

8 Indienen van het project

- Maak een tekstbestand *project.txt* Dit moet een gewoon tekstbestand zijn en dus geen Word-document of HTML-bestand. Deze tekst bevat extra gegevens over het ontwerp van het programma, de onderverdeling in klassen en hun onderlinge interacties. Je biedt hier een globaal overzicht dat de algemene structuur van de toepassing belicht.

Beschrijf welke componenten er door welke klassen worden voorgesteld, welke klassen er eventueel als “model” dienen (en voor welke views) en geef aan welke gebeurtenissen er door welke luisteraars worden opgevangen.

Geef eventueel extra informatie over andere eigenaardigheden van je programma die anders bij een eerste lezing niet te begrijpen zijn.

De hoofdbedoeling van deze tekst is de lezer een eerste weg te wijzen door de broncode van de toepassing. Het is belangrijk dat je duidelijk en overzichtelijk bent zonder al te veel detailbeschrijvingen of irrelevante opmerkingen. Verdere details schrijf je in commentaarlijnen van de klassen zelf.

- We bieden je een NetBeans-project aan. In dit project bevindt zich reeds de verplichte klasse `bozels.Bozels`, de bibliotheken `JBox2D` en `JDOM` en een bestand `project.txt` (in de map `etc`) waarin je bovenstaand vermelde tekst kan zetten. Het build-script van dit project is aangepast zodanig dat er automatisch een zipbestand `project.zip` wordt aangemaakt en in de map `dist` geplaatst wordt. Dit bestand wordt enkel aangemaakt als je Build kiest voor het project en niet als je gewoon Run selecteert. Dit bestand moet dan ingediend worden via <http://indiano.ugent.be>. Als je geen gebruik wenst te maken van NetBeans, dan verwachten we dat je ook zelfstandig in staat bent om een zipbestand te maken dat voldoet aan de punten omschreven in het volgende item. Naast de standaardstructuur van een NetBeans-project bevat het meegeleverde project dus het volgende:

- `project.txt` in de map `etc`;
- voorbeeldlevels in de map `etc`;
- `JBox2D` en `JDOM` in de map `lib`, maar ook reeds ingesteld als bibliotheken voor het project;
- klasse `Bozels` met `main`-methode in het package `bozels`;
- aangepast build-script.

- Als je het zipbestand indient, wordt het gecontroleerd op de geldigheid:
 - Het bestand `project.txt` bevindt zich in de rootmap van het zipbestand.
 - Het klassepad bevindt zich tevens in de rootmap en de mainklasse van de applicatie is `bozels.Bozels`.
 - Het zipbestand bevat geen class- of jar-bestanden.

Vervolgens wordt het programma gecompileerd waarbij de bibliotheken `JBox2D` en `JDOM` in het klassepad worden gezet. Als één van deze stappen mislukt, krijg je hier ook meteen feedback over.

- Bij het indienen op Indianio worden ook enkele stijlregels gecontroleerd zoals klassenamen met hoofdletters, variabelenamen met kleine letters, ... Dien dus zeker op tijd in zodat je nog tijd hebt om eventuele kleine vormfouten in je oplossing aan te passen.
- Het project moet worden ingediend ten laatste op maandag 7 mei om 8u30. Je kan echter zo vaak indienen als je wilt: we kijken enkel naar de laatste versie. Wacht dus ook niet tot 5 minuten voor de deadline om in te dienen. Als je tijdig probeert in te dienen, heb je nog voldoende tijd om eventuele fouten in je bestand op te lossen.
- Werk verplicht met een packagestructuur!
- Test zelf grondig of de versie die je wil indienen voldoet aan alle opgelegde specificaties!
- Elk bestand (.java-bestand, .xml-bestand, .properties-bestand, ...) dat je indient, moet ergens bovenaan in commentaar je **NAAM** en **VOORNAAM** bevatten.

9 Beoordeling

We hanteren o.a. onderstaande criteria bij de beoordeling van het project:

- Er zal gecontroleerd worden of het project op **strikt individuele** basis gemaakt is; elk gebruik van code van andere studenten zal dan ook bij alle betrokken studenten worden **bestraft**.
- De GUI moet verzorgd en overzichtelijk zijn.
- Er worden punten afgetrokken voor projecten die te laat werden ingediend of niet voldoen aan de vooropgestelde eisen en/of specificaties.
- Projecten die niet compileren, worden niet verbeterd!
- Het programma mag geen fouten bevatten: het moet foutloos compileren en het mag niet crashen bij de uitvoering. Jullie programma's worden onder Linux getest. Gebruik dus waar nodig het klassepapier om toegang te krijgen tot bepaalde bronnen.
- De broncode moet leesbaar en overzichtelijk zijn en de toepassing moet een overzichtelijke structuur hebben. Plaats commentaar in de broncode waar nodig, maar overdrijf ook niet. Verwijder alle code die niet (meer) wordt gebruikt, ook al staat ze tussen commentaarhaken.
- Programmeer steeds met efficiëntie in het achterhoofd.
- Programmeer helder en met stijl, gebruik voldoende klassen en methodes (in plaats van knip-en plakwerk), ...
- Indien dit nodig zou blijken, kan u gevraagd worden om uw project te komen toelichten.

10 Beschuwing

We doen beroep op je creativiteit. Dus, betast het probleem, bedenk en probeer manieren om dat probleem aan te pakken. Vertrouw hierbij op je eigenwaarde als creatieveling. Wij als lesgevers zijn animators, inspirators en trainers. Animators die je enthousiast willen maken, je de uitdaging laten aanvoelen tot het ook bij jou begint te jeuken. Inspirators die je leerstof aanreiken met technieken, concepten, Trainers die je expertise bijbrengen volgens het *learning by example*-principe gecombineerd met de Socratische *ontdek je eigen kunnen*-methodiek. Ook bespreking met collega's (uitwisseling van ideeën en ervaringen, relaas van experimenten en probeersels) kan verrijkend zijn. Laat je echter niet verleiden tot dom kopieerwerk maar maak er een eigen werk van waarop je fier bent en waarvan jezelf beter wordt.

Toon je langs je beste kant in een mooi afgewerkt project waaruit blijkt dat je zelf de expertise verworven hebt om dit probleem zelfstandig aan te pakken op een vakkundige en creatieve manier. Je mag dus gerust het warm water opnieuw uitvinden - dat wordt trouwens van jou verwacht - want waarschijnlijk zijn er geen 1001 fundamenteel verschillende manieren om dit probleem op te lossen. Wel zijn er meer dan 1001 verschillende manieren om eenzelfde oplossingsconcept te realiseren. En vooral: denk eerst na vooraleer blindelings te beginnen.

Bezint eer ge begint !

(Bron: Walter Bossaert)

Veel succes!!
Davy & Nico
Maart 2012