

Brugervejledning:

Formålet med denne kode er give nybegyndere et simpelt værktøj til collision control af objekter. Det er tiltænkt til at skulle bruges til 2D arkade spil.

Redigering af PhysicsEngine

Vi opfordrer brugere til at modificere vores kode til at matche behov. Hvis der menes at der fundet en forbedring til koden, er man velkommen til at kontakte kajense@ruc.dk for at få github adgang eller sende koden direkte.

Forudsætninger:

Denne 2D Physics Engine er lavet til Processing brugere og har derfor behov for at have Processing installeret, versionen af processing benyttet er 3.3.6 og selvom at det ikke anses som et problem at bruge tidligere versioner kan vi ikke garantere fuld funktionalitet ved brug af disse.

Installering af Processing kan gøres fra <https://processing.org/download/>

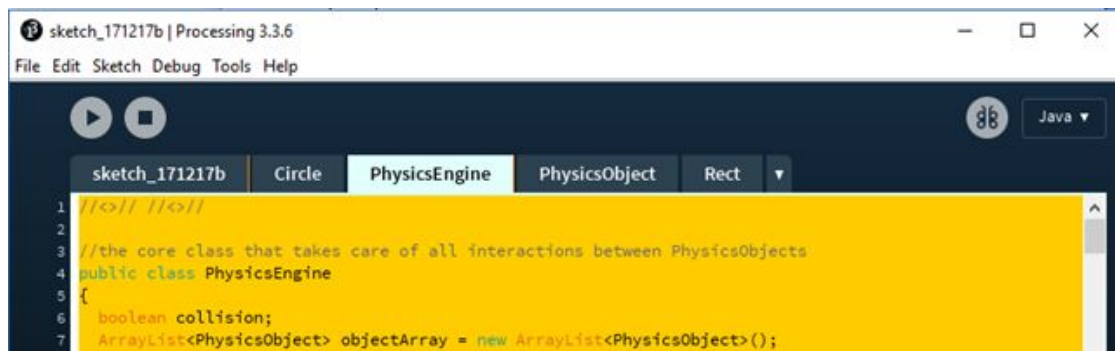
Herefter åbnes PhysicsEngine.pde som enten er medfølgende denne brugervejledning eller kan downloades på: <https://github.com/Henhunter/PhysicsEngine2D/>

Indholdet af PhysicsEngine.pde kan nu copy-pastes til ny tab i eget program i Processing, således at den bliver en del af det program der ønsket lavet.



```
1 //|| ||||
2
3 //the core class that takes care of all interactions between PhysicsObjects
4 public class PhysicsEngine
5 {
6     boolean collision;
7     ArrayList<PhysicsObject> objectArray = new ArrayList<PhysicsObject>();
```

eller yderligere opdeling i tabs kan gøres, hvis man påtænker at redigere i koden for overskuelighedens skyld.



Anvendelse:

Der er to forskellige måder hvorpå denne brugervejledning kan anvendes, den ene er hvis du er på workshopen Noob to Master og ønsker yderligere forståelse for vores program, i sammenhæng med undervisningen, den anden er hvis du som læser finder interesse i vores program og ønsker en mindre teknisk gennemgang af vores produkt. I begge scenarier, har denne vejledning formålet, at afklare, afdække og skabe forståelse for vores Physics engine, på et mindre teknisk niveau. I denne vejledning vil følgende områder af programmet blive dækket:

- Objektet PhysicsEngine.
- Objekter med fysiske parametre.
- Metoderne omhandlende kollision.
- Eksempel på hvordan en "Custom class" laves.
- Eksempel på reaktion efter kollision.
- Gennemgang af standardværdier i programmet og hvordan de ændres.
- Gennemgang af Booleans

Foruden denne vejledning, er der også skrevet kommentarer i projektet under hver metode, der kort forklarer hvad den gør og dens formål. Dette anbefaler vi også at kigge på, for at skabe en helhed og større forståelse for programmet.

Start opsætning

Aktivering af objektet PhysicsEngine.

Det er nødvendigt at tilføje PhysicsEngine objektet for at kunne tilgå dens funktioner, dette gøres ved at deklarere en variabel og skabe et nyt objekt, denne variabel peger på.

PhysicsEngine PE = new PhysicsEngine();

Skabelse af objekt med fysiske parametre.

Der kan håndteres to forskellige geometriske former, cirkel og rektangler. Disse skal ligesom PhysicsEngine skabes, før de kan bruges.

Circle har behov for 5 parametre for at bestemme dens plads i programmet. Den skal bruge x position, y position, diameter, vertikal hastighed og horizontal hastighed.

Circle circle = new Circle(posX, posY, diameter, velocityX, velocityY);

Rektangel har behov for 6 parametre, x position, y position, vidde, højde, vertikal hastighed og horizontal hastighed.

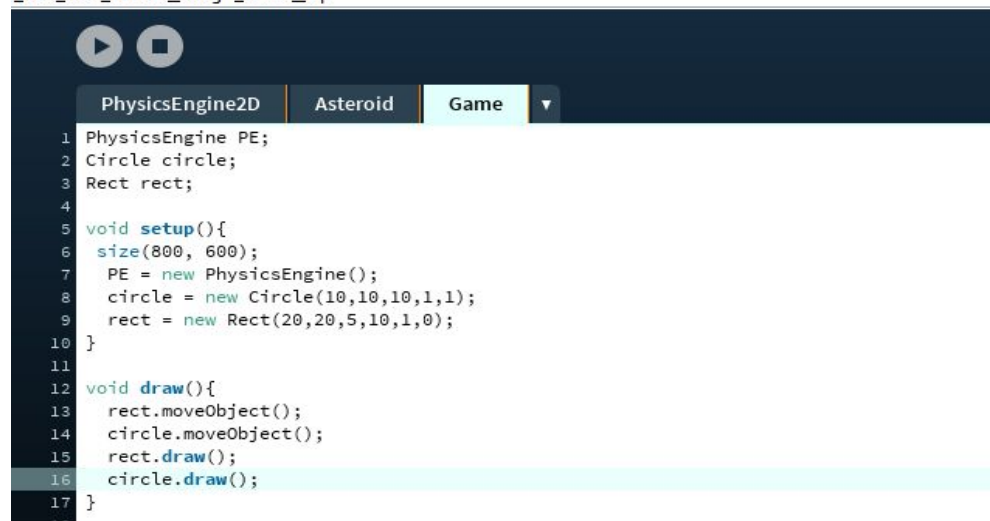
Rect rect = new Rect(posX, posY, Width, Height, velocityX, velocityY);

Både Rect og Cirkel extender PhysicsObjects, hvilket betyder at alle Rect og Cirkel kan bruges som parametre i en metode, der tager PhysicsObjects. For at tegne disse objekter benyttes deres egen tegne metode og for at flytte objecterne bruges moveObject metoden.

Rect.draw();

Rect.moveObject;

PhysicsEngine2D | Processing 3.3.6
File Edit Sketch Debug Tools Help



```
1 PhysicsEngine PE;
2 Circle circle;
3 Rect rect;
4
5 void setup(){
6   size(800, 600);
7   PE = new PhysicsEngine();
8   circle = new Circle(10,10,10,1,1);
9   rect = new Rect(20,20,5,10,1,0);
10 }
11
12 void draw(){
13   rect.moveObject();
14   circle.moveObject();
15   rect.draw();
16   circle.draw();
17 }
```

Ex. på ovenstående.

Yderligere funktionalitet

Collisionsmuligheder:

Vi tilbyder to forskellige muligheder for at teste om to objekter kolliderer. Den manuelle mulighed er at bruge en af PhysicsEngines metoder til at kontrollere om to specifikke objekter kolliderer ved hjælp af følgende metode der returner en boolean.

PhysicsObject.collissionDetection(PhysicsObject, Physicsobject);

Den anden metode er at tilføje objekter til PhysicsEngines liste over PhysicsObjects. Dette vil give mulighed for at sætte en række objekter til at køre med standard responses baseret på, hvilke triggers man har tilvalgt. (se Trigger booleans)

For at tilføje eller fjerne elementer bruges:

PE.add(PhysicsObject);

PE.remove(PhysicsObject);

Og herefter kan man benytte sig af en grænse (eng. border) kontrol med sine objekter og/eller kontrollere alle objekterne der er added for kollisioner og sørge for at de afstøder hinanden. Disse to metoder aktiveres henholdsvis:

`PE.BorderCollision();`

`PE.CollisionDetection();`



Ex. på ovenstående.

Making custom classes

Der kan tilføjes et rektangels eller en cirkels egenskaber til et brugerdefineret objekt. Dette kunne f.eks være et astroide objekt skulle ligne en astroide som kan benytte sig af en Circles egenskaber og muligheder.

Class CustomClass extends Circle

En vigtig og afgørende del her er, at overføre parametrene til Rect og Circle vha. Objektets konstruktor.

```
CustomClass(float posX, float posY, float diameter, float velX, float velY)
{
    super(posX, posY, diameter, velX, velY);
}
```



Ex. på ovenstående.

Brug af standard reaktion

Der er mulighed for at kunne lave en standard reaktion for en kollision. Denne kode frastøder de to objekter ud fra deres hastighed og størrelse.

```
collisionResponse(PhysicsObject, PhysicsObject);
```

Brug af bagomliggende værdier.

Alle standardværdierne på objekterne kan ændres efter de er skabt. Dette gøres med deres set metoder.

```
Rect.setPos(posX, posY);
Rect.setVelocity(velocityX, velocityY);
Rect.setWidth(width);
Rect.setHeight(height);
```

```
Circle.setPos(posX, posY);  
Circle.setVelocity(velocityX, velocityY);  
Circle.setDiameter(diameter);
```

For velocity er der yderligere mulighed for at tilføje hastighed til den eksisterende hastighed.

```
Circle.addVelocity(velocityX, velocityY);
```

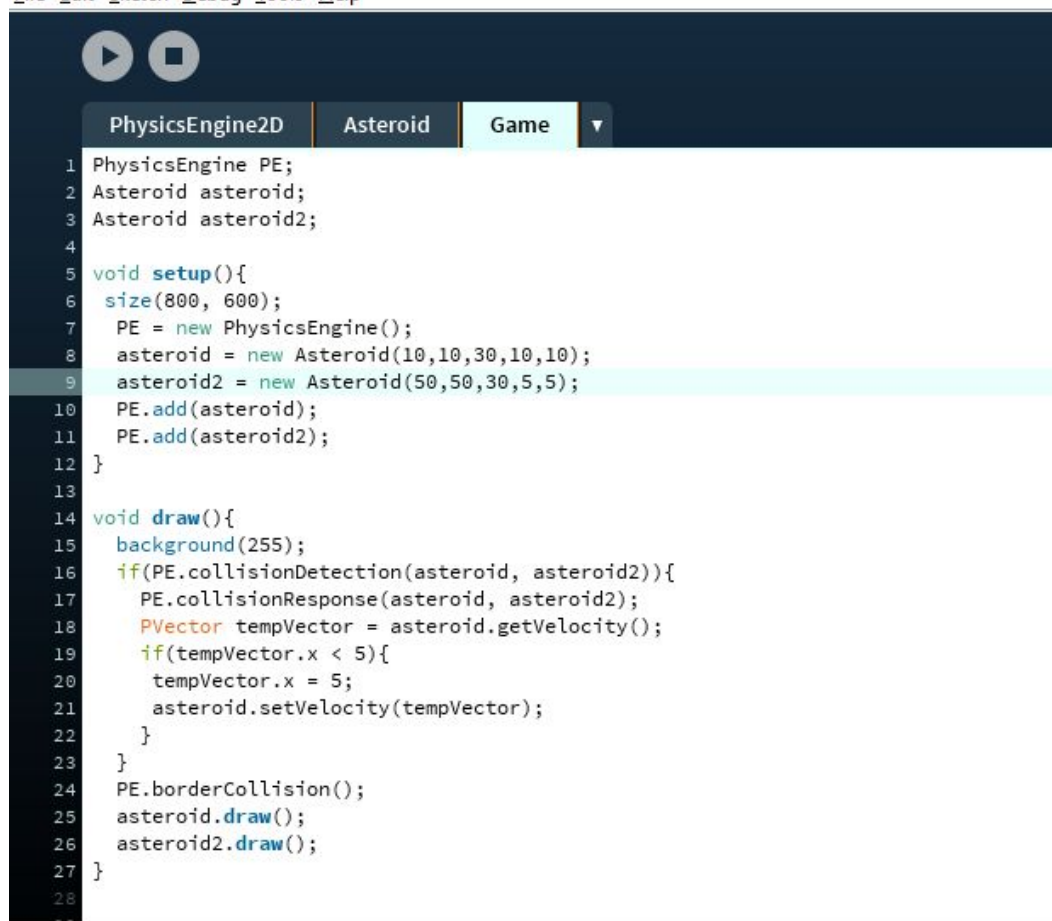
På samme måde kan man få position og velocity tilbage, men da physicsEngine bruger primært PVector, returner get metoderne en PVector.
Man kan tilgå x og y værdierne i en PVector med .x og .y

PVector.x

PVector.y

I nedenstående eksempel skabes der to brugerdefinerede objekter, som vi har beskrevet tidligere og bliver kontrolleret for kollision og bedt om at lave standard response og til sidst sørge for at Y's hastighed for asteroid aldrig kommer under 5.

PhysicsEngine2D | Processing 3.3.6
File Edit Sketch Debug Tools Help



```
1 PhysicsEngine PE;  
2 Asteroid asteroid;  
3 Asteroid asteroid2;  
4  
5 void setup(){  
6   size(800, 600);  
7   PE = new PhysicsEngine();  
8   asteroid = new Asteroid(10,10,30,10,10);  
9   asteroid2 = new Asteroid(50,50,30,5,5);  
10  PE.add(asteroid);  
11  PE.add(asteroid2);  
12 }  
13  
14 void draw(){  
15   background(255);  
16   if(PE.collisionDetection(asteroid, asteroid2)){  
17     PE.collisionResponse(asteroid, asteroid2);  
18     PVector tempVector = asteroid.getVelocity();  
19     if(tempVector.x < 5){  
20       tempVector.x = 5;  
21       asteroid.setVelocity(tempVector);  
22     }  
23   }  
24   PE.borderCollision();  
25   asteroid.draw();  
26   asteroid2.draw();  
27 }  
28
```

Trigger Booleans

Disse booleans kan aktiveres individuelt for alle objekter, således at det kan tilegnes specifik adfærd.

bounceOnImpact er en boolean der styre om et objekt skal lave collisionResponse, når det er i arrayet, det er som udgangspunkt slået til og kan slås fra ved at sætte værdien til false.

```
Circle.bounceOnImpact = false;
```

mouseUse er en boolean der styrer om du skal styre objektet med musen, den er som udgangspunkt slået fra og kan aktiveres ved at sætte værdien til true.

```
Circle.mouseUse = true;
```

keepXConstant er en boolean der styrer om objektet skal kunne bevæge sig i den horizontale retning, den er som udgangspunkt slået fra og kan aktiveres ved at sætte værdien til true.

```
Circle.keepXConstant = true;
```

keepYConstant er en boolean der styrer om objektet skal kunne bevæge sig i den vertikale retning, den er som udgangspunkt slået fra og kan aktiveres ved at sætte værdien til true.

```
Circle.keepYConstant = true;
```

vanishOnImpact er en boolean der styrer om objektet skal forsvinde, hvis det kolliderer med et andet objekt, den er som udgangspunkt slået fra og kan aktiveres ved at sætte værdien til true.

```
Circle.keepXConstant = true;
```

constrainedInFrame er en boolean der styrer om objektet skal forsvinde, hvis objekterne ryger ud fra skærbilledet og er som udgangspunkt slået til og kan slås fra ved at sætte værdien til false.

```
Circle.constrainedInFrame = false;
```