

Multimedia-Programmierung

Übung 7

Ludwig-Maximilians-Universität München
Sommersemester 2016

Today

- Physics
- Final Project
- Illustrated with



+



Exam – New Date

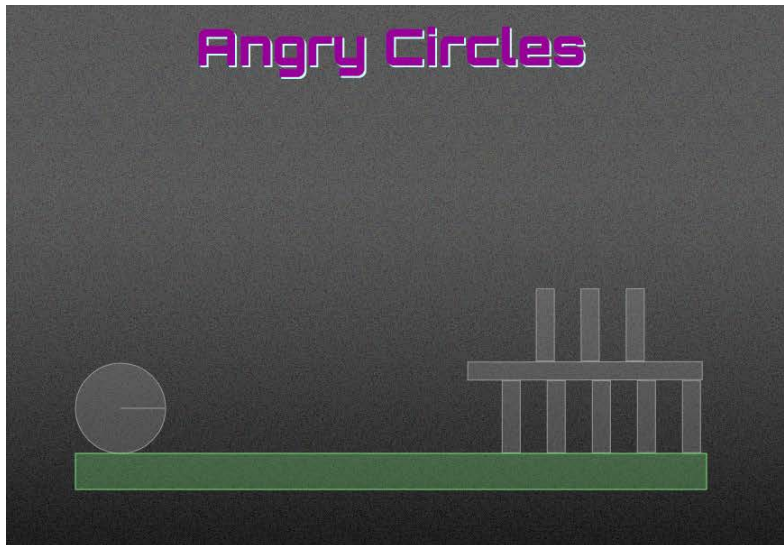
- 11.07.2016
- 10:00-12:00

Physics

Users have specific expectations

For example, if something hits a wall it should bounce or create some damage

Adding physics to applications helps to improve usability and user experience





Collision Detection in PyGame

- `Rect.collidepoint(point)` can be used to see whether a coordinate is within the area of a Rect object
- `pygame.sprite` has advanced methods to check for collisions
 - E.g. `pygame.sprite.collide_rect(a,b)` checks whether two sprites intersect

A simple collision detection



```
import pygame
from pygame.locals import *

...

pygame.init()

screen = pygame.display.set_mode((640, 480), 0, 32)
box = Box((255,0,0),(0,0))

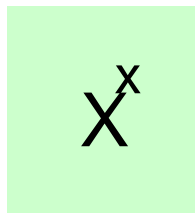
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        if event.type == MOUSEBUTTONDOWN:
            if box.rect.collidepoint(event.pos):
                print "in"
            else:
                print "out"

    box.update()
    screen.blit(box.image,box.rect)
    pygame.display.update()
```

Collision Detection

Rect

- Rect provides several methods to test collisions
<http://www.pygame.org/docs/ref/rect.html>
- `Rect.collidepoint(point)` tests whether a point is within the Rect's area



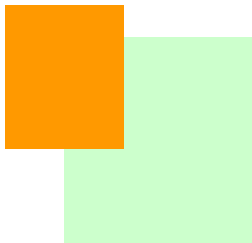
True



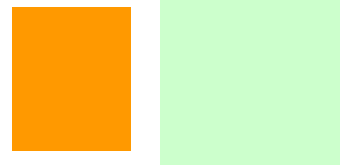
False

x X

- `Rect.colliderect(rect)` tests whether two Rects intersect



True



False

Collision Detection

Rect II

- `Rect.collidelist(list)` tests whether the Rect collides with **at least one** Rect in the given list
- `Rect.collidelistall(list)` tests whether the Rect collides with **all** Rects in the list
- `Rect.collidedict(dict)` tests whether the Rect collides with **at least one** Rect in the given dictionary
- `Rect.collidedictall(dict)` tests whether the Rect collides with **all** Rects in the dictionary

Collision Detection

Sprites

- The module `sprite` provides several methods to test collision
<http://www.pygame.org/docs/ref/sprite.html>
- `sprite.spritecollide(...)` returns a list of sprites within a group that intersect with a given sprite
- `sprite.collide_rect(a,b)` checks whether two sprites intersect (must have rects)
- `sprite.collide_circle(a,b)` checks whether the radius of two sprites intersect. Radius attribute should be defined in the sprite.



False



True

Collision Detection

Sprites 2

- `sprite.groupcollide(a,b)` returns a list of sprites of two groups that intersect
- `sprite.collide_mask(a,b)` checks whether two

```
if pygame.sprite.collide_mask(head1,head2):
    print "collide"
```



Collision Detection

Masks

- Masks are 1bit per pixel representations of areas that can collide
- Module mask contains functions and classes to create and use masks
<http://www.pygame.org/docs/ref/mask.html>
- `mask.from_surface(surface, threshold=127)` creates a mask of a surface. Threshold defines the alpha value that counts as collideable
- Class Mask contains methods to work with classes

Original



Mask



collision area



Collision Detection

Conclusion

- Pygame offers various ways to check for collisions
- **Choose your collision detection algorithm wisely depending on the task**
- Pixel based collision detection is precise but slow
- Rect or radius based collision detection is fast but imprecise



Programming Physics

(LOW-LEVEL)



Frameworks like Cocos2d-x offer physics engines (e.g. 3D game engines, Interpolators in Flash or Box2D for JavaScript (..and python))

In Python, **WE have** to do the physics!!

Tutorials

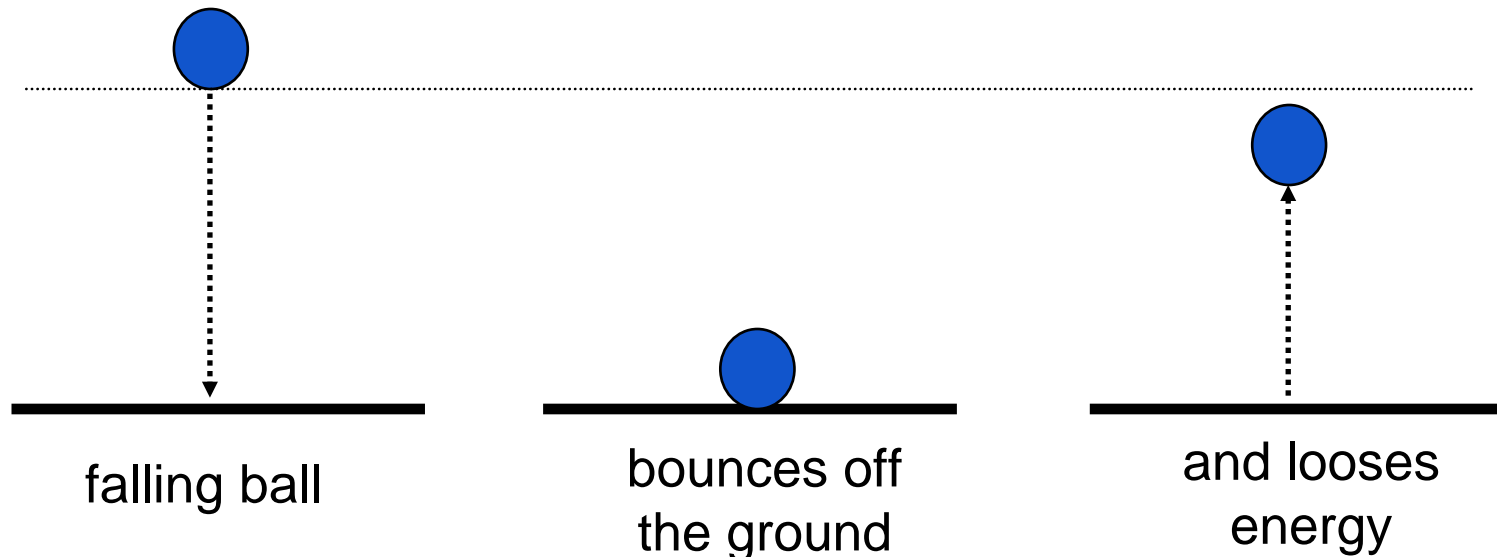
<http://pet.timetocode.org>

<http://www.petercollingridge.co.uk/pygame-physics-simulation>

Bouncing Ball Example 1

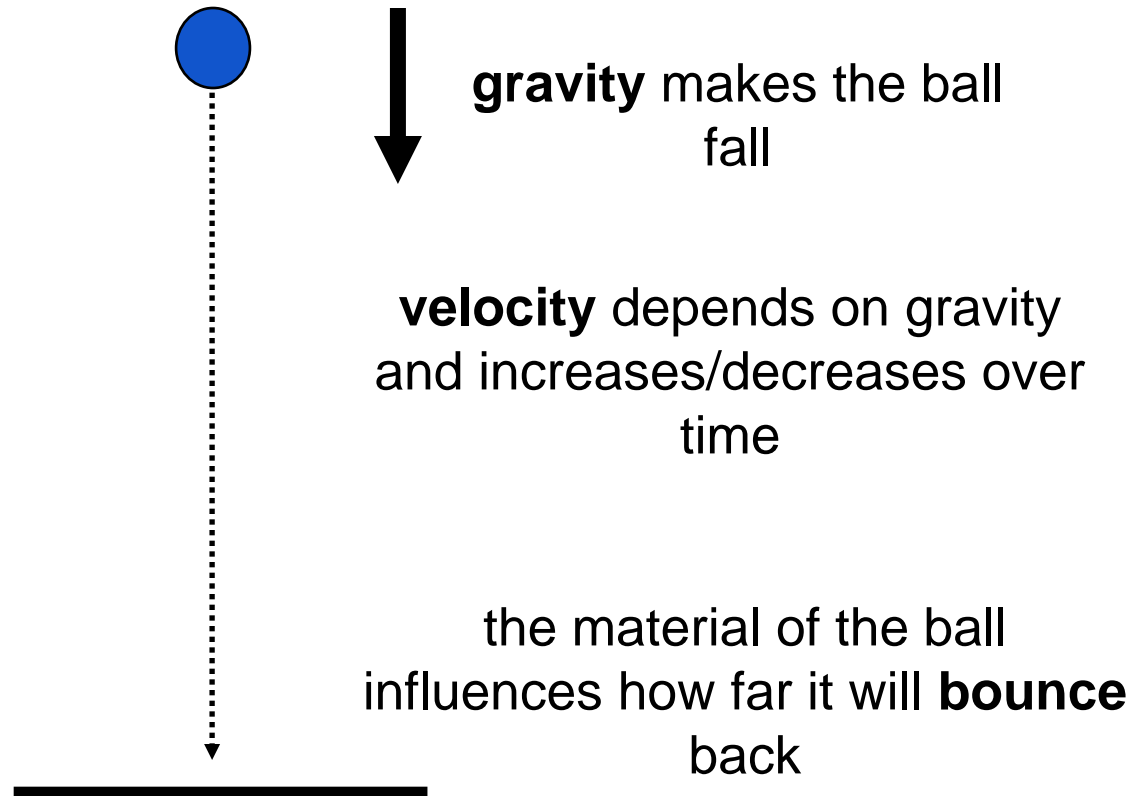
Let's make a ball bounce in a realistic way

1. We need a concept:



Bouncing Ball Example 2

2. What makes the ball fall and bounce?



Bouncing Ball Example 3



```
class Ball(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        size = 20
        self.gravity = 900
        self.velocity = 0
        self.bounce = 0.9

        self.image =
pygame.Surface((size,size),pygame.SRCALPHA,32)
        pygame.draw.circle(self.image,color,(size/2,size/2),size/2)
        self.rect = self.image.get_rect()
        self.rect.center = initial_position

    def update(self, time_passed, size):

        self.velocity += (self.gravity * time_passed)
        self.rect.bottom += int(self.velocity * time_passed)

        if self.rect.bottom >= size[1]:
            self.rect.bottom = size[1]
            self.velocity = -self.velocity * self.bounce
```

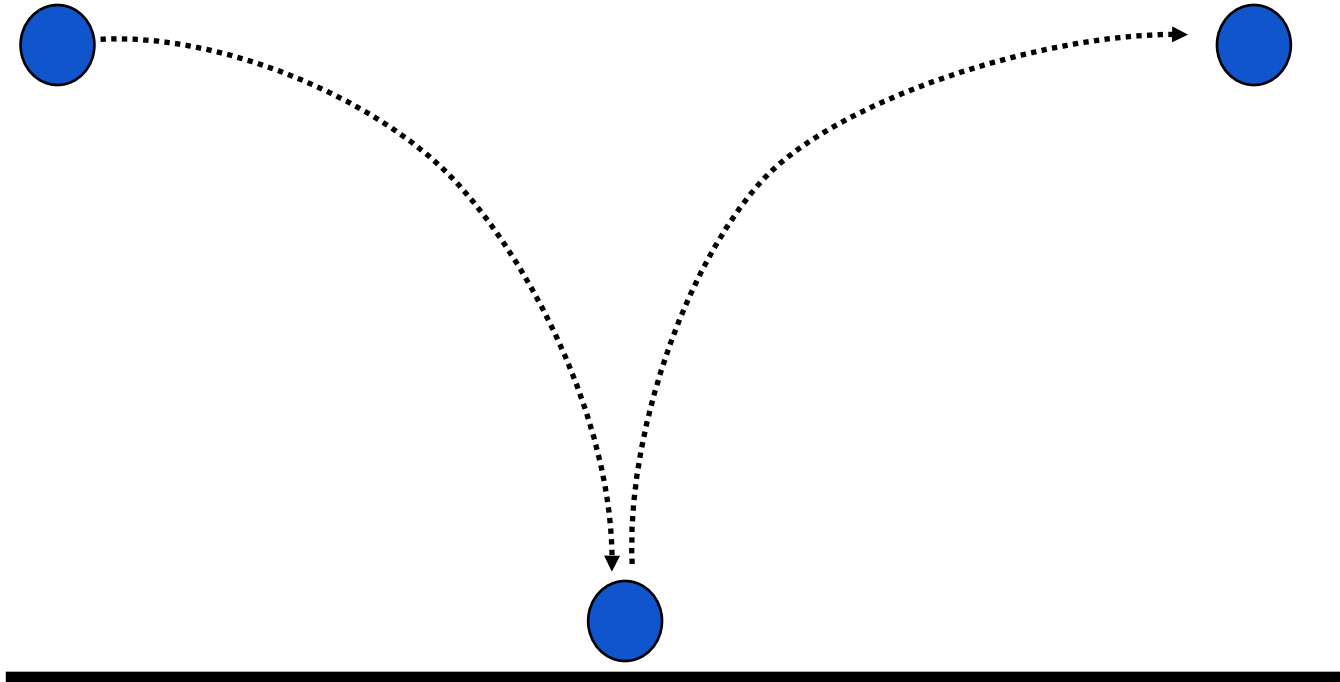
gravity per second,
current velocity and
bounce factor of the
material

velocity is
increased/decrease
d by the gravity

if the ball hits the
ground, reduce
velocity based on
the bounce factor

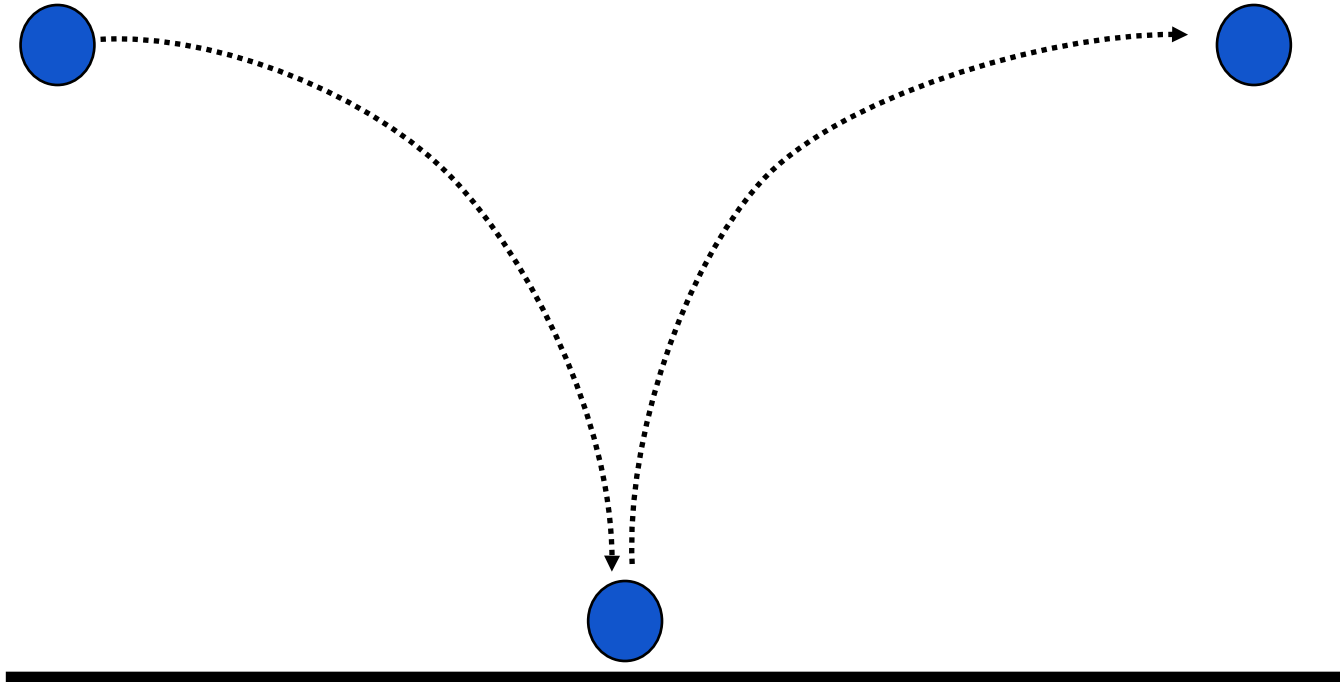
Bouncing Ball Example 4

Making the ball bounce and move vertically



In-class exercise

Implement this movement:



Bouncing Ball Example 5



```
class Ball(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        size = 20
        self.gravity = 900
        self.vx = 0
        self.vy = 0
        self.bounce = 0.9
        ...
    def update(self, time_passed, size):
        self.velocity += (self.gravity * time_passed)
        ydistance = int(self.vy * time_passed)
        self.rect.bottom += ydistance
        if ydistance == 0 and self.rect.bottom == size[1]: self.vx = 0
        self.rect.left += int(self.vx * time_passed)
        if self.rect.right >= size[0]:
            self.rect.right = size[0]
            self.vx = -self.vx
        if self.rect.left <= 0:
            self.rect.left = 0
            self.vx = -self.vx
        if self.rect.bottom >= size[1]:
            self.rect.bottom = size[1]
            self.vy = -self.vy * self.bounce
```

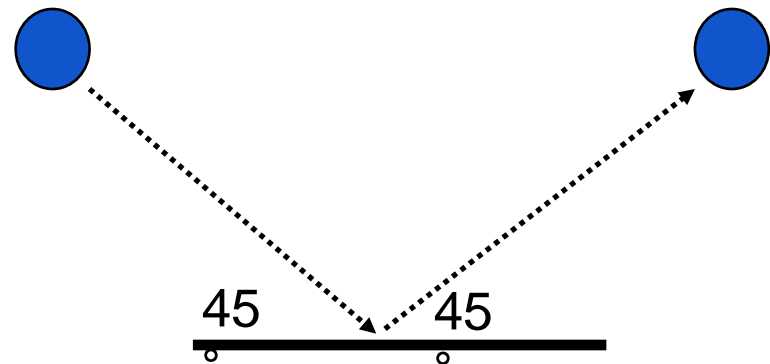
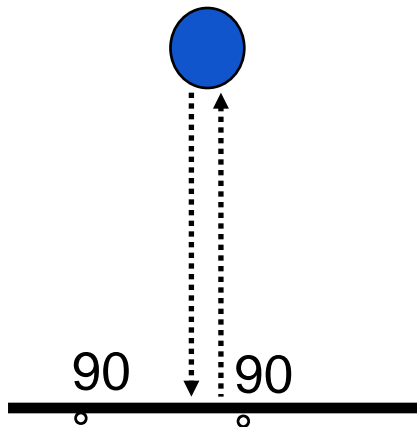
x and y velocity

clumsy way to make
the ball stop

if the ball hits the
sidewalls, make it
change the direction

Arrival Angle = Angle of Reflection

What if the Ball doesn't drop perfectly vertically?





Programming Physics

(HIGH-LEVEL)

When do you need a physics engine?

- You want to simulate real world situations
- You need a lot of **collision detection**, **gravity**, **elasticity** and **friction**
- You deal with many objects

- Often, using a physics engine **is not necessary** (e.g., simple gravity simulation, detecting rectangle collisions)
- **Good read:** *Daniel Shiffman, The Nature of Code* (<http://natureofcode.com/book/>)





Physics in Cocos

Two engines:

- Chipmunk (built-in)
- Box2D





Games based on physics engines

Intertwining of:

- Graphical world (displayed)
 - E.g., Cocos scenegraph
 - Physics world (simulated)
 - E.g., Box2D physics simulation
-
- 2D graphics/2D physics simulation
 - 3D graphics/3D physics simulation



Important concepts

Physics world:

- Coordinate systems and units can be different from the graphical rendering (mapping!)
- Forces, collisions etc. are calculated and solved in steps (update rate)
- With every update, graphical objects are moved and oriented according to the current state of the simulation

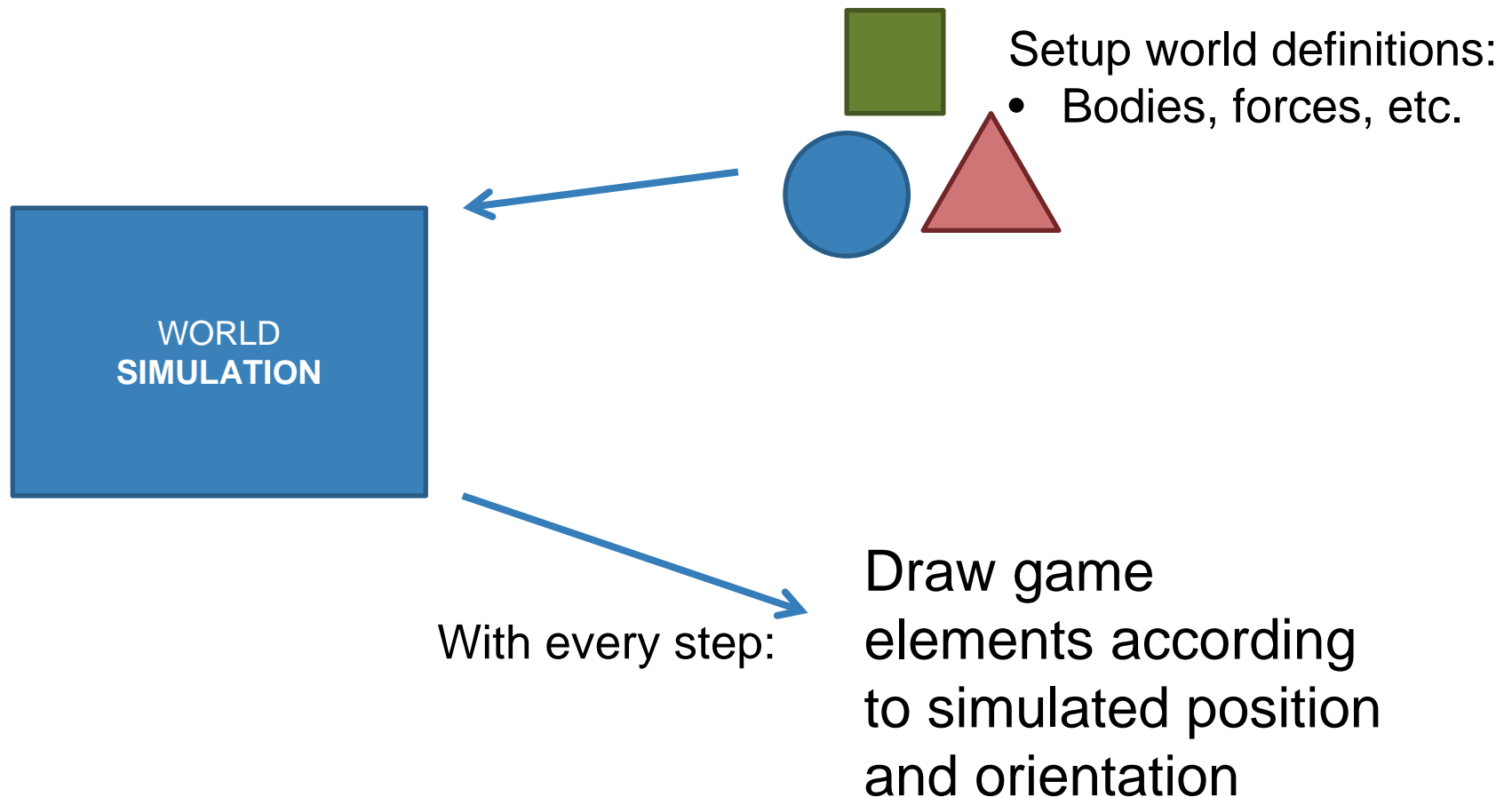


Important Aspects

- Bodies
- Shapes
- Materials
- Contacts/Joints
- World



Simulated World as a “Magic Box”





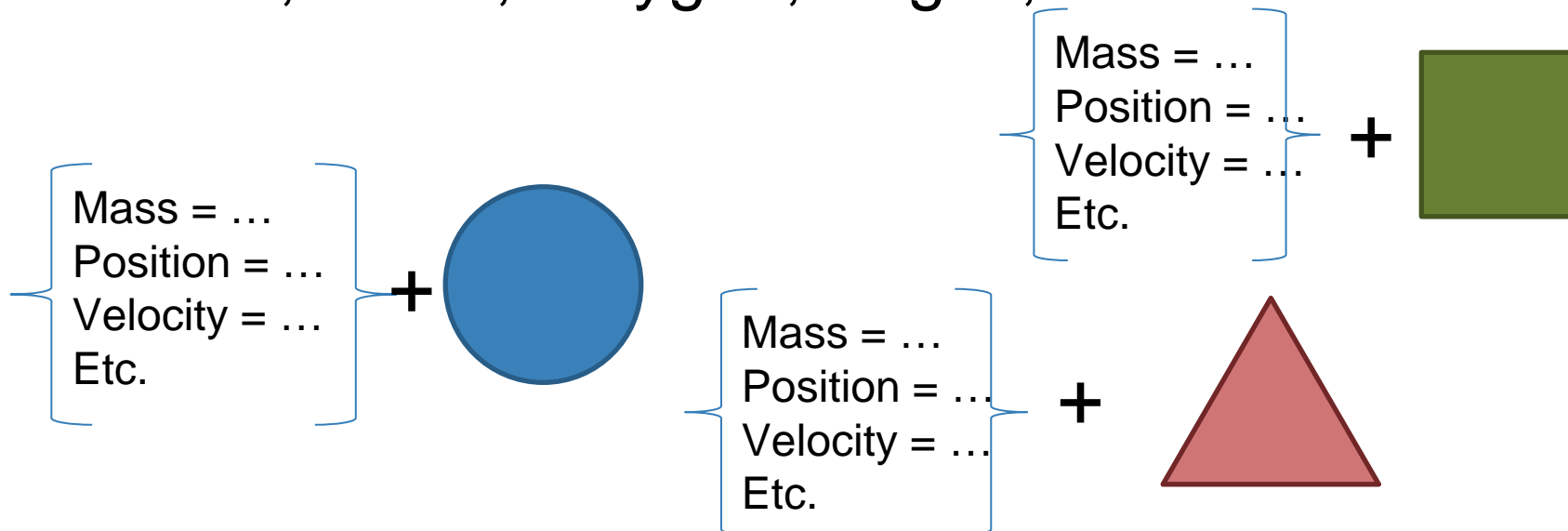
Bodies

- A *Body* defines the physical properties of an object, such as *mass*, *position*, *rotation*, *velocity*, *damping*
- Has no shape!
- *Static* bodies don't move in the simulation and behave like they have infinite mass
- *Dynamic* bodies are fully simulated and move according to simulated forces and/or manual input

Mass = ...
Position = ...
Velocity = ...
Etc.

Shapes

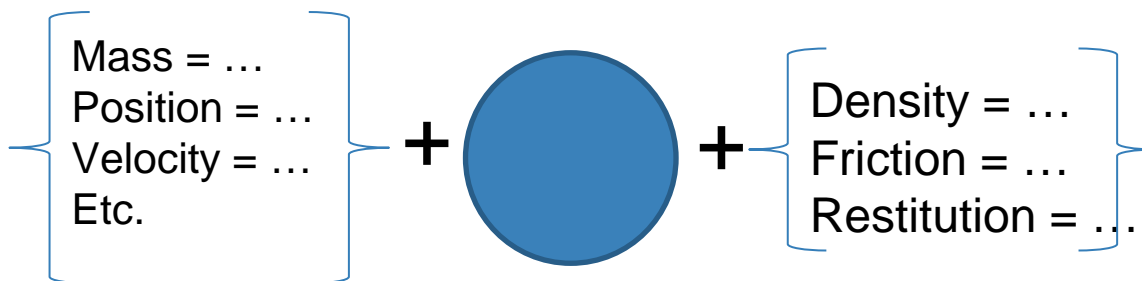
- *Shapes* describe collision geometry
- Are attached to bodies
- Predefined shapes in Box2D/Chipmunk:
 - Box, Circle, Polygon, Edges, ...





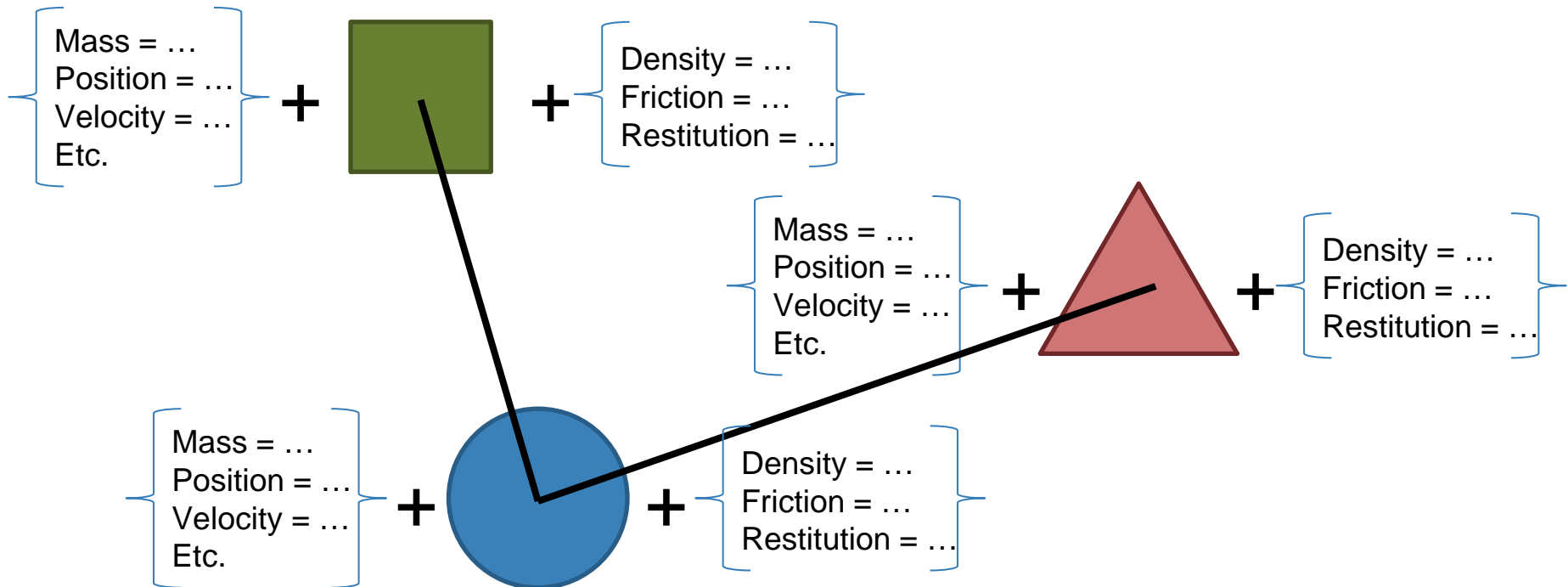
Materials

- *Materials* describe material properties:
- Density: mass properties of the parent body
- Restitution: bouncing properties of the parent body
- Friction: sliding properties of the parent body



Contacts/Joints

- Describe how bodies are attached to each other





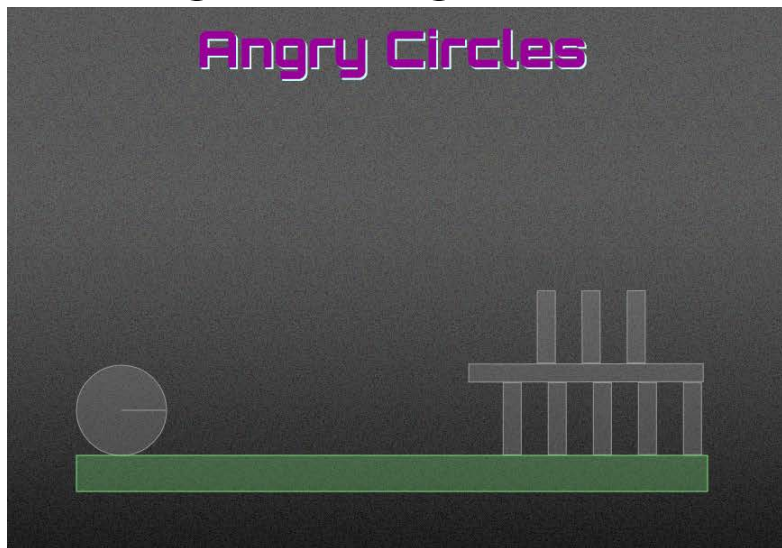
World

- The *World* object is the container for the simulation
- Physics bodies, shapes and constraints are added to it
- *World* updates control how all of the added objects interact together

- Important *World* properties:
 - *Gravity*
 - *Speed (of simulation)*
 - *Update rate*

Recap

Debug drawing of a box2d simulation:

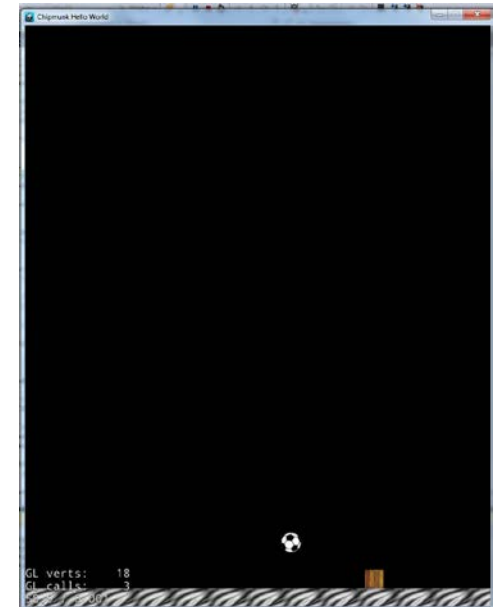


Static bodies?
Dynamic bodies?
Forces?
Updates?



Cocos and Chipmunk “Hello World”

- Chipmunk is integrated into Cocos
- World is based on pixels as units
- Deeply integrated with *Scene*





Creating a world

```
auto scene = Scene::createWithPhysics();  
scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);  
scene->getPhysicsWorld()->setGravity(Vec2(0.0f, -350.0f));
```



Creating a static body

```
auto groundBody = PhysicsBody::createBox(  
    Size(65.0f, 81.0f),  
    PhysicsMaterial(0.1f, 1.0f, 0.0f)  
);  
  
groundBody >setDynamic(false);
```

1. Defining a body
2. Defining and attaching a shape
3. Defining material properties



Attaching a body to a sprite

```
//add sprite to scene
_ground = GameSprite::gameSpriteWithFile("res/ground.png");
_ground->setPosition(Vec2(_center.x, 16.0f));
this->addChild(_ground);

//attach groundBody to the sprite
_ground->setPhysicsBody(groundBody);
```



Creating a dynamic body

```
//body definition
auto ballBody = PhysicsBody::createCircle(
    17.5f,
    PhysicsMaterial(0.1f, 0.4f, 0.0f)
);
ballBody->setMass(10.0f);

//sprite definition
_ball = GameSprite::gameSpriteWithFile("res/ball.png");
_ball->setPosition(Vec2(400.0f, 500.0f));
this->addChild(_ball);

_ball->setPhysicsBody(ballBody);
```



Applying a force

```
Vec2 force = Vec2(0.0f, 550.0f);  
_ball->getPhysicsBody()->applyImpulse(force);
```



Cocos and Box2D “Hello World“

- Box2D is very popular (e.g. Angry Birds)
- A lot of Documentation, Tutorials etc.
- Based on **MKS** (meters, kilograms, and seconds)





Box2D and Visual Studio

```
r *renderer, const Mat4& transform, uint32_t flags)
```

```
ransform, flags);
```

```
ector
```

```
cocos
```

```
RIX_S
```

```
IX_ST
```

```
dt)
```

```
10;
```

```
10;
```

```
yIter
```

```
ld->G
```

```
a())
```

```
te =
```

```
ion(d
```

```
ion(-
```

```
an(co
```

```
se) {
```

```
} {
```

```
getLo
```

Box2DHelloWorld-Eigenschaftenseiten

Konfiguration: Aktiv(Debug) Plattform: Aktiv(Win32)

- Konfigurationseigenschaften
 - Allgemein
 - Debugging
 - VC++-Verzeichnisse
 - C/C++
 - Linker
 - Allgemein
 - Eingabe
 - Manifestdatei
 - Debuggen
 - System
 - Optimierung
 - Eingebettete IDL
 - Windows-Metadaten
 - Erweitert
 - Alle Optionen
 - Befehlszeile
 - Manifesttool
 - Ressourcen
 - XML-Dokument-Generato
 - Informationen durchsuche
 - Buildereignisse
 - Benutzerdefinierter Buildsc
 - Codeanalyse

Zusätzliche Includeverzeichnisse	\$(COCOS_X_ROOT)\cocos2d-x-3.10
Zusätzliche #using-Verweise	
Debuginformationsformat	Program
Common Language Runtime-Unterstützung	
Windows-Runtime-Erweiterung verwenden	
Startbanner unterdrücken	Ja (In
Warnstufe	Stufe
Warnungen als Fehler behandeln	Nein (
Warnungsversion	
SDL-Prüfungen	
Kompilierung mit mehreren Prozessoren	Ja (/M

Zusätzliche Includeverzeichnisse
Gibt mindestens ein Verzeichnis an, das zum Includepfad
Verzeichnis Semikolons als Trennzeichen. (/I[Pfad])

Zusätzliche Includeverzeichnisse

\$(COCOS_X_ROOT)\cocos2d-x-3.10
\$(COCOS_X_ROOT)\cocos2d-x-3.10\tests\cpp-tests\Classes\Box2DTest
\$(COCOS_X_ROOT)\cocos2d-x-3.10\external\Box2D
\$(COCOS_X_ROOT)\cocos2d-x-3.10\cocos\audio\include
\$(COCOS_X_ROOT)\cocos2d-x-3.10\external
..\Classes
..
%(AdditionalIncludeDirectories)
\$_COCOS_HEADER_WIN32_BEGIN)
\$_COCOS_HEADER_WIN32_END)
\$(ProjectDir)

Ausgewerteter Wert:

C:\Cocos\Cocos2d-x\cocos2d-x-3.10
C:\Cocos\Cocos2d-x\cocos2d-x-3.10\tests\cpp-tests\Classes\Box2DTest
C:\Cocos\Cocos2d-x\cocos2d-x-3.10\external\Box2D
C:\Cocos\Cocos2d-x\cocos2d-x-3.10\cocos\audio\include
C:\Cocos\Cocos2d-x\cocos2d-x-3.10\external
..\Classes
..
%(AdditionalIncludeDirectories)

Vererbte Werte:

\$(EngineRoot)\cocos\editor-support
\$(EngineRoot)\cocos
\$(EngineRoot)\cocos\platform
\$(EngineRoot)\cocos\platform\desktop
\$(EngineRoot)\external\glfw3\include\win32
\$(EngineRoot)\external\win32-specific\gles\include\OGLES
\$(EngineRoot)\external\freetype2\include\win32\freetype2
\$(EngineRoot)\external\freetype2\include\win32\
\$(EngineRoot)\external

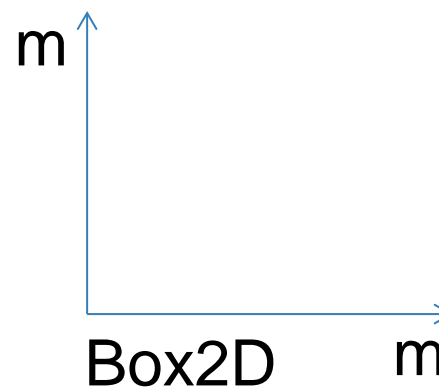
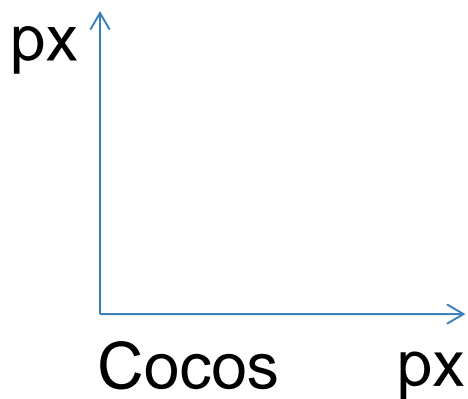
Vom übergeordneten Projekt erben oder Projektstandard

Makros >>

OK Abbrechen

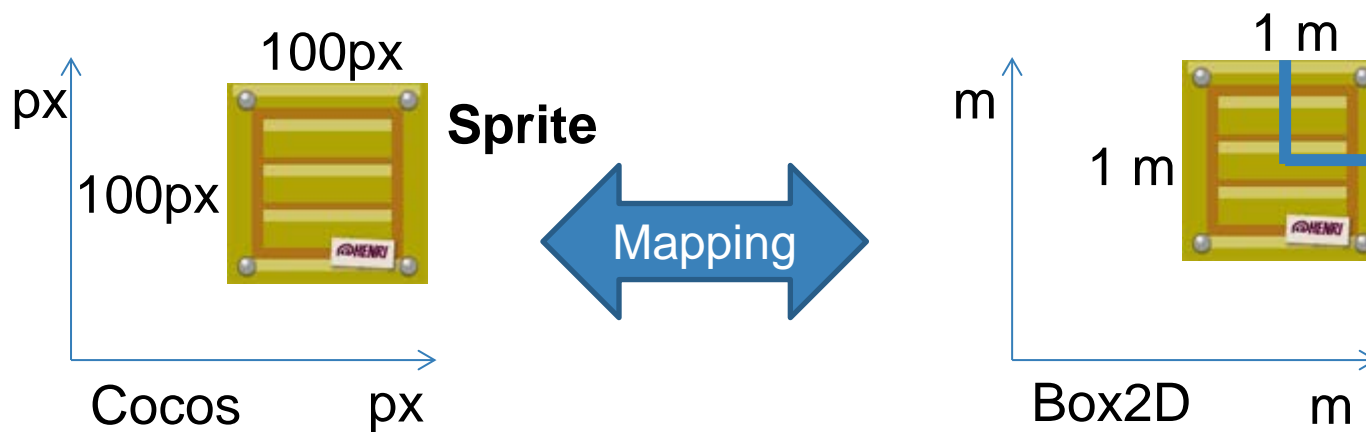


Cocos and Box2D “Hello World“





Cocos and Box2D “Hello World“



In **Box2D**:

A box (width = 0.5, height = 0.5)



Creating a world

```
// Create a world, define gravity  
b2Vec2 gravity = b2Vec2(0.0f, -8.0f);  
_world = new b2World(gravity);
```



Creating a static body

```
b2BodyDef groundBodyDef;  
groundBodyDef.position.Set(_center.x / SCALE_RATIO, 16.0f /  
SCALE_RATIO);  
_staticBody = _world->CreateBody(&groundBodyDef);  
  
b2PolygonShape groundBox;  
groundBox.SetAsBox(800.0f / 2 / SCALE_RATIO, 32.0f / 2 /  
SCALE_RATIO);  
  
_staticBody->CreateFixture(&groundBox, 0.0f);
```

1. Defining a body
2. Defining and attaching a shape
3. Defining material properties



Attaching a body to a sprite

- Not possible, you have to take care of that manually
- With every update of the simulation, go through list of bodies and manipulate associated sprites accordingly
- **Help:**
- `bodyDef.userData = Reference to Sprite;`
- `GameSprite *sprite = (GameSprite *)body->GetUserData();`



Attaching a body to a sprite

```
void GameLayer::update(float dt) {
//get current state of the world
_world->Step(dt, velocityIterations, positionIterations);
//iterate through bodies and update sprites
for (b2Body *body = _world->GetBodyList(); body != NULL; body = body-
>GetNext())
    if (body->GetUserData())
    {
        GameSprite *sprite = (GameSprite *)body->GetUserData();

        sprite->setPosition(ccp(body->GetPosition().x *
SCALE_RATIO,
SCALE_RATIO),
body->GetPosition().y *
SCALE_RATIO);
        sprite->setRotation(-1 * CC_RADIANS_TO_DEGREES(
body->GetAngle()));
    }
}
```



Creating a dynamic body

```
//create a dynamic body
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.userData = _box;
bodyDef.position.Set(xPos / SCALE_RATIO, yPos / SCALE_RATIO);
b2Body * box = _world->CreateBody(&bodyDef);

b2PolygonShape boxShape;
boxShape.SetAsBox(width / 2 / SCALE_RATIO, height / 2 / SCALE_RATIO);

b2FixtureDef fixtureDef;
fixtureDef.shape = &boxShape;
fixtureDef.density = 10.0f;
fixtureDef.friction = 0.4f;
fixtureDef.restitution = 0.1f;
box->CreateFixture(&fixtureDef);
```



Applying a force

```
Vec2 force = Vec2(0.0f, 550.0f);  
_dynamicBody->ApplyForce(  
    force.x,  
    force.y,  
    _dynamicBody->GetWorldCenter(),  
    true  
);
```



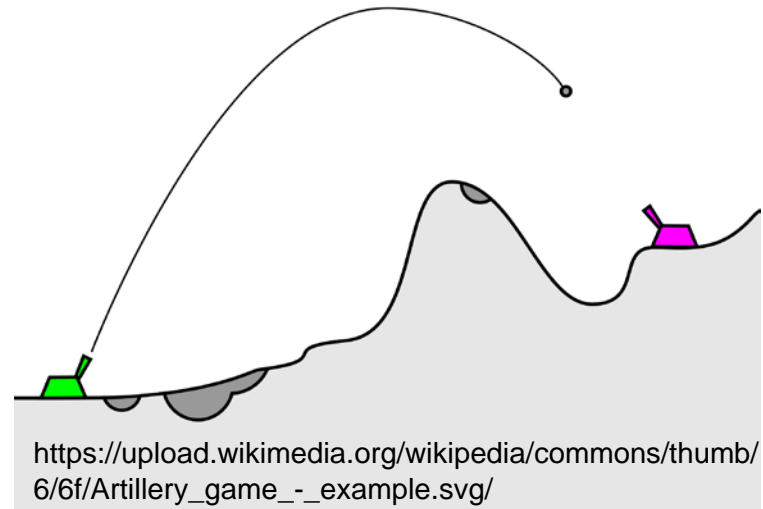

What's next?

- Check for collisions
- Joints
- Complex shapes
- **Tool:**

<https://www.codeandweb.com/physicseditor>



Bonus Project: Artillery game



Quick Facts

- Design and implement your own game (in a group)
- **Gain up to 10% bonus for the final exam**
- Games are developed in Cocos2d-X
- Project phase: 13.06. – 13.07.
- Early submissions are allowed

! All submissions will be checked for plagiarism !

Links

<http://www.cocos2d-x.org/wiki/Physics>

<http://box2d.org/documentation.html>

<http://box2d.org/manual.pdf>