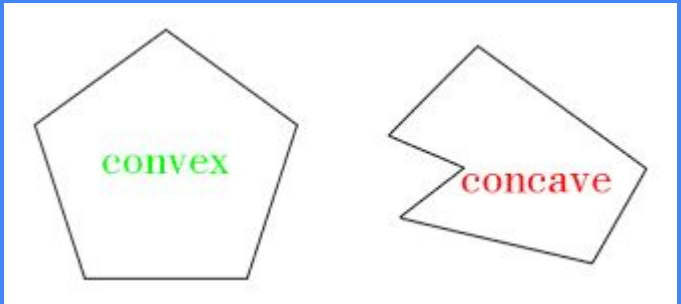
Physics 2 - Intro to Box2D

Ricard Pillosu - UPC

Documentation

- Be sure to have opened http://box2d.org/manual.pdf (included in the zip)
- Most of data and functions starts with “**b2**”
- Read the FAQ: https://code.google.com/p/box2d/wiki/FAQ
- Play with the testbed at home:
  https://www.youtube.com/watch?v=FiIZqOioRhQ

Intro to Box2D



- Box2D handles **rigid bodies** for collision detection and simulation
- **Shape**: circle, rectangle/box, *convex* polygon
- **(Rigid) Body**: A point in space that will be simulated
- **Fixture**: Binds a shape to a body, giving it mass. A body could have multiple fixtures.

Creating a world

- We need to create a **b2World**
- *A physics world is a collection of bodies, fixtures, and constraints that interact together.*
- Box2D supports the creation of multiple worlds.

```
b2Vec2 gravity(0.0f, -10.0f);

b2World world(gravity);
```

Simulating the world

- Every step / update the physics world calculates positions and rotations.
- We pick a time step, e. g. 1 / 60 of a second.
- This would be true for a game running at perfect 60 fps.
- Even if we do not, we generally want to keep fixed steps for physics.
- We pick velocity and iteration for the constraint solver (8 and 3 are good).
- We could call several time to “Step” in lower frame rate (minimize tunneling).

```
world->Step(1.0f / 60.0f, 8, 3);
```

Units

- Box2d uses *meters-kilograms-second (MKS)* and *radians* for rotation.
- Performs optimal for *dynamic* bodies between **0.1** and **10** meters.
- For *static* bodies, up to **50** meters.
- We need a formula to transform pixels to meters.
  - How many pixels will be considered a meter in the simulation world ?
- Remember that pixels are expressed with **int** and meters in **float**
- Create a macro (#define) to translate from one to the other

Creating bodies

- First, we need to create a bodyDef and fill it with information
- For now, only body type and position are relevant
  - Bodies can be *static*, *dynamic* and *kinematic*
  - Remember to translate from pixels to meters!

```
b2BodyDef body_def;

body_def.type = b2_staticBody; // or b2_dynamicBody

body_def.position.Set(PIXEL_TO_METERS(x), PIXEL_TO_METERS(y));

b2Body* body= world->CreateBody(&body_def);
```

Creating fixtures: shapes

- First create a shape
- We have plenty of classes for that: b2PolygonShape, b2CircleShape, etc...

```
b2CircleShape shape;

shape.m_radius = PIXEL_TO_METERS(radius);
```

Creating fixtures

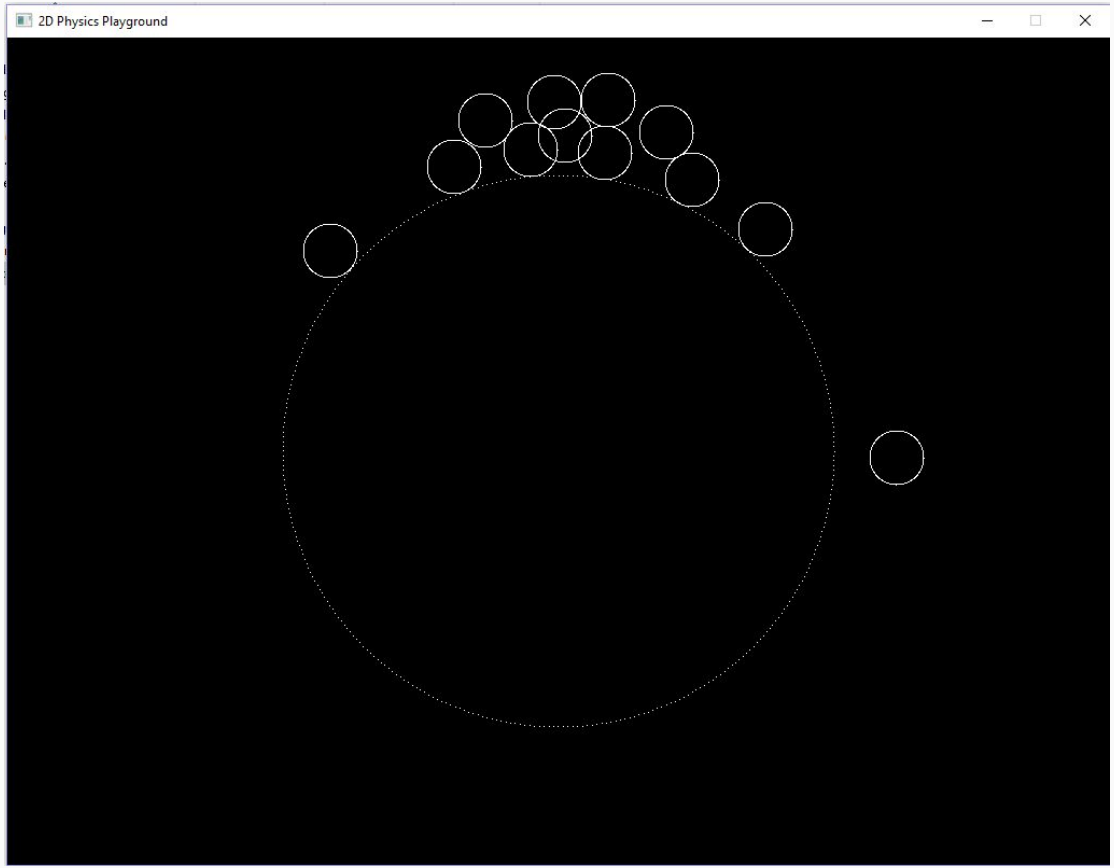
- In its simplest form, a fixture is a container for the shape
- Later, we can play with values like density and friction.
- Static bodies are considered to have zero mass
- Shape and fixture data are copied by Box2D, so we can discard them.

```
b2FixtureDef fixture;

fixture.shape = &shape;

body->CreateFixture(&fixture);
```

Execute Game/solution.exe (press space to spawn dynamic circle)



TODO 1

“Include Box2D header and library”

- All files are already inside the folder Box2D
- You should include the debug or release lib
- The macro **_DEBUG** is defined if we are in Debug mode
- Use #ifdef to include one or the other

TODO 2

“Create a private variable for the world. You need to send it a default gravity. You need init the world in Start(). Remember to destroy the world”

- Look in the documentation in how to create the world (section 2.1)
- Create on Start() destroy on CleanUp()

TODO 3

*“Update the simulation (**step** the world)”*

- Look in the documentation in how to step (section 2.4)
- For velocity and iteration use 8 and 3
- Later we will experiment with other values to see the effect

TODO 4

*“Create a big static circle as **ground**”*

- Will create a hardcoded static big circle in the middle
- Check documentation (section 2.2)
- … but we want a big circle!
- Try different radius to meet the solution.exe
- Create the macros METER_TO_PIXEL and PIXEL_TO_METER
- Now uncomment the *Bonus Code* in **PostUpdate()**

TODO 5

“On space bar press, create a circle on mouse position. You need to transform the position / radius”

- You will need to use again the METER_TO_PIXEL macros
- For mouse use GetMouseX/Y() methods from input module
- Experiment with different values to try mimic solution provided

Homework

- Try random radius for generated circles
- Try creating a box under the big circle as bottom ground