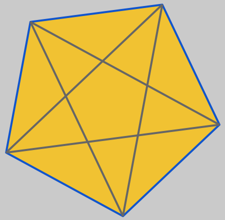
Shapes

Types of shapes

- ✗ Circles (remember from previous class!)
- ✗ Convex polygons
- ✗ Edges
- ✗ Chains

Shapes: Polygons

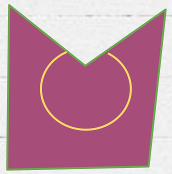
Polygons must be convex! Box2D polygons accept up to *b2_maxPolygonVertices* (8) and must be declared in counterclockwise order.



Convex

Box2D has a handy function to create rectangles:

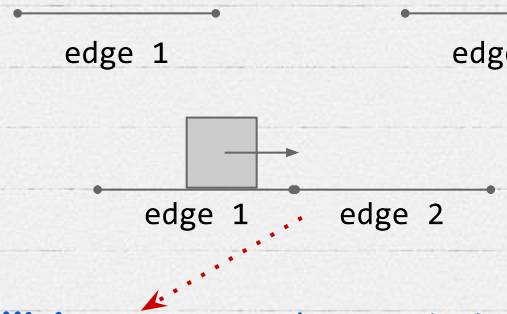
```
void SetAsBox(float32 hx, float32 hy);
```

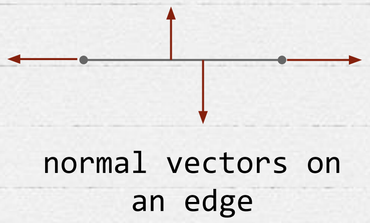


Concave

Shapes: Edges

Edges shapes are line segments. Useful to create a free-form static environment for our game. However…

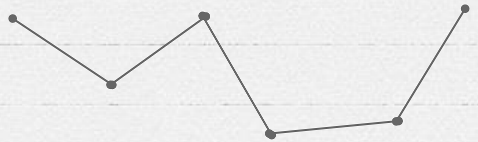


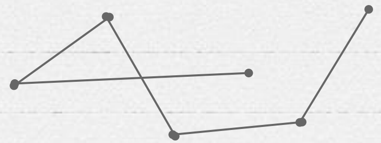


Ghost collisions are unpleasant. Can be avoided declaring *ghost vertices*, but creating edges that way turns into a tedious task. Fortunately, we have…

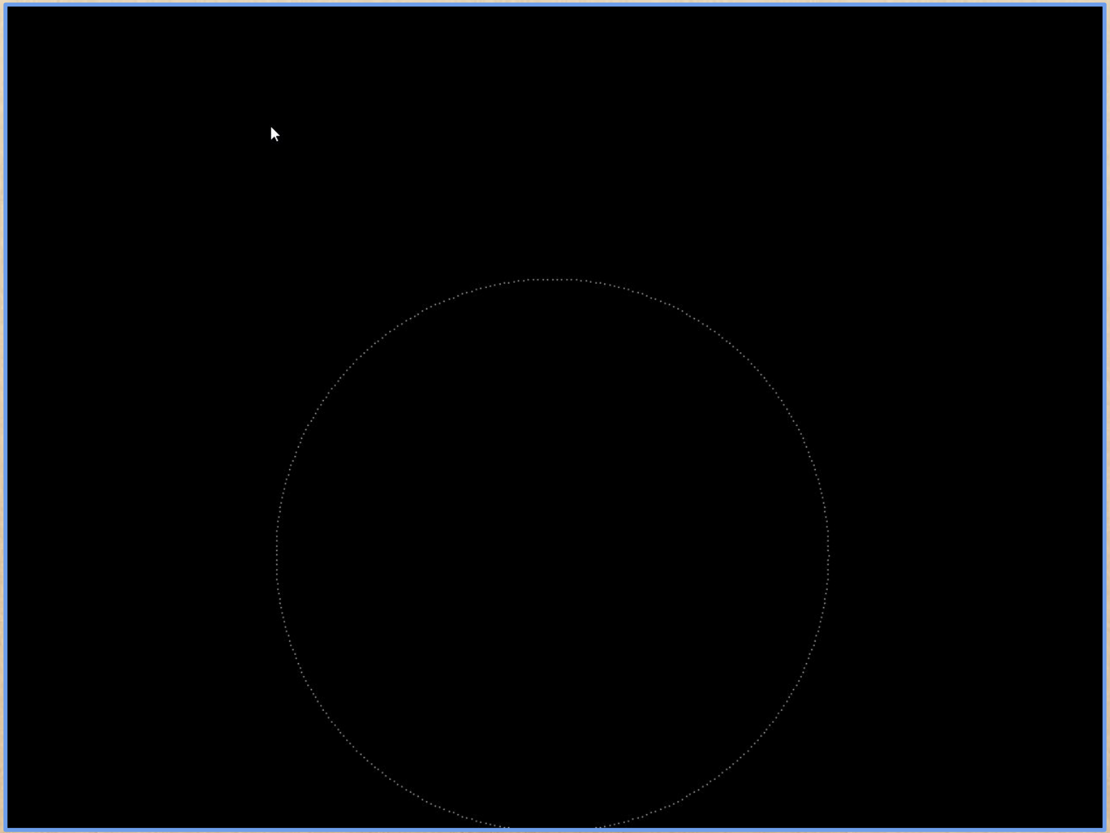
Shapes: Chains

It's an efficient way to concatenate edges. Eliminates ghost collisions and has two-sided collisions.

 ✓

 ✗

Chains and edges don't self-collide because they do not have volume.



Our goal

YOUR TURN !

TODO No 1

```
// TODO 1: When pressing 2, create a box on the mouse position
// To have the box behave normally, set fixture's density to 1.0f
```

It's the same code that you used to create a circle. But now… a box! So, you must change the *shape* and use the proper method.

Remember the pixels ⇔ meters conversion!

TODO Nº 2

```
// TODO 2: Create a chain shape using those vertices
// remember to convert them from pixels to meters!
```

- ✗ You need to fill an array of *b2Vec2*
- ✗ Once converted on the new array, call CreateLoop( ) method defined in the b2ChainShape Box2D's class

```
void CreateLoop(const b2Vec2* vertices, int32 count);
```

Which letter do you obtain?

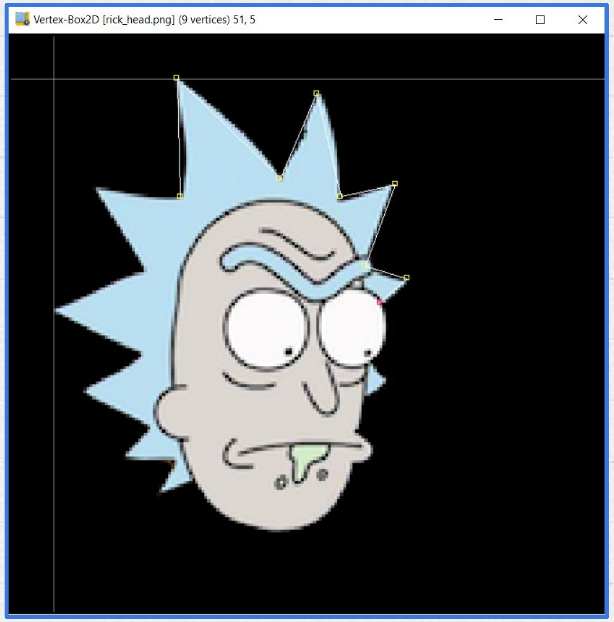
Our own shapes

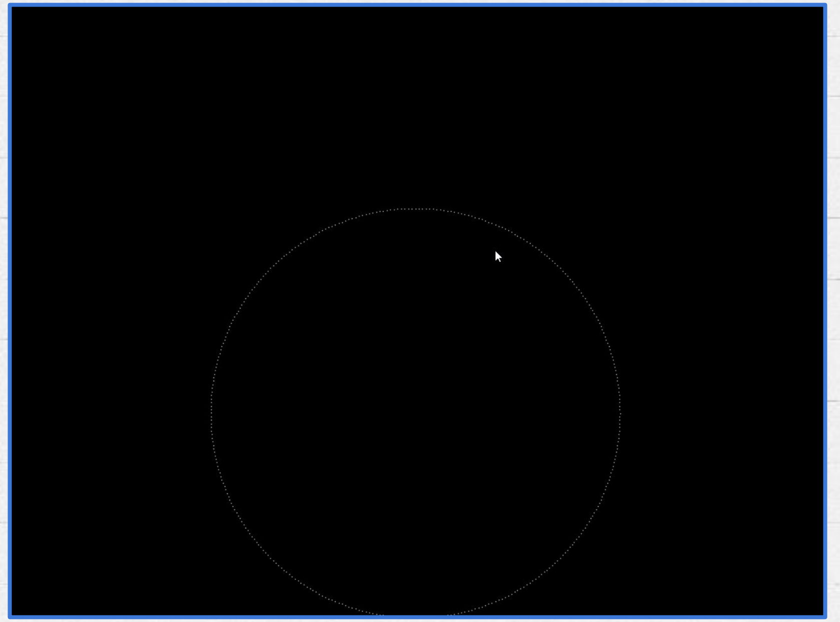
Now, it's time to create our **own shapes.**

Download Ric's shape creator. You can drag & drop any image that you want and clicking with the mouse, you will create all the edges surrounding the image that you've imported. Do not close the last segment and press ESC to save all the vertex's information to the buffer.

Give it a go with !

Our own shapes


Vertex-Box2D [rick_head.png] (9 vertices) 51, 5



TODO Nº 3 AND 4

```
// TODO 3: Move body creation to 3 functions to create
circles, rectangles and chains
```

```
// TODO 4: Move all creation of bodies on 1,2,3 key press
here in the scene
```

It's a matter of order and cleaning. The functionality remains the same but better structured.

TODO Nº 5

Now, we want to get rid of debug representation of the shapes and use a texture instead.

In order to do that, we need to access to the current position (in pixels) of all physics bodies and draw the corresponding texture at that position, each frame. So...

TODO Nº 5

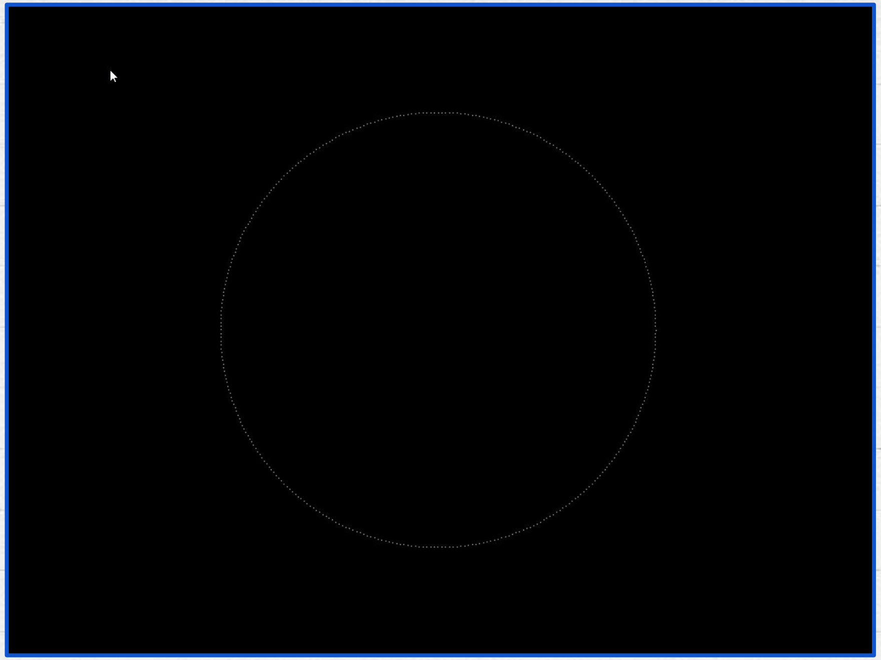
```
// TODO 5: Create a small class that keeps a pointer to the b2Body
// and has a method to request the position
// then write the implementation in the .cpp
// Then make your circle creation function to return a pointer to that class
```

- ✗ Declare a small class in *ModulePhysics.h* and define it in *ModulePhysics.cpp*
- ✗ This class keeps a *b2Body* pointer and a GetPosition( ) method
- ✗ Change the return type of void to a pointer to this new class on your circle creation method

TODO Nº 6

```
// TODO 6: Draw all the circles using "circle" texture
```

- ✗ First, declare a list that will contain all bodies, i.e., all pointers to the previous new class
- ✗ Afterwards, store each new circle created by pressing 1 to that list
- ✗ At the end of the Update method, iterate the list and use GetPosition() to to draw all the circles using the circle texture already loaded



Do you get it ?

Why do you have an offset between texture and debug render?

Try to fix it!

Homework

- ✗ Try to draw the others objects: boxes and "Rickies"
- ✗ Include a GetRotation( ) method on the class that you've created. The Box2D bodies rotate so the image must rotate accordingly
- ✗ Create a macro to convert the rotation angle from Radians to Degrees and pass it to the Blit function to rotate the texture.
- ✗ Start extracting the sprites of your game
- ✗ Create the physical edges of your scenario
- ✗ Add the ball at the top and look how they fall and collide

Next Week . . .

Collision Detection