Bouncing

The character moves around and bounces back at the borders.

We create a ball that moves on the screen at a speed of 150 pixels per second. When it reaches the border, it bounces back.

```
WIDTH = 640
HEIGHT = 480

ball = Actor('ball', anchor=('left', 'top'))
velocity = {'x': 150, 'y': 150}

def update(dt):
    if ball.x + ball.width >= WIDTH:
        velocity['x'] = -velocity['x']
    if ball.x < 0:
        velocity['x'] = -velocity['x']
    if ball.y + ball.height >= HEIGHT:
        velocity['y'] = -velocity['y']
    if ball.y < 0:
        velocity['y'] = -velocity['y']
    ball.x += velocity['x'] * dt
    ball.y += velocity['y'] * dt

def draw():
    screen.fill((255, 255, 153))
    ball.draw()
```

The ball has a velocity with two components: one for each axis. when the ball touches the border, one of both components is "inverted".

Bouncing

The character moves around and bounces back at the borders.

We create a ball that moves on the screen at a speed of 150 pixels per second. When it reaches the border, it bounces back.

```
WIDTH = 640
HEIGHT = 480

ball = Actor('ball', anchor=('left', 'top'))
velocity = {'x': 150, 'y': 150}

def update(dt):
    if ball.x + ball.width >= WIDTH:
        velocity['x'] = -velocity['x']
    if ball.x < 0:
        velocity['x'] = -velocity['x']
    if ball.y + ball.height >= HEIGHT:
        velocity['y'] = -velocity['y']
    if ball.y < 0:
        velocity['y'] = -velocity['y']
    ball.x += velocity['x'] * dt
    ball.y += velocity['y'] * dt

def draw():
    screen.fill((255, 255, 153))
    ball.draw()
```

The ball has a velocity with two components: one for each axis. when the ball touches the border, one of both components is "inverted".

Bouncing

The character moves around and bounces back at the borders.

We create a ball that moves on the screen at a speed of 150 pixels per second. When it reaches the border, it bounces back.

```
WIDTH = 640
HEIGHT = 480

ball = Actor('ball', anchor=('left', 'top'))
velocity = {'x': 150, 'y': 150}

def update(dt):
    if ball.x + ball.width >= WIDTH:
        velocity['x'] = -velocity['x']
    if ball.x < 0:
        velocity['x'] = -velocity['x']
    if ball.y + ball.height >= HEIGHT:
        velocity['y'] = -velocity['y']
    if ball.y < 0:
        velocity['y'] = -velocity['y']
    ball.x += velocity['x'] * dt
    ball.y += velocity['y'] * dt

def draw():
    screen.fill((255, 255, 153))
    ball.draw()
```

The ball has a velocity with two components: one for each axis. when the ball touches the border, one of both components is "inverted".

Bouncing

The character moves around and bounces back at the borders.

We create a ball that moves on the screen at a speed of 150 pixels per second. When it reaches the border, it bounces back.

```
WIDTH = 640
HEIGHT = 480

ball = Actor('ball', anchor=('left', 'top'))
velocity = {'x': 150, 'y': 150}

def update(dt):
    if ball.x + ball.width >= WIDTH:
        velocity['x'] = -velocity['x']
    if ball.x < 0:
        velocity['x'] = -velocity['x']
    if ball.y + ball.height >= HEIGHT:
        velocity['y'] = -velocity['y']
    if ball.y < 0:
        velocity['y'] = -velocity['y']
    ball.x += velocity['x'] * dt
    ball.y += velocity['y'] * dt

def draw():
    screen.fill((255, 255, 153))
    ball.draw()
```

The ball has a velocity with two components: one for each axis. when the ball touches the border, one of both components is "inverted".

In the `update` function, we check the current position and compare it to the windows size.

When creating the `ball`, we define the Actor to be *anchored* to the top left corner (the default being the center). This makes the comparison with the border simpler: we can diretly compare the top and left borders and can add the actor's width or height for the right and bottom borders.

The `x` and `y` parts of the new `ball`'s position are calculated by adding the corresponding velocity component, multiplied by `dt`. dt is the time elapsed since the last time the `update` function has been called: if we did not multiply the velocity by `dt`, the ball would move by 150 pixels on each call of `update` (very, very often!) and not every second.

When you push the "Play" button, you should see a bull bouncing around the window.

In the `update` function, we check the current position and compare it to the windows size.

When creating the `ball`, we define the Actor to be *anchored* to the top left corner (the default being the center). This makes the comparison with the border simpler: we can diretly compare the top and left borders and can add the actor's width or height for the right and bottom borders.

The `x` and `y` parts of the new `ball`'s position are calculated by adding the corresponding velocity component, multiplied by `dt`. dt is the time elapsed since the last time the `update` function has been called: if we did not multiply the velocity by `dt`, the ball would move by 150 pixels on each call of `update` (very, very often!) and not every second.

When you push the "Play" button, you should see a bull bouncing around the window.

In the `update` function, we check the current position and compare it to the windows size.

When creating the `ball`, we define the Actor to be *anchored* to the top left corner (the default being the center). This makes the comparison with the border simpler: we can diretly compare the top and left borders and can add the actor's width or height for the right and bottom borders.

The `x` and `y` parts of the new `ball`'s position are calculated by adding the corresponding velocity component, multiplied by `dt`. dt is the time elapsed since the last time the `update` function has been called: if we did not multiply the velocity by `dt`, the ball would move by 150 pixels on each call of `update` (very, very often!) and not every second.

When you push the "Play" button, you should see a bull bouncing around the window.

In the `update` function, we check the current position and compare it to the windows size.

When creating the `ball`, we define the Actor to be *anchored* to the top left corner (the default being the center). This makes the comparison with the border simpler: we can diretly compare the top and left borders and can add the actor's width or height for the right and bottom borders.

The `x` and `y` parts of the new `ball`'s position are calculated by adding the corresponding velocity component, multiplied by `dt`. dt is the time elapsed since the last time the `update` function has been called: if we did not multiply the velocity by `dt`, the ball would move by 150 pixels on each call of `update` (very, very often!) and not every second.

When you push the "Play" button, you should see a bull bouncing around the window.