Explaining the physics of PONG

Initially, we have the movement code of the **Player**

We use a 2D static body because we want it to participate in the collision system, but not react according to physics, like a 2D Rigid Body would.

The direction is defined from the input and is normalized so that it does not affect the speed

We need to use the Clamp function on the player to limit its vertical movement. It's more efficient to do it this way than to build a wall just to block your movement.

```python
position = position.clamp(Vector2.ZERO + Vector2(0, menu_up_height) +
object_size/2, screen_size - object_size/2)
```

We also have the **machine**, which varies in initial speed according to the difficulty level set by the player

```python
if GameParams.level == 1:
	speed = easySpeed
Elif GameParams.level == 2:
	speed = normalSpeed
Elif GameParams.level == 3:
	speed = hardSpeed
```

The machine is moved as follows:

If the height of the ball on the screen is greater than my current height (ball above the machine sprite), I should have my direction up. Otherwise, I should have my direction down.

If the distance between the ball and the machine is greater than how much I can move in the next frame (speed multiplied by delta), I move normally. Otherwise, I should only move the necessary distance.

Again we need to limit the position so as not to exceed the limits of the screen

```python
if ballY > machineY:
	direction -= 1
Elif ballY < machineY:
	direction += 1
if abs(dist) > speed * delta:
	moveBy = direction * speed * delta
else:
```

```
        moveBy = dist

position.y -= moveBy
```

But the most important thing is actually the ball. Its initial direction is random, but only between 4 pre-determined values, all divisible by 45 degrees. I also normalize it to avoid its size being larger than expected.

```
Python
direction = possibleDirections[randi() % possibleDirections.size()]
direction.normalized()
```

The ball also has an auxiliary sprite to highlight its direction. Rotation is determined from the inclination of the direction vector with the horizontal axis

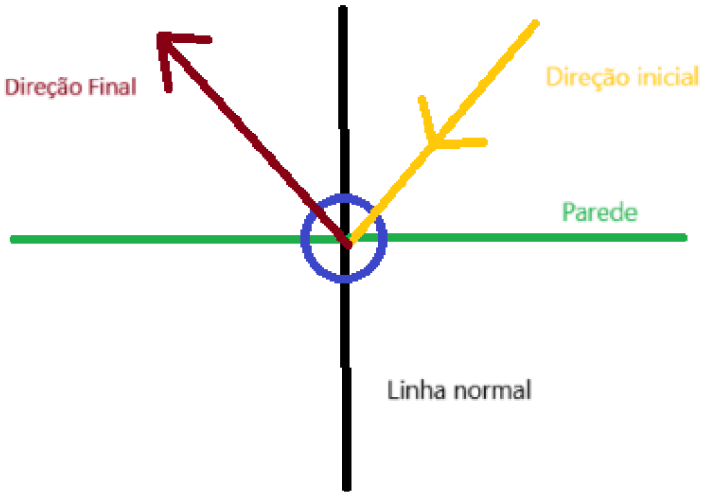
```
Python
pivotSprite.rotation = Vector2(1,0).angle_to(direction)
```

It would look something similar to the drawing below:

For the ball physics, we use a CharacterBody2D, as we want to control both movement and collision manually.

When the ball collides with the wall, we simply need to reflect its direction. To reflect, we need to know the central point. To get this point, you need to find the normal vector of the collision point and to reflect it, just call the bounce function

```
Python
direction = direction.bounce(collision.get_normal())
```



However, when the collision is with the player or the machine, the calculation must be done in a more unpredictable way. The further the ball is from the center of the player, the more intense the reflection will be. Furthermore, to not be extreme, we multiply by a decimal value below 1 to reduce its size, as per the equation below

```python
newDir.y = (dist / (collider.object_size.y / 2)) * MAX_Y_VECTOR
```