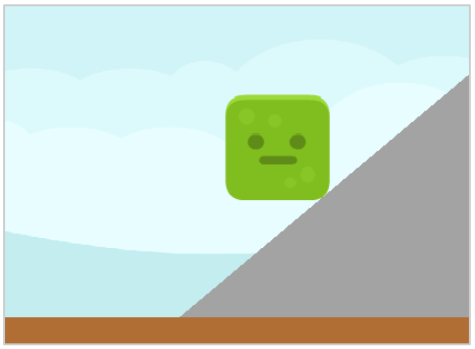
応用テキスト
A04



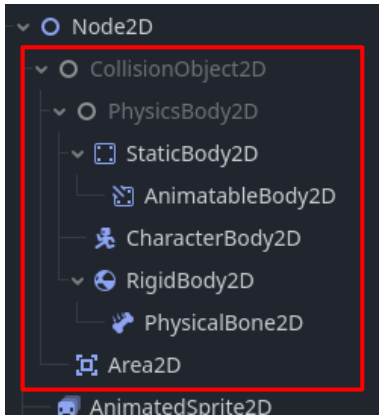
衝突

ver.1.0.0

バージョン4.2.1をベースに作成し、4.3.0にも対応しています。

本書はGodotエンジンの普及、発展を目的に
プログラボ教育事業運営委員会が制作したものです。

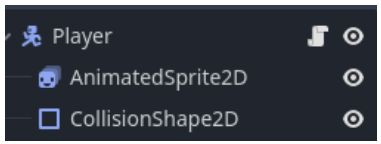
ゲーム内オブジェクト同士の衝突検出は、ゲームの要素としては重要なものの1つです。単純にオブジェクトの座標や面積から当たり判定をすることもできますが、ここではゲームエンジンの機能としてノードの衝突検出を利用します。



Godotで衝突を検出するには「CollisionObject2D」クラスを継承したノードを使います。

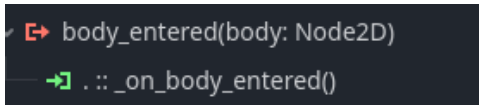
基本テキスト 1-3 衝突検出ノード

これらのノードは、それ単体では衝突範囲を持たないため、CollisionShape2D（CollisionPolygon2D）を子ノードに設定します。



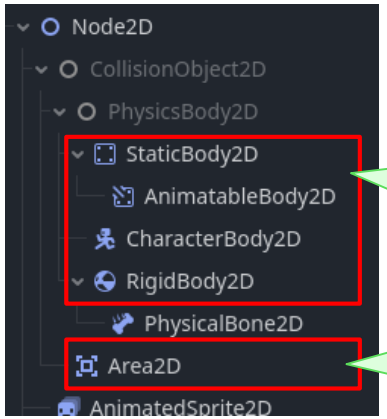
コリジョンノードは1つのシーンに複数配置することで、例えば、障害物との判定と敵との判定を分けて制御するなど、柔軟なゲーム設計を行えます。

また各ノードにあるシグナルを使えば、衝突時に特別なアクションを行うこともできます。



■物理衝突と範囲検出

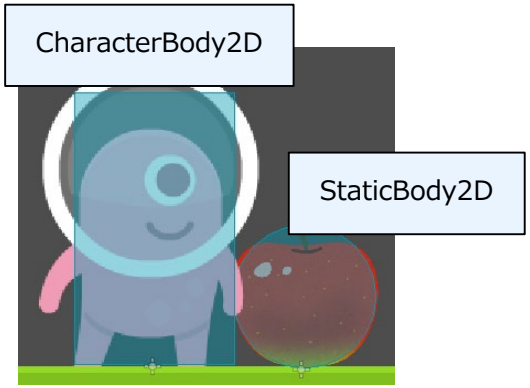
衝突には物理的な衝突を行うものと、重なり（範囲検出）の2種類があります。



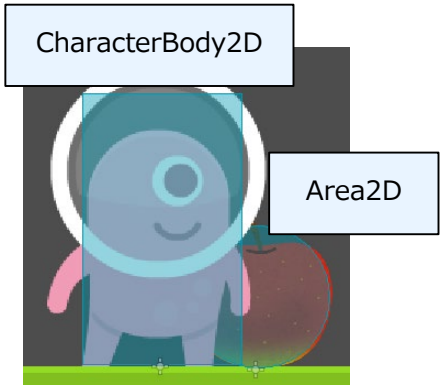
〇〇Body2Dの名前がつくノードは物理的に衝突します。

Area2Dノードはコリジョンの重なり（当たり判定）を検出します。

この例では、リンゴシーンのルートノードの違いによって、衝突の仕方が異なってきます。



物理的に衝突
（重ならない）



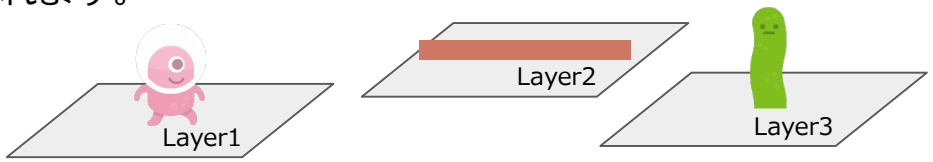
範囲の検出
（重なる）

ゲーム内容によって、衝突をどのように行うかが変わってきますので、適切なノードを選択するようにします。

ゲーム上では、衝突判定させたいものと衝突すべきでないものが混在してしまいます。例えばシューティングゲームでは自機から発射した弾が、自機には当たることはありません。レイヤーとマスクを利用すれば判定範囲を設定できます。

Layer（レイヤー）

レイヤーという言葉は、お絵描きソフトを使ったことがあれば、なじみがあるかもしれませんが、日本語では「層」と訳されます。



考え方としては、この図のように各シーンがそれぞれのレイヤー（層）に分かれているとします。ただしそのためには設定を行います。

[プロジェクト] > [プロジェクト設定...]

レイヤーの名前を、どの役割にするか分かりやすいように設定します。

ここでは、まだレイヤー名を決めただけですので、それぞれのシーンに対して、このレイヤーを割り当てます。

それぞれのシーンのルートノードを選択した状態で、インスペクターの「Collision」を確認すると、下図のような番号が書かれた升目があるのが分かります。

Mask（マスク）

マスクは訳すと「覆う、保護する」ような意味で使われます。ここでは、どのレイヤーとだけ衝突判定するのかを設定するために使います。

Layer：1（Playerレイヤー）
Mask ：2
Layer2、３と衝突判定する

Layer：2（groundレイヤー）
Mask ：1
Layer1と衝突判定する

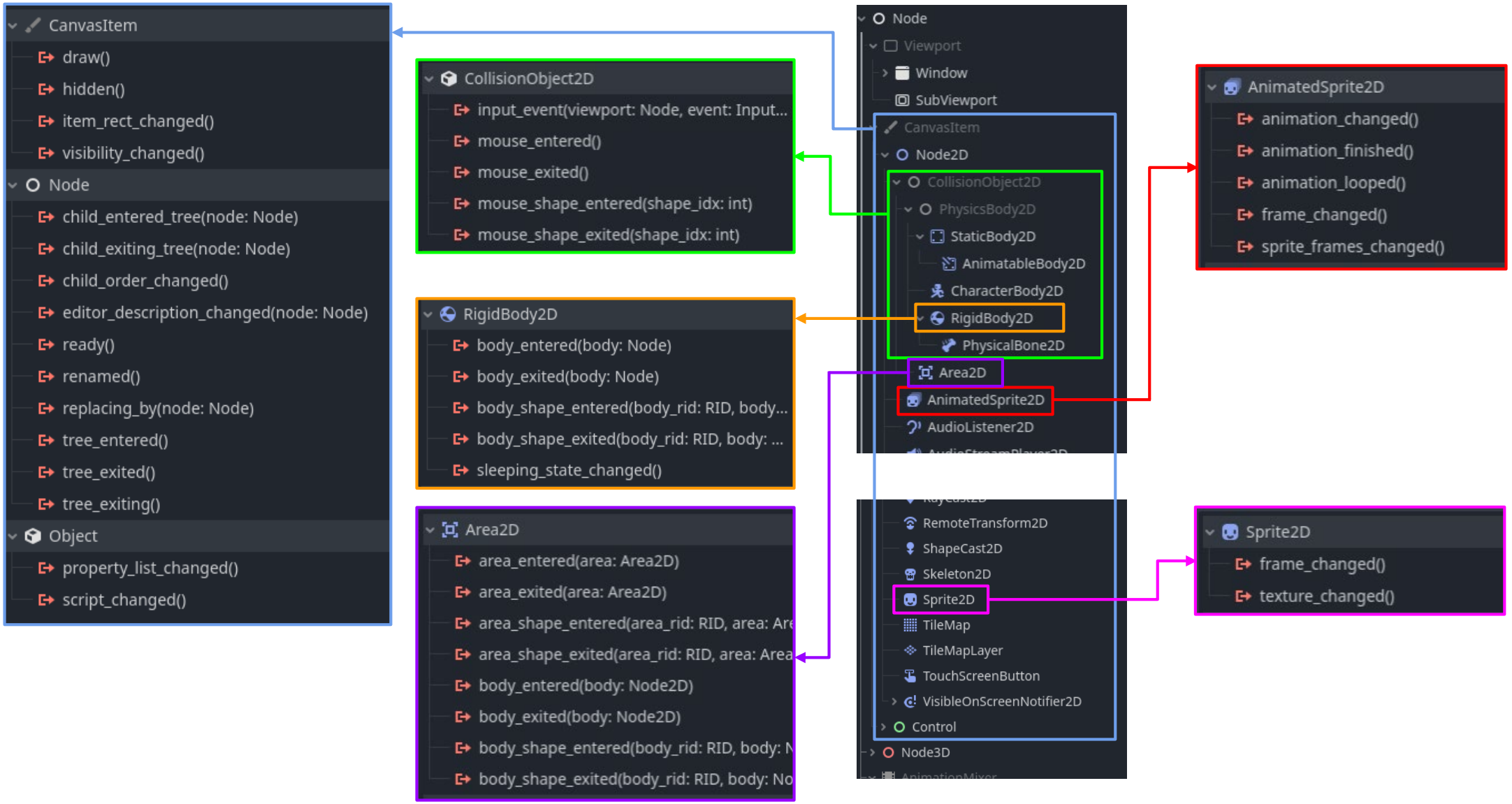
この設定により
地面と敵は衝突判定しません。

Layer：3（enemyレイヤー）
Mask ：1
Layer1と衝突判定する

レイヤーとマスクの設定をしないと、必要のない地面と敵の衝突判定をしてしまい、無駄な処理が発生してしまいます。

よく使うノードとシグナル

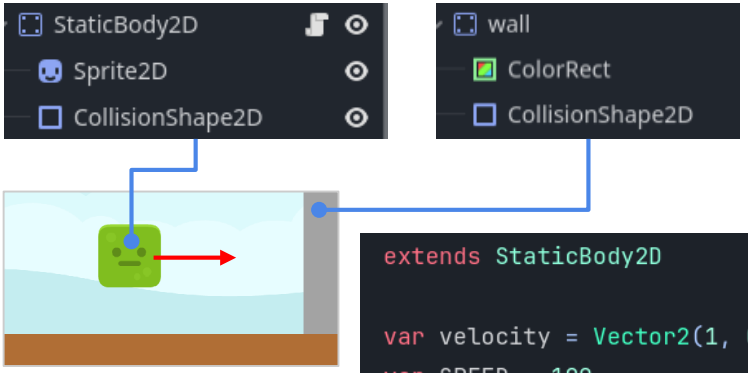
ノードは継承関係にあるため、継承元のノードのシグナルは継承先でも使用できます。
オブジェクト同士の衝突を検出してなんらかの処理を行う場合、シグナルが使えれば
シンプルに処理できますので、どのノードを使えば良いか迷った場合は、シグナルを参
考にしても良いでしょう。



■move_and_collide()

move_and_collide()メソッドは、PhysicsBody2Dクラスを継承するノード（右図参照）で使用できます。

例としてStaticBody2Dを動かします。
ノード構成とスクリプトは以下の通りです。




```gdscript
extends StaticBody2D

var velocity = Vector2(1, 0)
var SPEED = 100

func _physics_process(delta):
    move_and_collide(velocity * SPEED * delta)
```

実行すると緑色のキャラが、右方向にゆっくり移動していきます。他のオブジェクトとの衝突をスクリプトで検出するには、move_and_collide()で取得できるKinematicCollision2Dオブジェクトを利用します。

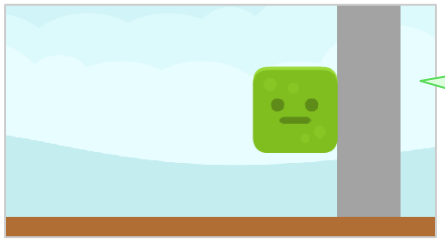
```gdscript
var collision_info = move_and_collide(velocity * SPEED * delta)
```

この変数に、KinematicCollision2Dオブジェクトが格納されます。

KinematicCollision2Dオブジェクトには、衝突に関する情報が含まれています。

衝突した相手の情報を取得するには、さらにget_collider()メソッドを呼び出します

```gdscript
if collision_info:
    var obj_name = collision_info.get_collider().name
    print(obj_name)
```



move_and_collide()で移動しているオブジェクトは衝突すると停止します。

実行結果

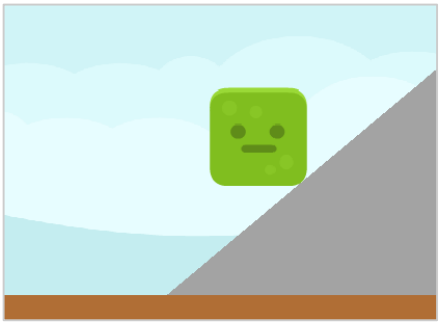
```
wall
```

取得した衝突情報から、get_collider()メソッドで衝突先のオブジェクトを取り出し、nameプロパティ（ノード名）である「wall」を表示しています。

このように、move_and_collide()を使えば衝突を検出し、その詳細情報を取得することができるため、ゲーム内で衝突後に特定の処理をしたい場合などに活用できます。

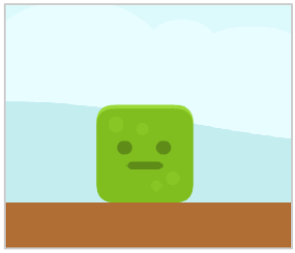
■move_and_slide()

move_and_slide()メソッドは、CharacterBody2Dノード固有のメソッドです。move_and_collide()と異なり、衝突しても停止せず、表面に沿ってスライドするように動きます。



例えばこのように坂道にぶつかった場合、move_and_slide()であれば、スムーズに坂道を登るような移動になります。

重力で地面に設置している場合も、ずっとvelocity.yに下方向の力がかかったままだと、move_and_collide()の場合は、地面に接触してしまい、左右に動けません。



このように、キャラクターをスムーズに動かしたい場合は、move_and_slide()を使うようにします。

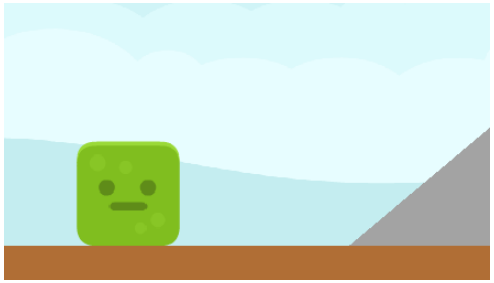
また他のオブジェクトとの衝突をスクリプトで検出するには、get_last_slide_collision()メソッドを使用して、KinematicCollision2Dオブジェクトを取得します。

```
move_and_slide()
var collision_info = get_last_slide_collision()
```

この変数に、KinematicCollision2Dオブジェクトが格納されます。

衝突した相手の情報を取得するには、さらにget_collider()メソッドを呼び出します

```
move_and_slide()
var collision_info = get_last_slide_collision()
if collision_info:
    var obj_name = collision_info.get_collider().name
    print(obj_name)
```



実行結果

```
ground
ground
ground
```

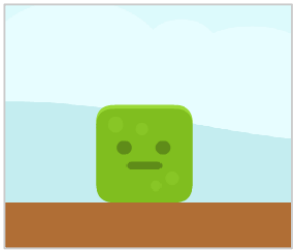
他にも、衝突検出した物体の数を取得して、それぞれに対して処理を行うような書き方もあります。

```
for i in get_slide_collision_count():
    var c = get_slide_collision(i)
    if c.get_collider() is RigidBody2D:
        print("RigidBody2Dと衝突")
```

衝突相手がRigidBody2Dの時になんらかの処理をするような場合。

■is_on_floor()

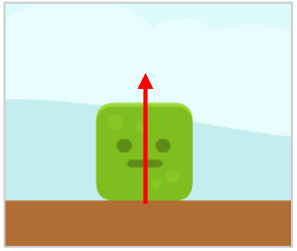
CharacterBody2Dノードでキャラクターを動かすときに、move_and_slide()メソッドを使用しましたが、その時に合わせてよく使うのが、is_on_floor()メソッドです。



floor（フロア）とは英語で「床」の意味を持ちますので、床（地面）に接触しているかどうかを判定するメソッドです。

trueが返ってくれば接している状態で、他にも天井や壁との接触を判定できます。

is_on_ceiling()	キャラクターが天井に接触しているかどうか
is_on_ceiling_only()	キャラクターが天井のみに接触しているかどうか
is_on_floor()	キャラクターが地面に接触しているかどうか
is_on_floor_only()	キャラクターが地面のみに接触しているかどうか
is_on_wall()	キャラクターが壁に接触しているかどうか
is_on_wall_only()	キャラクターが壁のみに接触しているかどうか

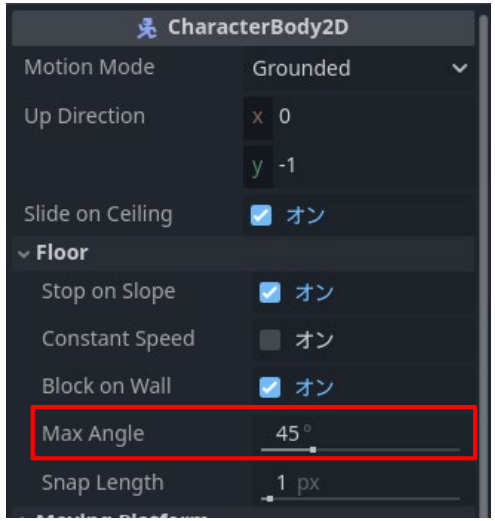


そもそも何を持って「床」と判定しているのでしょうか？
Godotは内部的に接触面に対して垂直に向かうベクトルを持っており、そのベクトルの向きで床や壁として認識するようにできています。

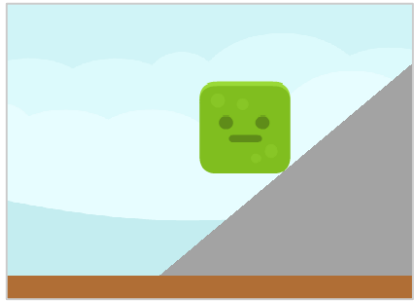
坂道の場合では、地面か壁かの区別をCharacterBody2DのMax Angleプロパティで設定されます。
デフォルトでは45度です。

スクリプトで変更する場合は、floor_max_angleに値を設定します。

```
floor_max_angle = 60
```



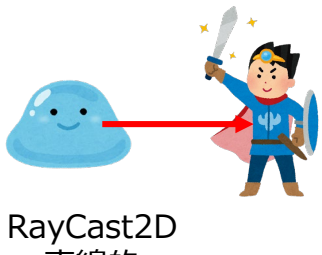
傾斜が最大設定角度を超えた場合は、壁と判定されてキャラクターは登ることはできません。



is_on_floor()をチェックして、床に触れているならジャンプ操作を受け付ける、逆に床に触れていないならジャンプできないなど、キャラクターの操作をつくるときに必須の要素になっています。

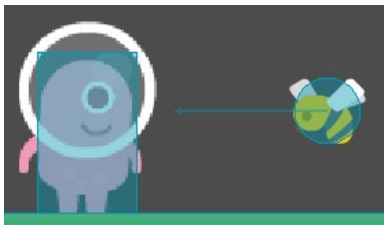
物理的な接触ではなく、範囲検出を行う際に便利なのがArea2Dノードです。同じように物理的に接触せずに衝突判定をできるものに、RayCast2D（レイ・キャスト2D）ノードがあります。
Ray（光線）Cast（投げる）という意味があり、目に見えない光線を放射して衝突情報を検知するしくみです。

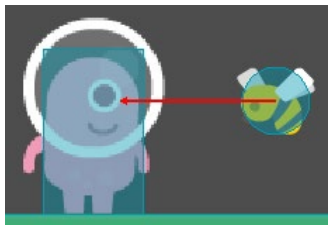

Area2D
範囲


RayCast2D
直線的

例として敵キャラクターにRayCast2Dノードを追加します。
矢印の向きはキャラクターの絵の向きに合わせています。

RayCast2Dの矢印に触れると、矢印が赤色に変化して反応していることが分かります。





スクリプト例です。プレイヤーが敵キャラのRayCastに触れると、敵キャラが追いかけてくるようになります。

```gdscript
extends CharacterBody2D

@export var speed: float = 100.0
@onready var raycast = $RayCast2D
@onready var player = null

func _physics_process(delta):
    if raycast.is_colliding():
        var target = raycast.get_collider()
        if target and target.name == "Player":
            player = target

    # プレイヤーを追跡
    if player:
        var direction = (player.position - position).normalized()
        velocity = direction * speed
    else:
        velocity = Vector2.ZERO

    move_and_slide()
```

raycast.is_colliding()で、コリジョンとの接触判定を行います。
このように毎フレーム接触判定を行えるのがポイント。

Area2Dノードでは接触したことをシグナルを使って通知し処理を行いますが、RayCast2Dノードでは好きなタイミングでチェックすることができ、ピンポイントに方向性を持たせた判定ができます。

いろいろな場面で使えますので、使いこなしたいノードと言えます。