CODE NINJAS®

Silver Belt Glossary & Documentation

Silver Belt Glossary & Documentation

All new terms and concepts introduced in the Godot curriculum have been listed below. A definition/description is provided for each term/concept and images are provided where necessary.

Contents

Activity 01: Robomania 5

apply_central_impulse() 5

Collision Layers 5

PhysicsMaterial 6

Activity 02: Find The Exit 7

atan2(y,x) 7

lerp_angle() 7

print() 8

RemoteTransForm3D 8

Activity 03: Shape Jam 9

get_viewport_rect() 9

Instantiation Code Pattern 9

Modulo % 10

range() 10

Rect2 10

translate() 11

Activity 04: Jungle Escape 12

Collision Mask 12

Enum 12

get_collider() 13

get_parent() 14

is_colliding() 15

Match statements 16
name 16
RayCast3D 16
Activity 05: Cloud Hop 17
apply_impulse() 17
move_and_slide() 17
move_toward() 18
Activity 06: Evil Fortress of Doctor Worm 19
connect() 19
emit_signal() 20
get_overlapping_bodies() 21
Not operator 21
Activity 07: CyberFu Part 1 22
clamp() 22
distance_to() 22
get_slide_collision_count() 22
look_at() 23
Activity 08: World of Color 24
Array 24
get_node() 24
Replayability 24
shuffle() 24
Activity 09: Amazing Ninja Worlds Part 1 25
call_deferred() 25
Activity 10: CyberFu Part 2 26
current_animation 26
is (keyword) 26
Activity 11: Labyrinth 27

Baking....27
get_next_path_position() 27
NavigationAgent3D....28
NavigationObstacle3D 28
NavigationRegion3D 29
Activity 12: Amazing Ninja Worlds Part 2 30
MarginContainer....30
Activity 13: Food Frenzy Part 1 31
Anchor....31
Anchor Points....32
class_name....32
Control Nodes 33
super()....33
Activity 14: Scavenger Hunt Deluxe 34
Array.size() 34
ColorRect 34
HBoxContainer 35
pick_random() 35
Unique Name 36
Activity 15: Amazing Ninja Worlds Part 3 37
No New Concepts 37
Activity 16: Food Frenzy Part 2 38
create_tween() 38
set_ease()....39
set_trans() 40
Tween 41
tween_Property() 41
Activity 17: Food Frenzy Part 3....42

const 42
Dictionary 42
file_exists() 43
open() 43
stringify() 44

ACTIVITY 01: ROBOMANIA

APPLY_CENTRAL_IMPULSE()

`apply_central_impulse()`**:** part of the RigidBody2D/3D classes, applies an impulse to the body without affecting its rotation. Impulses are instantaneous changes in the momentum of a body.

Parameters:

1. `impulse` ***(Vector2/3)*****:** the direction and magnitude of the impulse.

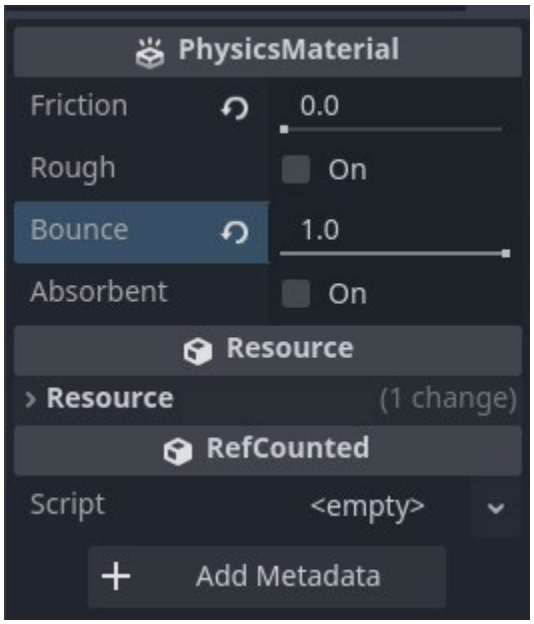
Returns *(void)*: this method is void, it returns nothing.

COLLISION LAYERS

Collision layers allow developers to tell objects what they are supposed to collide with. The projectile is on a different layer, so it doesn't collide with the crusher!

PHYSICSMATERIAL

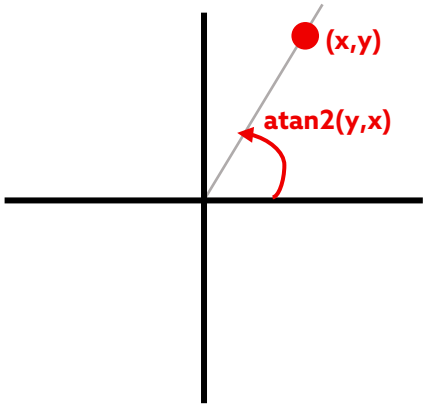
PhysicsMaterials can be attached to a **RigidBody** to set the RigidBody's friction and bounciness.



Activity 02: Find The Exit

atan2(y,x)

Calculates the angle between the positive x axis and the point (x,y), measured with the origin as the pivot.



lerp_angle()

Computes the angle value between the given from and to values given by weight.

`lerp_angle()`: computes the angle value between the given from and to values given by weight

Parameters:

1. **from *(int)*:** the start angle
2. **to *(int)*:** the destination angle
3. **weight *(float)*:** decimal (0<weight<1) representing percentage traveled between **from** and **to**

Returns *(float)*: The angle value that is (weight*100) percent of the way between **from** and **to**, starting at **from**.

```
15    if input_dir != Vector3.ZERO:
16        anim_player.play("Run")
17        var target_angle: float = atan2(input_dir.x, input_dir.z)
18        rotation.y = lerp_angle(rotation.y, target_angle, turn_rate)
19    else:
20        anim_player.stop()
```

PRINT()

Displays given arguments as text in the Output Bottom Panel of the editor while the

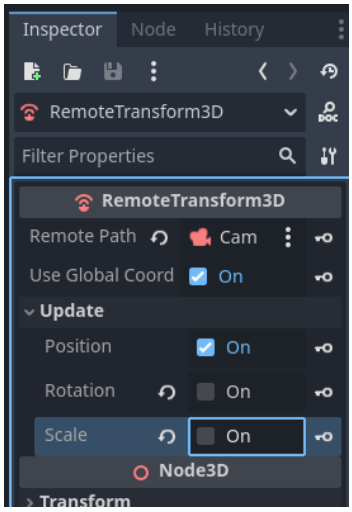
> **`print()`:** displays given arguments as text in the Output Bottom Panel of the editor while the game is running.
>
> **Parameters:**
>
> 2. **argument *(any)*:** any number of arguments of any type may be sent to a print call.
>
> **Returns *(void)*:** No return value but displays all arguments together as nicely as possible to the Output Bottom Panel, followed by a newline character.

```
5  func _physics_process(delta: float) -> void:
6      #global_position += Vector3.FORWARD * global_basis * move
7      linear_velocity = Vector3.FORWARD * move_speed
8
9      print(Input.get_axis("move_left", "move_right"))
```

game is running.

REMOTETRANSFORM3D

A node that sends its own transform information to another node. It can be adjusted so that only some information is sent, instead of all of it.



Activity 03: Shape Jam

get_viewport_rect()

> **`get_viewport_rect()`:** part of the CanvasItem class, returns the boundaries of the viewport as a Rect2.
>
> **Parameters:** none
>
> **Returns *(Rect2)*:** the boundaries of the viewport. Includes Vector2 properties **position, end,** and **size.**

Instantiation Code Pattern

Instantiating objects follows a certain pattern.

The pattern is:

1. Instantiate to memory
2. Handle object-specific stuff
3. Add to the scene tree
4. Handle scene-specific stuff

```
26  # --------------------
27  # TODO 2
28  # Enemy spawning
29  # --------------------
30  var enemy: Area2D = enemy_scene.instantiate()
31  enemy.waypoints = waypoints
32  add_child(enemy)
33  enemy.global_position = global_position
34
```

MODULO %

This operator computes the remainder of integer division. For example: **7 % 4 = 3, 12 % 6 = 0, 8 % 2 = 0.** It is mostly used to check if a number is even by comparing the output of the number modulo 2 to 0 or 1. If it is 0, that means there is no remainder, so the number is even. Otherwise, it's odd.

RANGE()

Returns an array of numbers starting at 0 to the parameter -1.

> **`range()`:** generates an array of integers starting at 0 and incrementing by 1 up to the given end.
>
> Parameters:
>
> 1. **end *(int)*:** the maximum value, excluded from the array
>
> **Returns *(Array)*:** An array of integers from 0 to end, excluding end

```
for i in range(5):
    pass
```

```
for (let value of list) {

}
```

RECT2

A class that stores bounds of a 2D rectangle with the Vector2 properties **position** (top-left coordinate), **end** (bottom-right coordinate), and **size** (width, height).

TRANSLATE()

`translate()`: part of the Node2D/3D classes, changes the node's position by the given Vector2D/3D offset.

Parameters:

1. **offset *(Vector2)*:** the direction and magnitude of the desired translation

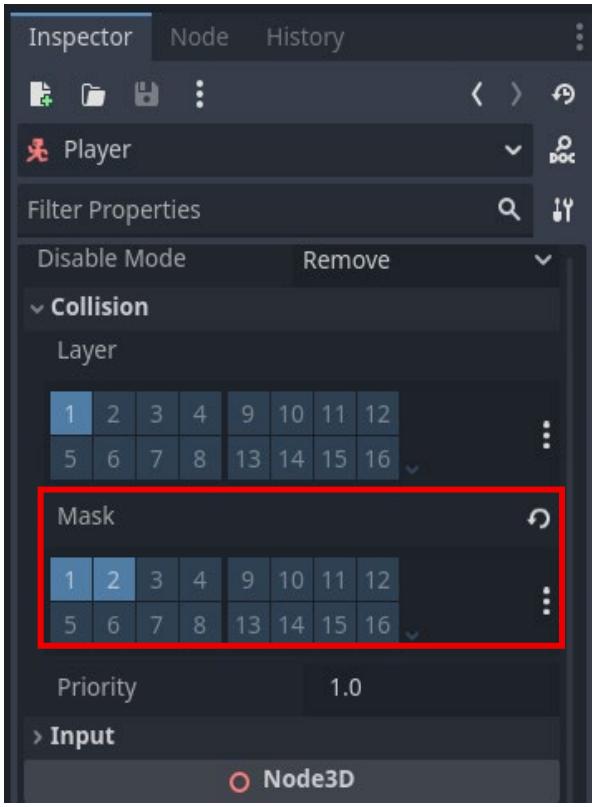
Returns *(void)*: No return value but updates the position of the node by adding the provided Vector2D/Vector3D offset to it.

```
1   extends Area2D
2
3   var speed: float
4   var direction: Vector2
5
6   func _process(delta):
7       translate(direction * speed * delta)
8
```

Activity 04: Jungle Escape

Collision Mask

A body's **Collision Mask** describes which **Collision Layers** the current body will scan for to handle collisions. The body will ignore all objects that do not lie on any of the **Collision Mask's** layers.



Enum

Enum, short for enumeration, is a user-defined data type. Enums are very helpful for making code easier to read and for variables that can only be equal to a small set of values.

```
20  enum AnimationState
21  {
22      IDLE,
23      WALKING,
24      JUMPING,
25  }
26
```

GET_COLLIDER()

`get_collider()`: part of the RayCast2D/3D classes, returns the first object that the ray intersects with, starting from its source.

Parameters: *none*

Returns (*Node*): the first object that the ray intersects with, starting from its source

```
97  # --------------------
98  # TODO 3
99  # Raycast collisions
100 # --------------------
101 if ray_disappear.is_colliding():
102     var other_object = ray_disappear.get_collider()
103     var platform_node = other_object.get_parent()
104     print("WARNING: " + platform_node.name)
105 else:
106     print("Ray is not colliding.")
107
```

GET_PARENT()

`get_parent()`: part of the Node class, returns the parent of this node.

Parameters: *none*

Returns (*Node*): the parent of the node. If there is no parent, returns null instead

```
97  # --------------------
98  # TODO 3
99  # Raycast collisions
100 # --------------------
101 if ray_disappear.is_colliding():
102     var other_object = ray_disappear.get_collider()
103     var platform_node = other_object.get_parent()
104     print("WARNING: " + platform_node.name)
105 else:
106     print("Ray is not colliding.")
107
```

IS_COLLIDING()

`is_colliding()`: part of the RayCast2D/3D classes, returns whether the given Raycast is colliding with any objects on its masked collision layers.

Parameters: *none*

Returns (*boolean*): whether the target is colliding with any objects on its masked collision layers

```
97  # --------------------
98  # TODO 3
99  # Raycast collisions
100 # --------------------
101 if ray_disappear.is_colliding():
102     var other_object = ray_disappear.get_collider()
103     var platform_node = other_object.get_parent()
104     print("WARNING: " + platform_node.name)
105 else:
106     print("Ray is not colliding.")
107
```

MATCH STATEMENTS

Match statements compare a variable to multiple possible values. It executes the corresponding block of code when a match is found. It's a cleaner and more efficient way of writing a series of if statements.

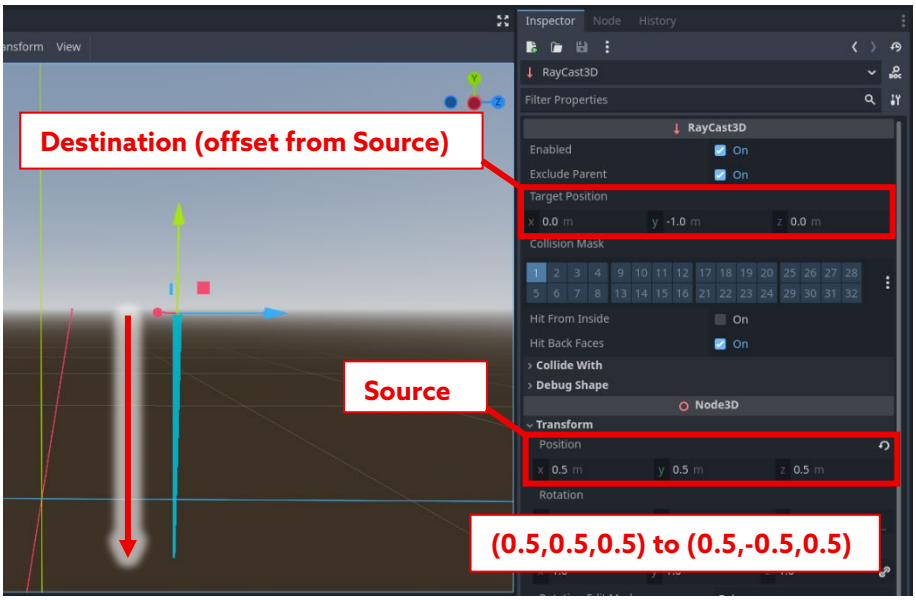
```
27  func handle_animation(anim_state: AnimationState) -> void:
28      match anim_state:
29          AnimationState.IDLE:
30              if anim.current_animation != "PlayerLibrary/IdlePose":
31                  anim.play("PlayerLibrary/IdlePose")
32          AnimationState.JUMPING:
33              if anim.current_animation != "PlayerLibrary/Jump":
34                  anim.play("PlayerLibrary/Jump")
35          AnimationState.WALKING:
36              if anim.current_animation != "PlayerLibrary/Walk":
37                  anim.play("PlayerLibrary/Walk")
38
```

NAME

Part of the Node class, **name** stores a node's name seen in the Scene panel.

RAYCAST3D

A node that creates a ray with a source and a destination, where the destination is an offset from the source. This ray can detect collisions.



Activity 05: Cloud Hop

apply_impulse()

Part of the RigidBody2D, applies a positioned impulse to the body.

> `apply_impulse()`**:** applies a positioned impulse to the body.
>
> **Parameters:**
>
> 1. **impulse *(Vector2)*:** the strength of the impulse.
> 2. **position *(Vector2)*:** the offset from the body origin in global coordinates.

move_and_slide()

Returns a list of intersecting PhysicsBody3Ds and GridMaps.

> `get_overlapping_bodies()`**:** Returns a list of intersecting PhysicsBody3Ds and GridMaps.
>
> For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved.
>
> **Parameters:** *None*
>
> **Returns:** An array of overlapping bodies.

MOVE_TOWARD()

`move_toward()`**:** Returns a new vector moved toward to by the fixed delta amount. Will not go past the final value.

Parameters:

1. **from *(Vector2)*:** the current value.
2. **to *(Vector2)*:** the target value.
3. **Delta *(float)***: the maximum amount to move in one step.

Returns: **Float**

ACTIVITY 06: EVIL FORTRESS OF DOCTOR WORM

CONNECT()

Connects a signal to a callable. The code may look different depending on whether the method is called on a **node with a signal** or **directly on a signal**.

```
18  # --------------------
19  # TODO  7
20  # connect switch to door
21  # --------------------
22  for switch in switches:
23      switch.connect("toggled", _switch_check)
24
```

```
16  # --------------------
17  # TODO  5
18  # _ready() method
19  # --------------------
20  func _ready() -> void:
21      area.body_entered.connect(_update_switch)
22      area.body_exited.connect(_update_switch)
23
```

EMIT_SIGNAL()

Emits a built in or custom signal by name.

`emit_signal()`: part of the Object class, emits the given signal by name. The signal must exist as a built-in signal within this class or one of its inherited classes, or a user-defined signal.

Parameters: *This method supports a variable number arguments, so parameters can be passed as a comma separated list.*

Returns: Nothing, unless the signal does not exist or parameters are invalid, then an error is returned.

```
11  # --------------------
12  # TODO  6
13  # Create signal
14  # --------------------
15  signal toggled(pressed: bool)
16
```

```
42      # --------------------
43      # TODO  3
44      # Toggle switch
45      # --------------------
46      if touching_switch_presser and not pressed:
47          pressed = true
48          pressed_audio.play()
49          emit_signal("toggled", true)
50      elif not touching_switch_presser and pressed:
51          pressed = false
52          unpressed_audio.play()
53          emit_signal("toggled", false)
54
```

GET_OVERLAPPING_BODIES()

`get_overlapping_bodies()`: Returns a list of intersecting PhysicsBody3Ds and GridMaps.

For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved.

Parameters: *None*

Returns: An array of overlapping bodies.

```
20
21  # --------------------
22  # TODO  1
23  # _update_switch() & variables
24  # --------------------
25  func _update_switch(_body):
26      var bodies = area.get_overlapping_bodies()
27      var touching_switch_presser = false
28
```

NOT OPERATOR

Part of the GDScript language, **not** is a logical operator that inverts a Boolean value.

The not operator:

if **true** (returns **true**)

if **false** (returns **false**)

if **not true** (returns **false**)

if **not false** (returns

if **true** and **not true**

→ if **true** and **false**

(returns **false**)

Activity 07: CyberFu Part 1

clamp()

`clamp()`: a mathematical utility used to restrict a value within a specified minimum and maximum range. It ensures that a given value will never be less than the minimum or greater than the maximum.

Parameters:

- **Value (*Variant*):** The value to be clamped.
- **Min (*Variant*):** The minimum allowed value.
- **Max (*Variant*):** The maximum allowed value.

Returns (*Variant*): A value not less than min and not more than max.

distance_to()

`distance_to()`: A built-in method that calculates the distance between two Vector3.

Parameter (*Vector*): The vector3 to compare to.

Returns (*float*): The distance between the two vectors.

get_slide_collision_count()

`get_slide_collision_count()`: checks if anything is colliding with this node.

Parameters: None

Returns: the number of collisions colliding with this node.

LOOK_AT()

Part of the Node2D class, **look_at()** rotates a node so that it faces a given position in world space.

`look_at()`: part of the RayCast2D/3D classes.

Parameters:

- **target (*Vector3*):** The position of the target to look at.
- **up (*Vector3*):** The direction of Up in relation to this node.
- **use_model_front (*bool*):** The direction of the z-axis being positive or negative. If true, the +Z axis is treated as forward.

Returns (*boolean*): whether the given RayCast is colliding with any objects on its

Activity 08: World of Color

Array

A built-in data type that stores an ordered list of values.

get_node()

Part of the Node class, **get_node()** returns a reference to another node in the scene tree.

```
for i in range(0, obstacle_count):
    var obstacle = obstacle_scene.instantiate()
    get_parent().add_child(obstacle)

    obstacle.position = Vector2(1200, obstacle_height * (i - 1))

    var obstacle_sprite = obstacle.get_node("CollisionShape2D/Sprite2D")
```

Replayability

The number of times a game can be played while still giving the player a new experience each time.

shuffle()

A part of the array class, it takes every item in an array and shuffles it in a random order, automatically setting the array to the new shuffled one.

`shuffle()`: Shuffles all elements of the array in a random order.

Parameters: None

Returns (*void*): This method is void; it doesn't return anything.

```
func spawn_obstacles():
    # --------------------
    # TODO 9:
    # Shuffle the color and shape arrays so obstacles are different each time.
    # --------------------
    color_array.shuffle()
    shape_array.shuffle()
```

ACTIVITY 09: AMAZING NINJA WORLDS PART 1

CALL_DEFERRED()

Part of the Object class, **call_deferred()** schedules a method to be called after the current frame or logic step finishes.

```
10  # --------------------
11  # TODO 4
12  # Create the change_level function
13  # --------------------
14  func change_level(string_path : String) -> void:
15      keys_left = 0
16      get_tree().call_deferred("change_scene_to_file", string_path)
```

Activity 10: CyberFu Part 2

current_animation

In the AnimationPlayer class; returns the name of the animation that is currently playing.

> `current_animation`: Can be used to check the key of the currently playing animation. If no animation is playing, the property's value is an empty string.
>
> **Parameters:** None
>
> **Returns (*Void*)**: This property is void; it doesn't return anything.

is (keyword)

In GDScript, the **is** operator functions differently than the **==** equality operator. Rather than checking if two things are equal, **is** checks if two things are the same type. It tests whether a variable extends a given class, or is of a given built-in type.

```
20    # TODO 3
21    func handle_body_entered(body: Node3D):
22        if body is PhysicalBone3D:
23
```

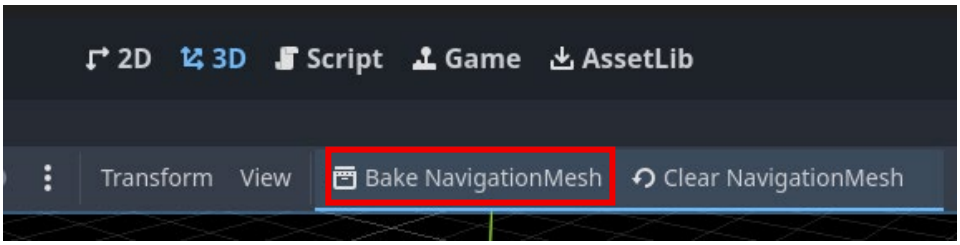
PhysicalBone3D

The **PhysicalBone3D** node represents a segment of an animated 3D model, which connects to other **PhysicalBone3D** nodes by joints on each end.

Activity 11: Labyrinth

Baking

Part of the Godot rendering and lighting system, **baking** refers to the process of precomputing lighting, shadows, or navigation data.



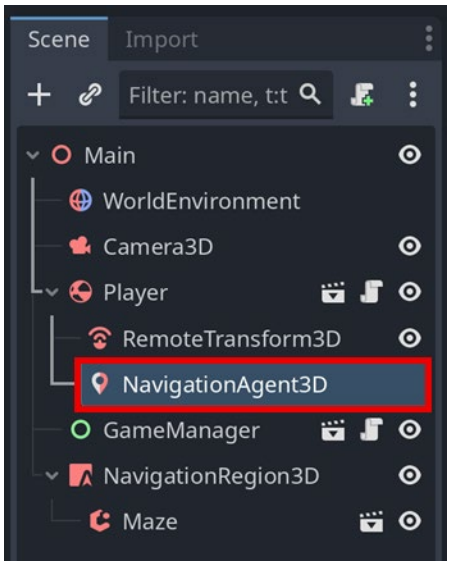
get_next_path_position()

Part of the NavigationAgent2D class, returns the next point along the agent's current navigation path.

> `get_next_path_position()`: Returns the next position in global coordinates that can be moved to, making sure that there are no static objects in the way. If the agent does not have a navigation path, it will return the position of the agent's parent.
>
> The use of this function once every physics frame is required to update the internal path logic of the NavigationAgent.
>
> **Parameters:** *None*
>
> **Returns:** Vector3D

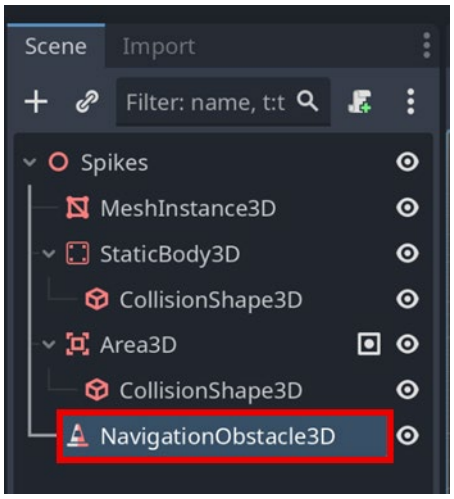
NavigationAgent3D

A **NavigationAgent3D** is a 3D agent used to pathfind a position while avoiding obstacles.



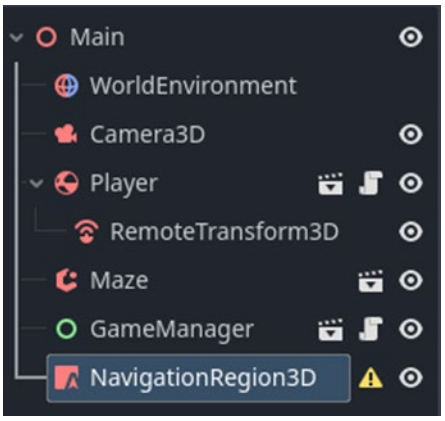
NavigationObstacle3D

Part of the Godot 3D navigation system, **NavigationObstacle3D** is a Node3D that defines an area or object which influences how navigation agents plan and follow paths.



NavigationRegion3D

A **NavigationRegion3D** is a traversable 3D region based on a NavigationMesh that NavigationAgents3Ds can use for pathfinding.



Activity 12: Amazing Ninja Worlds Part 2

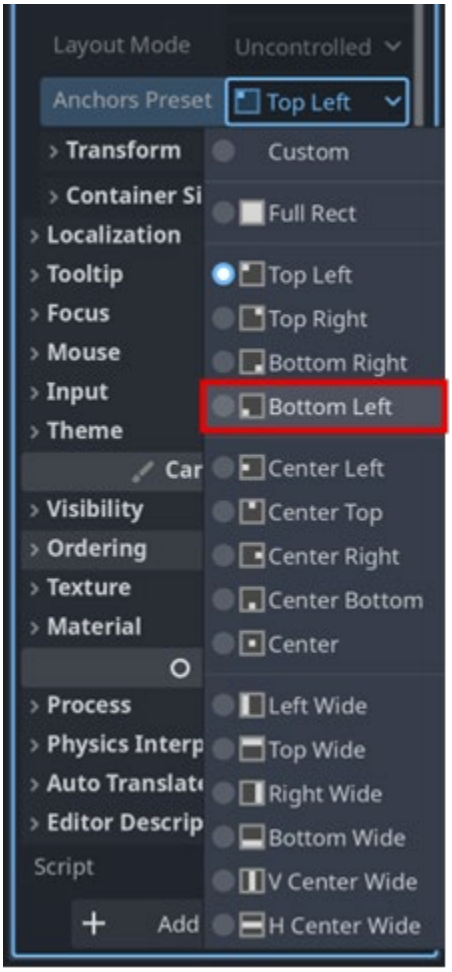
MarginContainer

This node adds an adjustable margin on each side of its child controls. It is made specifically to evenly adjust spacing around its child nodes to help align elements nicely. The margins are added around all children, not around each individual one.

Activity 13: Food Frenzy Part 1

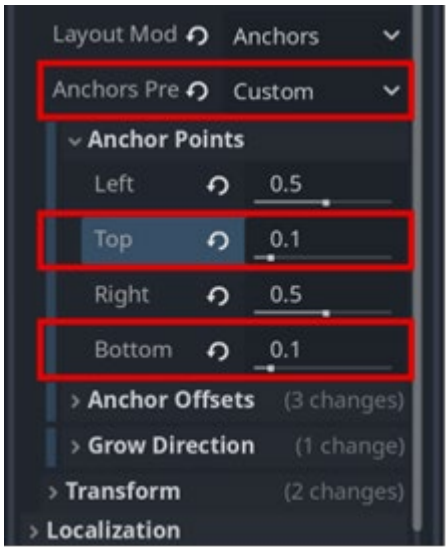
Anchor

Anchors help keep the position of a component within the UI while accounting for changes in size of the parent node. This is helpful when designing the UI for a game that may be played on different screen sizes, such as laptops and cellphones.



Anchor Points

There are four Anchor Points: Left, Top, Right, and Bottom. These points adjust from 0.0 to 1.0, with 0.5 being the center. Anchor points are relative to the parent control or viewport and are positioned relative to the top left corner of the bounding rectangle.



CLASS_NAME

The keyword class_name defines the script as a globally accessible class with the specified name, UI. This allows other scripts to extend the UI class and access the functions and variables within it.

```
1  extends CanvasLayer
2  class_name UI
3
4
5
```

Control Nodes

Control nodes are the base class for all UI (user interface) related nodes, such as Label and TextureRect. The Control class has a **bounding rectangle**, which defines its size, an **anchor position** relative to the parent control (usually the parent node) or viewport and an **anchor offsets** relative to the anchor position.

SUPER()

The super() method is used to call a function or method in the parent class.

```
 9
10  func _ready():
11      super()
12      for i in range(0, len(obstacle_types)):
13          num_obstacles_left += len(grid._getPiecesOfType(obstacle_types[i]))
14      # --------------------
15      # TODO 7
16      # set num_obstacles_left & num_moves
17      # --------------------
18      # set num_obstacles
19      # set num_moves
20
```

Activity 14: Scavenger Hunt Deluxe

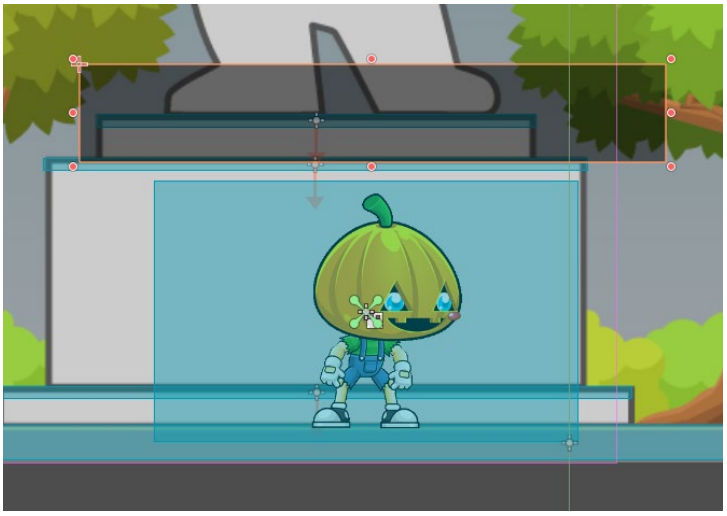
Array.size()

Part of the array class that gives the number of objects inside of an array. Does not return the biggest number in an array. (An array of 5 will return 5, not 4.)

```
# --------------------
# TODO 7:
# Create arrays of indexes for colors and shapes.
# Hint: fill each array with numbers 0...size-1
# --------------------
shape_array = []
shape_array.resize(shapes.size())
```

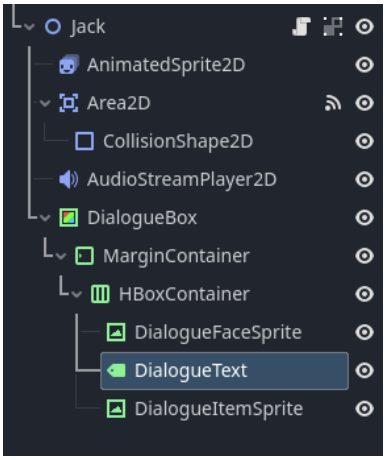
ColorRect

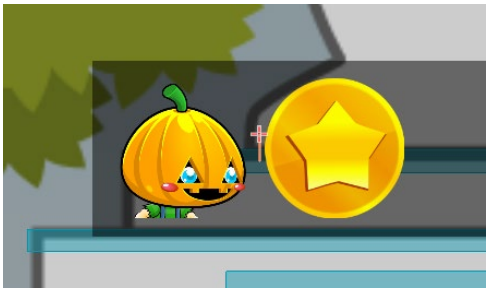
The **ColorRect** node has one property: **color**. It displays a rectangle that is filled with **color** which has support for transparency (alpha).



HBoxContainer

The **HBoxContainer** node determines the position of its child nodes by arranging them horizontally and removes their ability to set their anchor preset.





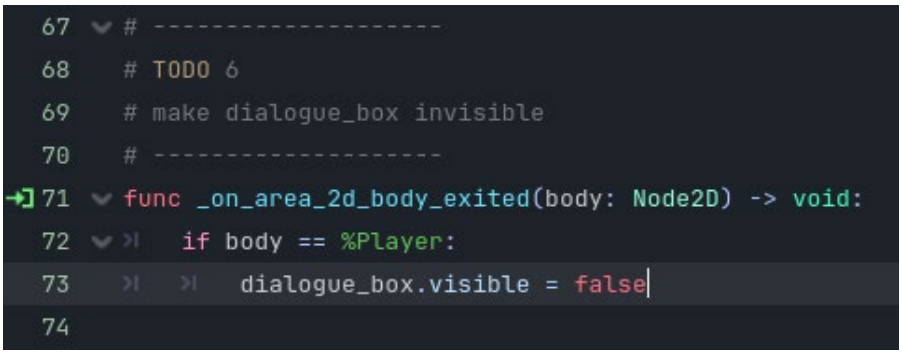
pick_random()

Part of the Array class, it returns a random member of the array. It acts as a shorthand for a random index selection.

> `pick_random()`: part of the Array class, it returns a random member of the array.
>
> **Parameters:** *none*
>
> **Returns (*Variant*)**: a random member of the array

Unique Name

Nodes can be accessed from scripts by using **Unique Names** instead of `get_node()` or `get_first_node_in_group()`. Unique Names can be granted to nodes in the **Scene** panel and are accessed in code by typing `%[node name]`.



Activity 15: Amazing Ninja Worlds Part 3

No New Concepts

This activity does not introduce any new concepts.

ACTIVITY 16: FOOD FRENZY PART 2

CREATE_TWEEN()

Creates and starts a new tween object.

`create_tween()`: Creates a new Tween and binds it to this node.

Parameters: *None*

Returns (`tween`): Returns the newly created Tween.

```
 99
100 #------
101  # TODO 5: game_over_animation() function
102  #------
103 func game_over_animation() -> void:
104     var end_y = game_over_UI.position.y
105
106     game_over_UI.position.y = - 200
107     game_over_UI.rotation_degrees = -45
108
109     var tween = create_tween()
110
```

SET_EASE()

A **Tween** method that defines how the animation accelerates/decelerates over time.

> **`set_ease()`:** Sets the type of ease used.
>
> **Parameters:**
>
> 1. **ease (`EaseType`):** type of ease
>
> **Returns (`PropertyTweener`):** Returns the PropertyTweener, which auto-generates the in-between frames (interpolates) of the object's property over time

EaseType:

- `EASE_IN`
- `EASE_OUT`
- `EASE_IN_OUT`
- `EASE_OUT_IN`

```
107    game_over_UI.rotation_degrees = -45
108
109    var tween = create_tween()
110
111    tween.tween_property(
112        game_over_UI,
113        "position:y",
114        end_y,
115        GO_animation_time
116    ).set_trans(Tween.TRANS_BOUNCE).set_ease(Tween.EASE)
117
```

EASE_IN
EASE_IN_OUT
EASE_OUT
EASE_OUT_IN
EaseType
TWEEN_PAUSE_BOUND
TWEEN_PAUSE_PROCESS

SET_TRANS()

A method of **Tween** that sets the transition type used.

> **`set_trans()`:** Sets the type of transition used.
>
> **Parameters:**
>
> 1. **trans *(`TransitionType`)*:** type of transition
>
> **Returns *(`PropertyTweener`)*:** Returns the PropertyTweener, which auto-generates the in-between frames (interpolates) of the object's property over time.

TransitionType:

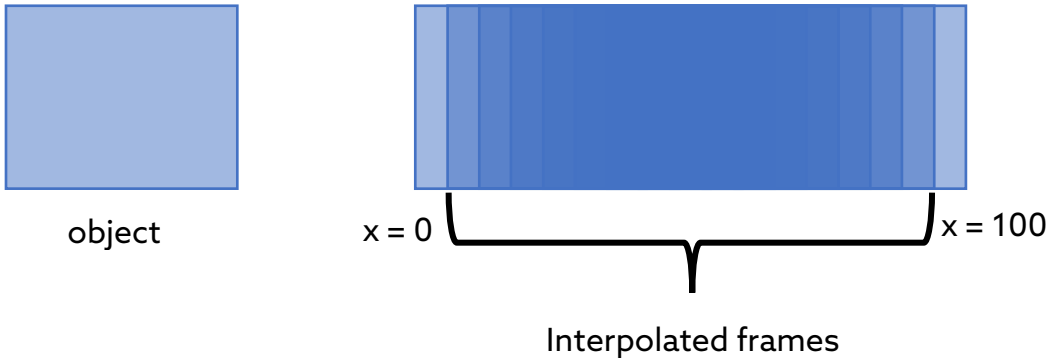
- `TRANS_ELASTIC`
- `TRANS_BOUNCE`
- `TRANS_BACK`
- `TRANS_SPRING`
- `TRANS_LINEAR`
- `TRANS_SINE`
- `TRANS_QUINT`
- `TRANS_QUART`
- `TRANS_QUAD`
- `TRANS_EXPO`
- `TRANS_CUBIC`
- `TRANS_CIRC`

```
103 func game_over_animation() -> void:
104     var end_y = game_over_UI.position.y
105
106     game_over_UI.position.y = - 200
107     game_over_UI.rotation_degrees = -45
108
109     var tween = create
110
111     tween.tween_proper
112         game_over_UI,
113         "position:y",
114         end_y,
115         GO_animation_t
116     ).set_trans(Tween.TRA)
```

```
.C TRANS_BACK
.C TRANS_BOUNCE
.C TRANS_CIRC
.C TRANS_CUBIC
.C TRANS_ELASTIC
.C TRANS_EXPO
.C TRANS_LINEAR
```

Tween

Tweens are used to smoothly interpolate (animate) values over time. Tweens will be used often for creating transitions in animations.



tween_property()

A method of Tween that animates an object's property from one value to another over a set time.

`tween_property()`**:** Creates and appends a PropertyTweener.

Parameters:

1. **object (`Object`):** the affected object
2. **property (`NodePath`):** the affected property
3. **final_var (`Variant`):** the final value
4. **duration (`float`):** span of time in seconds

Returns (`PropertyTweener`): Returns the PropertyTweener, which auto-generates the in-between frames (interpolates) of the objects property over time.

```
106    game_over_UI.position.y = - 200
107    game_over_UI.rotation_degrees = -45
108
109    var tween = create_tween()
110
111    tween.tween_property(
112        # object (Object),
113        # property (NodePath),
114        # final_var (Variant),
115        # duration (float)
116    )
117
```

ACTIVITY 17: FOOD FRENZY PART 3

CONST

The keyword **const** creates a constant, which is a value that cannot be changed when the game is running. Trying to assign a value to a constant after it is declared will give an error. Notice that when defining a constant, all capital letters are used with underscores separating the words.

```
1   extends Node
2
3   const SAVE_PATH: String = "res://star_data.json"
4
```

DICTIONARY

A Dictionary is a data structure that holds a collection of data as key-value pairs. Each key must be unique and have an associated value.

```
var level_stars: Dictionary = {"Level1": 1, "Level2": 2, "Level3": 3}
```

```
var level_stars: Dictionary = {"Level1": 1, "Level2": 2, "Level3": 3}

var level = level_stars["Level1"]
# level = 1
```

access a value

```
var level_stars: Dictionary = {"Level1": 1, "Level2": 2, "Level3": 3}

level_stars["Level1"] = 3
# {"Level1": 3, "Level2": 2, "Level3": 3}
```

assign a value

FILE_EXISTS()

A method used to check whether a file is present at a given path.

> **`file_exists():`** Checks if the file exists in the given path.
>
> **Parameters:**
>
> 1. **`path (String):`** the file path
>
> **`Returns (bool):`** Returns True if the file exists in the given path.

```
17
18  func load_stars() -> void:
19      print("Loading data...")
20
21      if FileAccess.file_exists(SAVE_PATH):
22          # var file = ???.???(???, ???.???)
```

OPEN()

The **`open()`** method is part of the **FileAccess** class. **`open()`** creates a new **FileAccess** object and opens the file for writing or reading.

> **`open():`** creates a new FileAccess object and opens the file for reading or writing depending on the flags.
>
> **Parameters:**
>
> 1. **`path (String):`** the file path
> 2. **`flags (ModeFlags):`** what is done with the file
>
> **`Returns (FileAccess):`** Returns the new FileAccess object or null if opening the file failed

```
7
8   func save_stars() -> void:
9       print("Saving data...")
10
11      var file = FileAccess.open(SAVE_PATH, FileAccess.WRITE)
12
13
```

STRINGIFY()

`stringify()` converts a Variant to JSON text and returns the result.

> `stringify():` Converts a Variant var to JSON text and returns the result as a String.
>
> **Parameters:**
>
> 1. `data (Variant):` data being converted
> 2. `indent (String):` optional parameter that controls how the data is indented
>
> `Returns (String)`

```
var file_data: String = JSON.stringify(level_stars, "\t")
```