# ICOLLISIONENGINE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/icollisionengine

The `ICollisionEngine` interface provides access to Construct's collision engine, such as to identify if two objects are intersecting. It is typically accessed via `runtime.collisions`.

## Collision APIs

### runtime

A reference back to the IRuntime script interface.

### testOverlap(instA, instB)

Returns a boolean that indicates if two instances, which must be derivatives of IWorldInstance, are overlapping at their current positions.

```
// Example code
if (runtime.collisions.testOverlap(instA, instB))
{
    console.log("Collision found!");
}
```

### testOverlapAny(inst, iterable)

Test if an IWorldInstance is overlapping any of the other `IWorldInstance`s provided by an iterable (which can be any kind of iterable object, such as an array). If an overlap is found, it returns the `IWorldInstance` of the first instance in `iterable` that overlaps `inst`. If no overlap is found with any of the provided instances, it returns `null`.

> *Note the return value can also be used as truthy or falsey, such as in an `if` statement.*

### testOverlapSolid(inst)

Test if an IWorldInstance is overlapping any other instance with the Solid behavior. If an overlap is found, it returns the `IWorldInstance` of the first found solid instance that overlaps `inst`. If no overlap is found it returns `null`.

*Note the return value can also be used as truthy or falsey, such as in an `if` statement.*

---

### setCollisionCellSize(width, height)

Construct optimizes collision checks by sorting all objects in to "cells". The default cell size is the viewport size. Changing the collision cell size adjusts the trade-off between collision performance, memory use, and overhead of moving objects. Usually the default works well for most projects, but projects where there are large numbers of objects testing collisions in a small area, such as "bullet hell" style games, may benefit from a smaller collision cell size. Use performance measurements to identify the optimal size. The collision cell size will also affect how many instances are returned by `getCollisionCandidates()`.

---

### getCollisionCellSize()

Returns `[width, height]` indicating the current collision cell size.

---

### getCollisionCandidates(iObjectClasses, domRect)

Efficiently retrieve only instances of the given object classes that are near the specified rectangular area in the layout. This uses Construct's "collision cells" optimization and allows substantially reducing the number of collision checks that need to be performed. For example instead of checking for collisions against all instances of a given object class, which could involve thousands of instances distributed across a large layout, this method allows efficiently retrieving only a smaller number of instances in the area which are possible candidates for collision checking. The parameters work as follows:

- `iObjectClasses` is either an IObjectClass, or an array of IObjectClass, specifying the object classes of interest, e.g. `runtime.objects.Enemy`. Only instances belonging to these object classes will be returned. Families can also be specified and all instances belonging to any of the object types in the family will be included.

- `domRect` is a DOMRect specifying a rectangular area in the layout. Instances near this area will be returned. All instances inside the specified rectangle will be returned, but some instances near to but outside the rectangle may also be returned, as the area checked is done at the resolution of the collision cell size; however instances far away from the area will not be included, which is the main purpose of this method.

The method returns an array of IWorldInstance with the instances near the specified area. Note also that due to the algorithm used, the returned array of instances **may include duplicates** - i.e. the same instance may appear more than once in the array. These can be filtered by creating a Set with the returned array, as a Set only stores unique items; however this will add a performance overhead. Eliminating duplicates may not be necessary depending on the algorithm used. For example if the aim is to identify any collision and then destroy the object, it does not matter if the same check is performed twice. However if the

aim is to add to the score for every overlap, then duplicates must be eliminated for the correct intended result.