Variables

In GDevelop, you can use *variables* to store temporary data, such as numbers and text. For example, you might use variables to store the player's name, current health, and score.

To learn how to store data that isn't temporary, such as a player's saved progress, refer to Storage.

Data types

All variables have a *data type*. The data type of a variable determines what type of data the variable can contain.

In GDevelop, variables can contain the following types of data:

- Number
- Text
- Structure
- Boolean
- Array

Number, Text and Boolean are *primitive types*: they store a value. Structure and Array are *collection types*: they store multiple variables.

Number

A variable with the *Number* data type can contain numeric values, such as `0`, `100`, and `-10`. You can perform mathematical calculations on variables that have this data type, such as multiplication or division.

Text

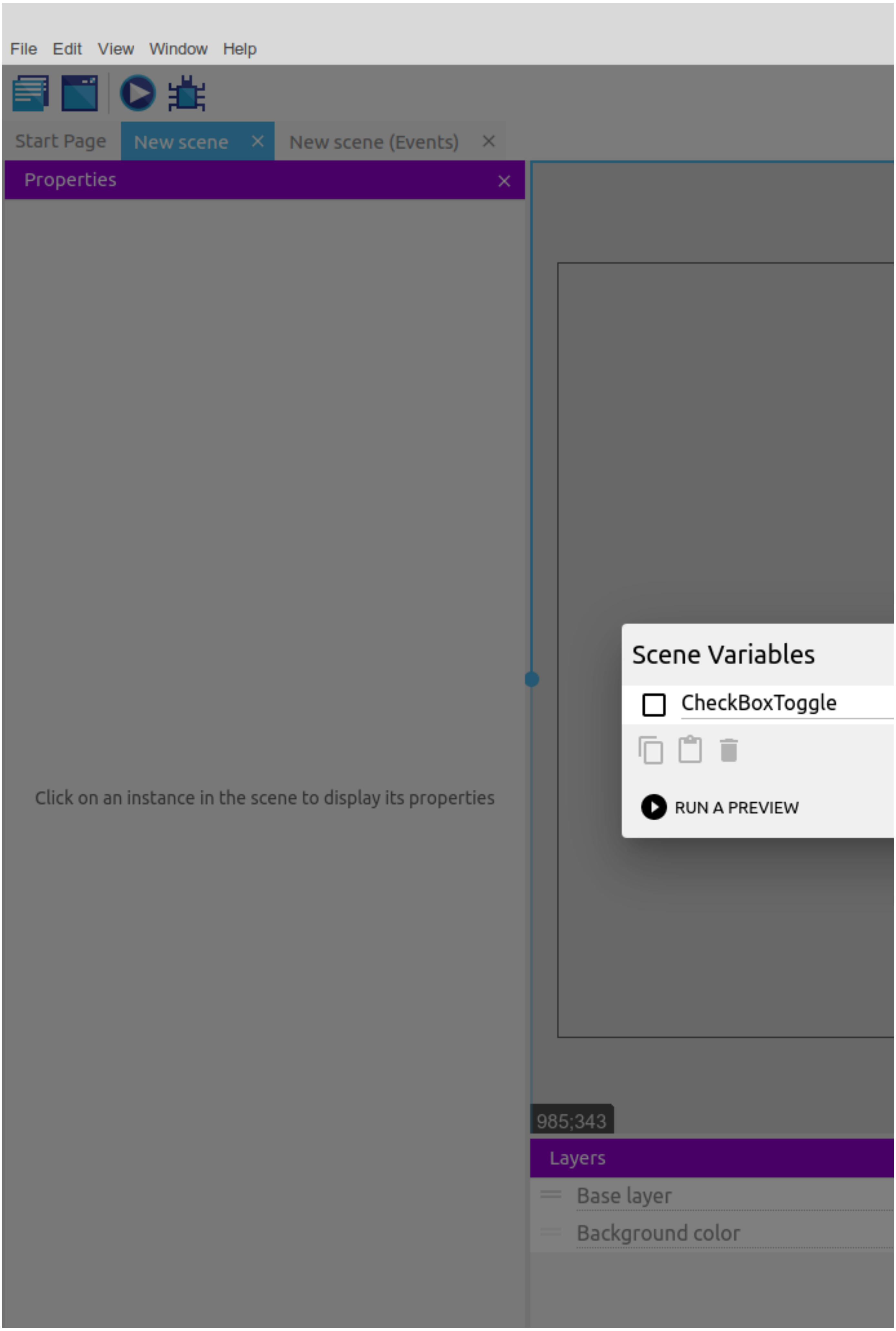
A variable with the *Text* data type can contain text, such as the words `Hello world`. In programming languages, this data type is often referred to as a *string*. In this documentation, the terms *text* and *string* are used interchangeably.

Boolean

A variable with the *Boolean* data type contains the simplest form of information: either yes or no, 1 or 0, true or false. They are useful as they can be easily toggled.

Adding a boolean

A boolean can be added by clicking the wrench **icon→Primitive types→Convert to boolean**.


File
Edit
View
Window
Help
Start Page
New scene
New scene (Events)
Properties
Click on an instance in the scene to display its properties
Scene Variables
CheckBoxToggle
RUN A PREVIEW
985;343
Layers
Base layer
Background color

Changing the value of the boolean

There are two ways you can change the state of the boolean.

Modify the boolean value

You can set the boolean value to `"true"` or `"false"`.

Toggle the boolean value

You can toggle between the two states. If the value was True, it will be changed to False and vice versa.

Checking the value of the boolean

Compare the boolean value

You can check if the boolean value is `"true"` or `"false"`.

To compare a boolean to another value, text or boolean, you can use **Compare the text of a variable**.

Structure

A Structure variable maps names to variables. For example, a simple structure can map the name “Hello” to one sub-variable and the name “World” to another sub-variable. You can use this data type to organize related variables within a single variable. In programming languages, this data type is often referred to as an *object*, *map*, *hash*, or *dictionary*.

Array

An Array variable, also sometimes called *list* in programming languages, is like a list of variables. Each variable in an Array has an index, which defines their position in the array. The indices begin at 0 and go up to however long the array is.

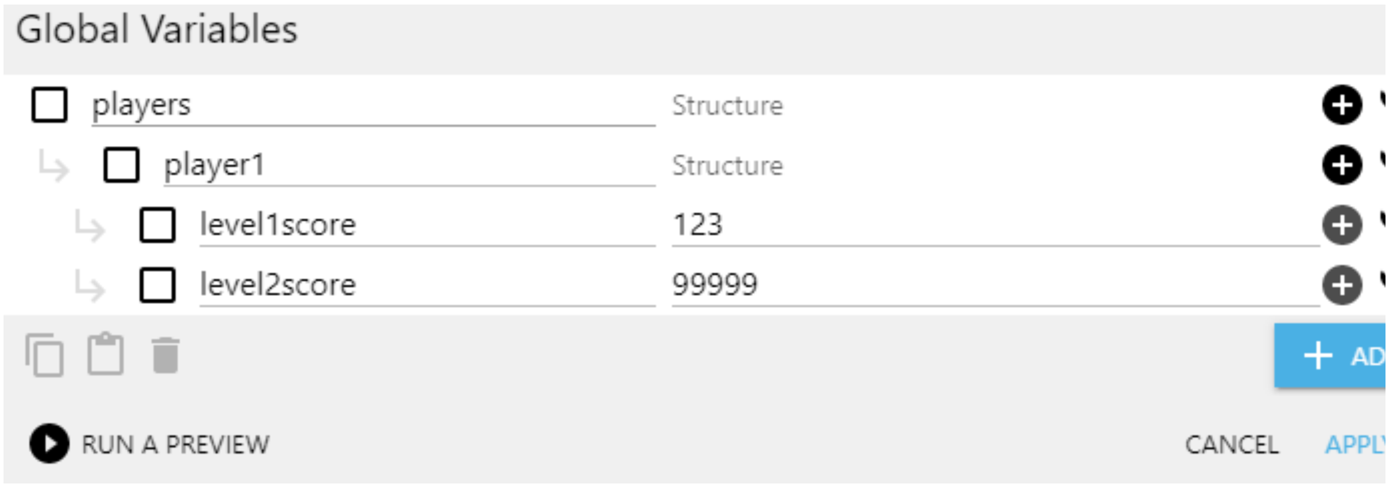
Don't try to use text as an array variable index, it will return the variable at index 0!

Accessing child variables

Variables that exist within a collection variable are known as *child variables*. To access the value of a child variable, use the following syntax in an expressions, replacing the values in angled brackets with variable names:

```
<parent_variable>.<child_variable>
```

Assume we have this structure:



players	Structure
player1	Structure
level1score	123
level2score	99999

To get the value `123` we can write the following expression

```
GlobalVariable(players.player1.level1score)
```

On structures, `<child_variable>` is the name of the child variable, and on Arrays it is the index of the child variable. Note that only numbers work as indices for Arrays.

If a child variable doesn't exist, GDevelop creates it.

Collection variables can contain other collection variables. This makes it possible to store complex data in a single variable. Just be careful the data doesn't become too difficult to manage.

Accessing child variables dynamically

You can use expressions to dynamically access child variables.

For example, imagine storing the the player's score for each level, called `Level1`, `Level2`, `Level3`. If you want to show the player's score for a specific level, you may store the current level number in a variable called `CurrentLevel`. You could then use the following syntax to access the score:

```
PlayerScore["Level"+ToString(Variable(CurrentLevel))]
```

Whatever is inside the square brackets will be interpreted as the name of the child.

If you need to use a variable to define part of the child path, all the subsequent children in the path will need to be in square brackets as well. In the above example if you wanted to address a child at PlayerScore.Level1.enemies.killbonus but still define the level dynamically, it would look like this:

```
PlayerScore["Level"+ToString(Variable(CurrentLevel))]["enemies"]["killbonus"]
```

Scopes

The *scope* of a variable determines:

- where the variable can be accessed from
- how long the variable is stored in memory
- the steps required to create the variable

In GDevelop, there are three variable scopes:

- Global
- Scene
- Object

Refer to the linked pages for more information about each variable scope.

Naming variables

Variable names should *not* contain dots (periods) or commas and begin with a letter. We recommend using alphanumerical characters (A-Z, 0-9).

Using variables without declaring them

You don't have to create (declare) variables before using them.

For example, if you reference a variable that doesn't exist in an action or condition, GDevelop automatically initializes the variable with a default value. The default value is determined by the data type of the variable:

- Numeric variables are initialized with a value of `0`.
- Text variables are initialized with a value of `""` (an empty string).

But while it's possible use variables without first creating them, we recommend creating them anyway, as it allows GDevelop to generate optimized events and helps you keep track of the variables in your game.

Debugging variables

When developing a game, bugs can occur because the value of a variable isn't what you expect it to be. If something in your game isn't working and you think the problem might relate to a variable, use GDevelop's debugger to figure out what's wrong.

For more information, refer to Game Debugger and Profiler.