



CodingPark



User Manual

Welcome onboard!

Coding Park is a fun platform that helps kids understand computer thinking through video games.

Designed for both students and teachers, the platform allows students to acquire the skills required in the game over time.

This document serves as a guide for parents, teachers, and trainers, wishing to introduce children to programming.

The guide specifically addresses the game Golden Quest, a treasure hunt in a world of robots and pirates. Golden Quest is the first game available on the Coding Park platform.



Instructions for parents, teachers, and pirates !

GoldenQuest is a treasure hunt that introduces children to the basics of computer programming.

We will acquire the following skills:

- Write sequences of instructions,
- Manipulate loops (repeat, while),
- Define and call procedures,
- Declare and use variables,
- Formulate conditions,
- ... and other more advanced concepts.

Game Rules

Objective

This is the story of Cody, a pirate robot in the quest for golden treasures. We will follow Cody from one island to another and we will be an active partner during the adventure.



Characters

The heroes



Cody

A pirate robot that sails from island to island in search of treasures. He uses his shovel to deter treasures and fight enemies.



Luna

A smart mechanic that helps Cody by providing tips and tricks for solving the challenges.



Chipset

A robotic parrot that always follows Cody. Chipset has the information of the treasure maps.

The enemies



Skeleton

A bandit who attacks Cody when he is close to him. He is often motionless, but he follows Cody with his eyes. He uses the sword planted in his head to attack.

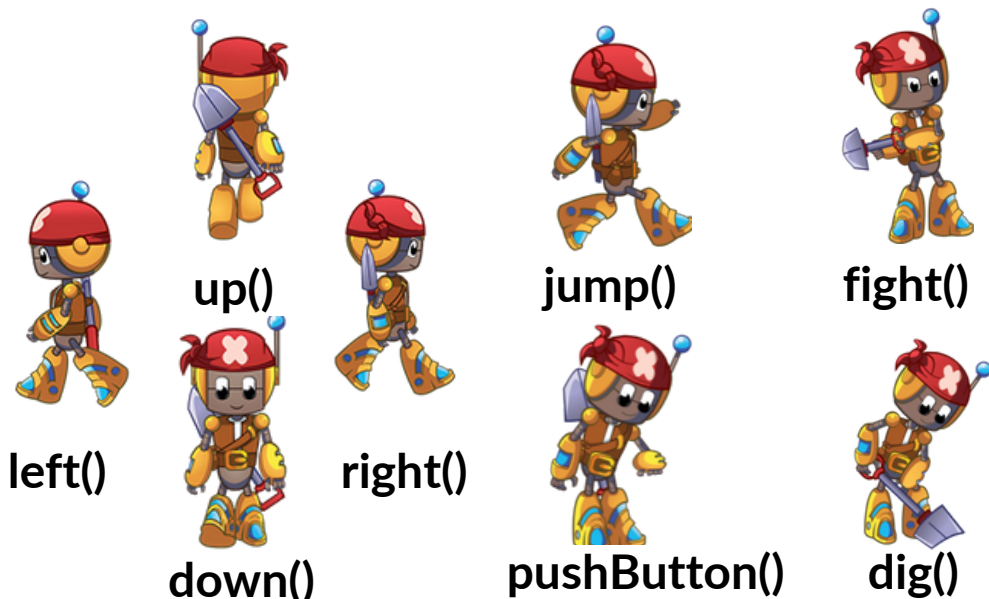
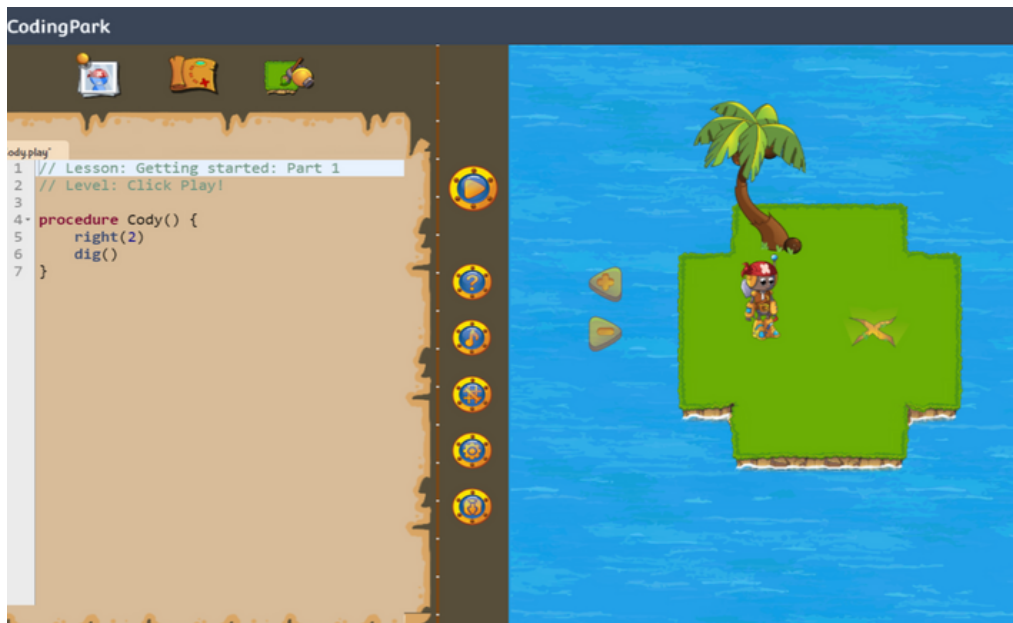


Voltar

Half-human and half-robot, Voltar tries also to get the treasures from the islands. Cody needs to be faster than Voltar to get the treasure in some levels.

Basic actions

We need to help Cody by writing the needed sequence of actions to get the treasure.
The code editor is on the left side. The Play button in the middle will make Cody start the introduced instructions.



Walking

For example, `right(2)` means to take two steps to the right. The steps can be easily measured through the grid shown in the island.

`right(2)` in the following island will make Cody to arrive to the treasure position.



`right()` does not mean to make Cody **turn** right, nor `left()` to turn left. Go right is from the view of the person who plays.

The level will be restarted if

- Cody falls into the water
- Cody walk close to an skeleton without fighting



Cody will not advance if there are blocking objects such as bushes, palm trees or barrels.



Fight

To destroy an enemy, Cody should be one step close to the enemy and then `fight()`. That means that Cody should not try to go to the exact position of an enemy but one step less and `fight()`.



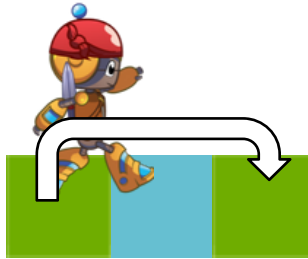
The fight direction is not important. For example, Cody can go `up()` and then `fight()` an enemy which is on its right.

Cody can only beat one enemy at a time. If there are several enemies around the position of Cody, Cody will beat one but the other one will attack and the level will be restarted.



Jump

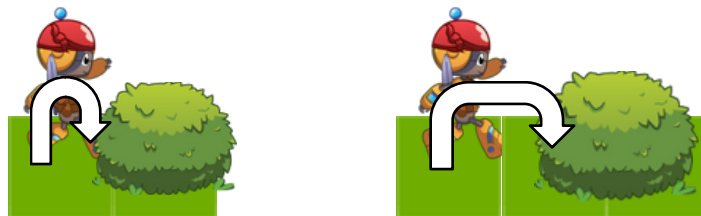
Jumping is specially useful when the next step is water. This way, Cody avoid to fall in the water. In general, when Cody jumps, it advances two steps.



Cody needs to take a little run to jump in a direction. Otherwise, it will jump vertically and fall in the same position.



Blocking objects (palm trees or bushes) cannot be jumped.

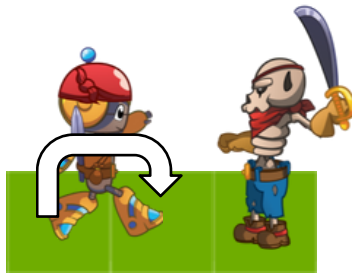


Cody will lose the little run if blocked.

Jump

Cody cannot attack during a jump. If Cody `jump()` and then `fight()`, the `fight()` will happen once Cody is on the ground.

Skeletons will attack while Cody is jumping if Cody is close to them. The level will be restarted.



Teleporters

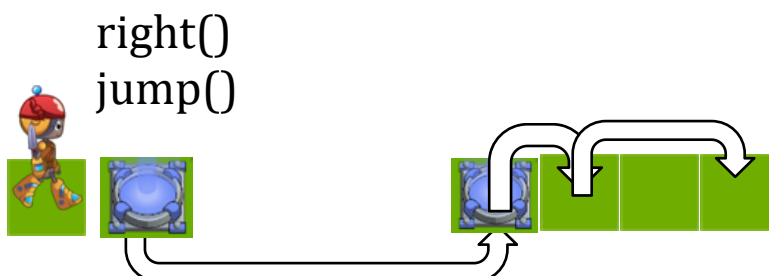
Teleporters come in pairs which are distinguishable by the color. If Cody is in the teleporter position (walking or while jumping) it will appear in the counterpart teleporter and it will be dropped one step further of its previous direction.



A teleporter pair is activated when it has light, otherwise, it is deactivated. A button in the island should be pushed to activate or deactivate them.

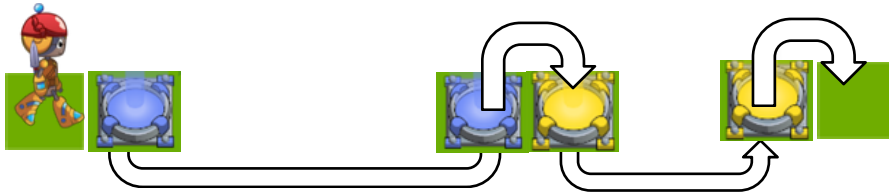


If Cody jumps in the position of a teleporter, the jump will happen after it landed at the other side. The little run is kept in the current direction.

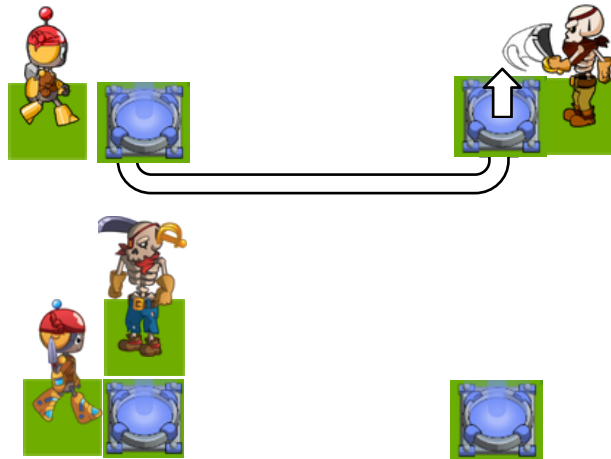


Teleporters

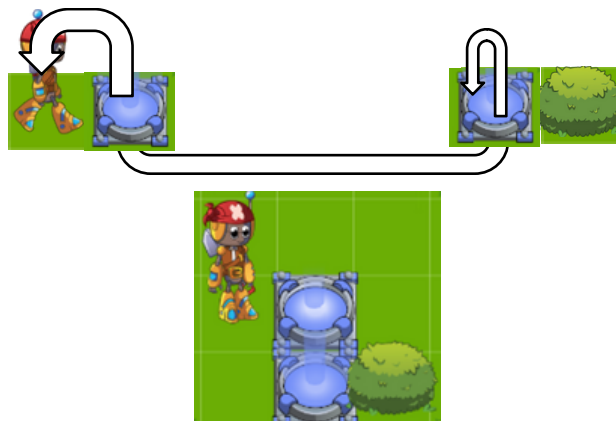
Teleporter pairs can be chained.



Skeletons will attack if Cody appear in one teleporter.



If there is a blocking element in the landing place, Cody will come back inside the teleporter in the opposite direction.

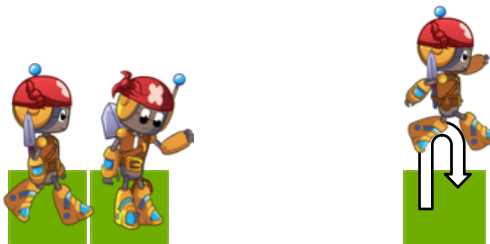


Push Button

Buttons serve to activate or deactivate a set of mechanisms associated with the color of the button. Calling the function `pushButton()` outside of a button position has no side effect.



In order to push a teleporter button, Cody needs to stop, and Cody will lose the little run for jumping.



Time limit

Some levels can have time limit to solve them. However, the level can be played again and the inserted code will be kept.

The time progress is shown in the form of Voltar progressing to a treasure.



Once Voltar reaches the treasure before Cody, a screen is displayed, and the timer can be reset.



Play language

Procedures

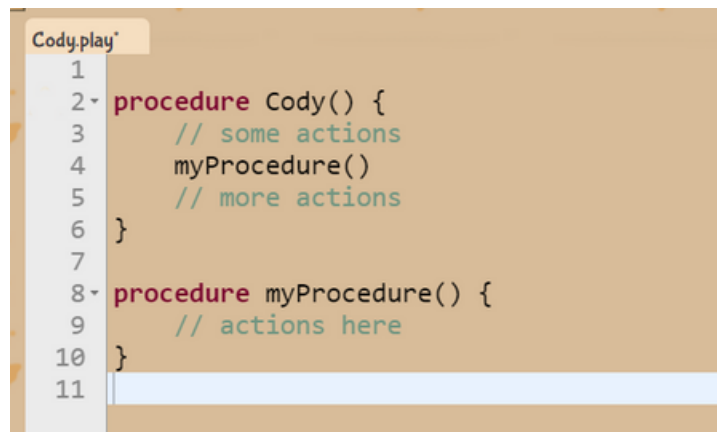
The play language (the language that Cody understands) is a case-sensitive procedural programming language.

The main procedure, the entry point to start the actions, is always:



```
Cody.play*
1
2
3- procedure Cody() {
4    // actions here
5 }
```

In the same way, we can define other procedures to be reused by calling them as many times as we want.



```
Cody.play*
1
2- procedure Cody() {
3    // some actions
4    myProcedure()
5    // more actions
6 }
7
8- procedure myProcedure() {
9    // actions here
10 }
11
```

We can use any name for the procedure (no whitespaces nor special characters) and names cannot be repeated if they take the same number of parameters.

Variables

The play language only allows numeric variables. For example, in the following code, we declare the variable `x` and we assign it the value of 2. This way, we can use `x` in the actions. In the example, to take 2 steps to the right.

```
Cody.play*
1
2 procedure Cody() {
3   var x = 2
4   right(x)
5 }
6
7
```

The value of `x` is 2 at the beginning of the program. By adding `x = x + 1` we are saying that the new value of `x` is the previous value of `x` (it was 2) plus 1. That means that in the following code, Cody will go right 2 steps and then 3 steps up.

```
Cody.play*
1
2 procedure Cody() {
3   var x = 2
4   right(x)
5   x = x + 1
6   up(x)
7 }
8
```

Parameters

The basic actions already have parameters. For example, in the following action 2 is the parameter that we use in this call to the right action.

```
Cody.play*  
1  
2 procedure Cody() {  
3   right(2)
```

We can declare our own procedures with parameters. This procedure will make Cody to go x steps to the right and x up.

```
Cody.play*  
1 procedure myProcedure(var x) {  
2   right(x)  
3   up(x)  
4 }
```

This way we can call this procedure with different values. In the following code we will make Cody to go 3 steps right and 3 up, and then 5 steps right and 5 up.

```
Cody.play*  
1 procedure Cody() {  
2   myProcedure(3)  
3   myProcedure(5)  
4 }
```

Parameters

Finally, we can also declare a procedure with as much input parameters as we want.

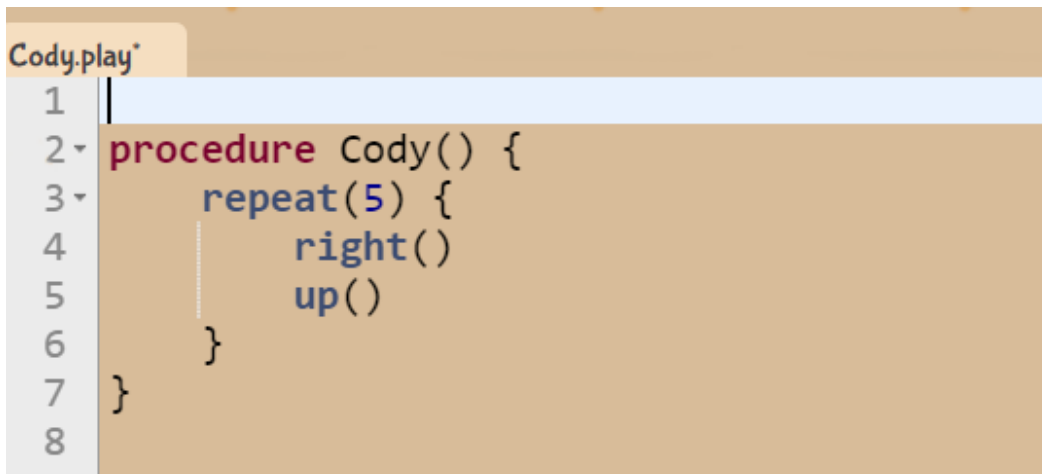
```
Cody.play*  
1 procedure other(var a, var b, var c) {  
2     // some actions  
3 }  
4
```

Loops: Repeat

The play language allows to write loops for repeating a sequence of actions. The language has two kinds of loops: repeats and whiles.

The following code will make to repeat 5 times the sequence of going right and then up.

The 5 can be replaced with other number or with a variable. If it was a variable and you change its value inside the repeat, it will have no effect in the number of times that it will be repeated. The number of times to repeat will be the first value it got when calling the repeat.



```
Cody.play*
1 |
2 | procedure Cody() {
3 |   repeat(5) {
4 |     right()
5 |     up()
6 |   }
7 | }
8 |
```

Loops: While

Compared to the repeat loop, when using a while loop, we have more control for when to finish the loop. A condition (a boolean true or false value) will be checked after the end of each repetition to decide if the loop will continue or not.

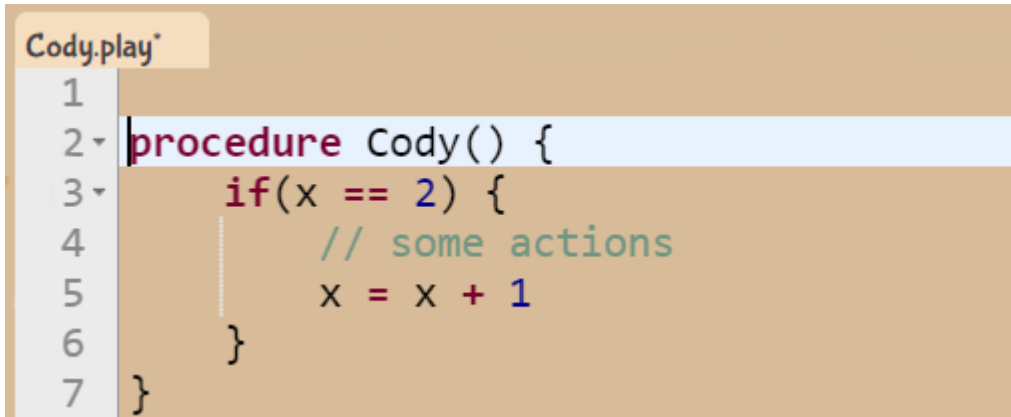
The following code shows a variable declaration and then a while where the variable is incremented by one in each iteration. The loop will repeat the sequence of actions 5 times because when x will get the value of 6, the condition will be false.

```
1  
2- procedure Cody() {  
3     var x = 1  
4-     while(x <= 5) {  
5         // some actions  
6         x = x + 1  
7     }  
8 }  
9
```

The value of x, can be used inside the actions of the loop. In the previous example, if we have right(x) inside the loop, the first iteration will go right one step, then 2 in the 2nd iteration, 3 in the 3rd and so on.

Conditionals

Conditions allow to execute one sequence of actions or not.



```
Cody.play*
1
2- procedure Cody() {
3-   if(x == 2) {
4-       // some actions
5-       x = x + 1
6-   }
7 }
```

The value of x , can be used inside the actions of the loop. In the previous example, if we have `right(x)` inside the loop, the first iteration will go right one step, then 2 in the 2nd iteration, 3 in the 3rd and so on.

Variables scope

The scope of a variable defines when we can make use of a variable. To use a variable, it should be accessible.

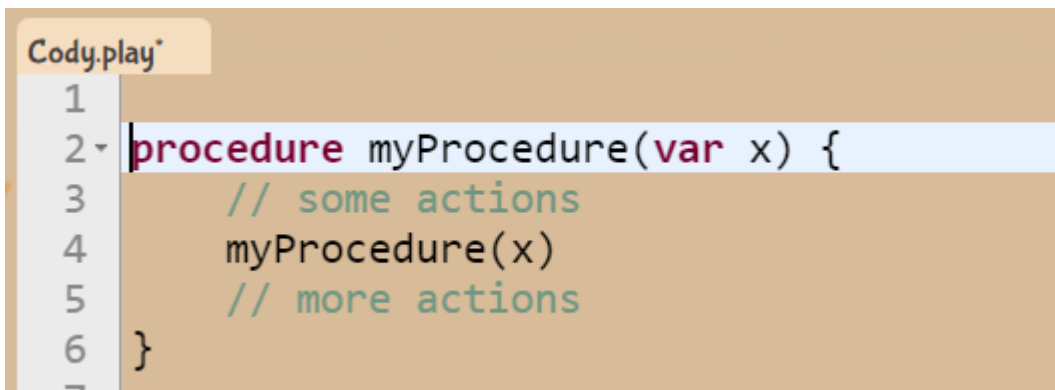
The Play language has 3 variable scopes:

- Global variables : variables accessible from anywhere,
- Procedure variables : variables accessible inside a procedure,
- Block variables : variables accessible inside a loop or a conditional expression (if)

Recursion

Recursion is a powerful concept in computer programming. When you define a procedure where, inside of itself, you call the same procedure, we have a case of recursion.

Here is a typical recursion example.



```
Cody.play
1
2 procedure myProcedure(var x) {
3     // some actions
4     myProcedure(x)
5     // more actions
6 }
7
```

To avoid infinite recursion we should include an exit condition that will make the procedure not to call itself again.

Operators

Numerical operators

The following math operators can be applied to numerical values and variables.

- + Addition
- Subtraction or negative number
- * Multiplication
- / Division
- % Remainder, modulo

Boolean operators

The following Boolean operators can be applied in conditionals (ifs, or whiles)

Between numerical values:

- == Equal
- != Not equal
- < Less than
- <= Less than or equal
- >= Greater or equal
- > Greater

Operators for booleans:

- and** the and operator
- or** the or operator
- not** negation of a Boolean value

Procedure overloading

We can declare the same procedure name with different number of parameters. For example, these three procedures can coexist in the same program.

```
4 ▾ procedure Cody() {  
5  
6     myProcedure(8)  
7     myProcedure(3, 4)  
8     myProcedure(3, 4, 8)  
9 }  
10  
11 ▾ procedure myProcedure(var x) {  
12     //do something.  
13 }  
14  
15 ▾ procedure myProcedure(var x, var y) {  
16     //do something.  
17 }  
18  
19 ▾ procedure myProcedure(var x, var y, var z) {  
20     //do something.  
21 }
```

Then, in Cody actions, if we call myProcedure(8) it will be the first one that will be executed as it is the one that expected one parameter. Then, myProcedure(3,4) will call the second procedure, and finally myProcedure(3,4,8) will call the third procedure as it expects 3 parameters.

The Big Picture



The Level Editor

Editing a level

The level editor can be opened from any level, or from the main screen. It is symbolized by the following icon.



We have a palette of game elements to add, move, remove or configure.

It is possible to configure the global theme of the challenge: grass, snow, or lava.

If the element can rotate (e.g. palm tree, bridge), we should click the game element with this element selected in the palette.

We can simultaneously edit the level and play the level.

Languages:
Quick
reference guide



up()
haut

left()
gauche



right()
droite

down()
bas

procedure()
procedure



jump()
sauter

repeat()
répéter



fight()
se battre

while()
tant que

if()
si



dig()
creuser

var
variable



up()
arriba

left()
izquierda



right()
derecha

down()
abajo

procedure()
procedimiento



jump()
saltar

repeat()
repetir



fight()
pelear

while()
mientras que

if()
si



dig()
excavar

var
variable