



# CodingPark



# Manuel Utilisateur

# Bienvenue à bord !

Coding Park est une plateforme ludique qui aide les enfants à comprendre la pensée informatique par le jeu vidéo.

Conçue à la fois pour les élèves et les enseignants, la plateforme permet aux élèves d'acquérir les compétences requises progressivement dans le jeu.

Ce document sert de guide aux parents, aux enseignants, et aux formateurs, souhaitant initier des enfants à la programmation.

Le guide adresse spécifiquement le jeu Golden Quest, une chasse aux trésors dans un univers de robots et de pirates. Golden Quest est le premier jeu disponible sur la plateforme Coding Park.



# Instructions pour les parents, les enseignants, et les pirates!

GoldenQuest est un jeu de chasse aux trésors qui permet d'initier les enfants aux fondements de la programmation informatique (codage).

Nous allons acquérir les notions suivantes :

- Ecrire des séquences d'instructions,
- Manipuler les boucles (repeat, while),
- Définir et appeler des procédures,
- Déclarer et utiliser des variables,
- Formuler des conditions,
- ... Et d'autres concepts plus avancés.

# Règles du Jeu

# Objectif

Voici l'histoire de Cody, un robot pirate à la recherche de trésors bien cachés. Nous suivrons Cody d'une île à l'autre et nous serons un partenaire actif tout au long de l'aventure.



# Personnages

# Les héros



## Cody

Un robot pirate qui navigue d'île en île à la recherche de trésors. Il utilise sa pelle à la fois pour déterrer les trésors et combattre les ennemis.



## Luna

Une mécano surdouée qui aide Cody en fournissant des conseils et astuces pour résoudre les challenges.



## Chipset

Un perroquet robotique qui suit Cody. Chipset détient les informations sur les cartes aux trésors.

# Les ennemies



## **Skeleton**

Un squelette qui attaque Cody quand il est proche de lui. Il est souvent immobile, mais il suit Cody des yeux. Il utilise l'épée plantée dans sa tête pour attaquer.



## **Voltar**

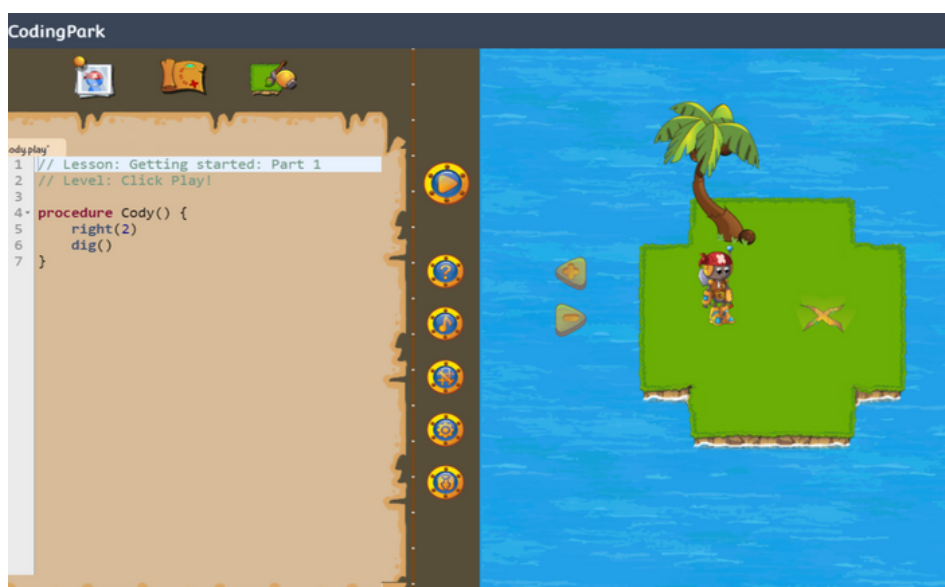
Mi-humain, mi-robot, Voltar essaie également de récupérer les trésors des îles à des fins maléfiques. Cody doit être plus rapide que Voltar pour obtenir le trésor à certains niveaux.



# Actions de base

Nous devons aider Cody en écrivant la séquence d'actions nécessaire pour atteindre le trésor.

L'éditeur de code est situé à gauche. Le bouton Lecture, situé au milieu, fera que Cody exécute les instructions écrites à gauche.



# Se déplacer

Par exemple, `right(2)` signifie faire deux pas vers la droite. Les étapes peuvent être facilement mesurées à travers la grille indiquée dans l'île. En écrivant `right(2)` dans l'île suivante, Cody arrivera à la position de trésor.



`right()` ne veut pas dire que Cody tourne à droite, idem pour `left()`. Le déplacement est du point de vue de la personne qui joue.

Le niveau redémarre si

- Cody tombe dans l'eau
- Cody est près d'un squelette sans se battre



Cody n'avancera pas s'il y a des objets bloquants tels que des buissons, des palmiers ou des tonneaux.



# Se battre

Pour vaincre un ennemi, Cody doit être proche de celui-ci et faire `fight()`. Cela signifie que Cody ne devrait pas essayer d'aller à la position exacte de l'ennemi, mais il doit être sur une case voisine de celui-ci.



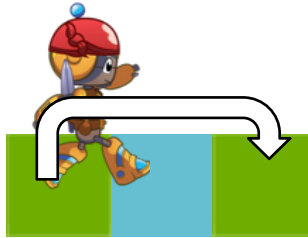
La direction n'est pas importante. Par exemple, Cody peut faire `up()` puis `fight()` sur un ennemi qui se trouve à sa droite.

Cody ne peut vaincre qu'un seul ennemi à la fois. S'il y a plusieurs ennemis autour de la position de Cody, Cody en battra un, mais l'autre l'attaquera et le niveau recommencera.



# Sauter

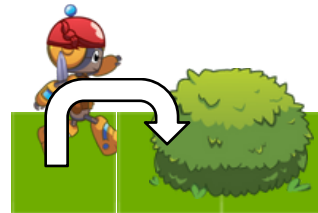
Le saut est particulièrement utile lorsque la prochaine case est l'eau. De cette façon, Cody évite de tomber dans l'eau. En général, lorsque Cody saute, il avance de deux cases.



Cody a besoin de prendre de l'élan pour sauter dans une direction donnée, sinon il sautera verticalement et tombera dans la même position.



Cody ne peut pas sauter lorsque la case suivante est un obstacle (un buisson, un palmier, un tonneau).

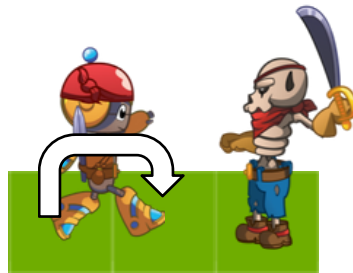


Cody perdra l'élan si la case suivante est un buisson ou un tonneau.

# Sauter

Cody ne peut pas attaquer pendant un saut. Si Cody fait `jump()` puis `fight()`, le combat aura lieu une fois que Cody sera au sol.

Les squelettes attaqueront pendant que Cody saute si Cody est proche d'eux. Le challenge sera alors redémarré.



# Se téléporter

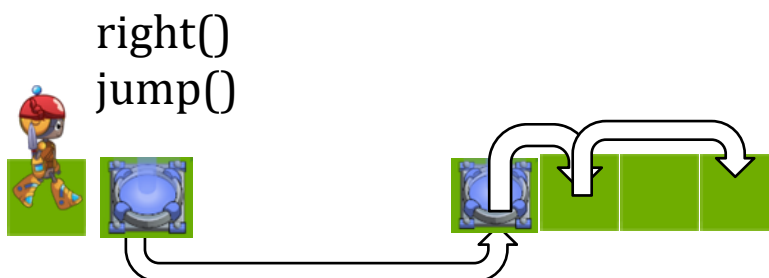
Les téléporteurs viennent en paires identifiées par leur couleur. Si Cody est sur un téléporteur (en train de marcher ou de sauter), il apparaîtra dans le téléporteur homologue et il sera jeté un peu plus loin de sa direction précédente.



Une paire de téléporteurs est activée ou désactivée par défaut. Il faut appuyer sur le bouton associé pour les activer ou les désactiver.

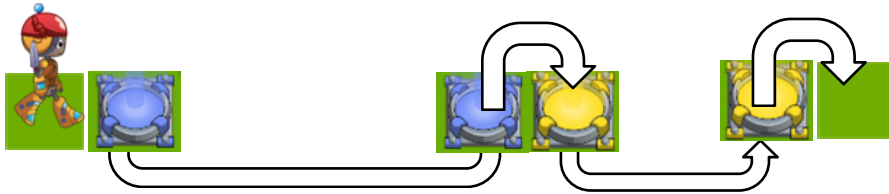


Si Cody saute sur un téléporteur, le saut aura lieu après son atterrissage de l'autre côté. L'élan est maintenu dans la direction initiale.

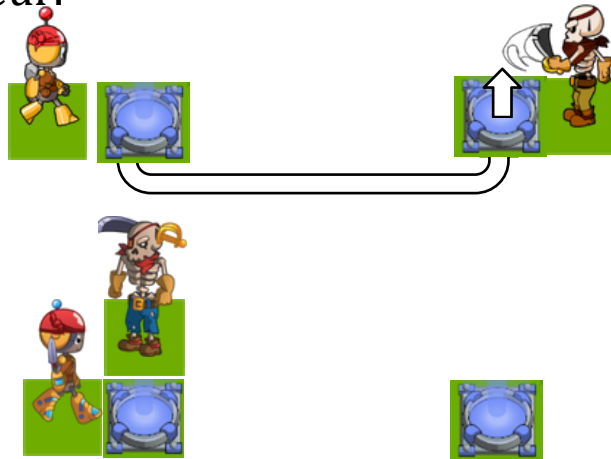


# Se téléporter

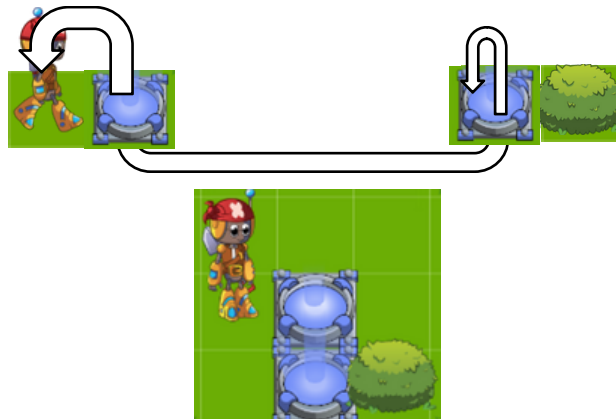
Les téléporteurs peuvent être chaînés.



Les squelettes attaqueront si Cody apparaît dans un téléporteur.



S'il y a un élément bloquant sur le lieu d'atterrissage, Cody reviendra à l'intérieur du téléporteur dans la direction opposée.



# Activer des téléporteurs

Les boutons permettent d'activer ou de désactiver les téléporteurs associés à la couleur du bouton. L'appel de la fonction `pushButton()` en dehors d'une position de bouton n'a aucun effet.



Pour appuyer sur un bouton de téléporteur, Cody doit s'arrêter et il perdra nécessairement son élan.





# Contre la montre

Certains challenges se jouent contre la montre, symbolisée par Voltar. Cependant, le challenge peut être rejoué et le code inséré est conservé.

Le progrès temporel est représenté sous la forme de Voltar s'approchant du trésor.



Une fois que Voltar a atteint le trésor avant Cody, un écran s'affiche et le chronomètre peut être réinitialisé.



# Le langage Play

# Procédures

Le langage Play (le langage que Cody comprend) est un pseudo langage de programmation procédural sensible à la casse.

La procédure principale, le point d'entrée pour démarrer les actions, est toujours:

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

De la même manière, nous pouvons définir d'autres procédures à réutiliser en les appelant autant de fois que nous le voulons.

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

Nous pouvons utiliser n'importe quel nom pour la procédure (pas d'espaces ni de caractères spéciaux) et les noms ne peuvent pas être identiques s'ils prennent le même nombre de paramètres.

# Variables

Play n'autorise que les variables numériques. Par exemple, dans le code suivant, nous déclarons la variable `x` et lui attribuons la valeur 2. De cette manière, nous pouvons utiliser `x` dans les actions. Dans l'exemple suivant, Cody fait 2 pas à droite.

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

La r  
exemple, en ajoutant `x = x + 1`, nous disons que la nouvelle valeur de `x` est la valeur précédente de `x` (elle était égale à 2) à laquelle on rajoute 1. Cela signifie que Cody effectuera 2 pas, puis 3 pas.

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

# Paramètres

Les actions de base ont déjà des paramètres. Par exemple, dans l'action suivante, 2 est le paramètre utilisé dans l'appel de la fonction `right()`.

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

Nous pouvons déclarer nos propres procédures avec leurs paramètres. Cette procédure fera que Cody se déplace de x pas à droite et x pas en haut.

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

De cette façon, nous pouvons appeler la procédure avec des valeurs différentes. Dans le code suivant, nous ferons en sorte que Cody aille 3 pas à droite et 3 pas en haut, puis 5 pas à droite et 5 en haut.

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

# Paramètres

Enfin, nous pouvons également déclarer une procédure avec autant de paramètres d'entrée que nous le souhaitons.

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

# Boucles: Repeat

Le langage Play permet d'écrire des boucles pour répéter une séquence d'actions. Il existe deux types de boucles: les boucles « repeat » et les boucles « while ». La première est utilisée lorsqu'on connaît le nombre de répétitions à priori. La deuxième est utilisée lorsque ce nombre n'est pas connu, mais on connaît une condition de sortie (exit) de la boucle.

Par exemple, le code suivant fera répéter 5 fois la séquence (i) aller droite puis (ii) aller en haut.

Le 5 peut être remplacé par un autre nombre ou par une variable. S'il s'agissait d'une variable et que vous modifiez sa valeur dans la répétition, cela n'aura aucun effet sur le nombre de répétitions. Le nombre de répétitions sera la première valeur obtenue lors de l'appel de la répétition.

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

# Boucles: While

Lors de l'utilisation d'une boucle « while », nous avons plus de contrôle sur le moment de terminaison la boucle. Une condition (une valeur booléenne vraie ou fausse) sera vérifiée après la fin de chaque itération pour décider si la boucle continuera ou non.

Le code suivant montre une déclaration de variable, puis la variable est incrémentée de 1 à chaque itération. La boucle répétera la séquence d'actions 5 fois, et lorsque x vaut 6, la condition sera fausse et la répétition se termine.

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

La valeur de x peut être utilisée dans les actions de la boucle. Dans l'exemple précédent, si nous avons right (x) à l'intérieur de la boucle, la première itération ira à droite d'un pas, puis 2 à la 2ème itération, 3 à la 3ème et ainsi de suite.



# Conditions

Les conditions permettent de décider si on exécute une séquence d'actions ou non.

Dans l'exemple suivant, la valeur de x peut être utilisée dans les actions de la boucle.

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

Dans l'exemple précédent (Conditions), si nous avons écrit `right(x)` à l'intérieur de la boucle « while », à la première itération Cody se déplacera à droite d'un pas, puis il se déplacera de 2 pas à la 2ème itération, puis 3 pas à la 3ème itération, et ainsi de suite.

# Portée des variables

La portée d'une variable définit quand on peut utiliser cette variable. Pour utiliser une variable, il faut qu'elle soit accessible.

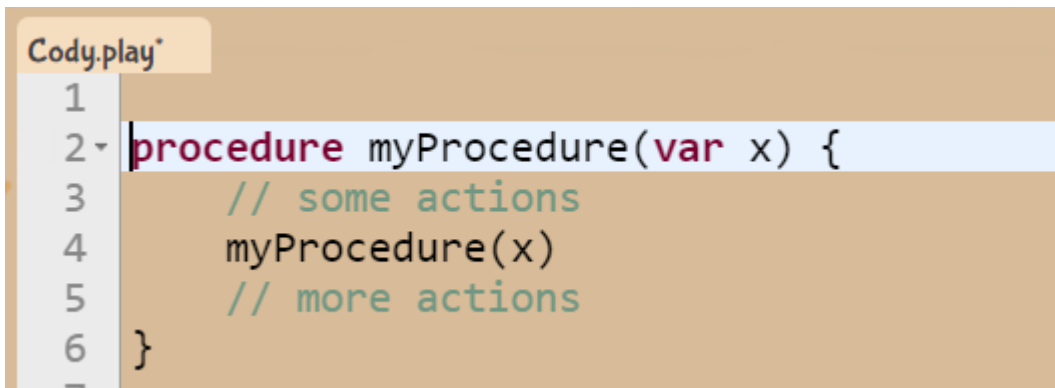
Le langage de jeu définit 3 portées de variables:

- Variables globales: variables accessibles depuis n'importe quel endroit du code,
- Variables de procédure: variables accessibles uniquement à l'intérieur d'une procédure,
- Variables de bloc: variables accessibles à l'intérieur d'une boucle ou d'une expression conditionnelle (if).

# Récurtivité

La récursivité est un concept avancé de l'informatique. Lorsque vous définissez une procédure dans laquelle vous appelez la même procédure, nous avons un cas de récursivité (la procédure s'appelle elle-même).

Voici un exemple typique de récursivité.

A screenshot of a code editor with a light brown background. The editor shows a file named 'Cody.play' at the top left. On the left side, there is a vertical line of numbers from 1 to 7. The code is as follows:

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

Pour éviter le cas d'une récursivité infinie, nous devons inclure une condition de sortie (exit) qui empêchera la procédure de s'appeler à nouveau.

# Opérateurs

## Opérateurs numériques

Les opérateurs mathématiques suivants peuvent être appliqués aux valeurs numériques et aux variables.

+ Addition

- Soustraction ou nombre négatif

\* Multiplication

/ Division

% Le reste d'une division, ou modulo

## Opérateurs booléens

Les opérateurs booléens suivants peuvent être appliqués aux valeurs booléennes dans des conditions (if) ou les boucles « while ».

*Entre valeurs numériques:*

== égal

!= Pas égal

< Inférieur à

<= Inférieur ou égal

> = Supérieur ou égal

> Strictement supérieur

*Entre valeurs booléennes:*

**and** conjonction « et » entre valeurs booléennes

**or** conjonction « ou » entre valeurs booléennes

**not** négation d'une valeur booléenne

# Surcharge de procédures

Nous pouvons déclarer le même nom de procédure avec un nombre différent de paramètres. Par exemple, ces trois procédures peuvent coexister dans le même programme.

```
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 }
```

Ensuite, dans les actions de Cody, si nous appelons myProcedure(8), ce sera la première procédure qui sera exécutée car c'est celle qui attend un seul paramètre. L'appel myProcedure(3,4) appellera la deuxième procédure, et enfin, l'appel myProcedure(3,4,8) appellera la troisième procédure.

# La photo de famille



L'éditeur de  
challenges

# Editer un challenge

L'éditeur de challenges peut être ouvert à partir de n'importe quel niveau ou de l'écran principal. Il est symbolisé par l'icone suivante.



Nous avons une palette d'éléments de jeu à ajouter, déplacer, supprimer, ou configurer.

Il est possible de configurer le thème global du challenge : herbe, neige, ou lave.

Si l'élément peut pivoter (exemple: un palmier, un ponton), vous devez cliquer sur l'élément du jeu avec cet élément sélectionné dans la palette.

Il est possible d'éditer le challenge et le jouer simultanément.



# Traduction des actions



**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*