GAnalyse à priori des situations

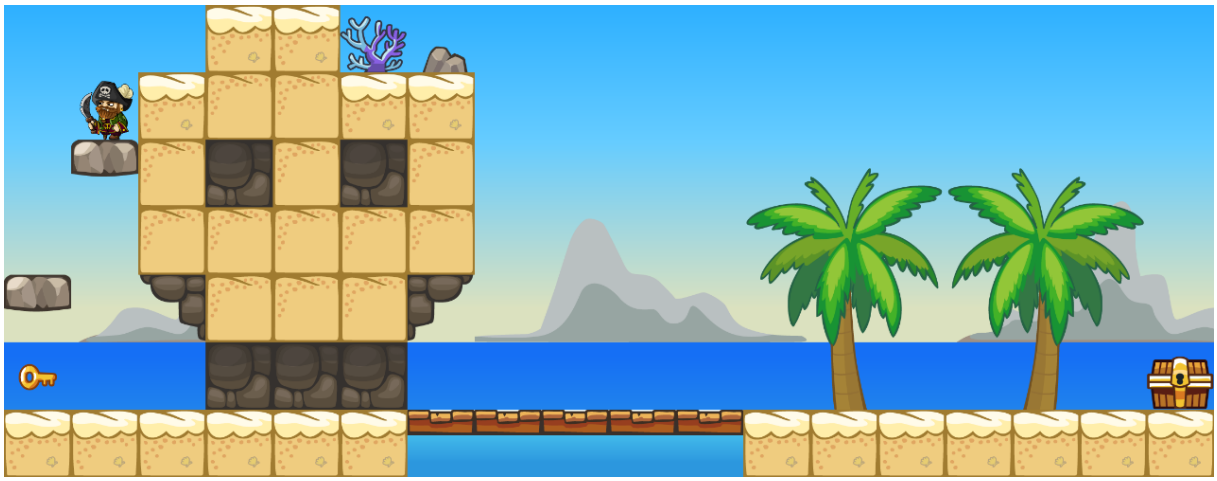
G.1 Niveau 1

G.1.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre.

Fonctions de contrôle : avancer(), droite(), gauche(), ouvrir()

Contraintes : Le programme ne doit pas dépasser 10 lignes.



G.1.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, droite, gauche, ouvrir)
Utiliser une boucle for sans variable de boucle (une seule instruction dans le corps de la boucle)

G.1.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Action à répéter 16 fois : avancer vers le coffre	Favorise l'utilisation d'une boucle for
Taille du programme limitée à 10 lignes	Rend nécessaire l'utilisation d'une boucle for

G.1.4 Stratégies gagnantes

Description	Code

[Stratégie visée] S1.1 (for-re) : Ramasser la clé en enchainant séquentiellement les instructions de déplacement. Parcourir la ligne droite de 16 blocs à l'aide d'une boucle for jusqu'au coffre. Ouvrir le coffre. Concepts : FOR-RE	1 avancer() 2 droite() 3 avancer() 4 gauche() 5 avancer() 6 droite() 7 for _ in range(16): 8 avancer() 9 ouvrir()
S1.1 (for-re_variante) : Il est possible d'avancer une fois de plus avant d'ouvrir le coffre. Concepts : FOR-RE	1 avancer() 2 droite() 3 avancer() 4 gauche() 5 avancer() 6 droite() 7 for _ in range(17): 8 avancer() 9 ouvrir()
S1.1 (for-re_variante) : Utiliser deux boucles for pendant le parcours Concepts : FOR-RE, FOR-RE	1 for _ in range(2): 2 avancer() 3 droite() 4 avancer() 5 gauche() 6 droite() 7 for _ in range(15): 8 avancer() 9 ouvrir()

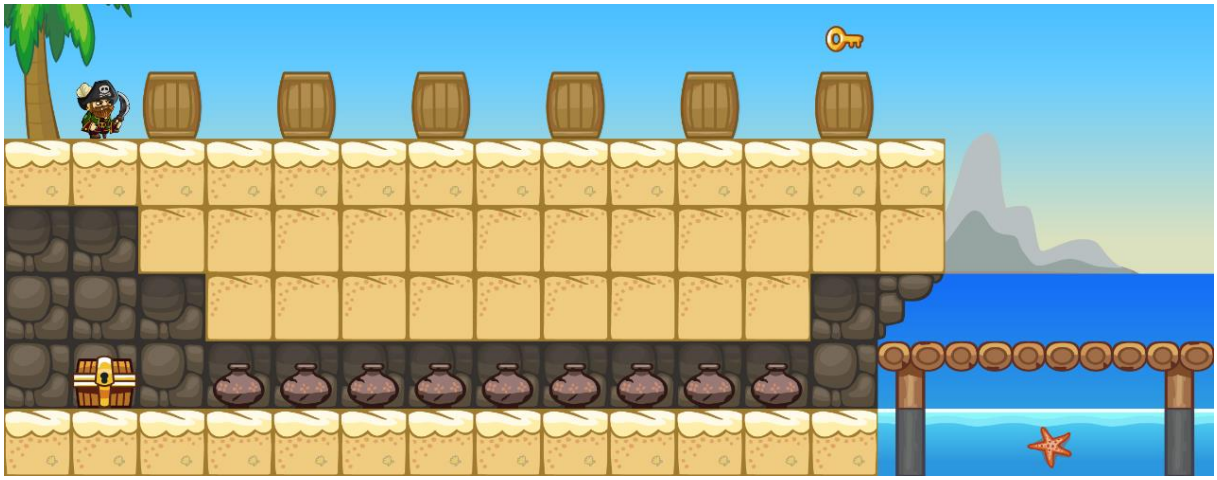
G.2 Niveau 2

G.2.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre.

Fonctions de contrôle : avancer(), droite(), gauche(), sauter(), coup(), ouvrir()

Contraintes : Le programme ne doit pas dépasser 14 lignes.



G.2.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, droite, gauche, sauter, coup, ouvrir)
Utiliser une boucle for sans variable de boucle (plusieurs instructions dans le corps de la boucle)

G.2.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Action à répéter 6 fois : sauter un tonneau puis avancer	Favorise l'utilisation d'une boucle for dont le corps contient plusieurs instructions
Action à répéter 9 fois : détruire un pot puis avancer	Favorise l'utilisation d'une boucle for dont le corps contient plusieurs instructions
Taille du programme limitée à 14 lignes	Rend nécessaire l'utilisation d'au moins deux boucles for

G.2.4 Stratégies gagnantes

Description	Code

[Stratégie visée]
S2.1 (for-re) :
Sauter par-dessus chacun des 6 tonneaux à l'aide d'une boucle for.
Descendre sur la plateforme inférieure.
Détruire les 9 pots à l'aide d'une boucle for.
Approcher puis ouvrir le coffre.
Concepts :
FOR-RE, FOR-RE
	`for _ in range(6):`
	`  sauter()`
	`  avancer()`
	`avancer()`
	`gauche()`
	`avancer()`
	`avancer()`
	`for _ in range(9):`
	`  coup()`
	`  avancer()`
	`avancer()`
	`ouvrir()`

S2.1 (for-re_variante) :
Faire le saut sur la plateforme inférieure en exécutant une fois de plus la boucle sauter-avancer.
Concepts :
FOR-RE, FOR-RE
	`for _ in range(7):`
	`  sauter()`
	`  avancer()`
	`gauche()`
	`avancer()`
	`avancer()`
	`avancer()`
	`for _ in range(9):`
	`  coup()`
	`  avancer()`
	`avancer()`
	`ouvrir()`

S2.1 (for-re_variante) :
Inversion des instructions dans la deuxième boucle.
Concepts :
FOR-RE, FOR-RE
	`for _ in range(6):`
	`  sauter()`
	`  avancer()`
	`avancer()`
	`gauche()`

	6 avancer() 7 for _ in range(9): 8 avancer() 9 coup() 10 avancer() 11 avancer() 12 ouvrir()
S2.1 (for-re) Utiliser une troisième boucle for pour avancer vers les pots. Concepts : FOR-RE, FOR-RE, FOR-RE	1 for _ in range(6): 2 sauter() 3 avancer() 4 avancer() 5 gauche() 6 for _ in range(2): 7 avancer() 8 for _ in range(9): 9 coup() 10 avancer() 11 avancer() 12 ouvrir()
S2.1 (for-re) : Utiliser une quatrième boucle for pour approcher le coffre Concepts : FOR-RE, FOR-RE, FOR-RE, FOR-RE	1 for _ in range(6): 2 sauter() 3 avancer() 4 avancer() 5 gauche() 6 for _ in range(2): 7 avancer() 8 for _ in range(9): 9 coup() 10 avancer() 11 for _ in range(2): 12 avancer() 13 ouvrir()
S2.1 (for-re) : Utiliser une cinquième boucle for pour avancer sur la plateforme. Concepts : FOR-RE, FOR-RE, FOR-RE, FOR-RE, FOR-RE	1 for _ in range(6): 2 sauter() 3 avancer() 4 for _ in range(2): 5 avancer() 6 gauche() 7 for _ in range(3): 8 avancer() 9 for _ in range(9): 10 coup() 11 avancer() 12 for _ in range(2): 13 avancer() 14 ouvrir()

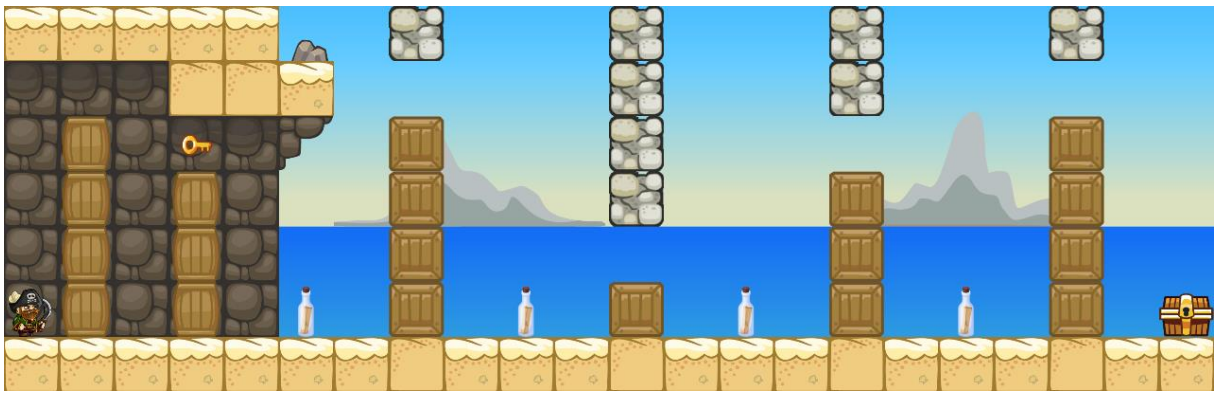
G.3 Niveau 3

G.3.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre. Les deux premières piles de tonneaux ont une hauteur fixe. Ensuite, la hauteur des piles de caisses est aléatoire et change à chaque exécution. Il faut donc lire le message dans la bouteille avant chaque pile qui indique sa hauteur.

Fonctions de contrôle : avancer(), sauter_hauteur(hauteur), lire_nombre(), ouvrir()

Contraintes : Le programme ne doit pas dépasser 14 lignes.



G.3.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, ouvrir)
Utiliser une fonction sans retour et avec argument (sauter_hauteur)
Utiliser une fonction avec retour et sans argument (lire_nombre)
Affecter et utiliser une variable
Manipuler une variable de type entier
Utiliser une boucle for sans variable de boucle (plusieurs instructions dans le corps de la boucle)

G.3.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Présence de plusieurs hauteurs de saut (de 1 à 5)	Rend nécessaire l'utilisation d'une fonction avec un argument (sauter_hauteur)
Hauteurs aléatoires des piles de caisses (entre 1 et 5) associé à la répétition de quatre piles donne $5^4 = 625$ parcours aléatoires différents	Augmente de coût de la stratégie consistant à coder un parcours particulier (S3.2) : $p = \frac{1}{5^4} = \frac{1}{625} = 0{,}16\%$ Favorise l'utilisation d'une fonction avec retour (lire_nombre pour connaitre la hauteur des piles)
Présence de « murs » au-dessus des caisses	Empêche l'enchaînement de 4 sauts de hauteur maximale
Distance d'1 bloc entre la bouteille et la pile de caisses	Rend nécessaire l'affectation d'une variable afin de garder en mémoire la valeur contenue dans la bouteille

Action à répéter 4 fois : lire le message dans la bouteille avancer et sauter de la hauteur indiquée dans le message	Favorise l'utilisation d'une boucle for dont le corps contient plusieurs instructions
Taille du programme limitée à 14 lignes	Rend nécessaire l'utilisation d'une boucle for

G.3.4 Stratégies gagnantes

Description	Code

[Stratégie visée]
S3.1 (var-af):
Sauter les deux premières piles de tonneaux.
Répéter quatre fois à l'aide d'une boucle for :
- lire la hauteur d'une pile de caisses dans la bouteille ;
- la stocker dans une variable ;
- avancer jusqu'à la pile ;
- sauter par-dessus la pile de caisse.
Ouvrir le coffre.
Concepts :
FOR-RE, VAR-AF
	`sauter_hauteur(4)`
	`avancer()`
	`sauter_hauteur(3)`
	`avancer()`
	`avancer()`
	`for _ in range(4):`
	`    message = lire_nombre()`
	`    avancer()`
	`    sauter_hauteur(message)`
	`    avancer()`
	`    avancer()`
	`ouvrir()`

S3.1 (var-af_variante)
Changement du motif répété dans la boucle.
Concepts :
FOR-RE, VAR-AF
	`sauter_hauteur(4)`
	`avancer()`
	`sauter_hauteur(3)`
	`for _ in range(4):`
	`    avancer()`
	`    avancer()`
	`    message = lire_nombre()`
	`    avancer()`
	`    sauter_hauteur(message)`
	`avancer()`
	`avancer()`
	`ouvrir()`

S3.1 (var-af_variante)
Changement du motif répété dans la boucle.
Concepts :
FOR-RE, VAR-AF
	`sauter_hauteur(4)`
	`avancer()`
	`sauter_hauteur(3)`
	`avancer()`
	`for _ in range(4):`
	`    avancer()`
	`    message = lire_nombre()`
	`    avancer()`
	`    sauter_hauteur(message)`
	`    avancer()`
	`avancer()`
	`ouvrir()`

S3.1 (var-af_variante):
Utilisation d'une deuxième boucle for pour avancer vers la première bouteille.
Concepts :
FOR-RE, FOR-RE, VAR-AF
	`sauter_hauteur(4)`
	`avancer()`
	`sauter_hauteur(3)`
	`for _ in range(2):`
	`    avancer()`
	`for _ in range(4):`

	7 message = lire_nombre() 8 avancer() 9 sauter_hauteur(message) 10 avancer() 11 avancer() 12 ouvrir()
S3.1 (var-af_variante): Utilisation d'une troisième boucle for pour avancer vers les bouteilles. Concepts : FOR-RE, FOR-RE, FOR-RE, VAR-AF	1 sauter_hauteur(4) 2 avancer() 3 sauter_hauteur(3) 4 for _ in range(2): 5 avancer() 6 for _ in range(4): 7 message = lire_nombre() 8 avancer() 9 sauter_hauteur(message) 10 for _ in range(2): 11 avancer() 12 ouvrir()
S3.1 (var-af_variante): Lecture inutile en dehors de la boucle de la première bouteille. Concepts : FOR-RE, VAR-AF, VAR-AF	1 sauter_hauteur(4) 2 avancer() 3 sauter_hauteur(3) 4 avancer() 5 avancer() 6 message = lireNombre() 7 for _ in range(4): 8 message = lireNombre() 9 avancer() 10 sauter_hauteur(message) 11 avancer() 12 avancer() 13 ouvrir()
S3.1 (var-af_variante): Lecture inutile de la première bouteille en dehors de la boucle. Concepts : FOR-RE, FOR-RE, VAR-AF, VAR-AF	1 sauter_hauteur(4) 2 avancer() 3 sauter_hauteur(3) 4 for _ in range(2): 5 avancer() 6 message = lire_nombre() 7 for _ in range(4): 8 message = lire_nombre() 9 avancer() 10 sauter_hauteur(message) 11 avancer() 12 avancer() 13 ouvrir()
S3.2 (no-var-af) : Codage d'un parcours « en dur » sans soucier de la hauteur aléatoire des caisses. Concepts : FOR-RE	1 sauter_hauteur(4) 2 avancer() 3 sauter_hauteur(3) 4 avancer() 5 avancer() 6 lire_nombre() 7 avancer()

```
sauter_hauteur(4)
avancer()
avancer()
for _ in range (12):
        avancer()
sauter_hauteur(2)
ouvrir()
```

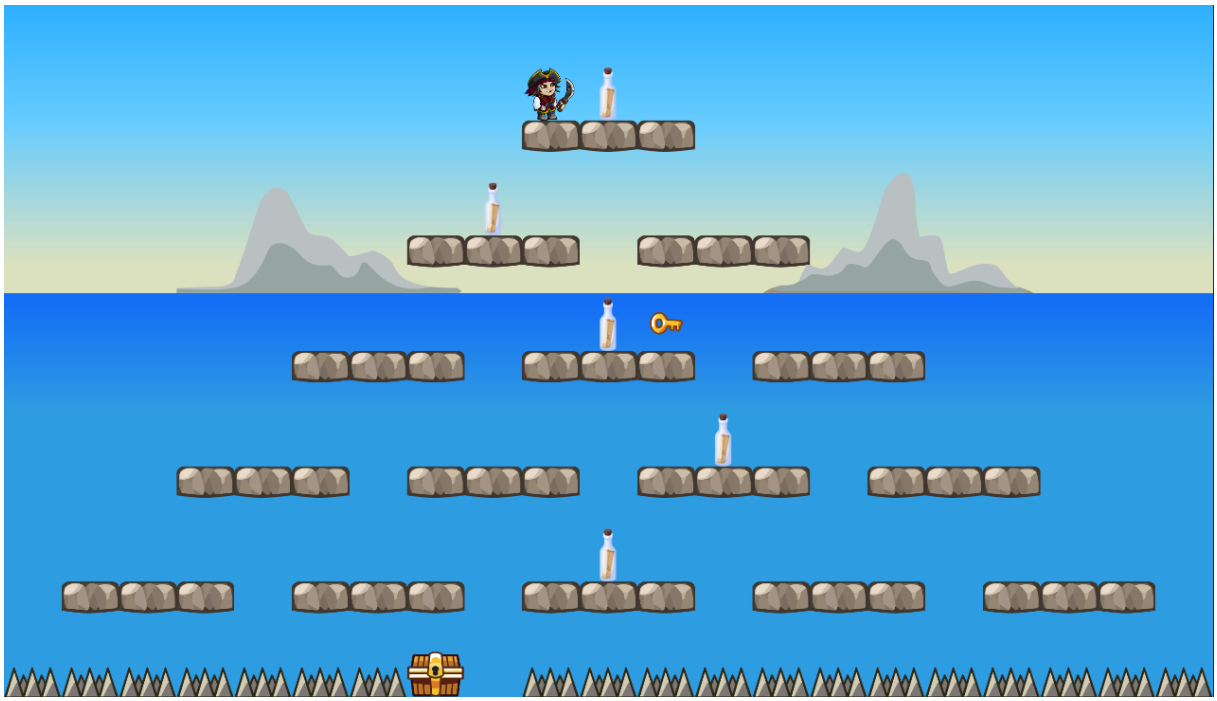
G.4 Niveau 4

G.4.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre. L'emplacement du coffre et de la clé sont aléatoires et changent à chaque exécution. Il faut donc lire les messages dans les bouteilles qui indiquent le chemin à suivre (gauche ou droite) à chaque étage. Le fait de tomber dans les piques fait perdre puis recommencer le niveau.

Fonctions de contrôle : avancer(), gauche(), droite(), lire_chaine(), ouvrir()

Contraintes : Le programme ne doit pas dépasser 18 lignes.



G.4.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Notions algorithmiques
Utiliser une fonction sans retour et sans argument (avancer, gauche, droite, ouvrir)
Utiliser une fonction avec retour et sans argument (lire_chaine)
Affecter et utiliser une variable
Manipuler une variable de type chaîne de caractères
Utiliser une structure conditionnelle à deux branches

Tester une égalité
Utiliser une boucle for sans variable de boucle (plusieurs instructions dans le corps de la boucle)

G.4.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Emplacement aléatoire de la clé, du coffre et des bouteilles sur 5 niveaux de plateforme. Donne $2^5 = 32$ parcours différents.	Augmente de coût de la stratégie consistant à coder un parcours particulier (S4.2) : $p = \frac{1}{32} \approx 3{,}1\%$ Favorise l'utilisation d'une fonction avec retour (lire_chaine pour connaitre le chemin à suivre) Favorise l'utilisation d'une structure conditionnelle à deux branches et d'un test d'égalité pour suivre l'indication du message.
Message dans les bouteilles de type chaînes de caractères	Rend nécessaire la manipulation de chaînes de caractères
Action à répéter 5 fois : lire le message dans la bouteille, s'orienter en fonction du message, et descendre de la plateforme	Favorise l'utilisation d'une boucle for dont le corps contient plusieurs instructions
Taille du programme limitée à 14 lignes	Rend nécessaire l'utilisation d'une boucle for

G.4.4 Stratégies gagnantes

Description	Code
[Stratégie visée] S4.1 (if-else) : Avancer jusqu'à la première bouteille. Utiliser une boucle for pour répéter cinq fois les instructions : - appeler la fonction lire_chaine et affecter une variable du retour de cette fonction ; - réaliser un test sur cette variable à l'aide d'une structure conditionnelle if-else permettant d'aller à gauche ou à droite ; - avancer de deux blocs pour descendre d'un étage. Ouvrir le coffre. Concepts : FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 for _ in range(5): 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 else: 7 droite() 8 avancer() 9 avancer() 10 ouvrir()
S4.1 (if-else_variante) : Utilisation d'une deuxième boucle for. Concepts : FOR-RE, VAR-AF, IF, ELSE,	1 avancer() 2 for _ in range(5): 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 else: 7 droite() 8 for _ in range(2): 9 avancer() 10 ouvrir()

S4.1 (if-else_variante) :
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, VAR-AF, IF, ELSE

```
avancer()
for _ in range(5):
    msg = lire_chaine()
    if msg == "gau":
        gauche()
        avancer()
        avancer()
    else:
        droite()
        avancer()
        avancer()
ouvrir()
```

S4.1 (if-else_variante) :
Modification de l'ordre des instructions dans la boucle for.
Concepts :
FOR-RE, VAR-AF, IF, ELSE

```
for _ in range (5):
    avancer()
    msg = lire_chaine()
    if msg == "gau":
        gauche()
    else :
        droite()
    avancer()
avancer()
ouvrir()
```

S4.1 (if-else_variante) :
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, VAR-AF, IF, ELSE

```
for _ in range (5):
    avancer()
    msg = lire_chaine()
    if msg == "gau":
        gauche()
        avancer()
    else:
        droite()
        avancer()
avancer()
ouvrir()
```

S4.1 (if-else_variante) :
Faire l'appel de fonction directement dans la condition.
Concepts :
FOR-RE, IF, ELSE

```
avancer()
for _ in range(5):
    if lire_chaine() == "gau":
        gauche()
    else:
        droite()
    avancer()
    avancer()
ouvrir()
```

S4.1 (if-else_variante) :
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, IF, ELSE

```
avancer()
for _ in range(3):
    if lire_chaine() == "gau":
        gauche()
        avancer()
        avancer()
    else:
        droite()
        avancer()
```

	10 `        avancer()` 11 `ouvrir()`
S4.1 (if-else_variante) : Modification de l'ordre des instructions dans la boucle Concepts : FOR-RE, IF, ELSE	1 `for _ in range (3):` 2 `    avancer()` 3 `    if lire_chaine() == "gau":` 4 `        gauche()` 5 `    else:` 6 `        droite()` 7 `    avancer()` 8 `avancer()` 9 `ouvrir()`
S4.1 (if-else_variante) : Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, IF, VAR-AF, ELSE	1 `for _ in range (3):` 2 `    avancer()` 3 `    if lire_chaine() == "gau":` 4 `        gauche()` 5 `        avancer()` 6 `    else:` 7 `        droite()` 8 `        avancer()` 9 `avancer()` 10 `ouvrir()`
S4.2 (if-if) : Utiliser deux structures conditionnelles if. (non optimal du point de vue de l'efficacité algorithmique) Concepts : FOR-RE, IF, VAR-AF, IF	1 `avancer()` 2 `for _ in range(4):` 3 `    msg = lire_chaine()` 4 `    if msg == "gau":` 5 `        gauche()` 6 `    if msg == "droi":` 7 `        droite()` 8 `    avancer()` 9 `    avancer()` 10 `ouvrir()`
S4.2 (if-if_variante) : Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, IF, VAR-AF, IF	1 `avancer()` 2 `for _ in range(4):` 3 `    msg = lire_chaine()` 4 `    if msg == "gau":` 5 `        gauche()` 6 `        avancer()` 7 `        avancer()` 8 `    if msg == "droi":` 9 `        droite()` 10 `        avancer()` 11 `        avancer()` 12 `ouvrir()`
S4.2 (if-if_variante) : Modification de l'ordre des instructions dans la boucle Concepts : FOR-RE, IF, VAR-AF, IF	1 `for _ in range (4):` 2 `    avancer()` 3 `    msg = lire_chaine()` 4 `    if msg == "gau":` 5 `        gauche()` 6 `    if msg == "droi":` 7 `        droite()` 8 `    avancer()`

	9 avancer() 10 ouvrir()
S4.2 (if-if_variante) : Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, IF, VAR-AF, IF	1 for _ in range (4): 2 avancer() 3 msg = lire_chaine() 4 if msg == "gau" : 5 gauche() 6 avancer() 7 if msg == "droi": 8 droite() 9 avancer() 10 avancer() 11 ouvrir()
S4.2 (if-if_variante) : Faire l'appel de fonction directement dans la condition. Concepts : FOR-RE, IF, IF	1 avancer() 2 for _ in range(4): 3 if lire_chaine() == "gau": 4 gauche() 5 if lire_chaine() == "droi": 6 droite() 7 avancer() 8 avancer() 9 ouvrir()
S4.2 (if-if_variante) : Modification de l'ordre des instructions dans la boucle. Concepts : FOR-RE, IF, IF	1 for _ in range (4): 2 avancer() 3 if lire_chaine() == "gau": 4 gauche() 5 if lire_chaine() == "droi": 6 droite() 7 avancer() 8 avancer() 9 ouvrir()
S4.3 (if-elif) Utiliser une structure conditionnelle if-elif. (non optimal du point de vue de l'efficacité algorithmique) Concepts : FOR-RE, VAR-AF, IF, ELIF	1 avancer() 2 for _ in range(4): 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 elif msg == "droi": 7 droite() 8 avancer() 9 avancer() 10 ouvrir()
S4.3 (if-elif variante) Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, VAR-AF, IF, ELIF	1 avancer() 2 for _ in range(4): 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 avancer() 7 avancer() 8 elif msg == "droi": 9 droite() 10 avancer()

	11 avancer() 12 ouvrir()
S4.3 (if-elif_variante) Modification de l'ordre des instructions dans la boucle Concepts : FOR-RE, VAR-AF, IF, ELIF	1 for _ in range (4) : 2 avancer() 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 elif msg == "droi": 7 droite() 8 avancer() 9 avancer() 10 ouvrir()
S4.3 (if-elif_variante) Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, VAR-AF, IF, ELIF	1 for _ in range (4) : 2 avancer() 3 msg = lire_chaine() 4 if msg == "gau": 5 gauche() 6 avancer() 7 elif msg == "droi": 8 droite() 9 avancer() 10 avancer() 11 ouvrir()
S4.3 (if-elif_variante) Faire l'appel de fonction directement dans la condition. Concepts : FOR-RE, IF, ELIF	1 avancer() 2 for _ in range(4): 3 if lire_chaine() == "gau": 4 gauche() 5 elif lire_chaine() =="droi": 6 droite() 7 avancer() 8 avancer() 9 ouvrir()
S4.3 (if-elif_variante) Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, IF, ELIF	1 avancer() 2 for _ in range(4): 3 if lire_chaine() == "gau": 4 gauche() 5 avancer() 6 avancer() 7 elif lire_chaine() =="droi": 8 droite() 9 avancer() 10 avancer() 11 ouvrir()
S4.3 (if-elif_variante) Modification de l'ordre des instructions dans la boucle Concepts : FOR-RE, IF, ELIF	1 for _ in range (4) : 2 avancer() 3 if lire_chaine() == "gau": 4 gauche() 5 elif lire_chaine() =="droi": 6 droite() 7 avancer() 8 avancer()

	`ouvrir()`

S4.3 (if-elif_variante)
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, IF, ELIF
	`for _ in range (4):`
	`  avancer()`
	`  if lire_chaine() == "gau":`
	`  gauche()`
	`  avancer()`
	`  elif lire_chaine() =="droi":`
	`  droite()`
	`  avancer()`
	`avancer()`
	`ouvrir()`

S4.4 (no-cond) :
Essayer un parcours particulier.
Concepts :
∅
	`gauche()`
	`avancer()`
	`avancer()`
	`avancer()`
	`avancer()`
	`avancer()`
	`avancer()`
	`avancer()`
	`ouvrir()`

S4.4 (no-cond_variante) :
Essayer un parcours particulier en utilisant une boucle.
Concepts :
FOR-RE
	`for _ in range(9):`
	`  avancer()`
	`ouvrir()`

S4.4 (no-cond_variante))
Essayer un parcours particulier en utilisant deux boucles
Concepts :
FOR-RE, FOR RE
	`for _ in range(6):`
	`  avancer()`
	`for _ in range(3):`
	`  avancer()`
	`ouvrir()`

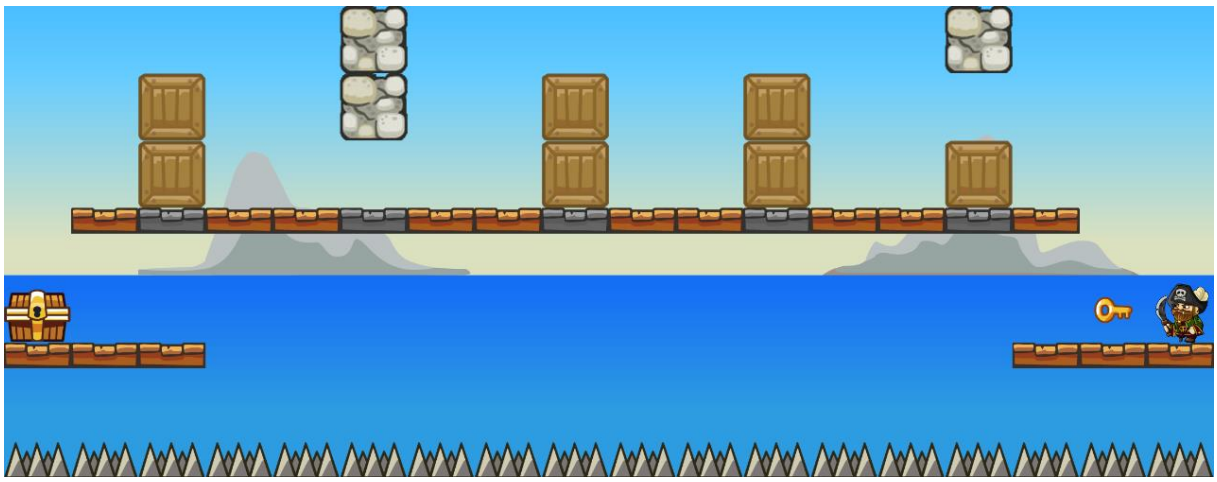
G.5 Niveau 5

G.5.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre. Sur la plateforme supérieure, la hauteur des caisses et des murs est aléatoire et change à chaque exécution. Le fait de tomber dans les piques fait perdre puis recommencer le niveau.

Fonctions de contrôle : avancer(), sauter(), sauter_haut(), mesurer_hauteur(), ouvrir()

Contraintes : Le programme ne doit pas dépasser 18 lignes.



G.5.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, sauter, sauter_haut, ouvrir)
Utiliser une fonction avec retour et sans argument (mesurer_hauteur)
Affecter et utiliser une variable
Manipuler une variable de type entier
Utiliser une structure conditionnelle à trois branches
Tester une égalité
Utiliser une boucle for sans variable de boucle (plusieurs instructions dans le corps de la boucle)

G.5.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Hauteurs aléatoires des piles de caisses (entre 0 et 2) associé à la répétition de cinq piles donne $3^5 = 243$ parcours aléatoires différents	Augmente de coût de la stratégie consistant à coder un parcours particulier (S5.4) : $p = \frac{1}{3^5} = \frac{1}{243} \approx 0{,}4\%$ Favorise l'utilisation d'une fonction avec retour (mesurer_hauteur pour connaitre la hauteur des piles des caisses) Favorise l'utilisation d'une structure conditionnelle et d'un test d'égalité
Pas d'accès à la fonction sauter_hauteur mais présence de trois fonction différentes pour passer les obstacles selon leur hauteur : avancer, sauter et sauter_haut	Favorise l'utilisation d'une structure conditionnelle à trois branches
Présence de « murs » au-dessus des caisses	Empêche l'enchaînement de 5 sauts de hauteur maximale
Action à répéter 5 fois : mesurer la hauteur des caisses,	Favorise l'utilisation d'une boucle for dont le corps contient plusieurs instructions
Taille du programme limitée à 18 lignes	Rend nécessaire l'utilisation d'une boucle for

G.5.4 Stratégies gagnantes

Description	Code

[Stratégie visée]
S5.1 (if-elif-else) :
Ramasser la clé sur la plateforme inférieure, puis monter à l'aide d'un saut haut.
Utiliser une boucle for pour répéter cinq fois les instructions :
- mesurer la hauteur des caisses se situant devant soi à l'aide de la fonction mesurer_hauteur puis affecter une variable à l'aide de cette valeur ;
- utiliser une structure conditionnelle if-elif-else afin d'effectuer l'action adéquate en fonction de la hauteur des caisses (avancer, sauter ou sauter_haut) ;
- avancer deux fois jusqu'aux prochaines caisses.
Avancer pour descendre sur la plateforme inférieure puis ouvrir le coffre.
Concepts :
FOR-RE, VAR-AF, IF, ELIF, ELSE
	`avancer()`
	`sauter_haut()`
	`for _ in range(5):`
	`    hauteur = mesurer_hauteur()`
	`    if hauteur == 0:`
	`        avancer()`
	`    elif hauteur == 1:`
	`        sauter()`
	`    else:`
	`        sauter_haut()`
	`    avancer()`
	`    avancer()`
	`ouvrir()`

S5.1 (if-elif-else_variante) :
Utiliser une boucle for supplémentaire
Concepts :
FOR-RE, VAR-AF, IF, ELIF, ELSE, FOR-RE
	`avancer()`
	`sauter_haut()`
	`for _ in range(5):`
	`    hauteur = mesurer_hauteur()`
	`    if hauteur == 0:`
	`        avancer()`
	`    elif hauteur == 1:`
	`        sauter()`
	`    else:`
	`        sauter_haut()`
	`    for _ in range(2):`
	`        avancer()`
	`ouvrir()`

S5.1 (if-elif-else_variante) :
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, VAR-AF, IF, ELIF, ELSE
	`avancer()`
	`sauter_haut()`
	`for _ in range(5):`
	`    hauteur = mesurer_hauteur()`
	`    if hauteur == 0:`
	`        avancer()`
	`        avancer()`
	`        avancer()`
	`    elif hauteur == 1:`
	`        sauter()`
	`        avancer()`
	`        avancer()`
	`    else:`
	`        sauter_haut()`
	`        avancer()`
	`        avancer()`
	`ouvrir()`

S5.1 (if-elif-else_variante) :

Détection d'obstacle tout au long de plateforme supérieure (et pas seulement sur les 5 emplacements affectés par le hasard).

Concepts :

FOR-RE, VAR-AF, IF, ELIF, ELSE

```
avancer()
sauter_haut()
for _ in range(15):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
    elif hauteur == 1:
        sauter()
    else:
        sauter_haut()
ouvrir()
```

S5.1 (if-elif-else_variante) :

Faire l'appel de fonction directement dans la condition du if et du elif.

Concepts :

FOR-RE, IF, ELIF, ELSE

```
avancer()
sauter_haut()
for _ in range(5):
    if mesurer_hauteur() == 0:
        avancer()
    elif mesurer_hauteur() == 1:
        sauter()
    else:
        sauter_haut()
    avancer()
    avancer()
ouvrir()
```

S5.1 (if-elif-else_variante) :

Non « factorisation » de la fonction avancer à la suite de la conditionnelle.

Concepts :

FOR-RE, IF, ELIF, ELSE

```
avancer()
sauter_haut()
for _ in range(4):
    if mesurer_hauteur() == 0:
        avancer()
        avancer()
        avancer()
    elif mesurer_hauteur() == 1:
        sauter()
        avancer()
        avancer()
    else:
        sauter_haut()
        avancer()
        avancer()
ouvrir()
```

S5.1 (if-elif-else_variante) :

Détection d'obstacle tout au long de plateforme supérieure (et pas seulement sur les 4 emplacements affectés par le hasard).

Concepts :

FOR-RE, IF, ELIF, ELSE

```
avancer()
sauter_haut()
for _ in range(14):
    if mesurer_hauteur() == 0:
        avancer()
    elif mesurer_hauteur() == 1:
        sauter()
    else:
        sauter_haut()
ouvrir()
```

S5.2 (if-if-if) :

```
avancer()
sauter_haut()
```

Utiliser trois structures conditionnelles if.
(non optimal du point de vue de l'efficacité algorithmique)
Concepts :
FOR-RE, VAR-AF, IF, IF, IF

```
for _ in range(5):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
    if hauteur == 1:
        sauter()
    if hauteur == 2:
        sauter_haut()
    avancer()
    avancer()
ouvrir()
```

S5.2 (if-if-if_variante) :
Non « factorisation » de la fonction avancer à la suite de la conditionnelle.
Concepts :
FOR-RE, VAR-AF, IF, IF, IF

```
avancer()
sauter_haut()
for _ in range(5):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
        avancer()
        avancer()
    if hauteur == 1:
        sauter()
        avancer()
        avancer()
    if hauteur == 2:
        sauter_haut()
        avancer()
        avancer()
ouvrir()
```

S5.2 (if-if-if_variante) :
Détection d'obstacle tout au long de plateforme supérieure (et pas seulement sur les 4 emplacements affectés par le hasard).
Concepts :
FOR-RE, VAR-AF, IF, IF, IF

```
avancer()
sauter_haut()
for _ in range(14):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
    if hauteur == 1:
        sauter()
    if hauteur == 2:
        sauter_haut()
ouvrir()
```

S5.2 (if-if-if_variante) :Faire l'appel de fonction directement dans la condition des if.
Concepts :
FOR-RE, IF, IF, IF

```
avancer()
sauter_haut()
for _ in range(5):
    if mesurer_hauteur() == 0:
        avancer()
    if mesurer_hauteur() == 1:
        sauter()
    if mesurer_hauteur() == 2:
        sauter_haut()
    avancer()
    avancer()
ouvrir()
```

S5.3 (if-elif-elif) :

Utiliser une structure conditionnelle if-elif-elif.

(non optimal du point de vue de l'efficacité algorithmique)

Concepts :

FOR-RE, VAR-AF, IF, ELIF, ELIF

```
avancer()
sauter_haut()
for _ in range(5):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
    elif hauteur == 1:
        sauter()
    elif hauteur == 2:
        sauter_haut()
    avancer()
    avancer()
ouvrir()
```

S5.3 (if-elif-elif_variante) :

Non « factorisation » de la fonction avancer à la suite de la conditionnelle.

Concepts :

FOR-RE, VAR-AF, IF, ELIF, ELIF

```
avancer()
sauter_haut()
for _ in range(5):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
        avancer()
        avancer()
    elif hauteur == 1:
        sauter()
        avancer()
        avancer()
    elif hauteur == 2:
        sauter_haut()
        avancer()
        avancer()
ouvrir()
```

S5.3 (if-elif-elif_variante) :

Détection d'obstacle tout au long de plateforme supérieure (et pas seulement sur les 5 emplacements affectés par le hasard).

Concepts :

FOR-RE, VAR-AF, IF, ELIF, ELIF

```
avancer()
sauter_haut()
for _ in range(14):
    hauteur = mesurer_hauteur()
    if hauteur == 0:
        avancer()
    elif hauteur == 1:
        sauter()
    elif hauteur == 2:
        sauter_haut()
ouvrir()
```

S5.3 (if-elif-elif_variante) :

Faire l'appel de fonction directement dans la condition du if et des elif.

Concepts :

FOR-RE, IF, ELIF, ELIF

```
avancer()
sauter_haut()
for _ in range(5):
    if mesurer_hauteur() == 0:
        avancer()
    elif mesurer_hauteur() == 1:
        sauter()
    elif mesurer_hauteur() == 2:
        sauter_haut()
    avancer()
```

	11 `    avancer()` 12 `ouvrir()`
S5.3 (if-elif-elif_variante) : Non « factorisation » de la fonction avancer à la suite de la conditionnelle. Concepts : FOR-RE, IF, ELIF, ELIF	1 `avancer()` 2 `sauter_haut()` 3 `for _ in range(4):` 4 `    if mesurer_hauteur() == 0:` 5 `        avancer()` 6 `        avancer()` 7 `        avancer()` 8 `    elif mesurer_hauteur() == 1:` 9 `        sauter()` 10 `        avancer()` 11 `        avancer()` 12 `    elif mesurer_hauteur() == 2:` 13 `        sauter_haut()` 14 `        avancer()` 15 `        avancer()` 16 `ouvrir()`
S5.3 (if-elif-elif_variante) : Détection d'obstacle tout au long de plateforme supérieure (et pas seulement sur les 5 emplacements affectés par le hasard). Concepts : FOR-RE, IF, ELIF, ELIF	1 `avancer()` 2 `sauter_haut()` 3 `for _ in range(14):` 4 `    if mesurer_hauteur() == 0:` 5 `        avancer()` 6 `    elif mesurer_hauteur() == 1:` 7 `        sauter()` 8 `    elif mesurer_hauteur() == 2:` 9 `        sauter_haut()` 10 `ouvrir()`
S5.4 (no-cond) : Essayer un parcours en particulier. Concepts : ∅	1 `avancer()` 2 `sauter_haut()` 3 `avancer()` 4 `sauter()` 5 `avancer()` 6 `avancer()` 7 `sauter()` 8 `avancer()` 9 `avancer()` 10 `sauter()` 11 `avancer()` 12 `avancer()` 13 `sauter()` 14 `avancer()` 15 `avancer()` 16 `avancer()` 17 `ouvrir()`
S5.4 (no-cond_variante) : Essayer un parcours en particulier en utilisant une boucle for Concepts : FOR-RE	1 `avancer()` 2 `sauter_haut()` 3 `for _ in range(14):` 4 `    avancer()` 5 `ouvrir()`

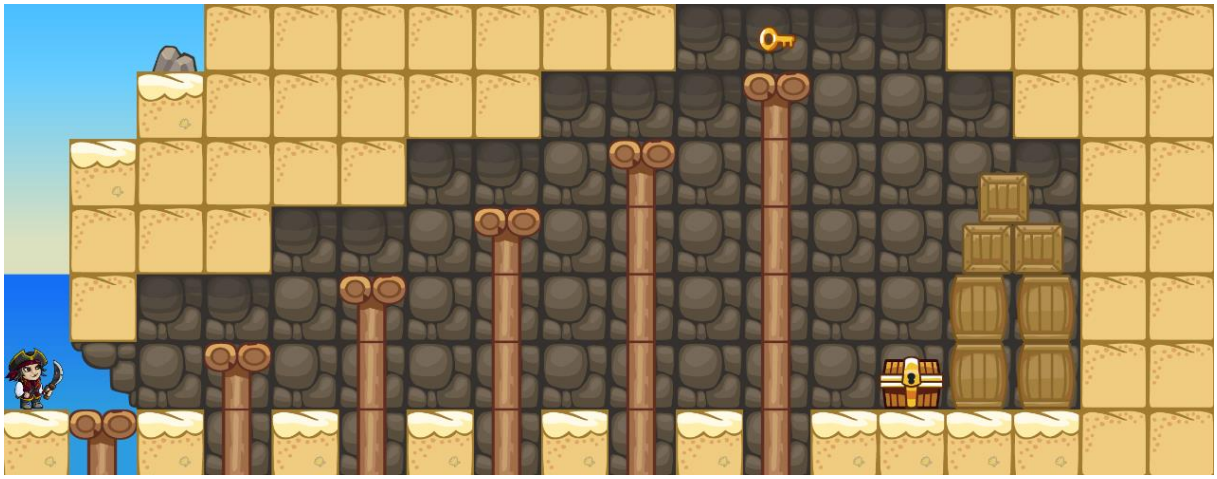
G.6 Niveau 6

G.6.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre, la hauteur des piliers est fixe.

Fonctions de contrôle : avancer(), sauter_hauteur(hauteur), ouvrir()

Contraintes : Le programme ne doit pas dépasser 4 lignes.



G.6.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, ouvrir)
Utiliser une fonction sans retour et avec argument (sauter_hauteur)
Utiliser une variable (variable de boucle)
Utiliser une boucle for avec variable de boucle (commence à zéro)

G.6.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Action à répéter 6 fois : sauter d'une hauteur qui augmente d'un bloc à chaque pilier	Favorise l'implémentation d'une boucle for en utilisant la variable de boucle
Permettre d'avancer en employant la fonction sauter_hauteur(0)	Favoriser l'utilisation d'une variable de boucle qui commence à zéro
Présence du plafond de la grotte « en escalier »	Empêche l'enchaînement de 6 sauts de hauteur maximale
Taille du programme limitée à 4 lignes	Rend nécessaire l'utilisation d'une boucle for avec variable de boucle

G.6.4 Stratégies gagnantes

Description	Code
S6.1 (for-co-0) : **[Stratégie visée]**	`1 for compteur in range(6):`

Gravir les piliers à l'aide d'une boucle en utilisant la variable de boucle afin de sauter à la bonne hauteur. Ouvrir le coffre. Concepts : FOR-VA-0	2 sauter_hauteur(compteur) 3 avancer() 4 ouvrir()
S6.1 (for-co-0_variante) : Utiliser la fonction sauter_hauteur pour avancer Concepts : FOR-VA-0	1 for compteur in range(6): 2 sauter_hauteur(compteur) 3 sauter_hauteur(0) 4 ouvrir()
S6.1 (for-co-0_variante) : Expliciter le début de la variable de boucle à 0 même si cela est inutile (valeur par défaut) Concepts : FOR-VA-0	1 for compteur in range(0,6): 2 sauter_hauteur(compteur) 3 avancer() 4 ouvrir()
S6.1 (for-co-0_variante) : Expliciter le pas d'incrémentation de la variable de boucle même si cela est inutile (valeur par défaut) Concepts : FOR-VA-0	1 for compteur in range(0,6,1): 2 sauter_hauteur(compteur) 3 avancer() 4 ouvrir()
S6.2 (for-co-n) : Implémenter une boucle for en utilisant la variable de boucle qui ne commence pas à zéro Concepts : FOR-VA-N	1 for k in range (1,6): 2 sauter_hauteur(k-1) 3 avancer() 4 ouvrir()

G.7 Niveau 7

G.7.1 Description

Objectif : Tirer sur les différentes séries de noix de coco qui apparaissent des deux côtés puis, à la fin, sur le coffre. L'autre pirate se place toujours derrière la dernière noix. Le fait de tirer sur ce pirate fait perdre puis recommencer le niveau. Il est possible d'atteindre plusieurs noix de coco à l'aide d'une seule balle.

Fonctions de contrôle : tourner, tirer(distance)

Contraintes : Le programme ne doit pas dépasser 5 lignes.

G.7.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (tourner)
Utiliser une fonction sans retour et avec argument (tirer)
Utiliser une variable (variable de boucle)
Utiliser une boucle for avec variable de boucle (ne commence pas à zéro)

G.7.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Action à répéter 7 fois : tirer sur des séries de noix de coco dont le nombre augmente d'une unité à chaque fois	Favorise l'implémentation d'une boucle for en utilisant la variable de boucle
La première série contient deux noix de coco et la fonction tirer ne peut être utilisée avec un paramètre nul	Favorise l'utilisation d'une variable de boucle qui ne commence pas à zéro
Présence d'un personnage derrière la dernière noix de coco	Empêche l'enchaînement de 7 tirs de distance maximal
Apparition finale du coffre (et du personnage) à une distance de trois blocs	Oblige à effectuer un tir en dehors de la boucle (et à bien calculer le nombre de tours de boucle)
Taille du programme limitée à 5 lignes	Rend nécessaire l'utilisation d'une boucle for avec variable de boucle

G.7.4 Stratégies gagnantes

Description	Code

S7.1 (for-co-n) : **[Stratégie visée]**
Tirer alternativement à droite puis à gauche et de plus en plus loin sur les différentes séries de noix de coco à l'aide d'une boucle for et de sa variable de boucle initialisée à 2 à prenant 8 pour dernière valeur.
Tirer sur le coffre.
Concepts :
FOR-VA-N
	`for compteur in range(2,9):`
	`  tirer(compteur)`
	`  tourner()`
	`tirer(3)`

S7.1 (for-co-n_variante) :
Utiliser deux boucles imbriquées pour tirer sur les noix de coco une à une.
Concepts :
FOR-VA-N, FOR-VA-N
	`for cpt in range(3,10):`
	`  for dist in range(1,cpt):`
	`  tirer(dist)`
	`  tourner()`
	`tirer(3)`

S7.1 (for-co-n_variante) :
Utiliser deux boucles imbriquées afin de tirer sur les noix de coco une à une (on tire sept fois quel que soit le nombre de noix)
Concepts :
FOR-RE, FOR-VA-N
	`for _ in range (9):`
	`  for dist in range (2,9):`
	`  tirer (dist)`
	`  tourner ()`

S7.1 (for-co-n_variante) :
Utiliser deux boucles imbriquées pour tirer sur les noix de coco une à une.
Concepts :
FOR-VA-N, FOR-VA-0

```
for cpt in range(2,9):
    for dist in range(cpt):
        tirer(dist+1)
    tourner()
tirer(3)
```

S7.2 (for-co-0) :
Utiliser une boucle for avec une variable de boucle commençant à zéro.
Concepts :
FOR-VA-0

```
for compteur in range (7):
    tirer(compteur+2)
    tourner()
tirer(3)
```

S7.2 (for-co-0_variante) :
Même chose avec des variables de boucle qui commencent à zéro
Concepts:
FOR-VA-0, FOR-VA-0

```
for i in range(7):
    for j in range(i+2):
        tirer(j+1)
    tourner()
tirer(3)
```

S7.2 (for-co-0_variante) :
Même chose avec une deuxième boucle qui commence à zéro
Concepts :
FOR-RE, FOR-VA-0

```
for _ in range (9):
    for dist in range (7):
        tirer(dist+2)
    tourner ()
```

S7.3 (for-re) :
Utiliser une boucle for pour la répétition est gestion « à la main » d'un compteur pour ajuster la distance de tir.
/!\ Nécessite de mettre deux instructions sur une seule ligne
Concepts :
FOR-RE, VAR-AF, VAR-AF

```
dist=2
for _ in range(7):
    tirer(dist); tourner()
    dist += 1
tirer(3)
```

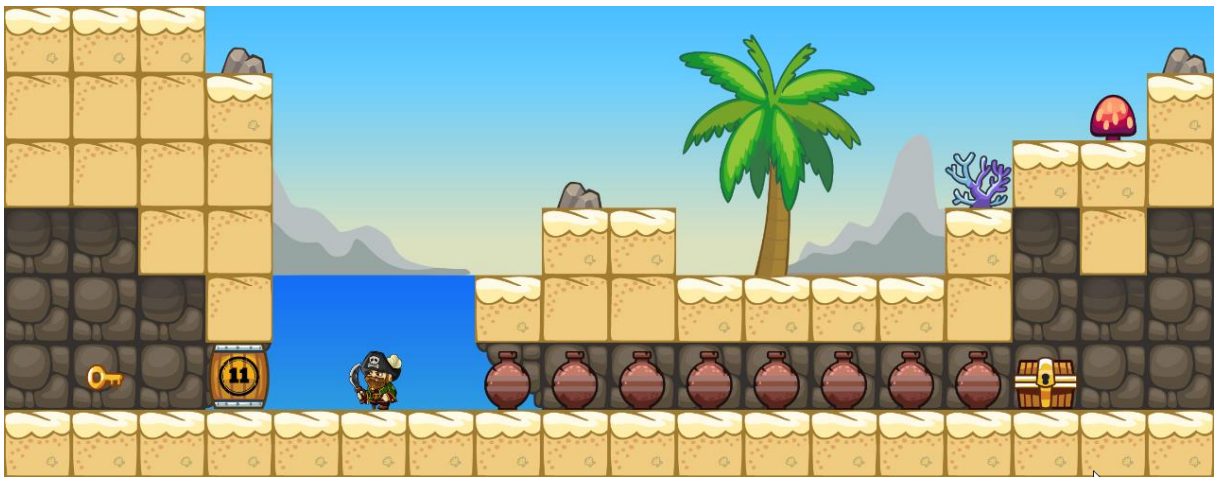
G.8 Niveau 8

G.8.1 Description

Objectif : Ramasser la clé puis ouvrir le coffre. La solidité du tonneau (nombre dans le cercle), le nombre de pots ainsi que l'emplacement du coffre sont aléatoires et changent à chaque exécution. Un fois le tonneau détruit, de la dynamite apparait à sa place. Il ne faut pas donner de coup dans cette dynamite sous peine d'une explosion qui fait perdre puis recommencer le niveau.

Fonctions de contrôle : avancer(), gauche(), droite(), coup(), detecter_obstacle(), ouvrir()

Contraintes : Aucunes



G.8.2 Savoirs en jeu

xxx = concepts introduits dans ce niveau

Concepts fondamentaux
Utiliser une fonction sans retour et sans argument (avancer, gauche, droite, coup, ouvrir)
Utiliser une fonction avec retour et sans argument (detecter_obstacle)
Affecter et utiliser une variable
Manipuler une variable de type booléen
Utiliser une boucle while
Utiliser une boucle for sans variable de boucle (plusieurs instructions dans le corps de la boucle)

G.8.3 Variables didactiques

Variables didactiques	Influence sur les procédures
Le tonneau possède une solidité aléatoire (entre 1 et 20) et le nombre de pots est également aléatoire (entre 1 et 10). Donne $20 \times 10 = 200$ parcours différents.	Augmente de coût de la stratégie consistant à coder un parcours particulier (S8.4) : $p = \frac{1}{20 \times 10} = \frac{1}{200} = 0{,}5\%$ Favorise l'utilisation d'une boucle while
Actions répétitives : donner plusieurs coups sur le tonneau, plusieurs coup/avancer pour détruire les pots	Favorise l'utilisation d'une boucle
Apparition de bâton de dynamite qui ne doit pas être détruit lorsque le tonneau est détruit.	Empêche d'assener un nombre de coups maximum sur le tonneau (20) quel que soit sa solidité Favorise l'utilisation d'une boucle while
Placement du coffre derrière le dernier pot	Empêche d'enchaîner les coup/avancer un nombre maximum de fois (10) quel que soit le nombre de pots. Favorise l'utilisation d'une boucle while

G.8.4 Stratégies gagnantes

Description	Code
S8.1 (while-while) : **[Stratégie visée]**	`avancer()`

Utiliser une boucle while afin de donner des coups de sabre tant qu'un obstacle (le tonneau) se trouve devant soi.

Avancer de trois blocs jusqu'à la clé.

Se retourner puis avancer de 5 blocs jusqu'au premier pot.

Utiliser une boucle while afin de porter un coup de sabre et d'avancer tant qu'un obstacle (un pot) se trouve devant nous.

Ouvrir le coffre.

Concepts :

VAR-AF, WHI, VAR-AF, FOR-RE, FOR-RE, VAR-AF, WHI, VAR-AF

```
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    obstacle = detecter_obstacle()
for _ in range(3) :
    avancer()
droite()
for _ in range(5):
    avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    avancer()
    obstacle = detecter_obstacle()
ouvrir()
```

S8.1 (while-while_variante) :

Avancer vers la clé sans boucle for.

Concepts :

VAR-AF, WHI, VAR-AF, FOR-RE, VAR-AF, WHI, VAR-AF

```
avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    obstacle = detecter_obstacle()
avancer()
avancer()
avancer()
droite()
for _ in range(5):
    avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    avancer()
    obstacle = detecter_obstacle()
ouvrir()
```

S8.1 (while-while_variante) :

Avancer vers les pots sans boucle for.

Concepts :

VAR-AF, WHI, VAR-AF, FOR-RE, VAR-AF, WHI, VAR-AF

```
avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    obstacle = detecter_obstacle()
for _ in range(3):
    avancer()
droite()
avancer()
avancer()
avancer()
avancer()
avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    avancer()
    obstacle = detecter_obstacle()
ouvrir()
```

S8.1 (while-while_variante) :

Avancer vers la clé et les pots sans boucle for.

Concepts :

VAR-AF, WHI, VAR-AF, FOR-RE, VAR-AF, WHI, VAR-AF

```
avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    obstacle = detecter_obstacle()
avancer()
avancer()
avancer()
droite()
avancer()
avancer()
avancer()
avancer()
avancer()
obstacle = detecter_obstacle()
while obstacle == True :
    coup()
    avancer()
    obstacle = detecter_obstacle()
ouvrir()
```

S8.1 (while-while_variante) :

Faire l'appel de fonction directement dans la condition de la boucle.

Concepts :

WHI, FOR-RE, FOR-RE, WHI

```
avancer()
while detecter_obstacle() == True:
    coup()
for _ in range(3):
    avancer()
droite()
for _ in range(5):
    avancer()
while detecter_obstacle() == True:
    coup()
    avancer()
ouvrir()
```

S8.1 (while-while_variante) :

Avancer vers la clé sans boucle for.

Concepts :

WHI, FOR-RE, WHI

```
avancer()
while detecter_obstacle() == True:
    coup()
avancer()
avancer()
avancer()
droite()
for _ in range(5):
    avancer()
while detecter_obstacle() == True:
    coup()
    avancer()
ouvrir()
```

S8.1 (while-while_variante) :

Avancer vers les pots sans boucle for.

Concepts :

WHI, FOR-RE, WHI

```
avancer()
while detecter_obstacle() == True:
    coup()
for _ in range(3):
    avancer()
droite()
```

	7 avancer() 8 avancer() 9 avancer() 10 avancer() 11 avancer() 12 while detecter_obstacle() == True: 13 coup() 14 avancer() 15 ouvrir()
S8.1 (while-while_variante) : Avancer vers la clé et les pots sans boucle for. Concepts : WHI, WHI	1 avancer() 2 while detecter_obstacle() == True: 3 coup() 4 avancer() 5 avancer() 6 avancer() 7 droite() 8 avancer() 9 avancer() 10 avancer() 11 avancer() 12 avancer() 13 while detecter_obstacle() == True: 14 coup() 15 avancer() 16 ouvrir()
S8.2 (for-cond-for-cond) : Utiliser des boucles for ainsi que des conditionnelles pour remplacer les deux boucles whiles Concepts : FOR-RE, VAR-AF, IF, FOR-RE, FOR-RE, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 for _ in range(21): 3 obstacle = detecter_obstacle() 4 if obstacle == True: 5 coup() 6 for _ in range(3): 7 avancer() 8 droite() 9 for _ in range(5): 10 avancer() 11 for _ in range(11): 12 obstacle = detecter_obstacle() 13 if obstacle == True: 14 coup() 15 avancer() 16 else: 17 avancer() 18 ouvrir()
S8.2 (for-cond-for-cond_variante) : Avancer vers la clé sans boucle for. Concepts : FOR-RE, VAR-AF, IF, FOR-RE, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 for _ in range(15): 3 obstacle = detecter_obstacle() 4 if obstacle == True: 5 coup() 6 avancer() 7 avancer() 8 avancer()

```
droite()
for _ in range(5):
    avancer()
for _ in range(11):
    obstacle = detecter_obstacle()
    if obstacle == True:
        coup()
        avancer()
    else:
        avancer()
        ouvrir()
```

S8.2 (for-cond-for-cond_variante) :
Avancer vers les pots sans boucle for.
Concepts :
FOR-RE, VAR-AF, IF, FOR-RE, FOR-RE, VAR-AF, IF, ELSE

```
avancer()
for _ in range(15):
    obstacle = detecter_obstacle()
    if obstacle == True:
        coup()
for _ in range(3):
    avancer()
droite()
avancer()
avancer()
avancer()
avancer()
avancer()
for _ in range(11):
    obstacle = detecter_obstacle()
    if obstacle == True:
        coup()
        avancer()
    else:
        avancer()
        ouvrir()
```

S8.2 (for-cond-for-cond_variante) :
Avancer vers la clé et les pots sans boucle for.
Concepts :
FOR-RE, VAR-AF, IF, FOR-RE, VAR-AF, IF, ELSE

```
avancer()
for _ in range(15):
    obstacle = detecter_obstacle()
    if obstacle == True:
        coup()
avancer()
avancer()
avancer()
droite()
avancer()
avancer()
avancer()
avancer()
avancer()
for _ in range(11):
    obstacle = detecter_obstacle()
    if obstacle == True:
        coup()
        avancer()
```

	20 `    else:` 21 `        avancer()` 22 `        ouvrir()`
S8.3 (for-cond-while) : Utiliser une boucle for ainsi qu'une conditionnelle pour remplacer la première boucle while Concepts : FOR-RE, VAR-AF, IF, FOR-RE, FOR-RE, VAR-AF, WHI, VAR-AF	1 `avancer()` 2 `for _ in range(15):` 3 `    obstacle = detecter_obstacle()` 4 `    if obstacle == True:` 5 `        coup()` 6 `for _ in range(3):` 7 `    avancer()` 8 `droite()` 9 `for _ in range(5):` 10 `    avancer()` 11 `obstacle = detecter_obstacle()` 12 `while obstacle == True :` 13 `    coup()` 14 `    avancer()` 15 `    obstacle = detecter_obstacle()` 16 `ouvrir()`
S8.3 (for-cond-while_variante) : Avancer vers la clé sans boucle for. Concepts : FOR-RE, VAR-AF, IF, FOR-RE, VAR-AF, WHI, VAR-AF	1 `avancer()` 2 `for _ in range(15):` 3 `    obstacle = detecter_obstacle()` 4 `    if obstacle == True:` 5 `        coup()` 6 `avancer()` 7 `avancer()` 8 `avancer()` 9 `droite()` 10 `for _ in range(5):` 11 `    avancer()` 12 `obstacle = detecter_obstacle()` 13 `while obstacle == True :` 14 `    coup()` 15 `    avancer()` 16 `    obstacle = detecter_obstacle()` 17 `ouvrir()`
S8.3 (for-cond-while_variante) : Avancer vers les pots sans boucle for. Concepts : FOR-RE, VAR-AF, IF, FOR-RE, VAR-AF, WHI, VAR-AF	1 `avancer()` 2 `for _ in range(15):` 3 `    obstacle = detecter_obstacle()` 4 `    if obstacle == True:` 5 `        coup()` 6 `for _ in range(3):` 7 `    avancer()` 8 `droite()` 9 `avancer()` 10 `avancer()` 11 `avancer()` 12 `avancer()` 13 `avancer()` 14 `obstacle = detecter_obstacle()` 15 `while obstacle == True :`

	16 coup() 17 avancer() 18 obstacle = detecter_obstacle() 19 ouvrir()
S8.3 (for-cond-while_variante) : Avancer vers la clé et les pots sans boucle for. Concepts : FOR-RE, VAR-AF, IF, VAR-AF, WHI, VAR-AF	1 avancer() 2 for _ in range(15): 3 obstacle = detecter_obstacle() 4 if obstacle == True: 5 coup() 6 avancer() 7 avancer() 8 avancer() 9 droite() 10 avancer() 11 avancer() 12 avancer() 13 avancer() 14 avancer() 15 obstacle = detecter_obstacle() 16 while obstacle == True : 17 coup() 18 avancer() 19 obstacle = detecter_obstacle() 20 ouvrir()
S8.3 (for-cond-while_variante) : Utiliser une boucle for ainsi qu'une conditionnelle pour remplacer la deuxième boucle while. Concepts : VAR-AF, WHI, VAR-AF, FOR-RE, FOR-RE, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 obstacle = detecter_obstacle() 3 while obstacle == True : 4 coup() 5 obstacle = detecter_obstacle() 6 for _ in range(3): 7 avancer() 8 droite() 9 for _ in range(5): 10 avancer() 11 for _ in range(11): 12 obstacle = detecter_obstacle() 13 if obstacle == True: 14 coup() 15 avancer() 16 else: 17 avancer() 18 ouvrir()
S8.3 (for-cond-while_variante) : Avancer vers la clé sans boucle for. Concepts : VAR-AF, WHI, VAR-AF, FOR-RE, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 obstacle = detecter_obstacle() 3 while obstacle == True : 4 coup() 5 obstacle = detecter_obstacle() 6 avancer() 7 avancer() 8 avancer() 9 droite()

	10 for _ in range(5): 11 avancer() 12 for _ in range(11): 13 obstacle = detecter_obstacle() 14 if obstacle == True: 15 coup() 16 avancer() 17 else: 18 avancer() 19 ouvrir()
S8.3 (for-cond-while_variante) : Avancer vers les pots sans boucle for. Concepts : VAR-AF, WHI, VAR-AF, FOR-RE, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 obstacle = detecter_obstacle() 3 while obstacle == True : 4 coup() 5 obstacle = detecter_obstacle() 6 for _ in range(3): 7 avancer() 8 droite() 9 avancer() 10 avancer() 11 avancer() 12 avancer() 13 avancer() 14 for _ in range(11): 15 obstacle = detecter_obstacle() 16 if obstacle == True: 17 coup() 18 avancer() 19 else: 20 avancer() 21 ouvrir()
S8.3 (for-cond-while_variante) : Avancer vers la clé et les pots sans boucle for. Concepts : VAR-AF, WHI, VAR-AF, FOR-RE, VAR-AF, IF, ELSE	1 avancer() 2 obstacle = detecter_obstacle() 3 while obstacle == True : 4 coup() 5 obstacle = detecter_obstacle() 6 avancer() 7 avancer() 8 avancer() 9 droite() 10 avancer() 11 avancer() 12 avancer() 13 avancer() 14 avancer() 15 for _ in range(11): 16 obstacle = detecter_obstacle() 17 if obstacle == True: 18 coup() 19 avancer() 20 else:

	21 avancer() 22 ouvrir()
S8.4 (no-while-no-if): Essayer un parcours en particulier Concepts : ∅	1 avancer() 2 coup() 3 coup() 4 avancer() 5 avancer() 6 avancer() 7 droite() 8 avancer() 9 avancer() 10 avancer() 11 avancer() 12 avancer() 13 coup() 14 avancer() 15 coup() 16 avancer() 17 ouvrir()
S8.4 (no-while-no-if_variante): Parcours particulier avec une boucle for Concepts : FOR-RE	1 avancer() 2 for _ in range(2): 3 coup() 4 avancer() 5 avancer() 6 avancer() 7 droite() 8 avancer() 9 avancer() 10 avancer() 11 avancer() 12 avancer() 13 coup() 14 avancer() 15 coup() 16 avancer() 17 ouvrir()
S8.4 (no-while-no-if_variante): Parcours particulier avec deux boucles for Concepts : FOR-RE, FOR-RE	1 avancer() 2 for _ in range(2): 3 coup() 4 for _ in range(3): 5 avancer() 6 droite() 7 avancer() 8 avancer() 9 avancer() 10 avancer() 11 avancer() 12 coup() 13 avancer() 14 coup() 15 avancer()

	16 ouvrir()
S8.4 (no-while-no-if_variante): Parcours particulier avec trois boucles for Concepts : FOR-RE, FOR-RE, FOR-RE	1 avancer() 2 for _ in range(2): 3 coup() 4 for _ in range(3): 5 avancer() 6 droite() 7 for _ in range(5): 8 avancer() 9 coup() 10 avancer() 11 coup() 12 avancer() 13 ouvrir()
S8.4 (no-while-no-if_variante): Parcours particulier avec quatre boucles for Concepts : FOR-RE, FOR-RE, FOR-RE, FOR-RE	1 avancer() 2 for _ in range(2): 3 coup() 4 for _ in range(3): 5 avancer() 6 droite() 7 for _ in range(5): 8 avancer() 9 for _ in range(2): 10 coup() 11 avancer() 12 ouvrir()