

---

# CONCEPTION ET ANALYSE D'UNE ACTIVITÉ INSTRUMENTÉE POUR LA DÉCOUVERTE DE LA NOTION DE PILE EN SPÉCIALITÉ NSI EN TERMINALE

---

**DECLERCQ Christophe<sup>1</sup>**

INSPE de l'académie de la Réunion, LIM, IREMI

**HOARAU Sébastien<sup>2</sup>**

UFR Sciences et Technologies, Université de la Réunion, LIM, IREMI

**CHANE-LUNE Sophie<sup>3</sup>**

Lycée Mémona Hintermann-Afféjee, IREMI de la Réunion

**NATIVEL Fabrice<sup>4</sup>**

Lycée Leconte de Lisle, IREMI de la Réunion

**SCHROEDER Siméon<sup>5</sup>**

Lycée Lislet Geoffroy, IREMI de la Réunion

**Résumé.** Nous proposons une activité instrumentée pour faire découvrir aux élèves la nécessité et le fonctionnement de la pile en spécialité NSI de la classe de terminale. Cette structure de données a de nombreux usages dont le plus caractéristique est la fonction *undo* telle qu'elle est mise en œuvre dans les logiciels d'édition de textes ou d'images. Nous proposons un jeu de recherche à résoudre en solitaire : le jeu de saute-mouton. Ce jeu a des règles de déplacements très simples, mais son déroulement amène à des situations de blocage où le joueur souhaiterait revenir en arrière pour tester un autre déplacement. Une mise en œuvre instrumentée permet d'effectuer l'activité en ligne à l'aide d'un navigateur, tout en recueillant les données d'apprentissage des élèves. Nous présentons tout d'abord une rapide analyse épistémologique et curriculaire, l'activité proposée, puis le cadre théorique de cette recherche, ses hypothèses et les expérimentations menées dans trois lycées de l'académie de La Réunion. Nous discutons ensuite les hypothèses à la lumière des expérimentations menées.

**Mots-clés.** Numérique et science informatique, analyse de traces, pile, ingénierie didactique.

**Abstract.** We provide a hands-on activity designed to familiarize students with the use-cases and operations of the stack data structure within the NSI specialty in 13th grade (classe terminale in France). The stack data structure serves various purposes, one of its most notable applications being the implementation of the undo function in text or image editing software. Our scientific experiment consist of a solitaire research game known as the leapfrog game. This game features straightforward movement rules, however its progression often leads to situations where players are facing obstacles, during wich the players wants to backtrack in order to test other movement. An instrumented implementation enables the activity to be conducted online via a browser, facilitating the collection of valuable student learning data. Our presentation begins with a brief epistemological and curricular analysis, followed by an exploration of the proposed activity, the theoretical framework underpinning this research, its hypotheses, and the experiments conducted in three high schools within the Réunion academy. In conclusion, we engage in a discussion evaluating the hypotheses in light of the experiments conducted.

**Keywords.** Computer science, high school, stack data structure, learning analytics.

---

<sup>1</sup> christophe.declercq@univ-reunion.fr

<sup>2</sup> seb.hoarau@univ-reunion.fr

<sup>3</sup> Sophie.Chane-Lune@ac-reunion.fr

<sup>4</sup> Fabrice-Bernard.Nativel@ac-reunion.fr

<sup>5</sup> Simeon.Schroeder@ac-reunion.fr

## Introduction

La manipulation de structures de données - dont la pile - est un point difficile du programme de la spécialité Numérique et Science Informatique (NSI) en terminale (MENJ, 2019). L'idée initiale de ce travail a été de proposer une situation ludique, où la nécessité de gérer un historique des positions de jeu vient de la difficulté à résoudre le jeu de manière directe.

Une première expérimentation auprès de 14 élèves a permis d'obtenir de premiers résultats mais aussi de montrer quelques faiblesses de l'activité et de l'outil d'analyse des traces d'activité des élèves. Une deuxième version plus robuste a été construite à partir des observations précédentes, puis a été de nouveau expérimentée dans deux lycées auprès de 38 élèves de classe de terminale. Nous présentons d'abord une analyse des notions en jeu dans la littérature et dans les programmes scolaires, nos choix de mise en œuvre puis l'expérimentation et ses résultats.

L'activité est disponible en ligne à l'adresse :

<https://iremi974.gitlab.io/python-de-la-fournaise/sautemouton.html>

## Analyse épistémologique et curriculaire

La partie « Structures de données » du programme de terminale de la spécialité NSI comprend les contenus suivants, concernant la pile en tant que structure de données (extrait).

Contenus	Capacités attendues	Commentaires
Structures de données, interface et implémentation.	Spécifier une structure de données par son interface. Distinguer interface et implémentation. Écrire plusieurs implémentations d'une même structure de données.	L'abstraction des structures de données est introduite après plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deux piles).
Listes, piles, files : structures linéaires.	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser.	On distingue les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files.

Il est à noter qu'il n'est pas fait référence au concept de « type abstrait de données » enseigné au niveau universitaire, ce que nous interprétons comme une nécessité de « transposition didactique » du savoir savant (Chevallard, 1991).

Le savoir savant concernant la pile en tant que type abstrait de données, tel qu'il est enseigné au niveau universitaire, figure par exemple à la page 74 de l'ouvrage (Froidevaux et al., 1990)

sorte Pile

utilise Booléen, Élément

opérations :

*pile-vide* :  $\rightarrow$  Booléen

*empiler* : Pile  $\times$  Élément  $\rightarrow$  Pile

*dépiler* : Pile  $\rightarrow$  Élément

*sommet* : Pile  $\rightarrow$  Élément

*est-vide* : Pile  $\rightarrow$  Booléen

Sachant qu'au niveau terminale, il est attendu seulement de distinguer interface et implémentation et de spécifier une structure de données par son interface, nous spécifions l'interface d'une pile par les fonctions suivantes :

```
pile_vide() : renvoie une pile vide  
empiler(element, pile) : modifie pile en ajoutant element à son sommet  
depiler(pile) : modifie pile en enlevant son sommet et renvoie cet élément  
est_vide(pile) : renvoie True si et seulement si pile est vide.
```

Nous choisissons de spécifier les interfaces en Python, car aucun autre formalisme ni théorie de types n'est au programme du lycée. Nous choisissons aussi de donner des spécifications pour le paradigme impératif (ou objet), car les implémentations sont suggérées dans ces paradigmes : « Le paradigme de la programmation objet peut être utilisé pour réaliser des implémentations effectives des structures de données, même si ce n'est pas la seule façon de procéder. »

Avec le paradigme objet, nous donnons la spécification suivante d'une classe `Pile` :

```
Pile() : crée et renvoie une pile vide  
pile.empiler(element) : modifie pile en ajoutant element à son sommet  
pile.depiler() : modifie pile en enlevant son sommet et renvoie cet élément  
pile.est_vide() : renvoie True si et seulement si pile est vide.
```

Sachant que la structure de données ne doit être introduite qu'après « plusieurs implémentations d'une structure simple », la première activité peut consister à découvrir l'idée de pile à partir de sa politique d'accès spécifique : dernier entré – premier sorti.

Nous avons choisi de faire implémenter aux élèves une pile, sans la nommer au préalable, et donc sans nommer ses méthodes, à partir des notions disponibles pour eux, à savoir les tableaux Python vus en première, complétés par les méthodes permettant de les revoir comme des listes en début de terminale.

L'activité de découverte que nous présentons et analysons dans cet article se situe au tout début de la progression suivante :

- Découverte de la notion de pile à partir d'une situation de type UNDO.
- Découverte de la notion de pile à partir d'une autre situation (évaluer une expression en notation polonaise inverse, vérifier le bon parenthésage...).
- Institutionnalisation de la notion de pile, par la spécification de son interface.
- Implémentation de la structure de données avec des listes Python.
- Implémentation de la structure de données en programmation objet.

Nous ne décrivons dans la suite que la première activité pour laquelle nous avons proposé une instrumentation spécifique.

Nous avons proposé par ailleurs une autre activité instrumentée pour permettre aux élèves de réutiliser une classe `Pile` fournie, pour concevoir et programmer un algorithme de construction de labyrinthe, nécessitant de mémoriser le chemin parcouru via une pile.

Cette dernière activité est disponible à l'adresse :

```
https://iremi974.gitlab.io/python-de-la-fournaise/labyrinthe.html
```

## L'activité initiale

### Règles et déroulement du jeu de saute-mouton

Le jeu comporte sept cases alignées numérotées de 0 à 6, trois jetons moutons blancs et trois jetons moutons noirs. Les moutons blancs (resp. noirs) peuvent se déplacer vers la droite (resp. gauche) vers une case vide ou en sautant par dessus un mouton noir (resp. blanc). L'objectif est d'inverser les moutons blancs et les moutons noirs (voir figure 1).


## Activité : Jeu de saute-mouton

**Règle du jeu :**  
Les moutons blancs peuvent se déplacer vers la droite vers une case vide ou en sautant par dessus un mouton noir.  
Les moutons noirs peuvent se déplacer vers la gauche vers une case vide ou en sautant par dessus un mouton blanc.  
Emmène tous les moutons blancs à droite et tous les moutons noirs à gauche.

Si tu es bloqué, essaie le bouton **UNDO** ! Si son fonctionnement ne te convient pas, modifie le programme pour que le bouton **UNDO** permette de revenir à la configuration précédente.

```
jeu = ['blanc', 'blanc', 'blanc', 'vide', 'noir', 'noir', 'noir']
historique = []

def saute(i):
    if jeu[i]=='blanc' and i<6 and jeu[i+1]=='vide':
        jeu[i], jeu[i+1] = 'vide', 'blanc'
    elif jeu[i]=='blanc' and i<5 and jeu[i+1]=='noir' and jeu[i+2]=='vide':
        jeu[i], jeu[i+2] = 'vide', 'blanc'
    elif jeu[i]=='noir' and i>0 and jeu[i-1]=='vide':
        jeu[i], jeu[i-1] = 'vide', 'noir'
    elif jeu[i]=='noir' and i>1 and jeu[i-1]=='blanc' and jeu[i-2]=='vide':
        jeu[i], jeu[i-2] = 'vide', 'noir'
def undo():
    global jeu
    jeu = ['blanc', 'blanc', 'blanc', 'vide', 'noir', 'noir', 'noir']
```



UNDO

Figure 1 : Interface initiale du jeu de saute-mouton.

L'interface proposée montre un code d'initialisation non modifiable (variables jeu et historique) et le code modifiable de deux fonctions. La fonction saute est appelée à chaque clic sur une case du jeu et permet de déplacer le mouton qui se trouve sur cette case si le déplacement est autorisé. La fonction undo est appelée à chaque clic sur le bouton UNDO et est programmée de manière naïve en ramenant directement à la position initiale.

Notons que l'ensemble du code peut aussi être recopié dans un éditeur de programme et testé simplement avec la console python, par exemple avec la suite d'appels suivante :

```
saute(2)
saute(4)
saute(3)
saute(1)
jeu
undo()
```

À l'issue des quatre premiers mouvements, nous arrivons à une situation de blocage caractéristique avec la variable `jeu` à `['blanc', 'vide', 'noir', 'blanc', 'blanc', 'noir', 'noir']` où deux moutons blancs font face à deux moutons noirs sans qu'aucun ne puisse plus avancer.

C'est la situation déclenchante qui va permettre de réfléchir à la nécessité d'un retour arrière avec la fonction `undo`. L'interface et les parties de code qui ne sont pas montrées à l'élève ne sont utiles que pour fournir un retour instrumental plus élaboré. Après chaque mouvement, la visualisation graphique est recalculée à partir des nouvelles valeurs contenues dans le tableau `jeu`.

### Analyse a priori de l'activité

Le bouton `UNDO`, tel qu'il est programmé initialement, a un comportement trop radical puisqu'il ramène à la configuration initiale. La réflexion sur le besoin de revenir seulement à la configuration précédente devrait amener les élèves à construire la nécessité d'une variable mémorisant cette configuration précédente.

Dans un premier temps, une variable peut effectivement suffire, reste à savoir quand l'initialiser - avant de jouer un coup - et quand utiliser la valeur mémorisée - au clic sur le bouton `UNDO` - pour remettre dans le tableau `jeu`, la configuration mémorisée.

Cette première étape peut être utile dans la résolution du problème initial et amènera alors à un obstacle quand, lors d'une partie de jeu, le joueur s'apercevra qu'il a besoin de revenir en arrière de plus d'un mouvement.

Se pose alors inductivement la question de la mémorisation de deux, trois puis de tous les mouvements depuis le début du jeu. C'est là que les élèves ont besoin de réfléchir sur le moment où ils risquent d'avoir besoin des valeurs mémorisées. Le premier appel de la fonction `undo` a besoin de la dernière valeur mémorisée, le second de la précédente...

La présence dans le programme fourni, de la variable `historique`, initialisée en tant que liste vide, est prévue pour inciter à l'usage de cette liste pour y mémoriser les configurations successives du jeu. Il ne reste plus qu'à fournir aux élèves les méthodes d'accès pour ajouter en fin de liste `append`, et retirer un élément à cette même fin de liste `pop`, pour leur permettre de réussir l'activité.

L'institutionnalisation prendra soin de préciser qu'on appelle « pile » cette manière particulière de mémoriser une série de données, lorsque la dernière à ajouter - à empiler - doit être la première à retirer - à dépiler.

Plusieurs solutions peuvent être imaginées pour enregistrer l'historique, l'historique des coups joués - [2, 4, 3, 1] dans notre exemple - ou l'historique des configurations du jeu. Chaque solution a ses avantages et ses inconvénients : l'historique des coups est une structure de données plus élémentaire mais nécessite le calcul des configurations ; l'historique des configurations est une liste (pile) de tableaux, ce qui nécessite en Python des précautions pour éviter les alias.

Une aide peut être fournie en précisant qu'il convient de faire une copie dans l'historique, de la configuration de jeu actuelle, pour éviter que cette mémorisation ne soit modifiée lors de la suite du déroulement du jeu.

## Correction de l'activité

Parmi les solutions possibles, nous proposons la suivante, en tant que corrigé :

```
def saute(i):
    historique.append(jeu.copy())
    if jeu[i]=='blanc' and i<6 and jeu[i+1]=='vide':
        jeu[i], jeu[i+1] = 'vide', 'blanc'
    elif jeu[i]=='blanc' and i<5 and jeu[i+1]=='noir' and jeu[i+2]=='vide':
        jeu[i], jeu[i+2] = 'vide', 'blanc'
    elif jeu[i]=='noir' and i>0 and jeu[i-1]=='vide':
        jeu[i], jeu[i-1] = 'vide', 'noir'
    elif jeu[i]=='noir' and i>1 and jeu[i-1]=='blanc' and jeu[i-2]=='vide':
        jeu[i], jeu[i-2] = 'vide', 'noir'
    else :
        historique.pop()
def undo():
    global jeu
    jeu = historique.pop()
```

Nous avons choisi ici de mémoriser les configurations et de réaliser la pile par les méthodes `append` et `pop` des listes. Le choix d'empiler systématiquement la configuration précédente, avant de tester si le coup est jouable, peut amener à empiler une configuration identique à la précédente. C'est la raison de la correction proposée ici, consistant à dépiler dans le `else` une configuration empilée inutilement. Une solution plus élémentaire consisterait à empiler uniquement dans les alternatives quand le coup est jouable. Cette réflexion est marginale et n'est bien sûr pas exigible des élèves. Si on omet le `else`, l'historique sera seulement un peu plus long (avec des configurations répétées) de même que la suite des appels à la fonction `undo`.

L'essentiel réside dans la compréhension du mécanisme : si la mémorisation de l'historique a bien été effectuée lors du jeu, la fonction `undo` consiste à dépiler la dernière configuration mémorisée pour la remettre en jeu.

### Remarque sur l'utilisation de variables globales et sur les données mutables

L'indication de variable globale pour `jeu` est fournie pour éviter un problème dû à la spécificité des variables mutables en Python associée à la déclaration automatique des variables à leur première affectation. Si on l'omet, en particulier dans la fonction `undo`, c'est une nouvelle variable locale qui est créée sans rapport avec la variable globale définie à l'initialisation et utilisée dans la fonction `saute`. Cette organisation logicielle est aussi adaptée aux paradigmes de programmation choisis. Dans l'environnement logiciel en ligne proposé, les fonctions `saute` et `undo`, sont utilisées en programmation événementielle, chacune étant associée à un bouton ou à un objet (case avec un mouton) de l'interface. La programmation événementielle incite à la manipulation d'objets globaux via les éléments de l'interface. Ces mêmes fonctions peuvent aussi être utilisées en programmation impérative directement depuis une console Python.

Le choix d'utiliser des variables globales, dans la partie de code fournie par l'enseignant, en exception aux consignes données habituellement aux élèves, est volontaire pour éviter d'aborder deux difficultés simultanément, la découverte d'une nouvelle structure de données et le passage de paramètres avec des structures de données mutables en Python. Ce dernier problème n'a en effet rien à voir avec la notion de pile et peut être abordé séparément. Le problème de l'aliasing peut survenir lors de la construction de listes de listes. Pour l'éviter, l'aide fournie recommande de recopier le jeu avant de le placer dans l'historique. En cas d'oubli de copie, le retour instrumental fourni lors de la seconde expérimentation, permet de visualiser le caractère erroné des informations mémorisées.

## **Protocole de recherche et résultats de la première expérimentation**

### **Problématique de recherche**

Nous nous situons dans le double cadre de la théorie des situations didactiques (Brousseau, 1998) et de l'approche instrumentale (Rabardel, 1995) (Nijimbere, 2013) pour analyser la situation et l'activité des élèves. Par rapport à l'utilisation de jeux pour l'apprentissage, il convient de se positionner par rapport au risque de « détruire le ludique au profit du sérieux éducatif » (Brougère, 2018). Dans le domaine de l'apprentissage de l'informatique, une posture spécifique est possible, qui consiste non pas à « ludifier » un apprentissage, mais à prendre un jeu en tant qu'objet d'étude et à étudier ses propriétés intrinsèques pour en programmer une version informatique ou une stratégie gagnante. C'est dans cette perspective que notre travail se situe dans la lignée de celui de Meyer et Modeste (Meyer & Modeste, 2020) (Meyer & Modeste, 2022), avec comme principale différence que nous proposons une situation instrumentée.

Notre approche se veut aussi résolument constructiviste. Il n'est pas question pour nous de révéler d'emblée le savoir savant concernant un type abstrait pile, mais de faire découvrir aux élèves l'essence de cette structure de données à partir de sa politique d'accès spécifique : dernier entré – premier sorti, dans une situation porteuse de sens.

Nos hypothèses de recherche sont les suivantes :

- Une situation ludique, de type jeu de solitaire, est de nature à engager les élèves dans l'activité.
- La difficulté de trouver directement la solution encourage à améliorer le jeu en programmant le bouton UNDO.
- La situation est signifiante par rapport à l'usage de la pile et est candidate pour être considérée comme une situation fondamentale.

### **Recueil des traces d'exécution**

Pour pouvoir mesurer, les temps de jeu et les temps de programmation, enregistrer les réussites des élèves par rapport au jeu et par rapport à la programmation et tenter de comprendre a posteriori leur cheminement, nous avons décidé d'enregistrer toutes les interactions de l'élève avec l'activité proposée.

Les logs d'activité consistent en un horodatage de toutes les actions sur l'interface du jeu (voir un exemple en annexe 2). Les actions comprennent d'abord les déplacements de moutons qui sont notées coup 0 à coup 6 et les actions sur le bouton UNDO. À chaque action est mémorisée son heure, la configuration du jeu, la valeur de la variable historique et si le programme a été modifié depuis la dernière action, le texte du programme. Pour pouvoir mesurer le temps de jeu et le temps de programmation, nous considérons comme temps de programmation, tous les temps précédant un moment où le programme a été modifié. À la lecture des logs, nous constatons en effet pendant les phases de jeu, des actions très rapprochées, puis des longues périodes sans interaction avant modification du programme que nous assimilons à des temps de réflexion et de programmation.

### **Analyse de la première expérimentation et préconisations**

L'activité a été testée pour la première fois avec une classe de spécialité NSI de terminale du lycée Lislet Geoffroy au mois de mai 2022. Le groupe comportait 14 élèves.

L'activité s'est déroulée selon le protocole décrit en annexe 1. L'enseignant a laissé les élèves en autonomie et a récolté les logs d'activité à l'issue de la séance qui a duré environ 50mn. Les résultats bruts de l'analyse des logs figurent en annexe 3.

Nous constatons que les temps d'activité des élèves varient entre 31 mn et 55 mn. Les temps les plus courts correspondent à une mise en route tardive, les plus longs à des débordements sur la pause inter-cours. Nous excluons ici de l'analyse l'élève LLG006 dont une partie de la session n'a pas été enregistrée suite à un arrêt de l'ordinateur.

Nous relevons une variation importante des temps de jeu et de programmation. Le temps de jeu varie de 5 mn pour l'élève LLG002 à 40 mn pour l'élève LLG007. Le temps de programmation varie de 3 mn pour l'élève LLG007 à 46 mn pour l'élève LLG101.

9 élèves sur 14 ont réussi le jeu en un temps variant entre 1 mn et 17 mn. Les 5 qui ont arrêté de jouer avant de gagner, figurent aussi parmi ceux qui ont passé le plus de temps à programmer. Nous pouvons supposer que pour eux la difficulté du jeu ait constitué une motivation pour programmer la fonction undo. Ceci est cependant contredit par le témoignage d'un élève (ayant réussi) : « le jeu est trop facile, pas besoin de programmer ».

L'engagement des élèves dans l'activité peut être évalué par le nombre d'interactions avec le système qui varie de 167 coups joués à 510.

### ***Analyse de la partie programmation***

Concernant la programmation, nous avons analysé principalement le dernier programme réalisé par chacun et codifié en 5 catégories :

- pas de programmation (pas de différence notable avec le programme donné) : 5 élèves
- programmation incohérente (impossibilité de comprendre l'intention du programmeur) : 4 élèves
- historique un coup (enregistrement du seul coup précédent) : 2 élèves
- historique sans copie du jeu (enregistrement de l'historique avec append mais sans copie du jeu ce qui, avec les alias, donne un historique comportant n fois le même jeu) : 2 élèves
- réussite : 1 élève.

Ce résultat plutôt décevant – 1 seule réussite et 4 réussites partielles – nous a amené à revoir certaines caractéristiques de l'activité. En particulier pour les 4 ayant seulement réussi partiellement, l'analyse détaillée des logs montre un début de construction de l'historique comportant des erreurs dues à l'absence de copie ou à un mauvais choix de l'information à mémoriser. Le problème est que ces élèves n'ont pas « vu » leurs erreurs, ce qui peut expliquer qu'ils ne les aient pas corrigées. Cette absence de « retour instrumental » est préjudiciable aux apprentissages, car l'élève ne peut pas apprendre de ses erreurs. Nous en déduisons que l'activité devrait visualiser en permanence la valeur de la variable historique, ce qui a été fait dans la version suivante. Nous notons aussi, en particulier dans les productions jugées incohérentes, un usage hasardeux des listes et des méthodes qui s'y appliquent. Nous proposons de fournir une aide comportant des explications sur les méthodes pouvant être utilisées.

### ***Analyse de l'ergonomie de l'interface et du recueil de données***

Au début du jeu, la fonction undo ramène à la position initiale, ce qui a été largement utilisé en particulier par les joueurs les plus téméraires. Mais dès que la fonction undo est modifiée, il n'y a plus d'autre moyen pour revenir à la situation initiale que de recharger la page. Cette possibilité a été utilisée et entraîne une réinitialisation du programme de l'élève à sa valeur par défaut.

Ceci constitue un défaut de l'interface, qui a fait perdre du temps aux élèves qui ont testé leur programme en cours de modification et se sont ensuite retrouvés bloqués.

Ceci entraîne de plus une difficulté supplémentaire pour l'analyse de l'activité à partir des logs, car ce qui se passe en dehors du navigateur – copie et modification du programme dans un éditeur - ne peut être capturé par les logs.



## Seconde campagne d'expérimentations

### Une nouvelle situation améliorée

Pour répondre aux difficultés rencontrées par les élèves lors de cette première expérimentation, nous avons décidé d'améliorer le retour instrumental en affichant en permanence l'historique (pour suggérer l'idée de la pile, nous avons choisi de la figurer de bas en haut en notant en marge les indices), et de fournir une aide spécifique sur les méthodes applicables aux listes avec un avertissement concernant la copie (voir annexe 4).

### Activité : Jeu de saute-mouton

**Joue :** Emmène tous les moutons blancs à droite et tous les moutons noirs à gauche ! Les moutons blancs peuvent se déplacer vers la droite vers une case vide ou en sautant par dessus un mouton noir. Les moutons noirs peuvent se déplacer vers la gauche vers une case vide ou en sautant par dessus un mouton blanc.

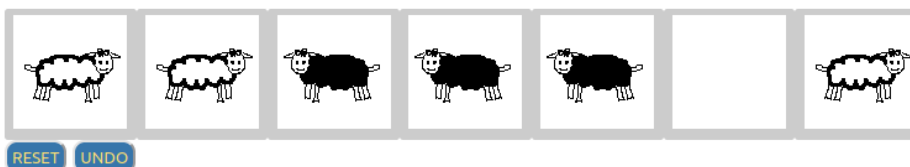
**Programme :** Le fonctionnement du bouton **UNDO** n'est pas satisfaisant. Modifie le programme pour qu'il permette de revenir à la configuration précédente.

**Enregistre :** A la fin de la session, [enregistre ton activité](#).

```
jeu = ["B", "B", "B", "O", "N", "N", "N"]  
historique = []
```

```
def saute(case):  
    """ Fait sauter le mouton présent dans la case en modifiant  
    le jeu et l'historique, enregistrés en variables globales  
    """  
    global jeu, historique  
    historique.append(jeu.copy())  
    if jeu[case]=="B" and case<6 and jeu[case+1]=="O":  
        jeu[case], jeu[case+1] = "O", "B"  
    elif jeu[case]=="B" and case<5 and jeu[case+1]=="N" and jeu[case+2]=="O":  
        jeu[case], jeu[case+2] = "O", "B"  
    elif jeu[case]=="N" and case>0 and jeu[case-1]=="O":  
        jeu[case], jeu[case-1] = "O", "N"  
    elif jeu[case]=="N" and case>1 and jeu[case-1]=="B" and jeu[case-2]=="O":  
        jeu[case], jeu[case-2] = "O", "N"  
    else :  
        historique.pop()  
  
def undo():  
    """ Remet la variable globale jeu à la dernière position enregistrée  
    en utilisant la variable globale historique
```

```
6 ['B', 'B', 'N', 'N', 'O', 'N', 'B']  
5 ['B', 'B', 'N', 'N', 'B', 'N', 'O']  
4 ['B', 'B', 'N', 'N', 'B', 'O', 'N']  
3 ['B', 'B', 'N', 'O', 'B', 'N', 'N']  
2 ['B', 'B', 'N', 'B', 'O', 'N', 'N']  
1 ['B', 'B', 'O', 'B', 'N', 'N', 'N']  
0 ['B', 'B', 'B', 'O', 'N', 'N', 'N']
```



**Historique**

[Aide pour gérer l'historique](#)

IREMI - LIM - INSPE de l'académie de la Réunion - 2022

Figure 2 : Interface du jeu de saute-mouton avec visualisation de l'historique

Concernant l'ergonomie, nous avons ajouté un bouton **RESET** dont l'effet - non modifiable par l'utilisateur - est au début identique à celui du bouton **UNDO**, à savoir remettre le jeu dans sa position initiale et vider l'historique, sans modifier le programme de l'élève.

Pour mieux comprendre l'activité des élèves, nous avons aussi ajouté l'enregistrement des rechargements de la page (action **LOAD**) ainsi que des réinitialisations (action **RESET**). Nous obtenons ainsi des logs horodatant toutes les actions réalisées sur l'interface : coup 0, ... coup 6, **LOAD**, **RESET**, **UNDO**.

La visualisation de l'historique en partie droite de la page permet à l'élève de repérer toute construction erronée qui ne permettrait pas à la fonction **undo** de faire son travail. Nous avons aussi modifié l'encodage du jeu, remplaçant respectivement blanc, vide et noir par **B**, **O** et **N**, afin d'alléger la visualisation de l'historique.

## Expérimentation

La nouvelle situation a été expérimentée dans deux lycées de l'académie, le lycée Mémona Hintermann (LMH) et le lycée Le Verger (LLV) le 20 septembre 2022. Les expérimentations ont été menées par les enseignant.e.s titulaires de leur classe et co-auteurs de l'article, avec l'observation d'un chercheur pour le lycée Mémona Hintermann, où l'activité a été menée successivement avec les deux demi-groupes de la classe comptant respectivement 13 et 10 élèves. Pour le lycée Le Verger, l'activité a concerné les 15 élèves de la classe. L'ensemble des logs d'activité ont pu être enregistrés à l'issue des séances. Les résultats bruts de l'analyse des logs figurent en annexe 5. Par rapport à la première expérimentation, plusieurs informations supplémentaires ont pu être calculées :

- Les comptages des nombres d'actions LOAD, RESET et UNDO au total (variables nL, nr et nu) et avant de gagner le jeu (variables nlg, nrg et nug).
- Le temps passé avant de débiter la programmation (variable deb\_prog)

Le codage de la qualité de la programmation (variable code) a été réalisé lors d'une séance collective d'analyse des résultats regroupant enseignants et chercheurs. Pour chaque élève a tout d'abord été noté une appréciation sur la gestion de l'historique, une appréciation sur la programmation de la fonction undo et une appréciation globale sur la programmation. Nous avons ensuite repris l'encodage 0 à 4 de la partie 2 pour résumer ces appréciations.

## Analyse des résultats

### Analyse quantitative

#### Engagement des élèves dans l'activité

Pour tenter de répondre à la première hypothèse sur l'engagement des élèves dans l'activité par le jeu, nous examinons d'abord les différentes métriques nous permettant de mesurer les temps et les nombres d'actions.

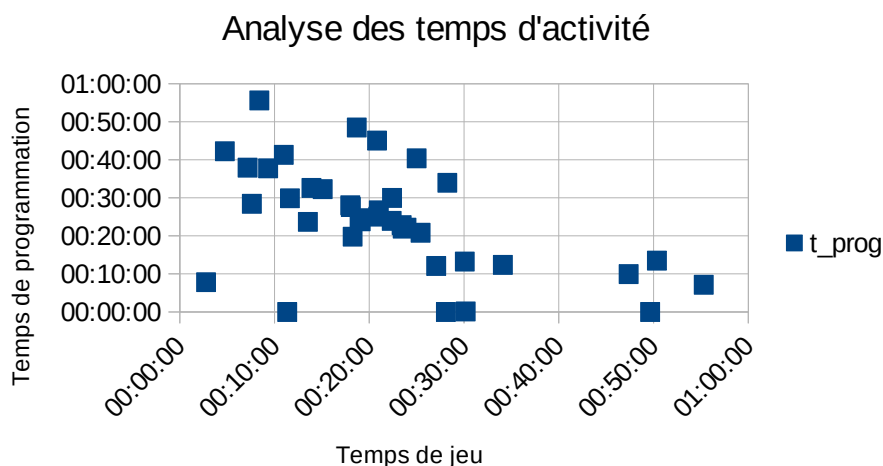


Figure 3 - Analyse des temps d'activité

Le temps d'activité moyen est de 46 mn (53 mn pour le lycée Le Verger et 41 mn pour le lycée Mémona Hintermann) et correspond aux durées des séances prévues par les enseignants. Les deux seuls élèves ayant passé un temps d'activité beaucoup plus faible sont les élèves LLV05 ayant cessé toute activité au bout de 11 mn et l'élève LMH202 ayant réussi la tâche de programmation au bout de 10 mn.

Concernant le rapport entre temps de jeu et temps de programmation, les valeurs moyennes sont très proches – 22 mn de jeu et 24 mn de programmation – mais cachent de fortes disparités.

Le graphique de la figure 3 représente les élèves en fonction de leur temps de jeu (en abscisse) et de leur temps de programmation (en ordonnée). Nous repérons les deux élèves déjà cités en bas à gauche. Seuls 4 élèves n’ont pas programmé. La corrélation linéaire ne fait qu’exprimer que la somme des temps de jeu et de programmation, correspond au temps total d’activité. Ceux qui ont joué plus longtemps (4 élèves ont joué plus de 45 mn) ont moins programmé et réciproquement. La tendance centrale correspond au groupe d’élèves qui a joué entre 15 et 25 mn et programmé entre 20 et 30 mn. Seuls 10 élèves sur 38 ont passé plus de 2/3 de leur temps à jouer. Donc presque les 3/4 des élèves ont consacré plus d’un tiers du temps d’activité à la programmation.

Le graphique de la figure 4 représente le nombre d’interactions de chaque élève avec le jeu en distinguant les coups correspondant à un déplacement d’un mouton et les actions LOAD, RESET ou UNDO.

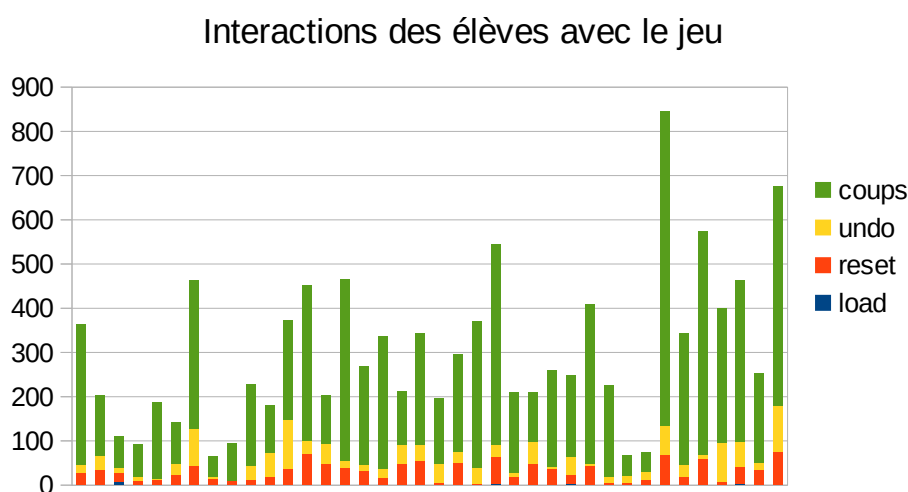


Figure 4 – Analyse des interactions des élèves

En moyenne, les élèves ont eu en peu plus de 300 interactions avec le jeu. Le plus acharné (élève LMH204) en a eu plus de 800 en 23 mn de jeu.

Ces deux types de mesures – temps et interactions – nous permettent d’attester d’un engagement fort des élèves dans l’activité et d’un partage du temps variable entre jeu et programmation.

### **Réussite du jeu**

31 élèves sur 38 ont réussi le jeu en un temps moyen de 12 mn. Le plus rapide (élève LMH112) a gagné en 39 s après 23 coups et 1 RESET, ce qui est remarquable sachant que le jeu nécessite 15 coups pour gagner. Les élèves les plus patients LMH108 et LMH210 n’ont pas hésité à réinitialiser le jeu 57 fois avant de gagner en respectivement 26 mn et 404 coups et 24 mn et 443 coups.

Nous notons l’élève LLV07 qui a utilisé 46 fois le bouton UNDO et a fini par gagner le jeu après l’avoir correctement programmé au bout de 33 mn.

### **Nécessité de la programmation**

Pour tenter d’approcher la nécessité ressentie ou non par les élèves de programmer, nous avons compté le nombre d’élèves ayant commencé à programmer avant la réussite du jeu. 8 élèves ont commencé à programmer au moment de leur réussite au jeu. 12 élèves ont commencé à programmer avant d’avoir gagné puis ont gagné par la suite. Il y a donc un nombre

significatif d'élèves qui ont ressenti cette nécessité avant d'avoir réussi le jeu.

### **Réussite de la programmation**

Sur les 38 élèves, 4 n'ont pas fait de programmation, 13 ont fait des tentatives infructueuses ou dont l'intention n'a pas été comprise, 11 ont programmé l'historique plus ou moins correctement mais pas la fonction undo et 10 élèves ont réussi l'activité de programmation de la fonction undo.

Nous notons en particulier qu'aucun élève n'a programmé un historique à un coup, comme cela avait été observé lors de la première expérimentation.

### **Analyse qualitative**

Les verbatims relevés par les enseignant.e.s au cours des séances ou après permettent de compléter et d'interpréter l'analyse. L'engagement de presque tous les élèves au-delà de leur participation habituelle à des activités de programmation a été relevée par les deux enseignants, ce qui semble confirmer notre hypothèse sur l'intérêt du jeu.

Lors de la séance d'institutionnalisation sur la pile qui a suivi cette activité au lycée Le Verger, un élève s'est exclamé : « en fait, le *undo* du saute-mouton, c'est comme le CTRL-Z », montrant qu'il avait compris cet usage fondamental de la pile.

La maîtrise des concepts de programmation permettant ou non de réaliser cette activité doit aussi nous interroger. Au bout de 30 mn d'activité, l'élève LMH112 s'est exprimé : « Ah, on peut mettre des listes dans des listes ! » et a réussi la programmation 5 mn après, son intervention montrant ainsi l'obstacle qu'il avait dû dépasser pour réussir.

L'analyse des programmes des élèves montre que la quasi-totalité ont choisi d'empiler dans l'historique les positions de jeu, à l'instar de l'élève LMH210 qui déclare après 30 mn d'activité : « je vais enregistrer dans l'historique la version précédente du jeu ».

Seul un élève (LMH202 déjà signalé pour sa réussite rapide à l'activité de programmation) s'est distingué en enregistrant sous forme de chaîne, le code python permettant d'inverser le dernier mouvement puis en l'utilisant via la fonction *exec* lors de l'appel à la fonction *undo*.

### **Analyse de la nouvelle interface de jeu**

Parmi les 38 élèves, seuls 2 ont été amenés à recharger une fois la page (action *LOAD*) au cours de l'activité, alors que ces rechargements intempestifs avaient été nombreux lors de la première expérimentation. Nous en déduisons qu'il était pertinent d'ajouter le bouton *RESET* qui a été largement utilisé pour réinitialiser le jeu sans perdre son programme.

L'intérêt de l'ajout de la visualisation de l'historique est confirmé par les résultats obtenus. En effet, plus aucun élève ne s'est contenté d'un historique à une position (code 2). Nous en déduisons que c'est le retour instrumental qui a permis aux élèves de voir leurs erreurs lors de la construction de l'historique, et aussi de corriger un historique où toutes les positions de jeu sont les mêmes à cause de l'absence de copie.

### **Une variable didactique à étudier**

Concernant l'impact de la difficulté du jeu sur la nécessité de la programmation de la fonction *undo*, il est difficile de conclure définitivement, sachant que les logs d'activité montrent un aller-retour entre jeu et programmation. L'engagement des élèves dans la programmation montre au moins qu'ils ont compris l'intérêt de programmer cette fonctionnalité. Une variable didactique de la situation aurait pu être utilisée pour forcer cette nécessité. La variation de la taille du jeu – passer de 3 à 4 ou 5 moutons de chaque couleur – permet de faire varier la difficulté. Une autre manière consiste à passer à la version à 2 dimensions, où 2 groupes de 8 moutons doivent se croiser via une case centrale unique, les moutons blancs allant vers le bas ou vers la droite et les

moutons noirs vers le haut ou vers la gauche. Cette dernière version a été programmée mais n'a pas encore été expérimentée. Elle est disponible à l'adresse :

<https://iremi974.gitlab.io/python-de-la-fournaise/sautemouton2d.html>

## Conclusion et perspectives

Nous avons proposé une activité ludique – le jeu de saute-mouton – pour faire découvrir aux élèves la pile en tant que structure de donnée inscrite au programme de la spécialité NSI en classe de terminale. Après une première phase d'expérimentation ayant permis d'améliorer l'activité et le recueil de données, nous avons conduit une seconde expérimentation permettant de mettre à l'épreuve nos hypothèses.

Au vu des résultats, nous pensons pouvoir conclure positivement aux deux premières hypothèses quant à l'intérêt d'une activité ludique pour l'engagement des élèves dans l'activité et par rapport à l'impact de la difficulté du jeu sur la nécessité de programmer pour l'améliorer.

La situation nous semble bien signifiante par rapport à l'usage de la pile, mais elle partage cette propriété avec toute activité faisant usage du CTRL-Z.

Il serait donc finalement présomptueux de considérer cette activité particulière comme une situation fondamentale pour l'apprentissage de la pile.

Parmi les perspectives de cette recherche, nous pensons développer dans deux directions complémentaires :

- décliner la situation à d'autres jeux de type solitaire, dont des casse-têtes plus difficiles où le CTRL-Z est indispensable.
- continuer à développer l'enregistrement automatique de l'activité des élèves lors d'activités de programmation en ligne, et développer des méthodes plus génériques d'analyse des logs à partir de cette première expérimentation.

Concernant la modalité de cette recherche-action, nous remercions l'IREMI de l'université de La Réunion, qui nous a permis de rassembler un groupe d'enseignant.e.s et de chercheurs autour de cette problématique.

## Références bibliographiques

- Brougère, G. (2018). *Jeu et éducation* (Numéro 3926, p. 28). PUF. <https://sorbonne-paris-nord.hal.science/hal-03628855>
- Brousseau, G. (1998). *Théorie des situations didactiques*. La pensée Sauvage.
- Chevallard, Y. (1991). *La transposition didactique, du savoir savant au savoir enseigné*. La pensée Sauvage.
- Froidevaux, C., Gaudel, M.-C., & Soria, M. (1990). *Types de données et algorithmes* (Mc Graw-Hill).
- MENJ. (2019). *Programme de l'enseignement de spécialité de numérique et sciences informatiques de la classe terminale de la voie générale*. Ministère de l'Education Nationale et de la Jeunesse. <https://www.education.gouv.fr/bo/19/Special8/MENE1921247A.htm>
- Meyer, A., & Modeste, S. (2022). Situation didactique autour d'un jeu de recherche : Expérimentation en classes de NSI. *L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ? Actes du colloque DIDAPRO 9*, 100. <https://hal.archives-ouvertes.fr/hal-03697943>
- Meyer, A., & Modeste, S. (2020, février). Analyse didactique d'un jeu de recherche : Vers une

situation fondamentale pour la complexité d'algorithmes et de problèmes. *Didapro 8 – DidaSTIC, L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation.* <https://hal.archives-ouvertes.fr/hal-03014372>

Nijimbere, C. (2013). Approche instrumentale et didactiques : Apports de Pierre Rabardel. *Adjectif.net.*

Rabardel, P. (1995). *Les hommes et les technologies; approche cognitive des instruments contemporains.* Armand Colin. <https://hal.archives-ouvertes.fr/hal-01017462>

# **Annexe 1**

## **Protocole de recueil de données – fiche enseignant**

### ***Déroulement de l'activité en classe***

Au début de l'activité, communiquer le lien aux élèves :

<https://iremi974.gitlab.io/python-de-la-fournaise/sautemouton.html>

Ne pas déposer le lien au préalable sur l'ENT, pour éviter que des élèves ne s'entraînent au préalable sans que leur activité ne soit enregistrée.

### **Lire la consigne suivante :**

L'activité comporte 2 parties :

- Pour la 1ere partie, le but du jeu est d'emmener tous les moutons blancs à droite et tous les moutons noirs à gauche ! Les moutons blancs peuvent se déplacer vers la droite vers une case vide ou en sautant par dessus un mouton noir. Les moutons noirs peuvent se déplacer vers la gauche vers une case vide ou en sautant par dessus un mouton blanc.
- La deuxième partie consiste à programmer : le fonctionnement du bouton UNDO n'est pas satisfaisant. Il faut modifier le programme pour qu'il permette de revenir à la configuration précédente.
- Ne pas oublier à la fin de la session, d'enregistrer l'activité.

L'activité est à faire individuellement pour permettre l'analyse à partir des seules traces enregistrées automatiquement. En cas d'activité en binôme, il faudrait un dispositif permettant d'enregistrer les conversations.

Laisser les élèves jouer et travailler en autonomie pendant le temps choisi (45 mn à 1 h).

### ***Recueil des traces***

Les traces d'activités sont loguées automatiquement dans le *localStorage* du navigateur. Le dispositif a été testé sous Firefox et fonctionne à condition que les paramètres de vie privée autorisent les données de sites.

Si la coche « supprimer les données de site à la fermeture de Firefox » est cochée, le recueil doit être fait immédiatement après l'activité, sans que les élèves ne ferment leur session de Firefox.

Le recueil de données peut être directement enregistré par l'élève en cliquant sur le lien Enregistre ton activité. Choisir un nom de fichier de la forme :

log\_saute\_mouton\_1.json en prenant soin de ne pas indiquer de nom d'élève mais seulement un numéro, ce qui permet à ce recueil de données d'être effectué sans aucune collecte de données personnelles.

Le recueil de données peut aussi être visualisé dans la page :

<https://iremi974.gitlab.io/python-de-la-fournaise/log.html>

Pour l'enregistrer, choisir « Enregistrer sous au format Fichier texte ».

Pour information, la trace est au format JSON et sera analysée par un code écrit en Python.

### ***Envoi des traces pour analyse***

Merci de m'adresser les fichiers de traces par mail à l'adresse :

[christophe.declercq@univ-reunion.fr](mailto:christophe.declercq@univ-reunion.fr)

Attention : le serveur de l'université bloque les archives zip !

## Annexe 2

### Exemple de trace d'activité

```
[('2022-09-19 10:45:22.091000', {'coup': 'Load', 'jeu': ['B', 'B', 'B', 'X', 'N', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:39.136000', {'coup': 2, 'jeu': ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:40.267000', {'coup': 4, 'jeu': ['B', 'B', 'N', 'B', 'X', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:41.715000', {'coup': 3, 'jeu': ['B', 'B', 'N', 'X', 'B', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:42.755000', {'coup': 1, 'jeu': ['B', 'X', 'N', 'B', 'B', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:43.404000', {'coup': 2, 'jeu': ['B', 'N', 'X', 'B', 'B', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:44.844000', {'coup': 0, 'jeu': ['X', 'N', 'B', 'B', 'B', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:46.242000', {'coup': 'Undo', 'jeu': ['B', 'B', 'B', 'X', 'N', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:48.118000', {'coup': 2, 'jeu': ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:49.029000', {'coup': 4, 'jeu': ['B', 'B', 'N', 'B', 'X', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:45:51.224000', {'coup': 'Reset', 'jeu': ['B', 'B', 'B', 'X', 'N', 'N', 'N'],
  'histo': []}),
('2022-09-19 10:46:14.153000', {'coup': 2, 'jeu': ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N']},
  'prog': 'def saute(i):\n      historique.append(jeu.copy())\n      if jeu[i]=="B" and i<6 and
jeu[i+1]=="X":\n          jeu[i], jeu[i+1] = "X", "B"\n      elif jeu[i]=="B" and i<5 and
jeu[i+1]=="N" and jeu[i+2]=="X":\n          jeu[i], jeu[i+2] = "X", "B"\n      elif jeu[i]=="N" and
i>0 and jeu[i-1]=="X":\n          jeu[i], jeu[i-1] = "X", "N"\n      elif jeu[i]=="N" and i>1 and
jeu[i-1]=="B" and jeu[i-2]=="X":\n          jeu[i], jeu[i-2] = "X", "N"\n      else :\n
historique.pop()\ndef undo():\n      global jeu\n      jeu = historique.pop()\n'}),
('2022-09-19 10:46:19.468000', {'coup': 4, 'jeu': ['B', 'B', 'N', 'B', 'X', 'N', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N'], ['B', 'B', 'X', 'B', 'N', 'N', 'N']]),
('2022-09-19 10:46:22.046000', {'coup': 3, 'jeu': ['B', 'B', 'N', 'X', 'B', 'N', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N'], ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  ['B', 'B', 'N', 'B', 'X', 'N', 'N']]),
('2022-09-19 10:46:30.761000', {'coup': 5, 'jeu': ['B', 'B', 'N', 'N', 'B', 'X', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N'], ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  ['B', 'B', 'N', 'B', 'X', 'N', 'N'], ['B', 'B', 'N', 'X', 'B', 'N', 'N']]),
('2022-09-19 10:46:33.940000', {'coup': 'Undo', 'jeu': ['B', 'B', 'N', 'X', 'B', 'N', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N'], ['B', 'B', 'X', 'B', 'N', 'N', 'N'],
  ['B', 'B', 'N', 'B', 'X', 'N', 'N']]),
('2022-09-19 10:46:41.365000', {'coup': 'Undo', 'jeu': ['B', 'B', 'N', 'B', 'X', 'N', 'N'],
  'histo': [['B', 'B', 'B', 'X', 'N', 'N', 'N'], ['B', 'B', 'X', 'B', 'N', 'N', 'N']])]
```



### Annexe 3

#### Résultats de la première expérimentation

ident	t_total	t_jeu	t_prog	gagne	t_gagne	n_gagne	n	appréciation	code
LLG001	44:34	09:00	35:33	True	06:47	82	254	historique sans copie du jeu	3
LLG002	37:44	05:35	32:08	False			185	programmation incohérente	1
LLG003	51:08	19:14	31:54	True	02:34	90	343	programmation incohérente	1
LLG004	52:45	12:19	40:25	False			201	historique sans copie du jeu	3
LLG005	54:19	13:15	41:03	True	01:02	33	168	programmation incohérente	1
LLG006	12:47	01:50	10:57	True	01:35	25	41	pas de programmation	0
LLG007	43:36	40:11	03:25	True	10:40	163	444	pas de programmation	0
LLG008	42:40	09:23	33:16	False			226	réussite	4
LLG101	55:53	09:20	46:33	False			175	historique un coup	2
LLG102	45:52	13:48	32:03	True	10:39	74	402	pas de programmation	0
LLG103	43:35	24:01	19:34	True	17:39	222	348	programmation incohérente	1
LLG104	45:07	19:16	25:50	False			237	historique un coup	2
LLG105	31:28	25:21	06:06	True	12:55	137	167	pas de programmation	0
LLG106	33:37	13:35	20:02	True	12:56	354	510	pas de programmation	0

## **Annexe 4**

### **Aide à propos des listes**

#### **Mémo sur les listes**

L'historique est au départ une liste (tableau) vide.

Les méthodes suivantes sont disponibles pour ajouter des éléments à une liste ou en supprimer :

- `list.append(x)` : Ajoute un élément à la fin de la liste.
- `list.insert(i, x)` : Insère un élément à la position indiquée.
- `list.pop(i)` : Enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de retour.
- `list.pop()` : Enlève de la liste le dernier élément et le renvoie en valeur de retour.

Attention : si une même liste est utilisée et modifiée plusieurs fois dans un programme, il est prudent d'en réaliser des copies :

- `list.copy()` : Renvoie une copie de la liste.

## Annexe 5

### Résultats de la deuxième expérimentation

ident	t_total	t_jeu	t_prog	gagne	t_gagne	nlg	nrg	nug	ng	nl	nr	nu	n	modif	deb_prog	code
LLV01	00:43:20	00:30:05	00:13:15	True	12:30	1	21	1	265	1	27	19	317	True	21:49	3
LLV02	00:47:43	00:21:00	00:26:43	True	06:01	1	5	1	39	1	33	33	136	True	06:01	4
LLV03	00:52:23	00:22:24	00:29:59	False						8	20	10	72	True	03:49	1
LLV04	00:38:02	00:18:15	00:19:47	True	15:38	1	3	2	23	1	9	10	73	True	10:32	1
LLV05	00:11:20	00:11:20	00:00:00	True	11:20	1	12	1	171	1	12	1	172	False		0
LLV06	00:52:17	00:10:59	00:41:18	True	17:26	1	16	10	62	2	23	23	94	True	03:32	1
LLV07	01:02:11	00:28:14	00:33:57	True	38:34	2	27	46	287	2	43	82	335	True	00:00	4
LLV08	01:04:01	00:08:24	00:55:37	False						1	13	5	47	True	02:56	1
LLV09	00:49:39	00:49:39	00:00:00	True	16:30	1	6	1	66	2	7	1	85	False		0
LLV10	01:02:26	00:55:18	00:07:08	False						1	11	31	185	True	39:47	1
LLV11	01:05:23	00:25:00	00:40:23	True	05:03	1	6	8	36	1	19	54	107	True	00:33	4
LLV12	01:05:51	00:20:49	00:45:02	True	05:54	1	2	12	75	1	37	109	225	True	02:29	4
LLV13	00:57:18	00:47:22	00:09:56	True	23:45	1	26	3	190	2	68	31	352	True	36:48	1
LLV14	01:07:10	00:18:41	00:48:29	False						1	46	47	110	True	00:00	1
LLV15	01:03:49	00:50:21	00:13:29	True	34:04	1	24	11	224	2	37	16	411	True	04:15	1
LMH101	00:42:55	00:19:06	00:23:49	True	07:31	1	14	0	136	1	31	13	223	True	13:03	1
LMH102	00:46:00	00:20:57	00:25:03	True	07:25	1	11	0	113	1	16	20	299	True	07:25	3
LMH103	00:47:05	00:09:20	00:37:45	True	04:52	1	3	3	41	1	47	44	121	True	04:52	3
LMH104	00:43:37	00:18:59	00:24:38	True	17:44	1	34	1	244	1	53	38	253	True	17:44	3
LMH105	00:45:05	00:07:09	00:37:56	True	02:45	1	0	8	61	1	4	43	149	True	02:45	3
LMH106	00:46:02	00:18:00	00:28:03	True	17:29	1	30	8	152	1	50	25	221	True	05:53	3
LMH107	00:30:18	00:30:10	00:00:08	True	10:01	1	1	24	201	2	1	37	331	True	23:12	0
LMH108	00:39:07	00:27:03	00:12:04	True	26:15	1	57	26	404	2	63	26	454	True	15:31	1
LMH109	00:36:02	00:07:36	00:28:26	True	04:32	1	9	1	108	2	16	11	181	True	04:32	3
LMH110	00:47:22	00:15:05	00:32:18	True	03:10	1	1	3	32	1	48	49	111	True	03:10	1
LMH111	00:45:26	00:23:31	00:21:55	True	29:56	1	32	3	193	1	37	3	219	True	17:57	3
LMH112	00:37:12	00:13:31	00:23:41	True	00:39	1	1	0	23	2	20	43	184	True	02:26	4
LMH113	00:28:05	00:28:05	00:00:00	True	19:44	1	34	2	271	1	44	3	361	False		0
LMH201	00:41:30	00:11:38	00:29:52	True	08:19	1	0	5	142	1	4	13	207	True	05:27	3
LMH202	00:10:36	00:02:47	00:07:49	False						1	6	14	45	True	00:00	4
LMH203	00:46:59	00:04:45	00:42:14	False						1	12	17	44	True	03:26	4
LMH204	00:46:13	00:23:25	00:22:48	False						1	67	67	712	True	20:38	4
LMH205	00:46:14	00:25:25	00:20:49	True	01:42	1	3	4	64	1	18	28	297	True	11:29	4
LMH206	00:46:27	00:34:06	00:12:21	True	05:01	1	8	1	77	1	60	9	505	True	23:16	3
LMH207	00:46:05	00:23:56	00:22:09	True	02:08	1	2	1	32	1	8	86	304	True	01:00	4

LMH208	00:46:18	00:22:22	00:23:56	True	02:25	1	2	0	33	4	37	58	364	True	04:17	3
LMH209	00:46:31	00:13:54	00:32:36	True	10:09	1	21	0	134	1	33	17	203	True	10:09	1
LMH210	00:45:39	00:18:03	00:27:36	True	24:34	2	57	24	443	2	75	102	496	True	07:44	1
<b>Min</b>	00:10:36	00:02:47	00:00:00		00:00:40	1	0	0	23	1	1	1	44		00:00:00	
<b>Max</b>	01:07:10	00:55:18	00:55:37		00:38:34	2	57	46	443	8	75	109	712		00:39:48	
<b>Moy</b>	00:46:18	00:22:01	00:24:17		00:12:41	1	15	7	140	2	30	33	237		00:09:41	