Exploration d'un jeu de taquin

Christophe Declercq, INSPE de la Réunion, LIM, IREMI

22 mai 2022

Introduction

La scène de jeu est un damier de 3 par 3 dont les cases peuvent être colorées en noir ou blanc. L'idée a été proposée dans un sujet de brevet avec un jeu d'instruction limité ne permettant pas d'atteindre toutes les configurations du jeu.

On propose un jeu d'instruction complet permettant avec seulement trois instructions d'atteindre les 512 configurations du jeu. C'est un jeu de solitaire dont l'objectif est de réussir à atteindre une configuration donnée en un nombre de coups minimal.

Il peut être utilisé pour découvrir la programmation au cycle 3. On notera cependant la difficulté d'anticiper les mouvements complexes du jeu.

Description du jeu

Le jeu comporte une scène à droite et une interface de programmation à gauche.

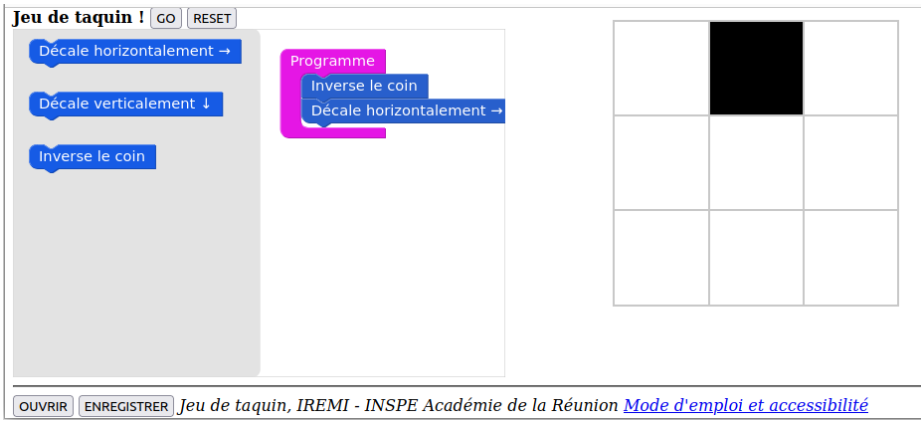

Figure 1: Interface du jeu de taquin

Pour jouer, on dispose uniquement des trois instructions suivantes :

- `Inverse le coin` : inverse la couleur du coin en haut à gauche.
- `Décale horizontalement` : décale horizontalement de gauche à droite et de manière circulaire toutes les cases du damier.
- `Décale verticalement` : décale verticalement de haut en bas et de manière circulaire toutes les cases du damier.

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

`https://iremi974.gitlab.io/blocks4school/taquin.html`

Analyse de l'accessibilité des configurations du jeu

La difficulté du jeu consiste à anticiper les mouvements globaux - à la manière du jeu de reversi. En effet après avoir introduit une case noire, puis l'avoir décalée horizontalement et/ou verticalement, l'introduction de nouvelles cases noires et leur déplacement entraîne aussi le décalage des cases précédemment introduites.

On peut proposer des activités progressives consistant à obtenir un motif à une ou deux cases noires pour découvrir ce fonctionnement.

On peut ensuite proposer des activités plus difficiles, dont en particulier des motifs de lignes horizontales, verticales ou de diagonales, puis le motif en croix avec 5 cases noires.

Analyse des configurations à 5 cases noires

Les configurations à 5 cases noires constituent un sous-problème intéressant du problème général d'accessibilité.

En effet certaines peuvent être obtenues avec un programme minimal contenant exactement 9 instructions, dont 5 instructions d'inversion entrecoupées de décalages. Les 6 configurations contenant la case en haut à gauche et celle en bas à droite sont les suivantes.

Figure 2: Configurations V-V-H-H, V-H-V-H et V-H-H-V

Figure 3: Configurations H-V-V-H, H-V-H-V et H-H-V-V

Accessibilité de toutes les configurations

Pour prouver que les 512 configurations sont bien accessibles, il suffit de prouver que le programme d'exploration en largeur des configurations accessibles termine avec 512 configurations.

```
def r(jeu): # retourne le coin haut gauche
    return (((jeu[0][0]+1)%2, jeu[0][1], jeu[0][2]),
            jeu[1],
            jeu[2])
def h(jeu): # décale horizontalement vers la droite
    return ((jeu[0][2], jeu[0][0], jeu[0][1]),
            (jeu[1][2], jeu[1][0], jeu[1][1]),
            (jeu[2][2], jeu[2][0], jeu[2][1]))
def v(jeu): # décale verticalement vers le bas
    return ((jeu[2][0], jeu[2][1], jeu[2][2]),
            (jeu[0][0], jeu[0][1], jeu[0][2]),
            (jeu[1][0], jeu[1][1], jeu[1][2]))

initial = ((0,0,0), (0,0,0), (0,0,0))
etats = {initial:[]}      # dictionnaire : état -> chemin
file_etats = [initial]    # file des états à explorer
while file_etats != []:
    etat = file_etats.pop(0) # exploration du premier état
    for op in (r, h, v):     # essai des 3 opérations
        next_etat = op(etat)
        if next_etat not in etats : # ajout du nouvel état
            chemin = etats[etat].copy()
            chemin.append(op.__name__)
            etats[next_etat] = chemin  # dans le dictionnaire
            file_etats.append(next_etat) # dans la file
```

Le programme construit un dictionnaire des états accessibles et associe à chacun le plus court chemin pour y parvenir. Le parcours en largeur consiste à gérer une file des états à examiner et à calculer leurs successeurs par une des opérations.

L'exécution du programme ci-dessus se termine avec 512 états, on obtient ainsi la preuve de l'accessibilité de toutes les configurations et la longueur du plus petit programme permettant d'atteindre chacune.

La requête suivante permer d'obtenir le programme optimal en 13 opérations pour obtenir la croix :

```
>>> etats[((1,0,1), (0,1,0), (1,0,1))]
['r', 'h', 'r', 'h', 'v', 'r', 'h', 'h', 'v', 'r', 'h', 'h', 'r']
```

Conclusion

On a présenté une activité de type jeu de solitaire pouvant être utilisé au cycle 3 pour une découverte de la programmation associée à une activité ludique.

L'exploration des configurations accessibles peut, quant à elle, fournir une activité de projet en spécialité NSI en classe de première ou terminale nécessitant de manipuler des listes, des tuples et des dictionnaires. Ce peut aussi être une manière de faire découvrir l'algorithme de parcours en largeur d'un graphe. Ce graphe n'est pas explicite mais est construit implicitement par le calcul d'accessibilité des configurations.

On espère ainsi continuer à montrer le potentiel des activités ludiques pour l'appentissage de la programmation à tous les niveaux de l'enseignement scolaire.