UNIVERSITÉ DE ROUEN NORMANDIE

UFR Sciences et Technique

Projet : Jeu du Solitaire Chinois

Rapport

Réalisé par :

- **MELLIKEHCE Massinissa**
- **GHEZZAZ Mahdi**

2024-2025

Table des matières

Jeu du Solitaire Chinois

1. Introduction Historique

Le Solitaire, également appelé "Peg Solitaire", est un jeu de stratégie solitaire qui remonte au XVIIe siècle. Il consiste à déplacer des pions (généralement des billes ou des fiches) sur un plateau dans le but de n'en laisser qu'un seul à la fin de la partie. Bien que l'origine exacte du Solitaire soit incertaine, il semble être une évolution d'un jeu de pions de type "chasse", tel que le jeu du renard et des poules, où l'objectif était de capturer les pions adverses en sautant par-dessus eux, un mécanisme similaire à celui des dames.

Le jeu du Solitaire a été popularisé en France à la fin du XVIIe siècle. La première description écrite du jeu apparait dans le *Mercure galant* d'août 1697, où il est décrit comme un jeu très en vogue à la cour, notamment auprès des femmes. Ce jeu, apparemment d'origine américaine selon certaines sources de l'époque, fait rapidement fureur et devient une tendance à Paris, se propageant à travers la société de l'époque. Bien qu'il soit souvent associé à un public féminin, des gravures de l'époque montrent aussi des hommes jouant à ce jeu, témoignant de son attrait pour tous les segments de la population.

Les variantes du jeu se multiplient au fil du temps, mais la version la plus connue reste celle sur un plateau en forme de croix.

2. Structures de Données Utilisées

Pour résoudre ce jeu, nous avons utilisé plusieurs structures de données simples :

- **Le plateau de jeu** : Il est représenté par une grille 7x7. Chaque case est soit occupée par une bille (1), soit vide (0), soit inaccessible (-1) pour les cases situées dans les coins du plateau, qui ne font pas partie du jeu.

  Voici deux exemples de plateaux :

  - **Plateau classique ajusté** : C'est le plateau traditionnel en forme de croix avec une case vide au centre.

    - **Plateau cercle centré** : Une variante où le plateau forme une sorte de cercle avec une case vide au centre.
- **Directions de saut** : Les directions dans lesquelles les billes peuvent se déplacer sont définies par quatre tuples : droite, gauche, haut et bas. Ces directions sont utilisées pour vérifier si un saut est possible.
- **États visités** : On garde un ensemble (visited_states) pour mémoriser les configurations de plateau déjà testées. Cela permet d'éviter de refaire les mêmes mouvements.
- **Mouvements** : Les mouvements sont enregistrés dans une liste de tuples, chacun indiquant d'où vient la bille et la direction dans laquelle elle a sauté.

3. Algorithmes Principaux

3.1. Vérification des Mouvements

Cet algorithme vérifie si un mouvement est valide :

- La case d'origine contient une bille (1).
- La case entre les deux positions contient aussi une bille.
- La case cible (où la bille doit atterrir) est vide.

```python
# Vérifie si un mouvement est valide
def is_valid_move(board, x, y, dx, dy):
    nx, ny = x + dx, y + dy  # Coordonnées de la case sautée
    nnx, nny = x + 2 * dx, y + 2 * dy  # Coordonnées de la case cible

    # Vérifications des limites et conditions du jeu
    if not (0 <= nx < BOARD_SIZE and 0 <= ny < BOARD_SIZE and
            0 <= nnx < BOARD_SIZE and 0 <= nny < BOARD_SIZE):
        return False

    return (
        board[x][y] == 1  # Case d'origine contient une bille
        and board[nx][ny] == 1  # Case sautée contient une bille
        and board[nnx][nny] == 0  # Case cible est vide
    )
```

3.2. Effectuer un Mouvement

Une fois validé, un mouvement est réalisé en retirant la bille d'origine, retirant la bille sautée et en posant la bille sur la case vide.

```python
# Effectue un mouvement
def make_move(board, x, y, dx, dy):
    nx, ny = x + dx, y + dy
    nnx, nny = x + 2 * dx, y + 2 * dy
    board[x][y] = 0  # Retirer la bille d'origine
    board[nx][ny] = 0  # Retirer la bille sautée
    board[nnx][nny] = 1  # Placer la bille à la destination
```

3.3. Annuler un Mouvement

Pour la résolution du jeu, nous utilisons un algorithme de backtracking (retour en arrière). Cela nécessite d'annuler un mouvement si celui-ci ne mène pas à une solution. L'algorithme restaure l'état du plateau en remettant la bille à sa place d'origine.

```python
# Annule un mouvement
def undo_move(board, x, y, dx, dy):
    nx, ny = x + dx, y + dy
    nnx, nny = x + 2 * dx, y + 2 * dy
    board[x][y] = 1  # Remettre la bille d'origine
    board[nx][ny] = 1  # Remettre la bille sautée
    board[nnx][nny] = 0  # Vider la case de destination
```

3.4. Vérification de la Victoire

Pour savoir si le jeu est résolu, on compte le nombre de billes restantes. Si une seule bille reste sur le plateau et qu'elle est à l'endroit prévu (cible), alors le jeu est terminé.

```python
# Vérifie si une solution est atteinte
def is_solved(board, x, y):
    count = sum(row.count(1) for row in board)
    return count == 1 and board[x][y] == 1
```

3.5. Backtracking pour la Résolution Automatique

L’algorithme principal explore toutes les configurations possibles du plateau en essayant tous les mouvements valides. Si un mouvement conduit à une impasse, il revient en arrière pour essayer d’autres options. Si une solution est trouvée, il arrête l’exploration.

```
if not solve(board, moves, target_x, target_y):
    moves.pop()
    undo_move(board, x, y, dx, dy)
else:
    return True
```

4. Choix de Programmation

- **Affichage en Couleur** : Pour rendre l’expérience utilisateur plus agréable, on a utilisé la bibliothèque colorama afin d’ajouter des bordures colorées au plateau.
- **Flexibilité pour l'utilisateur** : on a ajouté des options pour que l’utilisateur puisse choisir le plateau de départ, changer la position de la case vide initiale et définir une position cible pour la dernière bille. Cela permet à l'utilisateur de personnaliser la partie.

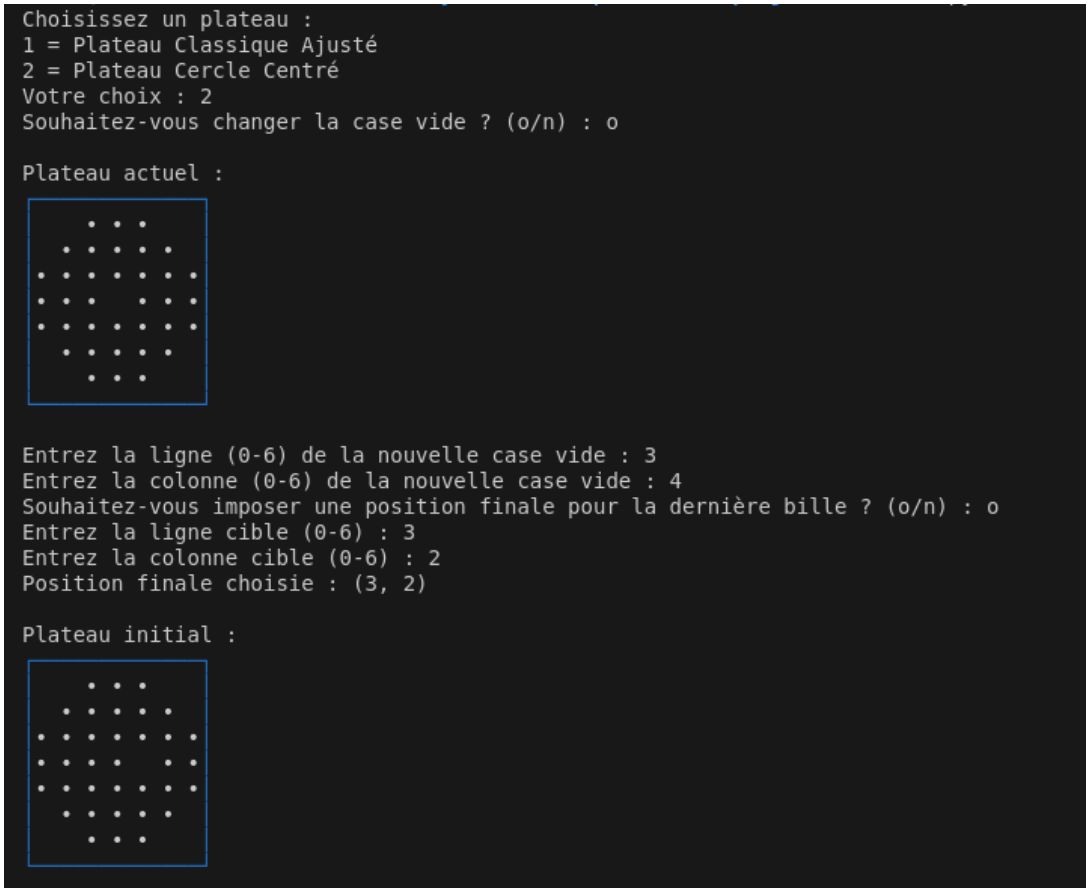
5. Difficultés Rencontrées

- **Changer la case vide initiale** : Il fallait permettre à l'utilisateur de changer la position de la case vide au début du jeu, ce qui nécessitait de vérifier que la nouvelle position choisie était correcte.
- **Éviter les répétitions** : Pour que l’algorithme ne répète pas inutilement les mêmes configurations, il a fallu mettre en place un système pour mémoriser les états déjà visités. Cela permet d’accélérer la recherche de solution.
- **Complexité du backtracking** : L'algorithme de backtracking explore toutes les possibilités, et cela peut devenir très long O(4^m×n). C’est pourquoi il est essentiel d’optimiser en évitant les répétitions inutiles.

6. Mode d'Utilisation

- **Choix du plateau** : Au démarrage, vous choisissez entre le plateau classique ou une variante en forme de cercle.

- **Changer la position de la case vide** : Si vous le souhaitez, vous pouvez changer la position de la case vide avant de commencer la partie.
- **Position finale de la dernière bille** : Vous pouvez aussi imposer une position spécifique pour la dernière bille, sinon elle reste par défaut au centre.



- **Résolution automatique** : Le programme résout ensuite le jeu en testant tous les mouvements possibles. Si une solution est trouvée, il affiche les mouvements à réaliser.

- ## 7. Conclusion

  Ce projet montre comment utiliser un algorithme de backtracking pour résoudre un jeu de solitaire de manière automatique. Il démontre aussi l'importance d'optimiser les algorithmes de recherche pour éviter les répétitions.

Exemple de résolution automatique :

```
Solution trouvée avec les mouvements suivants :
Bille de (1, 3) sautée en direction (3, 3)
Bille de (2, 1) sautée en direction (2, 3)
Bille de (0, 2) sautée en direction (2, 2)
Bille de (0, 4) sautée en direction (0, 2)
Bille de (2, 3) sautée en direction (2, 1)
Bille de (2, 0) sautée en direction (2, 2)
Bille de (2, 4) sautée en direction (0, 4)
Bille de (2, 6) sautée en direction (2, 4)
Bille de (3, 2) sautée en direction (1, 2)
Bille de (0, 2) sautée en direction (2, 2)
Bille de (3, 0) sautée en direction (3, 2)
Bille de (3, 2) sautée en direction (1, 2)
Bille de (3, 4) sautée en direction (3, 2)
Bille de (3, 6) sautée en direction (3, 4)
Bille de (3, 4) sautée en direction (1, 4)
Bille de (0, 4) sautée en direction (2, 4)
Bille de (4, 2) sautée en direction (2, 2)
Bille de (1, 2) sautée en direction (3, 2)
Bille de (4, 0) sautée en direction (4, 2)
Bille de (4, 3) sautée en direction (4, 1)
Bille de (5, 4) sautée en direction (3, 4)
Bille de (4, 6) sautée en direction (4, 4)
Bille de (6, 2) sautée en direction (4, 2)
Bille de (3, 2) sautée en direction (5, 2)
Bille de (6, 4) sautée en direction (6, 2)
Bille de (6, 2) sautée en direction (4, 2)
Bille de (4, 1) sautée en direction (4, 3)
Bille de (4, 3) sautée en direction (4, 5)
Bille de (2, 4) sautée en direction (4, 4)
Bille de (4, 5) sautée en direction (4, 3)
Bille de (5, 3) sautée en direction (3, 3)

Plateau final :
```