Rapport : Projet d'année Slideways

Polonski Sébastien 499415

Année académique 2019/2020

Contents

1 Introduction

Le but de ce projet d'année est l'implémentation en Python 3 du jeu de société "Slideways" dont les règles sont disponibles à cette adresse. Le jeu est une variante des jeux OXO et Puissance 4.

Le développement est découpé en 4 parties au cours de l'année.

- Partie 1 : Jeu simplifié, affichage en terminal
- Partie 2 : Jeu complet + IA basique, affichage en terminal
- Partie 3 : Réalisation d'une interface graphique pour le jeu à l'aide de la librairie PyQt.
- Partie 4 : Différentes améliorations au choix de l'étudiant(e), créativité encouragée.

La partie 1 ne contient que le jeu de base, sans IA ou option de décalage et avec un affichage en terminal. Le jeu ressemble donc plus au "OXO" mais avec un plateau différent.

La partie 2 nous a permis de découvrir le jeu dans son intégralité par l'ajout de l'option de décalage et l'implémentation d'une intelligence artificielle qui se base sur l'algorithme "minimax".

La partie 3 a surtout eu pour objectif de se focaliser sur la transposition de l'affichage en terminal par l'affichage avec une fenêtre grâce aux modules de PyQt5.

Le rapport se focalisera donc surtout sur la dernière partie à savoir la partie 4. Cette-dernière contient des fonctionnalités obligatoires et certaines facultatives visant notamment à améliorer l'intelligence artificielle que ce soit au niveau du temps où de l'exactitude des coups joués.

2 Méthodes obligatoires et exemples

2.1 Modification des règles de jeu

Une des règles à ajouter est la suivante : "Lors d'une même partie, on ne pourra pas rencontrer deux fois exactement le même plateau de jeu". Pour cela nous allons utiliser le code suivant afin de récupérer l'information d'un plateau sous forme de chaine de caractère.

```
def get_values(self):
    values = ""
    for row in self.grid:
        for elem in row:
            values = values + str(elem)  # ajout de chaque element du plateau
    for offset in self.offset:
        values = values + str(offset)  # ajout de chaque decalage
    return values
```

En effet, le stockage de l'information prend ainsi moins de place en mémoire que la copie du plateau et de la liste de décalages en entier. Cette fonction sera utilisé dans la méthode "get valid actions" dans laquelle on testera chaque coup supposé valide, transformer le plateau après coup en chaine de caractères et voir à l'aide de la fonction ci-dessus si la chaine de catactère est déja présente dans l'ensemble des plateaux joués. Si le plateau a déja été joué, le coup est retiré des coups valides et dans tous les cas l'action sur le plateau est annulée.

2.2 Sauvegarde et rediffusion d'une partie

2.2.1 Sauvegarde d'une partie

Tout d'abord pour la sauvegarde d'une partie, j'ai décidé de faire celle-ci lorsque la case "Save game" est cochée et que la partie se termine avec un ou deux gagant(s). Cela évite ainsi quelconque ambiguïté quant à la fin de la rediffusion et permet au joueur de savoir clairement quand il joue et quand il s'agit d'une rediffusion. La sauvegarde crée donc un fichier format *.sldws pour que celui-ci soit plus facilement détectable lors de la rediffusion. Celui-ci est structuré de cette manière: "4A,+0,4B,+2".

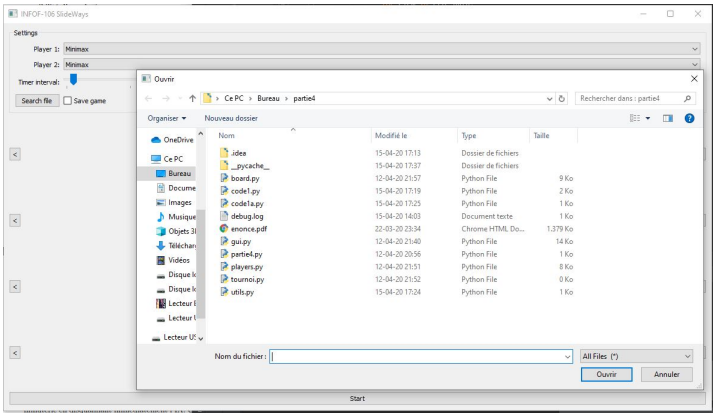

Figure 1: Recherche de fichier avec QfileDialog

2.2.2 Rediffusion d'une partie

En ce qui concerne la rediffusion d'une partie, le fichier est ensuite lu grâce à la fonction "get file content" appelée par "replay game" lorsque l'utilisateur appuie sur le bouton "Search file" et sélectionne un fichier format *.sldws, sinon rien ne se passe. La fonction replay game parcours ensuite le fichier si celui-ci est valide et joue les coups qui y figurent en annonçant le(s) gagant(s) à la fin. Bien entendu, pour éviter d'interrompre une partie déjà en cours, un paramètre "game is playing" est ajouté à la classe "App" afin de savoir si une partie est déja en cours. La rediffusion est donc possible seulement si personne ne joue déjà.

```
def get_file_content(self):
    """
    Fonction permettant de choisir un fichier via QfileDialog et d'en extraire le contenu s
    :return: data : liste contenant les coups a jouer si le fichier est valide, -1 sinon
    """
    dialog = QFileDialog()
    dialog.setFileMode(QFileDialog.ExistingFile)
    data = []
    if dialog.exec_():
        file_name = dialog.selectedFiles()
        if file_name[0].endswith('.sldws'):  # fichier dedie contenant les coups joues
            with open(file_name[0], 'r') as f:
                for line in f:
                    for elem in line.split(","):
                        data.append(elem)  # chaque coup joue dans l'ordre
        else:
            data = -1
    return data
```

2.3 Minuterie

Enfin, la dernière fonctionnalité obligatoire à implémenter est la suivante : "Si une IA ne retourne pas un coup dans le temps imparti, elle perd la partie. Il est évident que cette limitation ne s'applique pas à un joueur humain".
Pour cela il nous suffit de mesurer le temps avant calcul du coup et le soustraire au temps après calcul du coup et voir si le résultat est inférieur à une seconde. Si tel est le cas, la partie s'arrête.

```
def play(self, event=None):
    '''
    Joue le meilleur coup selon l'arbre de possibilites minimax
    :param event: ignore
    '''
    starting_time = time.time()  # depart de la minuterie, avant que l'IA calcule son coup
    action = self.minimax()[0]
    if time.time() - starting_time < 1:  # L'ia a maximum 1 seconde pour jouer
        self.board.act(action, self.player)
        self.board.played_moves.append(action)  # ajoute l'action jouee
        self.board.played_boards.add(self.board.get_values())
    else:
        print("IA Disqualifiee")  # Disqualification de l'IA, reset du plateau
        self.board.reset()
```

3 Améliorations facultatives de l'IA

3.1 Élagage Alpha-Beta

Pour ce qu'il s'agir de l'amélioration de l'IA, la base pour la fonction minimax était fournie dans la partie 3. Celle-ci contenait déja des modifications, notamment l'ajout de pénaliités en fonction de la profondeur de la recherche des meilleurs coups possibles. Ici, il s'agir juste de comparer les valeurs actuelles et la valeur d'alpha et beta et de ne pas continuer dans les branches qui ne respectent pas les conditions. Cela améliore donc la rapidité de la fonction.

```
def minimax(self, depth=2, maximize=True, penalty=0, alpha=-1000, beta=1000):
    if depth == 0:  # fin de l'arbre, pas de coup gagnant
        return (None, DRAW - penalty)
    if maximize:
        # joueur max
        best_score = -INF
        player = self.player

    else:
        # joueur min
        best_score = +INF
        player = self.other_player
    best_actions = []

    valid_actions = self.board.get_valid_actions(player)
    for action in valid_actions:  # pour chaque action valide
        self.board.act(action, player)
        winner = self.board.winner()
        if len(winner) == 0:
            score = self.minimax(depth - 1, not maximize, penalty + 1, alpha, beta)[1]
        else:  # si le coup est gagnant
            score = WIN - penalty if winner.pop() == self.player else LOSS + penalty
        self.board.undo()

        if score > best_score:
            if maximize:
                best_score = max(alpha, score)  # compare le score actuel avec alpha
                if not (beta <= alpha):
                    best_actions = [(action, score)]
                alpha = max(best_score, alpha)

        elif score < best_score:
            if not maximize:
                best_score = min(score, beta)  # compare le score actuel avec beta
                if not (beta <= alpha):
                    best_actions = [(action, score)]
                beta = min(best_score, beta)

        else:
            best_actions.append((action, score))
    return random.choice(best_actions)
```

4 Conclusion

Pour finir, ce projet d'année à permis de parcourir différents outils, méthodes et niveaux de difficultés aux cours de l'année. En effet, lors de la partie 1, la demande d'implémentation était relativement basique mais totalement adaptée à notre expérience de débutant(manipulation de matrice, création de fonctions). Beaucoup d'aide était fournie que ce soit au niveau des tests ou des différentes fonctions à implémenter.

En ce qui concerne la partie 2, il en va presque de même mise à part une réfléxion plus poussée afin de permettre la fonctionnalité de décalage et de déja s'intéresser à une IA qui utilise un algorithme récursif avec lequel nous n'avions pas encore beaucoup d'expérience(récursivité). Là aussi, peu à peu, de moins en moins d'aide était fournie. Il suffit de prendre comme exemple l'implémentation du décalage dont la réalisation était laisée à notre esprit logique.

Voici arrivée le temps de la partie 3, où le plus grand défi à relever était d'apprendre par soi-même les concepts d'interface graphique et d'apprentissage d'une libraire annexe. Là aussi, de moins en moins de guidance quand aux choix des fonctions à utiliser. L'autonomie de la réalisation était réelle.

C'est donc naturellement que viens s'annexer la partie finale de ce projet. Où il est libre de choisir ou non certaines fonctionnalités, voire même d'en ajouter si on en a envie(et qu'elles sont approuvées). Cette dernière partie, dont la remise se situe aux alentours de la fin de l'année reprends aussi les concepts des cours d'algorithmiques et donc permet d'en faire le lien.

Je pense donc que ce projet d'année était bien répartit en difficulté au cours des quadrimestres et nous a permis en tant que débutants et étudiants en première année d'informatique de toucher un peu à tout : manipulation de matrices, algorithme et IA, interface graphique et rédaction de rapport en LaTeX.