Rapport de projet

Binôme : Poueyto Clément et Lecavelier Maëva.

Groupe 2.

Notre projet était de faire un jeu des allumettes grâce au language de programmation Python et plus particulièrement avec son module graphique Turtle. Maëva avait déjà de l'expérience dans le développement d'un projet grâce au baccalauréat option ISN. Quant à Clément, il n'avait pas vraiment d'expérience dans la programmation. Nous nous étions alors mis d'accord sur le compromis de faire la version +, c'est-à-dire avec différentes règles tirées aléatoirement. Mais lors de notre travail, notre motivation et notre implication nous ont mené à développer la version ++++ du jeu, c'est-à-dire celle avec l'IA qui a une stratégie gagnante.

De plus, nous avions à choisir un thème pour ce jeu. Aimant tous les deux l'univers fantastique de Game of Thrones, notre choix a été rapide. Le décor reprend ainsi l'univers de cette série, et plus particulièrement la zone en dehors du mur de glace, que l'on peut apercevoir sur la droite du décor. Il y aussi des dragons dans la série. Ceux-ci se retrouvent dans le ciel, au niveau du mur de glace. Ainsi, nos allumettes n'ont pas la forme d'allumettes mais d'épées et le pommeau de celles-ci est de couleur aléatoire pour correspondre à la couleur d'une « maison » (famille) de Game of Thrones.

Les règles :

Bienvenue dans l'univers impitoyable de Game of Thrones dans lequel les conflits se règlent autour du « jeu des allumettes ». Le but de ce jeu que nous avons développé est de retirer la dernière allumette. Le jeu n'offre la possibilité que de jouer contre l'ordinateur. C'est donc un jeu 1 contre 1. Chacun leur tour, les joueurs doivent retirer un certain nombre d'allumettes, et celui qui retire la dernière a gagné la partie.

Dans notre version, il existe deux modes de jeu : la version classique et la version aléatoire. Les différences entre les deux versions sont les règles et le nombre d'allumettes possible au début de la partie.

La version classique n'a qu'une seule règle possible : on peut enlever 1,2 ou 3 allumettes dans la limite des allumettes restantes. Le nombre d'allumettes au départ est tiré au hasard entre 15 et 20. Ainsi la partie n'est ni trop rapide, ni trop longue.

La version aléatoire a énormément de règles possibles. En effet, la règle peut être composée de 3 à 8 valeurs, selon le tirage aléatoire. Ensuite, elle est composée de chiffres aléatoires entre 3 et 8. Afin d'éviter de bloquer le jeu, les chiffres 1 et 2 composent n'importe quelle règle. Exemple de règles possibles : [1,2,4,8] ; [1,2,3,4,7] ; [1,2,3,4,5,6] etc… Dans la version aléatoire, le nombre d'allumettes est tiré au hasard entre 15 et 40.

Dans la logique, nous avons fait en sorte qu'il soit impossible de retirer plus d'allumettes qu'il n'en reste, d'en retirer un nombre qui ne soit pas présent dans les règles et également être obligé de retirer au moins une allumette par tour.

En plus des deux modes de jeu, on propose aussi deux modes de difficulté de l'IA : facile et difficile. Pour le premier, l'ordinateur n'a aucune stratégie gagnante alors que dans la seconde celui-ci s'adaptera toujours à la règle spécifiée en début de jeu. Il s'adapte également à nos actions.

La structure du projet :

Le projet se structure en quatre domaines majeurs : le code du jeu, celui de l'IA, celui du décor fixe et enfin des décors dynamiques.

Nous nous sommes donc réparti le travail afin d'avoir deux domaines à traiter chacun. Clément s'est occupé du code du jeu et de l'IA et Maëva des différents décors. Bien sûr, tout est en corrélation car le décor dynamique dépend du déroulement du jeu, et l'IA également. Nous avons beaucoup communiqué lors de ce projet afin de tenir l'autre au courant des avancées du projet.

Nous nous sommes donc organisé comme suit :

Modules	Code du jeu	IA	Décor fixe	Décor dynamique	Son	Plateau
Maëva L			X	X	X	X
Clément P	X	X			X	X

Etudions le déroulement d'une partie au niveau du code :

- Le décor fixe (paysage, forêt, mur, dragons) s'affiche.
- Le joueur choisit le mode de jeu et la difficulté de l'IA.
- La partie commence : le plateau (pancarte) et les allumettes (épées) s'affichent.
- Les joueurs jouent chacun leur tour.
- Le plateau se retrace par-dessus l'ancien. Les allumettes se redessinent pour correspondre au nombre restant. Et du texte indique ce que vient de jouer l'ordinateur, le nombre d'allumettes qui reste ainsi que la règle du jeu en cours.
- Lorsqu'il n'y a plus d'allumettes, le jeu s'arrête et un message de victoire ou de défaite s'affiche.

L'IA

Ce module rassemble les fonctions qui participent à la stratégie gagnante de l'IA dans le cas où la difficulté choisie est « difficile ».

Comportement de l'IA :

Pour le mode de jeu classique :

L'ordinateur cherchera constamment à tirer des allumettes de manière à se ramener à un multiple de 4, si le nombre d'allumettes est déjà un multiple de 4 alors l'ordinateur sera contraint de retirer des allumettes de manière aléatoire car la règle est [1,2,3]. De plus s'il reste 1,2 ou 3 allumettes alors l'ordinateur en tirera un nombre convenable pour lui assurer la victoire.

Pour le mode de jeu « aléatoire » :

L'IA réalise un schéma identique à chaque tour afin de déterminer quelle sera la meilleure valeur à jouer. L'ordinateur commence par vérifier s'il peut gagner, sinon il continue.

Ensuite il cherche toutes les valeurs pour lesquelles il peut perdre au prochain tour (fonction Anticipation()). Ces valeurs sont ajoutées dans une liste (« got »), et il ne pourra pas tirer l'une de ces valeurs durant ce tour.

De manière analogue au mode classique, il s'adapte à la règle pour se ramener à un multiple spécial. Ainsi le modulo dépendra de la variable « suite », par exemple pour une règle [1,2,3,5], suite est égal à 3 donc il se ramènera à un modulo 4. Idem pour [1,2,3,7,8].

Une fois la « règle du modulo » définie, le programme place dans une liste (appelée valar) les valeurs permettant de se ramener à ce multiple. Il tire une valeur au hasard parmi la liste « valar », puis vérifie qu'elle n'appartient pas aux valeurs interdites.

S'il ne peut pas se ramener à un modulo, alors il tire une valeur aléatoire dans les règles. Si elle est interdite, alors il recommence, jusqu'à tomber sur une valeur convenable.

Si toujours aucune valeur ne convient alors il finira pas tirer soit 1, soit 2.

Le code du jeu :

Le module « codeJeu » comprend toutes les fonctions qui permettent de faire fonctionner la base du jeu. Du tirage des règles en passant par le fait de retirer des allumettes, l'ensemble de ces fonctions sera utilisé dans la fonction « JEU » qui permettra de lancer le jeu via le module « allumettes.py ».

```
########################################################################
# FONCTION PARTIE #
########################################################################

def JEU(nbAllum,mdj,difficulte,regle,regleOrdi,endgame,ordiRetire,n,long,haut,x_epee,y_epee): # "n" indique qui joue. Pair = joueur, impair = ordi.
    while endgame==0: #tant que les conditions de victoire ne sont pas réunies
        if n%2==0: #on fait jouer le joueur
            dessineEpee(x_epee,y_epee,nbAllum,long,haut,ordiRetire,regle)#trace le plateau et les épées (selon le nombre d'allumettes)
            nbAllum=retireAllum(nbAllum,regle,mdj)# retireAllum demande au joueur combien d'épées il veut retirer et met à jour directement le nombre d'épées
            os.system(r'afplay sword_cut.wav&') #lance le son d'épée du joueur
            ordiRetire=0 #on réinitialise la variable d'ordiRetire. Utilité dans la fonction dessineEpee
            dessineEpee(x_epee,y_epee,nbAllum,long,haut,ordiRetire,regle)#affiche le nouveau nombre d'épées
            endgame=jeuValide(nbAllum) #vérifie si le jeu continue
            if endgame==1: #si la partie se finit
                elemFinal(x_epee,y_epee,nbAllum,long,haut,ordiRetire,regle,n)


        else:    #l'ordinateur joue
            textinput(" ","Au tour de l'ordinateur ? ")#marque une pause
            if difficulte==1:
                ordiRetire=ordinateurFacile(regleOrdi,nbAllum,mdj)
            else:   #difficulté 2
                ordiRetire=IA(nbAllum,regleOrdi,mdj,regle)
            nbAllum=playOrdinateur(nbAllum,ordiRetire)
            os.system(r'afplay sword_ia.wav&')
            endgame=jeuValide(nbAllum) #vérifie si le jeu continue
            if endgame==1:
                elemFinal(x_epee,y_epee,nbAllum,long,haut,ordiRetire,regle,n)
        n=n+1 #change de tour
```

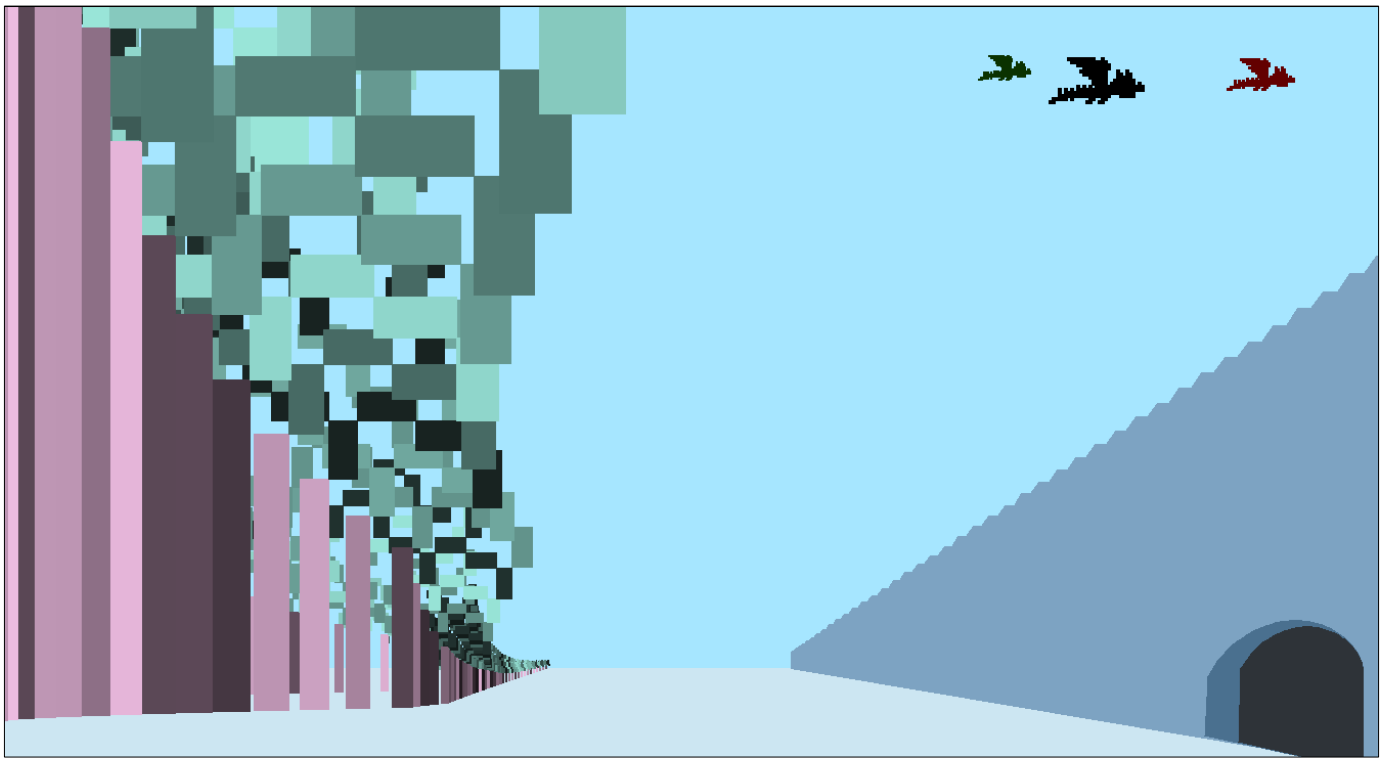
- retireAllum(nbAllum,regle,mdj) : cette fonction permet de demander le nombre d'allumettes que le joueur souhaite retirer puis elle met à jour le nombre d'allumettes restantes
- Chaque tour se termine par la fonction « endgame » qui vérifie si les conditions de fin de partie sont réunies.
- Selon la difficulté, la fonction qui fera retirer un nombre X d'épées à l'ordinateur n'est pas la même, en effet la fonction « ordinateurFacile » fera jouer un nombre aléatoire tandis que « IA » optera pour une stratégie gagnante si la difficulté est difficile.
- playOrdinateur met à jour le nombre d'allumettes restantes durant le tour de l'ordinateur.
- Lorsque la partie est terminée la fonction « elemFinal » trace le panneau sans épée, affiche un message et joue un son de victoire ou de défaite.

Le décor

Voyons plus en détail comment fonctionnent les différents décors :

Le décor fixe :

Le décor fixe est composé du ciel, du sol, du mur, des forêts et des dragons. L'ordre d'appel des fonctions est important pour permettre la superposition des différents dessins sans en cacher d'autres. En effet, appeler le ciel après la forêt cacherait une partie de celle-ci. Voici le décor fixe :



Il est composé de cinq fonctions différentes :

- Celle des dragons (appelé trois fois pour dessiner trois dragons)
- Celle de la forêt, qui appelle elle-même la fonction arbre qui appelle également d'autres fonctions. La fonction foret() est appelée deux fois pour faire une impression de forêt épaisse.
- Celle du mur, qui dessine le mur sur la droite (on a pas eu le temps de faire la texture du mur, on ne savait pas non plus comment s'y prendre exactement). Clément a su mettre une porte et rendre le haut du mur irrégulier.
- Celle du sol qui dessine un rectangle selon la longueur et la hauteur de la fenêtre
- Celle du ciel, similaire à celle du sol

Le décor dynamique :

Le décor dynamique n'est composé que de trois éléments majeurs : les épées, la pancarte (le plateau de jeu) et le texte qui affiche ce qu'a joué l'ordinateur, le nombre d'épées restantes et les règles du jeu. Voici notre interface graphique dynamique :

Voici la structure de ce décor :

- On trace le plateau avec la fonction plateau() qui dessine la pancarte marron
- Ensuite on dessine les épées pour cela :
    - La fonction epeePleine (qui utilise les fonctions garde(), carré(), et facteur_taille() (pour adapter la taille selon le nombre d'allumettes)) trace une épée
    - La fonction dessineEpee dessine le nombre d'épées qu'il faut. Pour cela, elle appelle la fonction epeePleine, repeteEpee et facteur_distance pour séparer les épées d'une certaine distance pour que ce soit esthétique et que toutes les épées rentrent dans le cadre. Moins il y a d'épées, plus fact_distance et longueurBase sont élévés.
- Enfin, un message apparaît sur la pancarte pour indiquer ce que vient de jouer l'ordinateur, le nombre d'allumettes restantes et rappeler les règles du jeu actuelles.

Nous avons également diviser la mise à jour du nombre d'allumettes en différentes étapes :

- Le nombre d'allumettes avant le tour du joueur
- Le nombre d'allumettes après le tour du joueur mais avant celui de l'ordinateur
- Le nombre d'allumettes après le tour de l'ordinateur mais avant celui du joueur

Ainsi, l'avancée du jeu est plus visible. De plus ceci nous a également permis de rajouter le son d'épée du tour de l'ordinateur, car sinon il jouait immédiatement après celui du joueur qui empêchait de voir la différence entre le tour du joueur et le tour de l'ordinateur. Pour ceci, nous avons fait appel à un « textinput » qui permet de faire apparaître une fenêtre pour que le joueur puisse valider quand il souhaite que l'ordinateur joue. Cette fenêtre agit comme une pause.

Les musiques :

Pour améliorer l'ambiance médiévale du programme nous avons ajouté une musique de fond qui est le générique de la série Game of Thrones. De plus lorsque l'ordinateur ou le joueur retire des allumettes, un bruit d'épée différent est joué. A la fin de la partie un son de victoire ou de défaite est lancé. Ces sons ne fonctionnent que sur MacOs.

Pour les faire fonctionner nous avons utilisé les modules préinstallés de python qui sont « os », « subprocess » et « signal ». Les commandes ont été trouvées sur différents forums sur internet.

Les difficultés rencontrées :

Durant ce projet nous avons rencontré de nombreuses difficultés, notamment sur la conception de l'IA avec une stratégie gagnante.

En effet le programme doit s'adapter à l'utilisateur qui peut choisir entre deux difficultés et deux modes de jeu différents. On se retrouve donc avec un nombre important de fonctions et de lignes à gérer. Nous nous sommes vite rendu compte que l'utilisation de module était indispensable pour s'y retrouver. Il fallait retranscrire en code le raisonnement d'un humain ce qui fut compliqué à mettre en place compte tenu du peu d'expérience que nous avions sur Python. La réelle difficulté était de faire en sorte que le programme s'adapte à la situation du jeu et que l'ordinateur tire le nombre optimal d'allumettes à chaque tour.

Nous avions pour objectif de jouer un extrait de la musique de « Game of thrones » , cependant nous avons rencontré des difficultés à l'installation du module pygame qui permet de gérer l'audio. Nous nous sommes donc orientés vers le module subprocess et os pour lancer le fichier audio, cependant le seul moyen de l'arrêter était de « kill » le programme. Nous avons alors dû créer 2 versions, une avec la musique et l'autre sans la musique, car sous Linux, la commande « kill » fermait la session.

Avec plus de temps nous aurions pu programmer une difficulté « moyenne » pour l'IA, une option afin de choisir ses propres règles du jeu et même essayer de faire jouer deux humains l'un contre l'autre plutôt que contre un ordinateur.