3e édition

Snak'O'tron

But du jeu

Ce jeu reprend les base du célèbre jeu **Snake** en plaçant dans une arène deux serpents devant se déplacer en évitant les murs, l'adversaire ainsi que lui-même. Vous devrez ainsi écrire le code permettant de contrôler l'un de ces serpents et faire en sorte que celui-ci reste en vie le plus longtemps pour gagner la partie.
C'est un concours de programmation, où on fera s'affronter les programmes de tous les participants dans un grand tournoi. Que le meilleur gagne !!

Règles de base

- Taille de l'arène : l'arène est de taille variable **L x H**, transmise par le serveur au début du jeu
- Position de départ : le **joueur 0** (qui joue le premier) commence à la position **(2, H/2)** et le **joueur 1** à la position **(L-3, H/2)**.
- Changement de taille du serpent : Chaque serpent gagne une case de taille tous les **10 tours** (tour 0 compris).

⚠ Tous les mouvements seront exécutés sur un serveur distant qui renverra les informations de mises à jour aux joueurs via l'API fournit. Vous ne pourrez donc pas exécuter de mouvements invalides.

Planning

- 28 novembre : Présentation et salle informatique (327) disponible (session d'aide)
- 5 décembre : Salle informatique (327) disponible (session d'aide)
- **12 décembre : Déroulement du tournoi (salle 327)**

Utilisation de l'API

Pour communiquer avec le serveur de jeu vous aurez accès à une API (Application Programming Interface) vous permettant d'envoyer et de recevoir les informations utiles au bon déroulement de la partie.

Cette API est uniquement disponible en **C** et en **Python**, où on vous fournit le code source nécessaire pour communiquer avec le serveur (quelques fonctions simples pour initier la communication, démarrer une partie, jouer un coup, recevoir le coup de l'adversaire, etc.)

Il est possible de participer au concours en utilisant un autre langage de programmation, mais dans ce cas, vous devrez réécrire l'API dans ce langage en vous basant sur les fichiers C fournis (à vous de vérifier le bon fonctionnement de votre code sur les machines de l'école).

API en C :

Utilisez les fonctions de l'api venant du fichier **snakeAPI.h** (les fichiers snakeAPI.h, snakeAPI.c, clientAPI.h et clientAPI.c seront requis)

```
void connectToServer(char* serverName, int port, char* name)
```

Initialise la connexion avec le serveur ou quitte le programme si elle ne peut être établit.

- **serverName** : Adresse du serveur (Serveur de test : polydev.cia-polytech-sorbonne.fr)
- **port** : Port du serveur de jeu (Serveur de test : **8080**)
- **name** : Nom du joueur (⚠ Le nom doit être unique et identifiable pour chaque participant)

```
void closeConnection()
```

Ferme la connection avec le serveur (doit être effectuée pour éviter tout problème)

```
void waitForSnakeGame(char* gameType, char* gameName, int* sizeX,
                      int* sizeY, int* nbWalls)
```

Fonction bloquante attendant que le serveur démarre le jeu (attente des participants). Cette fonction donne les informations nécessaires au démarrage de la partie, en remplissant les variables associées.

- **gameType** : Commande de création de la partie : on peut préciser si on veut participer à un tournoi ou jouer contre un bot existant. On peut peut aussi préciser des paramètres spéciaux afin de reproduire certaines conditions. Chaine de caractère sous la forme "COMMANDE key1=val1 key2=val2 …" (exemple : "RANDOM_PLAYER difficulty=2 timeout=100 seed=123 start=0")

    - la commande peut être un nom de bot (ici **RANDOM_PLAYER** ou **SUPER_PLAYER**) ou bien “TOURNAMENT xxxxx“ pour participer au tournoi xxxxx (il vous sera possible de créer un tournoi depuis le serveur)
    - les clés peuvent prendre les valeurs suivantes:
    - *difficulty* : indique le niveau de difficulté entre 0 et 3 (relatif au nombre de murs dans l’arène)
    - *timeout* indique le temps (en seconde) avant que le serveur nous considère comme perdant (on peut le fixer quand on joue contre un bot pour s'entraîner, il sera fixer à 10s pour le tournoi)
    - *start* indique le numéro du joueur qui commence (ici le Joueur 0, c’est à dire vous)
    - *seed* initialise à une certaine valeur la graine du générateur de nombre aléatoire. Fixer cette valeur permet de rejouer toujours la même partie (utile quand on déboggue son programme pour rejouer les mêmes parties)
- **gameName** : La variable dans laquelle la fonction donnera le nom de la partie
- **sizeX** : La variable dans laquelle la fonction donnera la largeur de l'arène
- **sizeY** : La variable dans laquelle la fonction donnera la hauteur de l'arène
- **nbWalls** : La variable dans laquelle la fonction donnera le nombre de murs de l'arène (excluant les bords de l’arène)

`int getSnakeArena(int* walls)`

Récupère l’emplacement des murs de l’arène et le joueur qui commence.

- **walls** : Reçoit un tableau de taille (**4 x nbWalls**) contenant l’emplacement murs sous la forme suivante (case1.x, case_1.y, case2.x, case2.y)
- *Retourne 0 si vous commencez, 1 sinon*

`t_return_code getMove(t_move* move)`

Attend le mouvement de l’adversaire depuis le serveur.

- **move** : La variable dans laquelle la fonction donnera la direction du mouvement de l’adversaire (NORTH, WEST, SOUTH, EAST)
- Retourne le type de mouvement *(NORMAL_MOVE, WINNING_MOVE, LOOSING_MOVE)* qui nous indique l’adversaire a gagné, perdu ou si le jeu continue

`t_return_code sendMove(t_move move)`

Envoie votre mouvement au serveur.

- **move** : La direction dans laquelle vous souhaitez vous déplacer (NORTH, WEST, SOUTH, EAST)
- Retourne le type de mouvement *(NORMAL_MOVE, WINNING_MOVE, LOOSING_MOVE)* pour savoir si ce coup nous a permi de gagner, de perdre ou si le jeu continue

`void printArena()`

Affiche l'état actuel de l'arène qui est contenu dans le serveur.

`void sendComment(char* comment)`

Envoie un commentaire au serveur et aux autres joueurs (utile pour vanner l'adversaire).

API en python :

Utilisez la class **SnakeAPI** venant de **snakeAPI.py** (les fichiers snakeAPI.py et clientAPI.py seront requis)

`connectToServer(serverName, port, name)`

Initialise la connexion avec le serveur ou quitte le programme si elle ne peut être établit.

- **serverName** : Adresse du serveur (Serveur de test : polydev.cia-polytech-sorbonne.fr)
- **port** : Port du serveur de jeu (Serveur de test : **8080**)
- **name** : Nom du joueur (⚠ Le nom doit être unique et identifiable pour chaque participant)

`closeConnection()`

Ferme la connection avec le serveur (doit être effectuée pour éviter tout problème)

`gameName, sizeX, sizeY, nbWalls = waitForSnakeGame(gameType)`

Fonction bloquante attendant que le serveur démarre le jeu (attente des participants). Cette fonction donne les informations nécessaires au démarrage de la partie, en remplissant les variables associées.

- **gameType** : Commande de création de la partie : on peut préciser si on veut participer à un tournoi ou jouer contre un bot existant. On peut peut aussi préciser des paramètres spéciaux afin de reproduire certaines conditions. Chaine de caractère sous la forme "COMMANDE key1=val1 key2=val2 …" (exemple : "RANDOM_PLAYER difficulty=2 timeout=100 seed=123 start=0")
  - la commande peut être un nom de bot (ici **RANDOM_PLAYER** ou **SUPER_PLAYER**) ou bien "TOURNAMENT xxxxx" pour participer au tournoi xxxxx (il vous sera possible de créer un tournoi depuis le serveur)
  - les clés peuvent prendre les valeurs suivantes:
  - *difficulty* : indique le niveau de difficulté entre 0 et 3 (relatif au nombre de murs dans l'arène)
  - *timeout* indique le temps (en seconde) avant que le serveur nous considère comme perdant (on peut le fixer quand on joue contre un bot pour s'entraîner, il sera fixer à 10s pour le tournoi)

    - *start* indique le numéro du joueur qui commence (ici le Joueur 0, c'est à dire vous)
    - *seed* initialise à une certaine valeur la graine du générateur de nombre aléatoire. Fixer cette valeur permet de rejouer toujours la même partie (utile quand on déboggue son programme pour rejouer les mêmes parties)
- **gameName** : La variable dans laquelle la fonction donnera le nom de la partie
- **sizeX** : La variable dans laquelle la fonction donnera la largeur de l'arène
- **sizeY** : La variable dans laquelle la fonction donnera la hauteur de l'arène
- **nbWalls** : La variable dans laquelle la fonction donnera le nombre de murs de l'arène (excluant les bords de l'arène)

`walls, player = getSnakeArena()`

Récupère l'emplacement des murs de l'arène et le joueur qui commence.

- **walls** : Reçoit un tableau de taille (**4 x nbWalls**) contenant l'emplacement murs sous la forme suivante (case1.x, case_1.y, case2.x, case2.y)
- *Retourne 0 si vous commencez, 1 sinon*

`move, ret = getMove()`

Attend le mouvement de l'adversaire depuis le serveur.

- **move** : La variable dans laquelle la fonction donnera la direction du mouvement de l'adversaire (NORTH = 0, WEST = 3, SOUTH = 2, EAST = 1)
- Retourne le type de mouvement *(NORMAL_MOVE = -1, WINNING_MOVE = 1, LOOSING_MOVE = 0)* qui nous indique l'adversaire a gagné, perdu ou si le jeu continue

`ret = sendMove(move)`

Envoie votre mouvement au serveur.

- **move** : La direction dans laquelle vous souhaitez vous déplacer (NORTH = 0, WEST = 3, SOUTH = 2, EAST = 1)
- Retourne le type de mouvement *(NORMAL_MOVE = -1, WINNING_MOVE = 1, LOOSING_MOVE = 0)* pour savoir si ce coup nous a permi de gagner, de perdre ou si le jeu continue

`printArena()`

Affiche l'état actuel de l'arène qui est contenu dans le serveur.

`sendComment(comment)`

Envoie un commentaire au serveur et aux autres joueurs (utile pour vanner l'adversaire).