

# LP MiAR – Programmation répartie (UE03EC2)

Olivier Boutin – olivier.boutin@univ-nantes.fr

Réalisation du « *net code* » d'un jeu vidéo simple  
Présentation du projet

## 1 Objectif du projet

Le code d'un jeu vidéo très simple impliquant quatre personnages vous est fourni. Dans la version actuelle de ce code, un personnage est contrôlé au clavier et les trois autres sont contrôlés par un générateur de nombres aléatoires. Votre objectif sera de permettre que ces trois derniers personnages soient contrôlés par d'autres joueurs à travers le réseau.

## 2 Description du jeu fourni

Le jeu qui vous est fourni est un jeu de course à pied dans lequel il faut appuyer le plus vite possible sur la barre espace pour faire arriver son personnage le premier au bout du parcours. Il se déroule en 5 étapes, qui sont décrites dans cette partie telles qu'elles fonctionnent actuellement. Chacune de ces étapes devra être plus ou moins fortement modifiée au cours de ce projet pour permettre le jeu par le réseau.

Le code vous sera distribué en début de première séance.

Pour tester le jeu il suffit de récupérer ce code et (à condition d'avoir Go sur votre machine) d'utiliser `go build` dans le répertoire où se trouve le fichier `go.mod`. Ceci construit un exécutable `course` qui permet de lancer le jeu.

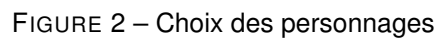
### 2.1 Écran d'accueil

Au lancement du jeu, un écran d'accueil s'affiche (figure 1). Il suffit d'appuyer sur espace pour passer à l'étape suivante.

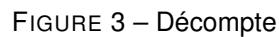


FIGURE 1 – Écran d'accueil

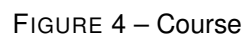
La deuxième étape est la sélection de quatre personnages (figure 2). Le joueur peut sélectionner l'un d'entre eux (P0) à l'aide des flèches gauche et droite puis valider son choix avec la touche espace. Les trois autres personnages sont sélectionnés au hasard. Une fois les quatre personnages sélectionnés, le jeu passe à l'étape suivante.



La troisième étape est un décompte avant le début de la course. Le joueur n'a rien à faire durant cette étape, on voit simplement 3, 2, 1 s'afficher à l'écran (figure 3). Une fois le décompte terminé le jeu passe à l'étape suivante : la course.



La quatrième étape est la course en elle même (figure 4). Le joueur contrôle le personnage qu'il a préalablement sélectionné en appuyant le plus vite possible sur espace jusqu'à atteindre la ligne d'arrivée. Les autres personnages sont contrôlés aléatoirement. Une fois les quatre personnages arrivés, la course se termine et le jeu passe à l'étape suivante.



La dernière étape est celle des résultats. Le temps de course de chacun des personnages est affiché, dans l'ordre de leur arrivée (figure 5). Après cela, le joueur peut appuyer sur espace pour redémarrer une course (retour à l'étape du décompte).


Press SPACE to restart		
1. P0	3+154	
3. P1	4+394	
2. P2	3+821	
4. P3	4+838	

FIGURE 5 – Résultats

### 3 Description du jeu attendu

Il vous faudra modifier le code fourni afin de permettre à quatre joueurs de contrôler chacun un personnage du jeu et de jouer ensemble à travers le réseau. Le principe sera que chaque joueur démarre le jeu sur sa machine (en indiquant l'adresse d'un serveur) et contrôle un personnage (le plus simple étant qu'il contrôle toujours P0) comme actuellement. Par contre, au lieu d'être contrôlés aléatoirement, les trois autres personnages reflètent ce que font les trois autres joueurs sur leurs instances respectives du jeu. Ceci implique que le jeu de chaque joueur retransmette à destination des autres jeux, via un serveur, des informations sur ce que fait le joueur.

### 4 Ce que vous aurez à faire

Pour obtenir la moyenne, il vous faut écrire un code Go qui réponde, au minimum, aux demandes exprimées dans les deux premiers guides. Ce code reposera sur les sources qui vous sont fournies.

Votre code devra bien sûr être commenté pour indiquer l'utilité de chacune de vos fonctions (sur le modèle du code fourni) et, si besoin, décrire l'implémentation des plus complexes d'entre elles.

Pour obtenir une note supérieure à la moyenne il vous faudra ajouter des améliorations, en particulier parmi celles qui sont suggérées dans le 3<sup>e</sup> guide (plus vous en ajouterez, plus votre note sera élevée).

En plus du bon fonctionnement de votre programme, la qualité de votre code et de vos commentaires pourront bien sûr aussi améliorer ou réduire votre note. Du moment que le jeu tourne toujours de manière fluide (c'est-à-dire aux alentours de 60 frames par seconde) les performances de votre code ne seront par contre pas prises en compte de manière négative.

### 5 Hypothèses simplificatrices

Pour simplifier l'implémentation on considérera les hypothèses suivantes.

- Aucun joueur n'est déconnecté durant toute la durée du jeu.
- Il y a un faible délai entre l'émission et la réception d'un message à travers le réseau.
- Aucun message envoyé sur le réseau n'est perdu.
- Les messages arrivent dans l'ordre où ils ont été émis (si  $m_1$  puis  $m_2$  sont envoyés par un programme  $p_1$  à destination d'un programme  $p_2$ , alors  $p_2$  recevra toujours  $m_1$  avant  $m_2$ ).

### Crédits de ce document

Cette œuvre est largement basée sur les supports de cours réalisés par Loïc Jezequel, pour l'enseignement de la ressource 3.05 (Programmation système) en 2<sup>e</sup> année de BUT, à l'IUT de Nantes. Elle est par ailleurs mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale 4.0 International. Pour voir une copie de cette licence, visitez <https://creativecommons.org/licenses/by-nc/4.0/deed.fr>.