IN405 – Système d'exploitation
Projet – Sea of Devs

S. Gougeaud

2018/2019

1 Objectif

Le projet consiste en l'implémentation d'une bataille navale. Un nombre donné de joueurs, chacun représenté par un navire, doivent parcourir un océan à la recherche des autres pour être le dernier survivant. La vie du navire est symbolisée par une valeur C appelée *coque*, et son endurance par une valeur K appelée *kerosene*. Chaque action effectuée par le navire aura une incidence sur sa *coque* et son *kerosene*. Si l'une de ces ressources vient à manquer, le navire coule. Le jeu fonctionne en tour par tour : durant un tour, (1) chaque joueur détermine la prochaine action à effectuer puis en informe le serveur ; (2) ce dernier effectue les actions en fonction de différents critères (priorité par exemple) ; (3) puis il informe les joueurs de leur nouvel état (nouvelles coordonnées, perte de points de *coque* ou de *kerosene*, etc.). Le jeu se termine lorsqu'il ne reste plus qu'un seul survivant, ou si un nombre de tours prédéfini a été atteint.

Chaque projet doit se faire en **binôme**. Le rendu du projet doit se faire avant le **21 Avril 2019 à 23h59** (heure d'été, France métropolitaine) par mail (avec en copie les membres du groupe) à l'adresse suivante, en fonction de votre chargé de TD :

`hugo.bollore@uvsq.fr`
`kevin.camus@uvsq.fr`
`mohammed-salah.ibnamar@uvsq.fr`
`mathieu.tribalat@uvsq.fr`

Le mail doit contenir une archive tar.gz composée :
- des sources du projet faites en langage C ;
- des directives de compilation pour le projet (script, Makefile) ;
- d'un rapport de deux/trois pages décrivant brièvement votre implémentation ainsi que les problèmes que vous avez rencontrés.

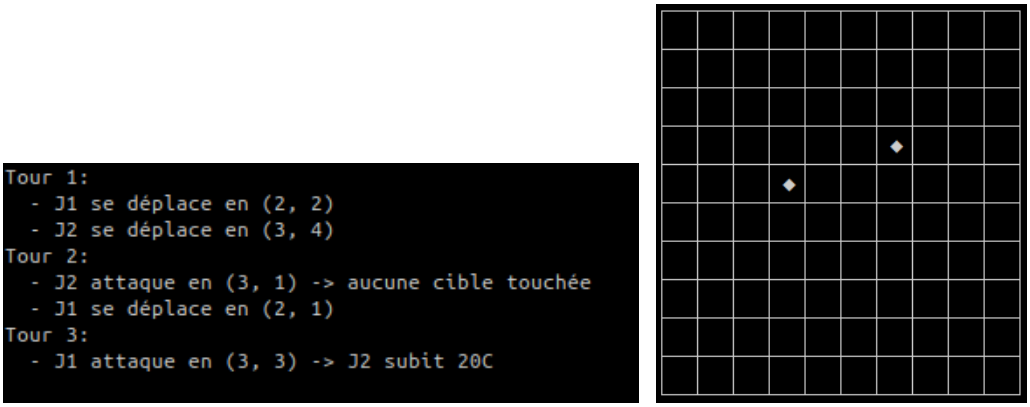
Chaque binôme sera soumis à une soutenance de 10 à 15 minutes durant laquelle il devra présenter son code, faire une démonstration et répondre aux questions qui lui seront posées.

2 Contenu du projet

Le projet est découpé en trois parties, chacune donné durant le semestre tel que vous ayez 2 à 3 semaines pour implémenter l'une des parties. La première partie nécessitera la connaissance du système de fichiers (TD 2-3). La seconde celle des processus et des tubes (TD 4). La dernière celle des threads (TD 5-6). Cet énoncé sera donc mis à jour durant le semestre. Une annonce sera faite lorsque la mise à jour sera opérationnelle.

2.1 Version Alpha : Serveur et gestion de la partie

Dans cette première version, vous devez implémenter de manière simple le serveur de jeu. Prenant en argument un fichier d'entrée, son rôle est de créer la carte navale fournie dans le fichier d'entrée et d'effectuer un nombre donné de tours de jeu, constitué d'actions pré-définies dans votre programme. Un exemple est donné dans la sous-section 2.1.2. Pour l'affichage du déroulement de la partie, vous pouvez opter pour un affichage en forme de log (cf figure 1a), soit un affichage de carte dynamique (cf figure 1b).



(a) Affichage de type log (b) Affichage de type carte

FIGURE 1 – Affichages possibles

Seules deux actions sont pour le moment possibles : le déplacement MOV et l'attaque ATK. Il est aussi possible de n'effectuer aucune action.

La gestion de la carte navale est fournie avec le projet dans l'archive navalmap.tar.gz. Cette archive est composée d'un fichier d'en-tête, de fichiers sources, d'un Makefile et d'un Readme. Les informations nécessaires à son utilisation sont décrites dans le fichier Readme.

2.1.1 Fichier d'entrée du programme

Le programme prend en entrée un fichier constitué de deux lignes :

1. la caractérisation de la carte, avec son type, sa taille en longueur et en hauteur.

```
1 #typeCarte;tailleX;tailleY
2 rectangle;10;10
```

2. la caractérisation de la partie, avec le nombre de joueurs, les valeurs initiales de *coque* et de *kerosene* et le nombre de tours maximum joués.

```
1 #nbJoueurs;Cmax;Kmax;nbTours
2 2;100;100;20
```

2.1.2 Exemple programme version Alpha

Soit une carte 10x10, et deux navires A et B placés initialement en (1,2) et (8,7) et possédant $50C$ et 100 K, les actions sont décrites dans l'encadré ci-dessous :

```
Tour  1 : A -> MOV (+1, 0) -- B -> MOV ( 0,+2)
Tour  2 : A -> MOV ( 0,-1) -- B -> MOV (+1, 0)
Tour  3 : A -> ATK ( 3, 3) -- B -> MOV ( 0,-2)
Tour  4 : A -> MOV (+2, 0) -- B -> MOV ( 0,-2)
Tour  5 : A -> MOV (+1, 0) -- B -> MOV (-2, 0)
Tour  6 : A -> ATK ( 7, 3) -- B -> MOV ( 0,-1)
Tour  7 : A -> MOV ( 0,+1) -- B -> MOV (-1, 0)
Tour  8 : A -> ATK ( 5, 6) -- B -> ATK ( 5, 2)
Tour  9 : A -> MOV ( 0,+2) -- B -> MOV ( 0,-1)
Tour 10 : A -> MOV (-2, 0) -- B -> ATK ( 5, 4)
Tour 11 : A -> ATK ( 6, 3) -- B -> MOV ( 0,+2)
Tour 12 : A -> ATK ( 6, 2) -- B -> ATK ( 4, 5)
```

2.2 Version Beta : Joueur et algorithme de décision

Dans cette seconde version, vous devez implémenter le côté joueur, ayant en charge un navire.

Le serveur et chaque joueur agiront dans des processus distincts. A l'initialisation de votre programme, une fois le fichier d'entrée lu, le processus principal générera n processus fils, avec n le nombre de joueurs demandés. Les communications joueur/serveur devront se faire par tube (nommé ou anonyme).

A la réception de la mise à jour de son état par le serveur, le joueur doit déterminer l'action pour le prochain tour qu'il devra transmettre au serveur. Vous pouvez dans un premier temps implémenter l'algorithme naïf suivant pour le choix de l'action :

```
si (position adverse non connue) ou (radar trop vieux)
    alors radar
sinon si (position adverse a portee d'attaque)
    alors attaque
sinon
    deplacement vers position adverse
```

Rien ne vous empêche d'écrire un algorithme plus efficace dans un second temps.

Une nouvelle action est ajoutée au jeu : le radar SCN. Il permet d'obtenir la position de l'ennemi le plus proche, ainsi que ses valeurs C et K.

Le serveur, une fois toutes les actions recueillies, procède au déroulement du tour de jeu, comme précédemment dans la version Alpha, puis informe les joueurs de leur nouvel état (valeurs C et K, position effective, cible touchée, etc.).

2.3 Version 1.0 : Dernières actions et équipe contre équipe

Dans cette dernière version, vous devez implémenter un mode *équipe contre équipe*.

En mode *seul contre tous*, chaque joueur était représenté par un processus. Ici, chaque équipe est représenté par un processus, et chaque joueur d'une même équipe, un thread créé par ce processus. Le fonctionnement du jeu est identique (chaque joueur communique son ordre du tour au serveur, qui exécute les ordres, etc.), à la différence que la mémoire partagée peut être utilisée pour permettre aux joueurs d'une même équipe de communiquer des informations.

Par exemple, si un thread A a trouvé la position d'un adversaire, alors il peut mettre en mémoire partagée ses coordonnées. Ainsi tous les threads faisant partie de son équipe, et étant à portée de cet adversaire peuvent l'attaquer.

Deux nouvelles actions sont ajoutées au jeu : la charge BST et la réparation RPR. Elles permettent respectivement de se déplacer plus loin dans une direction axiale et d'augmenter la valeur de sa coque C en échange de kérosène K.

2.4 Version 1.2 : Pour aller plus loin – Optionnel

Si vous avez le temps (et l'envie), vous pouvez également implémenter les fonctionnalités suivantes :

- Ajout des autres actions disponibles dans le glossaire.
- Amélioration de l'algorithme de décision du joueur.
- Possibilité que l'un des joueurs soit joué par vous (et non simulé).
- Ajout d'autres types de cartes (en se basant sur le code fourni dans la bibliothèque) et d'autres actions (que vous imaginerez).

3 Consignes d'implémentation

Les consignes suivantes doivent être respectées :

- l'appel au programme doit se faire comme suit :
  `./SoD fichier`
- les appels systèmes doivent être utilisés pour les fonctionnalités suivantes :
  - la gestion des fichiers d'entrée/sortie ;
  - la gestion des processus ;
  - la gestion des tubes.
- les constantes sont les seules variables globales autorisées ;
- chaque allocation doit être libérée avant la fin du programme ;
- chaque création de processus/thread doit être 'attendue'.

Rappel : Pour savoir si une fonction est un appel système ou non, il suffit de regarder le numéro d'indexage de cette dernière dans `man`. 2 signifie que c'est un appel système, 3 une fonction issue d'une bibliothèque.

4 Glossaire des actions

Il existe trois catégories d'actions dans Sea of Devs : les déplacements [M], les attaques [A] et les supports [S]. Nous définissons les règles suivantes :

- L'ordre de traitement des actions des joueurs dépend de leur catégorie. Sont d'abord traitées les attaques, puis les déplacements et enfin les supports.

— Si plusieurs actions font partie d'une même catégorie, elles sont alors traitées en même temps.
— Dans le cas où le mouvement de deux navires les amène sur la même case, les deux retournent à leur position de départ et perdent $10C$.
— Dans le cas où le mouvement d'un navire l'amène sur une case où se trouve déjà un autre navire, le premier retourne à sa position de départ et les deux perdent $5C$.

La figure 2 représente les différentes actions possibles ainsi que leur zone de portée.

Aucune action – NON Le joueur reste sur place et n'effectue aucune action. La non action utilise $1K$.

[A] Attaque – ATK Le joueur cible une case située à une distance comprise entre 2 et 4 de sa case initiale, infligeant $40C$ au navire présent. Les navires présents sur une case adjacente (en croix) à la case ciblée subissent $20C$. L'action utilise $5K$.

[A] Bombardier – BBS Le joueur cible une case située à une distance comprise entre 1 et 5 de sa case initiale, infligeant $30C$ au navire présent. L'action utilise $3K$.

[M] Charge – BST Le joueur avance sur une case située à une distance comprise entre 4 et 5 de sa case initiale. Cette case est obligatoirement alignée verticalement ou horizontalement avec la case initiale. Si un autre navire est située sur la case ciblée, il subit alors $50C$. Le navire du joueur actif subit $5C$. L'action utilise $3K$.

[M] Déplacement – MOV Le joueur avance sur une case située à une distance comprise entre 1 et 2 de sa case initiale. L'action utilise $2K$.

[M] Mine – MIN Le joueur dépose sur sa case initiale une mine qui infligera $20C$ au prochain navire s'arrêtant dessus. Puis le joueur avance sur une case située à une distance de 1 de sa case initiale. L'action utilise $3K$.

[S] Radar – SCN Le joueur récupère la position ainsi que la valeur C du navire ennemi le plus proche. L'action utilise $3K$.

[S] Radar+ – SC+ Le joueur récupère l'ensemble des entités situées à une distance comprise entre 1 et 3 de sa case initiale. L'action utilise $5K$.

[S] Réparation – RPR Le navire du joueur actif récupère $25C$. L'action utilise $20K$.

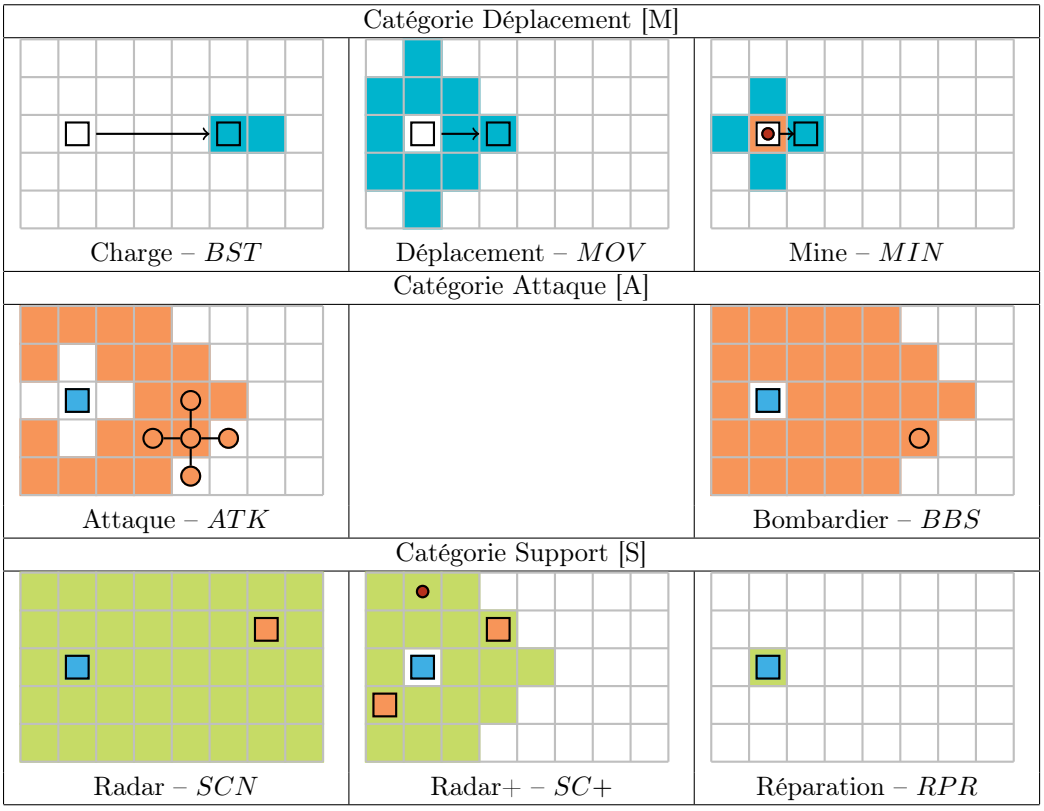


FIGURE 2 – Actions de jeu