Hes·so FRIBOURG/FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Systèmes embarqués

Projet Intégré

Analyse préliminaire

Auteur :

Romain Froidevaux, T-2a, Groupe 5 romain.froidevaux@edu.hefr.ch

Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Mars 2014

Table des matières

I. Introduction

Dans le cadre du cours de Systèmes Embarqués 2, nous devons réaliser un projet par groupe de 3 personnes. Cela nous permettra de mettre en application toute la matière vue au cours théorique ou en travaux pratiques.

Chacun des membres du projet a des tâches bien définies. Ce rapport a pour but de nous faire analyser nos différents points pour nous mener plus facilement à la phase de conception.

Mes points sont les suivants :

- Afficher les différentes vues sur l'écran LCD de la cible
- Implémenter les commandes de la Shell

Ainsi à la fin de cette analyse, j'aurai une vue plus précise des possibilités d'implémentation de mes fonctions sur l'i.MX27.

II. Rappel du projet

Toute la classe a une base commune pour le projet avec des points obligatoires :

- Affichage d'informations sur l'écran LCD
- Lancement des applications via l'écran tactile
- Affichage en temps réel de la température sur des 7 segments
- Réalisation d'un chronomètre avec les boutons poussoirs *(via interruptions)*
- Implémentation de la commande Ping pour vérifier la connectivité à distance
- Création d'un jeu multi-joueurs au travers du réseau
- Implémentation d'une Shell pour lancer/stopper/lister les applications
- Utilisation d'un noyau temps réel coopératif

Dans mon groupe, composé de L. Gremaud, D. Rossier et moi-même, nous avons choisi de développer le jeu de la bataille navale en multi-joueurs. Nous devrons donc gérer les tours de jeu et afficher en alternance la grille du joueur 1 et celle du joueur 2.

Comme fonctionnalités optionnelles qui seront développées selon notre avancement dans le projet, nous avons rajouté :

- Deuxième jeu : Tic Tac Toe
- Application de visualisation des scores
- Utilisation d'un noyau préemptif

Pour connaître la répartition des tâches précise, se référer au cahier des charges.

III. Architecture générale de notre projet

Afin que tout le monde puisse coder en parallèle ses composants et minimiser les problèmes d'interaction entre nous tous, nous avons décidé d'appliquer un modèle semblable au design pattern MCV. Ainsi dans la phase de spécification, nous

définirons très précisément quelles sont les méthodes et paramètres utilisés pour l'interaction entre les composants.

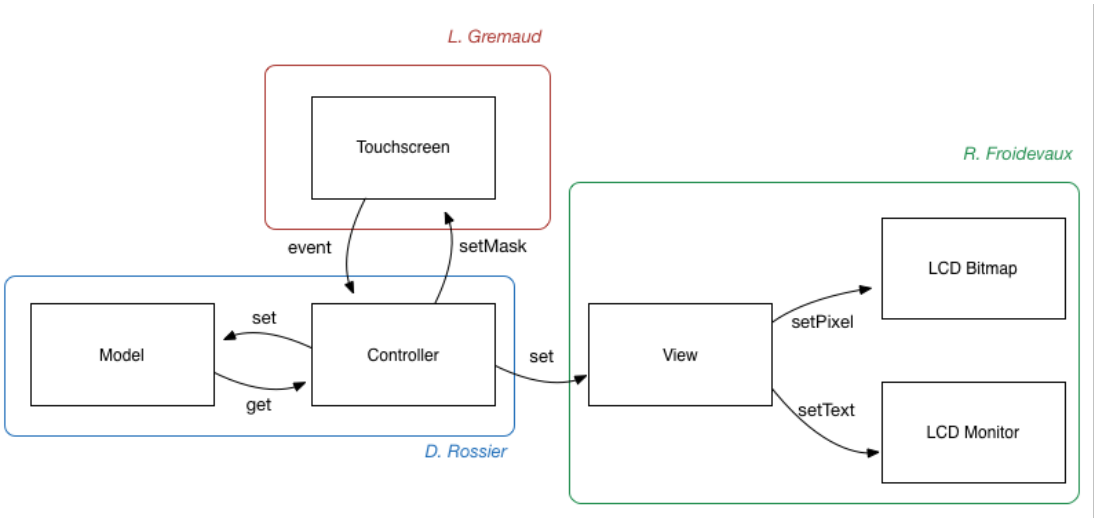

Fig. 01 : Modèle de développement

De plus avec une structure de développement telle que celle-ci, tous nos composants seront développés de manière générique et nous pourrons facilement les réutiliser plus tard dans d'autres implémentations.

IV. Interfaces graphiques

a. Contexte

La partie des interfaces graphiques est un point clé dans notre application. C'est au travers des vues que l'utilisateur va naviguer dans le système.
C'est sur la base de ces vues également que l'écran tactile sera calibré.

Au sein de notre programme, les interfaces graphiques ont un comportement totalement passif. Elles seront simplement gérées par le Contrôleur qui va faire modifier les vues selon les différentes phases du programme.

Dans ce cas présent, l'affichage des informations sur l'écran devra également être *relativement* rapide. Un utilisateur s'attend à ce que son action soit répercutée rapidement sur la vue sans quoi il pourrait penser que le programme s'est planté.

b. Croquis

1. Menu principal

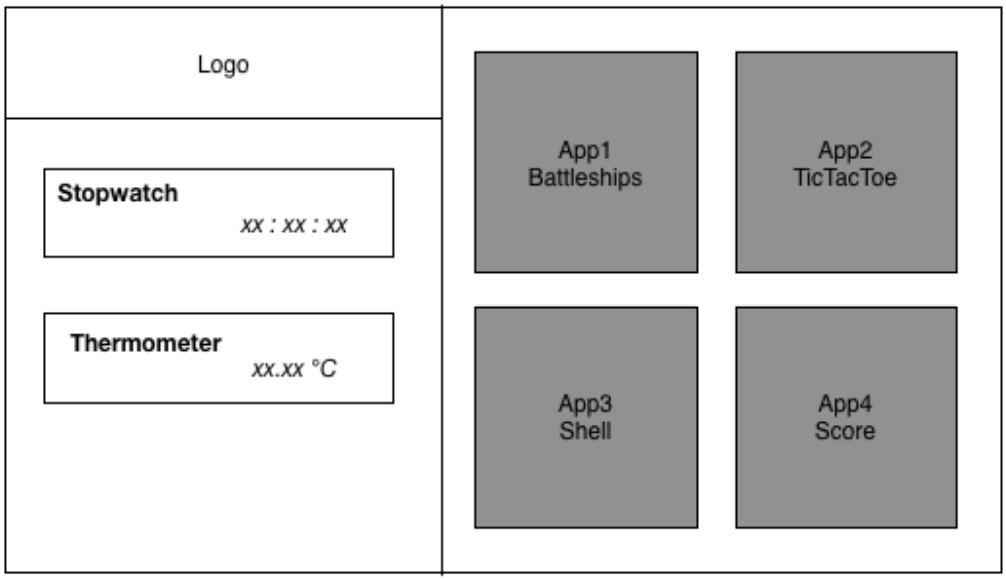

Fig. 02 : Croquis du menu principal

Le menu principal disposera de 4 boutons tactiles *(en gris sur le croquis)* afin de permettre de lancer les différentes applications.

La partie de gauche sera dynamique mais non tactile. Le chronomètre se lancera/s'arrêtera au moyen de boutons *(via des interruptions sur le GPIO)* et le thermomètre affichera la température en temps réel avec une précision de 0.5 °C.

2. Bataille navale : Positionnement des bateaux

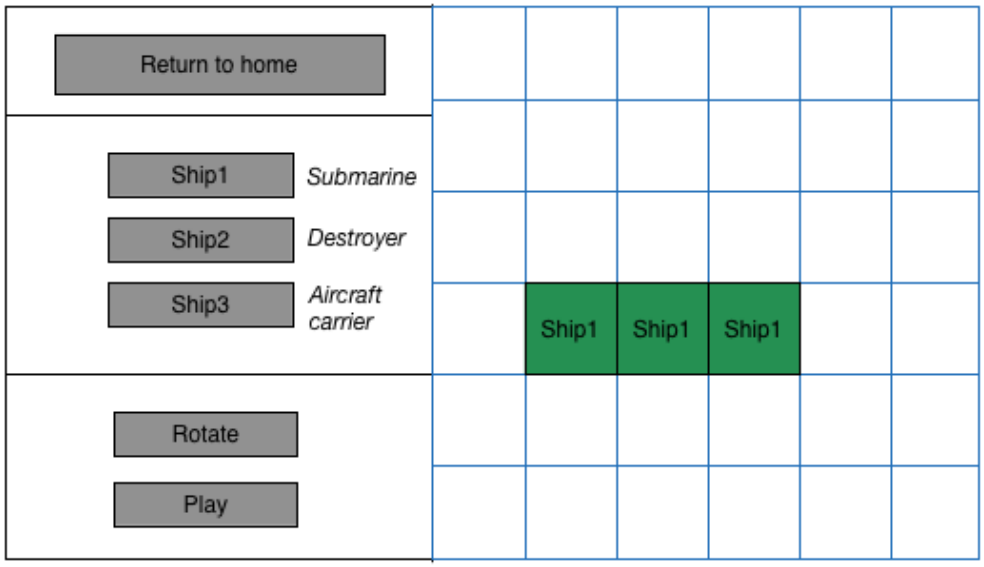

Fig. 03 : Positionnement des bateaux

Lors de l'étape de positionnement des bateaux, l'écran affichera sur la partie de droite une grille *(bleue = grille du joueur)*. Sur la partie de gauche, apparaitront 6 boutons tactiles :

- Return to Home — Permet au joueur de retourner à l'écran d'accueil. Cela implique la fin immédiate de la partie.
- Ship1 / Ship2 / Ship3 — En appuyant sur l'un de ces boutons, le joueur sélectionnera le bateau en question. Il devra directement ensuite choisir une case dans la grille ce qui aura pour effet de colorer les cases en vert afin de marquer que le bateau est posé.

  Le contrôleur se chargera bien sûr de vérifier que le positionnement est autorisé à cet endroit.
- Rotate — Permet au joueur de faire une rotation de son bateau *(Horizontal → Vertical et inversement)*
- Play — Une fois que tous ces bateaux sont positionnés, l'utilisateur doit appuyer sur `Play` pour indiquer qu'il est prêt à jouer.

3. Bataille navale : Tour du joueur de la cible

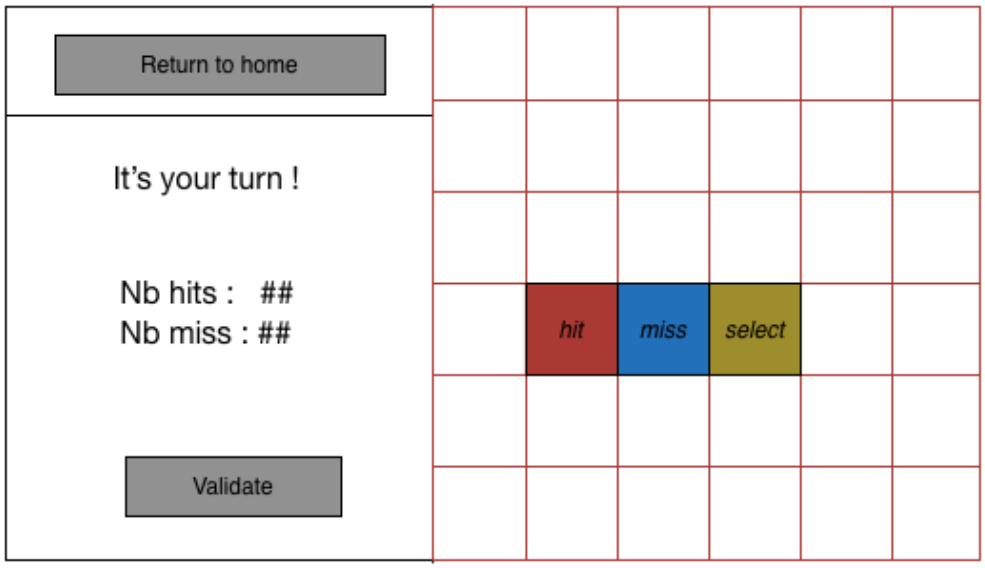

Fig. 04 : Au tour du joueur de la cible

Sur la partie de droite, la grille s'affiche en rouge pour indiquer qu'il ne s'agit pas de sa propre grille, mais de celle de son ennemi sur laquelle il doit viser.

Pour jouer, le joueur doit appuyer sur une case. Celle-ci s'affichera en jaune pour indiquer qu'elle est sélectionnée. Il peut ainsi changer d'avis s'il le souhaite en en sélectionnant une autre…

Une fois son choix définitif, il doit appuyer sur le bouton `Validate`. La case sélectionnée changera de couleur et deviendra rouge s'il a touché un bateau ou bleue s'il est à l'eau. Au même moment, les deux compteurs de la partie de gauche de l'écran s'incrémenteront.

4. Bataille navale : Tour du joueur distant (ennemi)

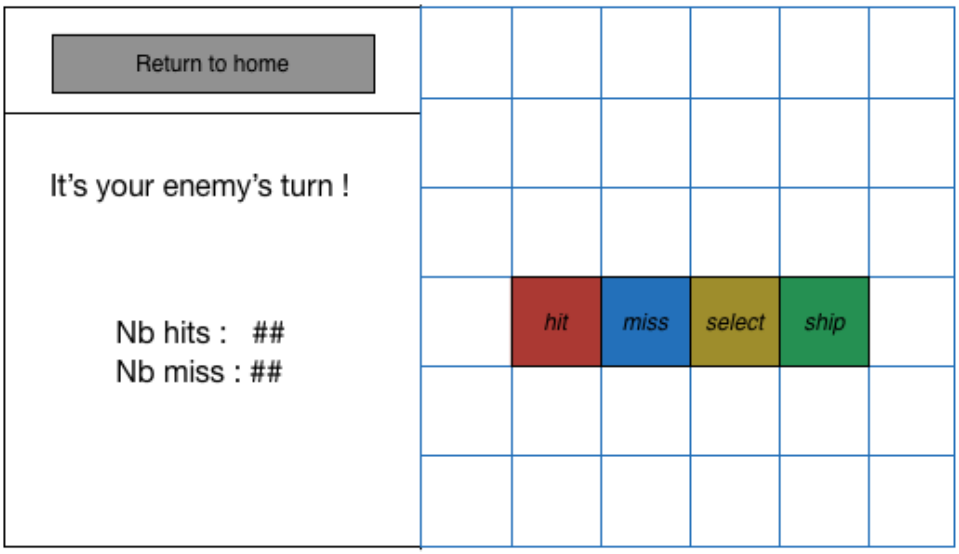

Fig. 05 : Au tour de l'ennemi

Lorsque c'est au tour de l'ennemi de jouer, la grille s'affiche en bleu sur la partie de droite. Cela indique au joueur qu'il s'agit de sa propre grille. La position des bateaux du joueur est signalée à l'aide des cases vertes.

Lorsque l'ennemi sélectionne une case, celle-ci s'affiche en jaune tant qu'il n'a pas validé son choix. S'il change le positionnement, bien sûr que la case jaune se déplace.
Une fois qu'il a appuyé sur le bouton `Validate`, la case sélectionnée devient rouge ou bleue selon qu'il ait touché un bateau respectivement rien touché.

A tout moment de la partie, les joueurs ont la possibilité de revenir au menu principal avec le bouton `Return to Home`, ce qui a pour effet d'annuler la partie en cours chez les deux joueurs.

5. Bataille navale : Score à la fin d'une partie

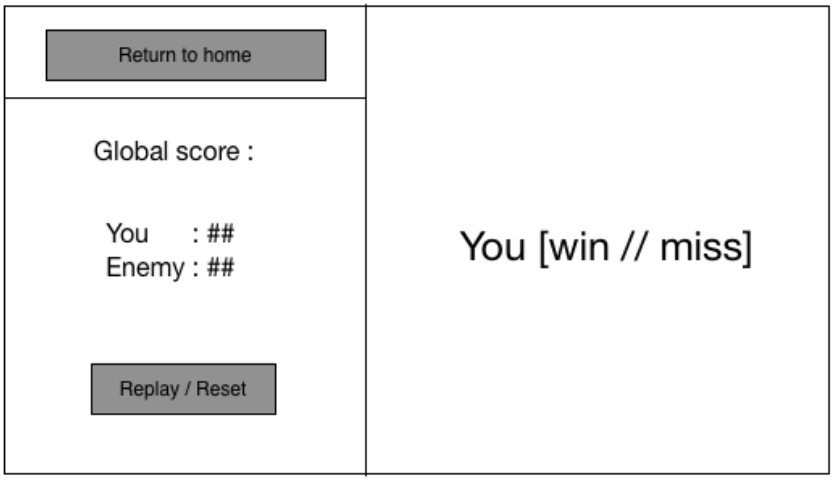

Fig. 06 : Score à la fin de partie

A la fin de la partie, les deux joueurs voient s'afficher sur leur écran le récapitulatif des scores.

Dans la partie droite de l'écran, l'énoncé `You miss` ou `You win` selon que le joueur ait remporté respectivement perdu la partie.
Sur la partie de gauche, un récapitulatif global des scores. Chaque partie gagnée fait remporter un point au joueur.

Chaque joueur à la possibilité de retourner sur le menu d'accueil en appuyant sur le bouton `Return to Home`.

Les utilisateurs peuvent appuyer sur le bouton `Replay` s'ils désirent refaire une nouvelle partie.

6. Application Scores

L'affichage est identique à celui du point précédent *(Bataille navale : Score à la fin d'une partie)* aux différences près :

- Le texte `You win` ou `You miss` est basée sur les résultats globaux (ceux affichés dans la partie gauche de l'écran). De ce fait, on a l'apparition de la mention `Equality` en cas d'égalité des scores.

- Le bouton `Replay` devient `Reset`, qui permet de remettre à zéro les résultats globaux.

7. Application Shell

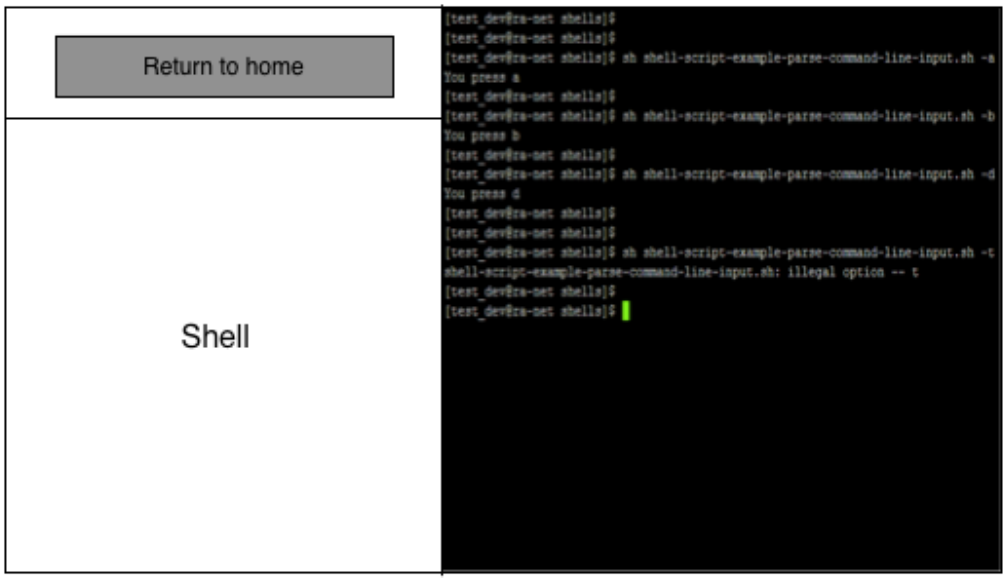

Fig. 07 : Application Shell

La vue est très basique. Sur la partie de droite, l'affichage des résultats des commandes entrées en Shell.
A gauche, le bouton tactile `Return to Home` permet de revenir au menu principal.

8. Jeu Tic Tac Toe : Déroulement

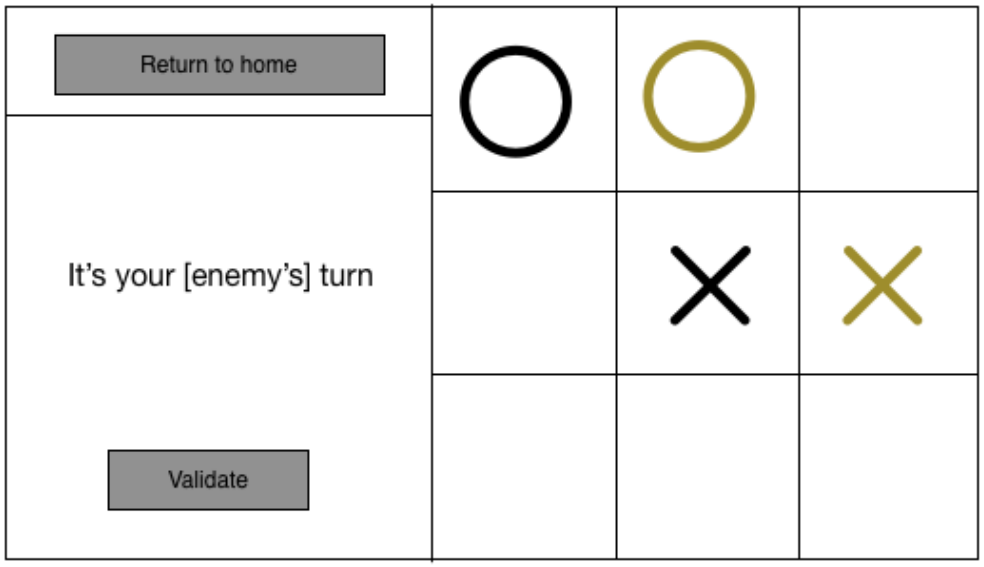

Fig. 08 : Vue du jeu Tic Tac Toe

Sur la partie de droite de l'écran, un damier *(3x3)* commun aux deux joueurs sera affiché.

A son tour, le joueur appuie sur une case pour positionner son élément *(croix ou rond selon le joueur)*. L'élément s'affiche en couleur jaune. Il peut le déplacer autant de fois qu'il veut. Une fois sûr de son choix, il peut le valider en appuyant sur le bouton `Validate` en bas à gauche de l'écran.

En haut à gauche, on retrouve le bouton `Return to Home` qui permet de retourner au menu principal en arrêtant la partie en cours pour les deux joueurs.

Le label du tournus passe de `It's your turn` à `It's your enemy's turn` selon les tours de jeu.

9. Jeu Tic Tac Toe : Affichage des scores fin de partie

La vue est strictement identique au point 5 *(Bataille navale : Score à la fin d'une partie)*, sauf que la notion d'égalité dans le label est aussi présente.

c. Etude d'implémentation

Pour afficher sur le LCD les différentes images, je vais utiliser le même procédé d'affichage que l'on a vu au TP N°5, soit tout d'abord convertir les images *(JPG ou PNG)* en fichiers XPM avant de les envoyer sur l'écran de la cible, pixel par pixel.

Pour les textes, nous avons à notre disposition une classe `font_8x8.h` qui contient les matrices unitaires de chacune des lettres et symboles courants. Ainsi, un 1 représente un pixel noir et un 0 un pixel blanc. Le tout représenté sur 8 bits en hauteur et 8 bits en largeur.

Finalement, pour les éléments telles que les grilles, une méthode spécialisée les dessinera pixel par pixel.

Très vite, je vais devoir faire face aux contraintes suivantes :

1. Eléments mobiles

Je pense principalement aux cases colorées. Etant donné que l'utilisateur pourra sélectionner une case, mais aussi en sélectionner une autre en s'il change son choix *(possible tant qu'il n'a pas validé)*, je dois pouvoir afficher puis faire disparaître la coloration de la case si nécessaire.

Après analyse du problème, je pense créer deux méthodes distinctes pour la bataille navale :

- `setColorCase (int x, int y, int colorId) ;`

Avec `x` et `y` les coordonnées de la case dans le tableau et `colorId` l'id représentant la couleur désirée *(vert, jaune, bleu ou rouge)*.

- `resetColorCase (int x, int y) ;`

Avec `x` et `y` les coordonnées de la case dans le tableau.

Ainsi, le contrôleur pourra simplement choisir de colorer une case de la couleur voulue et en cas de changement, de réinitialiser la couleur d'une case avant d'en recolorer une autre.

Pour le jeu Tic Tac Toe, le même principe serait appliqué, avec les deux méthodes suivantes :

- `setSymbolCase (int x, int y, int symbolId) ;`
- `resetSymbolCase (int x, int y) ;`

2. Alternation des vues dans le jeu de la bataille navale

Dans le jeu de la bataille navale, selon le tournus du jeu, il faut faire alterner les grilles sur l'écran *(Joueur 1 ←→ Joueur 2)*.

Ainsi, j'ai analysé deux possibilités :

1) Recréation de la vue depuis le modèle

Soit à chaque changement de tour, la vue est entièrement régénérée par le contrôleur avec les informations du modèle et les éléments sont réaffichés successivement.

L'avantage de cette solution est qu'elle est très simple à implémenter. Cependant, le fait d'aller rechercher à chaque fois l'intégralité des informations nous coûterait très cher en ressources et nécessiterait un nombre important d'instructions à réexécuter périodiquement.

2) Mémorisation parallèle des vues

L'idée serait de réserver un espace dans la mémoire de même taille que celui utilisé pour représenter les informations à l'écran.

Ainsi, à chaque changement de tour, le modèle n'aurait qu'à appeler une seule et unique méthode :

```
swapView() ;
```

qui échangerait les valeurs contenues dans la mémoire utilisées actuellement pour afficher les informations sur le LCD avec celles de l'autre joueur.

Cette solution est bien sûr plus complexe à implémenter au niveau de la vue et consomme plus d'espace mémoire, mais nécessite probablement moins de travail du contrôleur et sera beaucoup plus rapide. C'est donc cette solution que je préconise d'utiliser.

N.B. :

Comme nous l'avons vu en cours d'algorithmique l'année dernière, le fait de swapper deux éléments nécessite une espace mémoire servant de tampon (buffer), de taille égale aux des autres éléments.

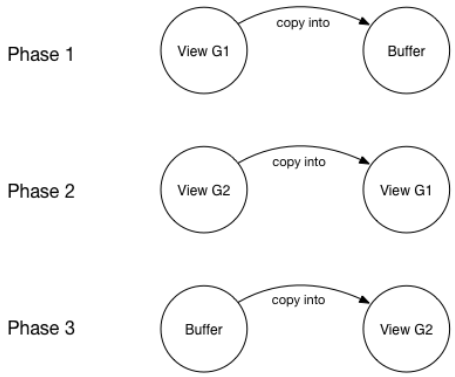

Fig. 09 : Algorithme de copie

3) Résultat des commandes Shell

Comme indiqué dans la donnée, l'utilisateur doit avoir la possibilité d'avoir affiché sur l'écran un output des commandes Shell passées au travers du port série de la cible.

Selon l'interface décrite au point IV.b.7, il serait tout à fait possible d'utiliser le même procédé d'affichage que partout, soir utiliser la classe `font_8x8.h` et d'afficher directement les textes sur l'écran LCD à l'emplacement désiré.

Cependant, afin d'utiliser une fois le concept de `Graphic Window on Screen`, il pourrait être judicieux d'afficher les textes non plus directement sur l'écran LCD mais sur un `Graphic Window` qui sera lui-même affiché sur l'écran LCD.

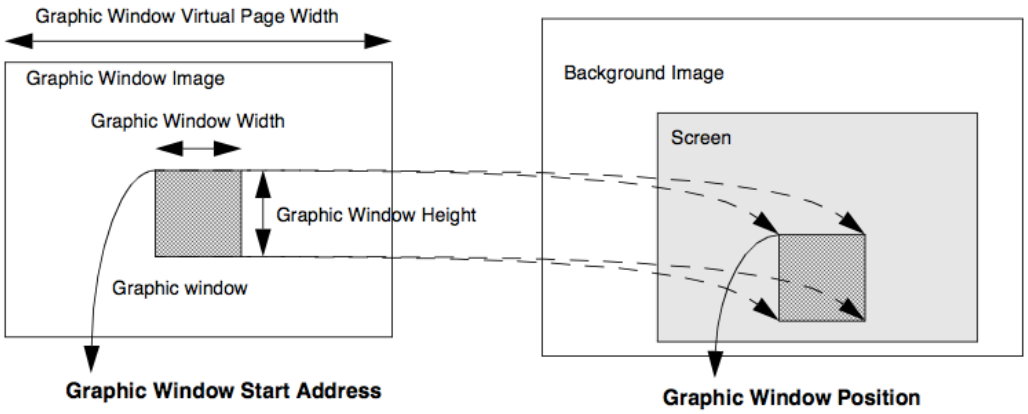

Fig. 10 : Concept du Graphic Window on Screen

Cela permettrait par exemple, dans une évolutivité éventuelle des fonctionnalités de notre cible, de ne plus créer une vue tout entière dédiée au Shell, mais d'afficher le résultat des commandes en *pop-up* par dessus la page du menu principal.

Dans notre cas cette variante n'étant pas nécessaire, elle sera implémentée si le temps le permet.

Pour l'affichage des informations, je devrais développer une méthode pour écrire du texte dans l'output Shell. Cette méthode devrait prendre en compte le fait que toutes les lignes *peuvent* être utilisées. Il faudra donc utiliser l'un des deux mécanismes suivants si nécessaire :

- Réinitialiser l'espace des écritures

  Très simple à implémenter, si l'output est plein, on supprime tout le texte puis on recommence à écrire de bas en haut. Cependant, ce n'est pas le comportement habituel d'une Shell.

- Décalage automatique des écritures

  Comme nous avons l'habitude de le voir dans une Shell, lorsqu'une ligne vient s'ajouter à l'affichage déjà complet, toutes les lignes sont rehaussées puis la dernière ligne vient s'ajouter à la suite. C'est cette technique qui sera privilégiée.

V. Implémentation des commandes Shell

a. Contexte

Depuis un ordinateur connecté sur le port série de la cible, il doit être possible de lister, lancer et arrêter les applications. L. Gremaud est en charge d'interpréter les commandes entrées par l'utilisateur dans le terminal, et moi de développer les méthodes permettant ensuite de traiter la commande et d'exécuter les actions nécessaires.

b. Listing des commandes à implémenter

- `ping` *`ipAddr`* Envoi un paquet ICMP vers un adresse IP afin de voir si le périphérique distant répond.
- `set ip` *`ipAddr`* Définition d'une adresse IP pour la cible
- `get ip` Retourne l'adresse IP actuelle de la cible
- `start` *`appName`* Lance l'application *appName*
- `stop` *`appName`* Arrête l'application *appName*
- `list app` Liste toutes les applications disponibles sur la cible
- `list run` Liste toutes les applications en cours d'exécution sur la cible
- `clr` Remet à zéro l'affichage (clean)
- `help` Affiche l'aide (toutes les commandes disponibles)

c. Etude d'implémentation

1. Ping

Envoi un ICMP à destination d'une adresse IP via une méthode permettant d'envoyer un paquet IP *(Méthode créée par D. Rossier qui développe les échanges UDP/IP)*.

Vérifier si un paquet vient en retour de cette même adresse IP afin de valider ou pas le ping. Pour ceci, j'utiliserai la méthode de D. Rossier afin de savoir si un paquet est arrivé sur l'interface réseau et pouvoir vérifier s'il s'agit d'une réponse au Ping.

La structure d'un paquet ICMP que pourra se charger de transporter le FEC sera la suivante :

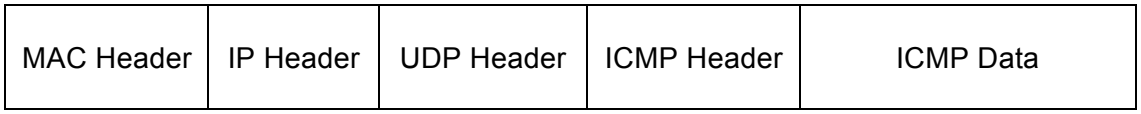

Fig. 11 : Schéma d'un paquet ICMP

D. Rossier développera la méthode permettant d'envoyer un paquet IP, donc d'encapsuler toutes les couches supérieures.

Je devrai donc encoder les parties suivantes du paquet ICMP, dont voici les schémas:

- UDP Header

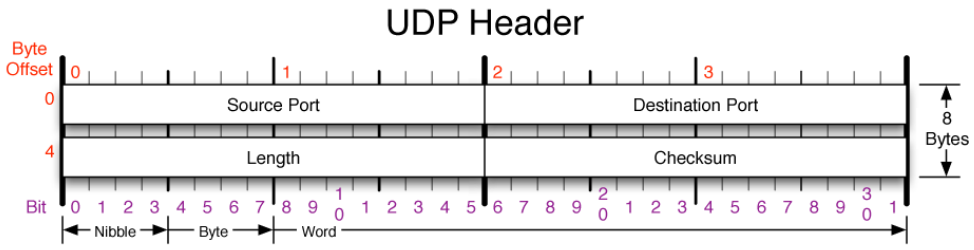

Fig. 12 : Schéma d'un en-tête UDP

Pour l'envoi, le port source sera généré aléatoirement entre 1024 et 65'535. Le port de destination quant à lui sera 1.

- ICMP Header & Data

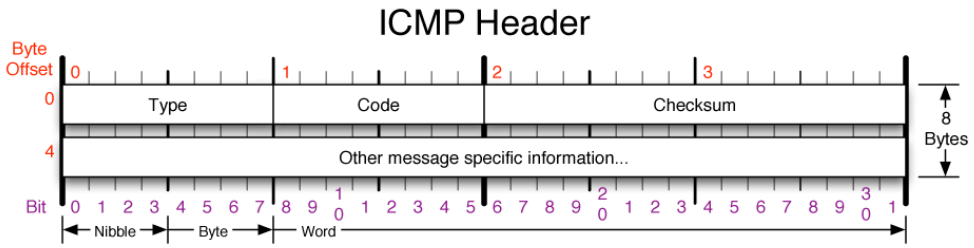

Fig. 13 : Schéma d'un en-tête ICMP

C'est avec le champ type que l'on peut définir la nature du paquet ICMP, donc entre-autre le résultat du Ping. Voici la description des principales combinaisons types/codes que l'on va implémenter :

Type	Code	Désignation
0	0	Echo Reply
3	1	Host Unreachable
8	0	Echo Request
11	0	Time Exceeded

2. Get/set IP

La commande `get ip` retournera simplement au Shell le contenu de l'endroit où sera stocké l'adresse IP de la cible *(p.ex. Une variable globale ou une structure selon choix de D. Rossier)*.
La commande `set ip ipAddr` quant à elle inscrira l'adresse IP passée en paramètre, pour autant que celle-ci soit une adresse IP conforme aux standards, dans son emplacement mémoire.

3. Clr

L'affichage des commandes shell étant géré par la vue, cette commande aura simplement pour effet d'appeler une méthode de type `resetShellCmdView()`, afin de supprimer toutes les anciennes écritures dans la Shell *(équivalent de la commande `clear` sur OS X)*.

4. Help

Le résultat étant simplement le listing de toutes les commandes disponibles sur notre cible, la commande `help` aura pour effet de retourner à la méthode de la vue responsable de l'écriture des caractères dans la Shell la liste des commandes implémentées.

5. List app

La commande retournera une liste des applications disponibles sur notre cible. Cette liste sera simplement contenue dans un tableau à une dimension, stocké quelque part en mémoire.

6. List run

Pour retourner la liste des applications qui sont actuellement en cours d'exécution sur notre cible, je devrai aller consulter la liste des processus qui tournent.

Chaque processus étant représenté par *(entre autre)* un numéro d'identification et un nom et un état *(running, blocked, ready)*, c'est ces deux informations que je vais retourner à la vue itérativement pour chaque processus avec l'état *running* pour afficher dans la Shell.

Le fait d'aller rechercher la liste des processus avec état Running sera traité après avoir vu la matière sur le noyau temps réel élémentaire en mode coopératif.

7. Start/stop appName

Pour l'application spécifiée en paramètre, je vais devoir créer un nouveau processus *(si l'application n'est pas déjà en cours d'exécution)* ou stopper le processus en question.

Cela sera également traité après avoir vu la matière sur le noyau temps réel élémentaire en mode coopératif.

VI. Conclusion

Au travers de cette analyse, j'ai pu étudier les variantes possibles pour implémenter chacune de mes tâches. Cela a été un très bon complément à la matière travaillée durant les travaux pratiques.

J'ai beaucoup apprécié la liberté que nous a laissé le projet. Sur la base des points obligatoires, nous avons pu ajouter des fonctionnalités intéressantes à notre cible et ainsi pouvoir implémenter par la suite ces fonctions intéressantes.

Après cette analyse, j'ai maintenant une idée plus précise de la manière dont je pourrai concevoir mes différentes tâches dans les phases prochaines du projet.

Grâce à cette phase d'analyse, au niveau des interfaces graphiques nous avons maintenant toute la structure logique des jeux et des différentes interactions. Ceci nous servira de fil conducteur tout au long du projet.

VII. Références

- Ficher 02_ARM_i.MX27_Reference_Manual, mis à disposition sur Cyberlearn

---

Signature

Froidevaux Romain