

Programmation concurrente

Jackpot

David Gonzalez - Claudio Sousa

1 février 2017

1 Introduction

Ce TP de deuxième année consiste à implémenter une machine à sous de Casino multi-tâches.

1.1 Spécification fonctionnelle

Chaque partie débute avec l'insertion d'une pièce. Ensuite, 3 roues tournent à des vitesses différentes. Chaque roue est arrêtée consécutivement (de gauche à droite) soit manuellement par l'utilisateur, soit automatiquement après 3 secondes.

Lorsque toutes les roues sont arrêtées, le résultat est ajouté à la caisse de la machine. Ce résultat est calculé selon le nombre de chiffres identiques, les résultats possibles sont :

- aucun chiffre identique (perdu) ;
- 2 chiffres identiques (2 pièces gagnées) ;
- 3 chiffres identiques (moitié des pièces en caisse).

Ce résultat est affiché pendant 5 secondes, puis revient à l'insertion de la pièce.

1.1.1 Entrées

Les entrées sont gérées à l'aide de signaux générés par la console avec les touches suivantes :

- CTRL-Z (SIGTSTP) : insertion d'une pièce ;
- CTRL-C (SIGINT) : arrêt d'une roue manuellement ;
- CTRL-\ (SIGQUIT) : arrêt du jeu.

1.1.2 Threads

Les threads sont divisés ainsi :

- 1 thread pour le contrôleur du jeu (qui sera le seul à recevoir les signaux) ;
- 1 thread pour l'affichage ;
- N threads, 1 pour chaque roue.

2 Development

2.1 Architecture

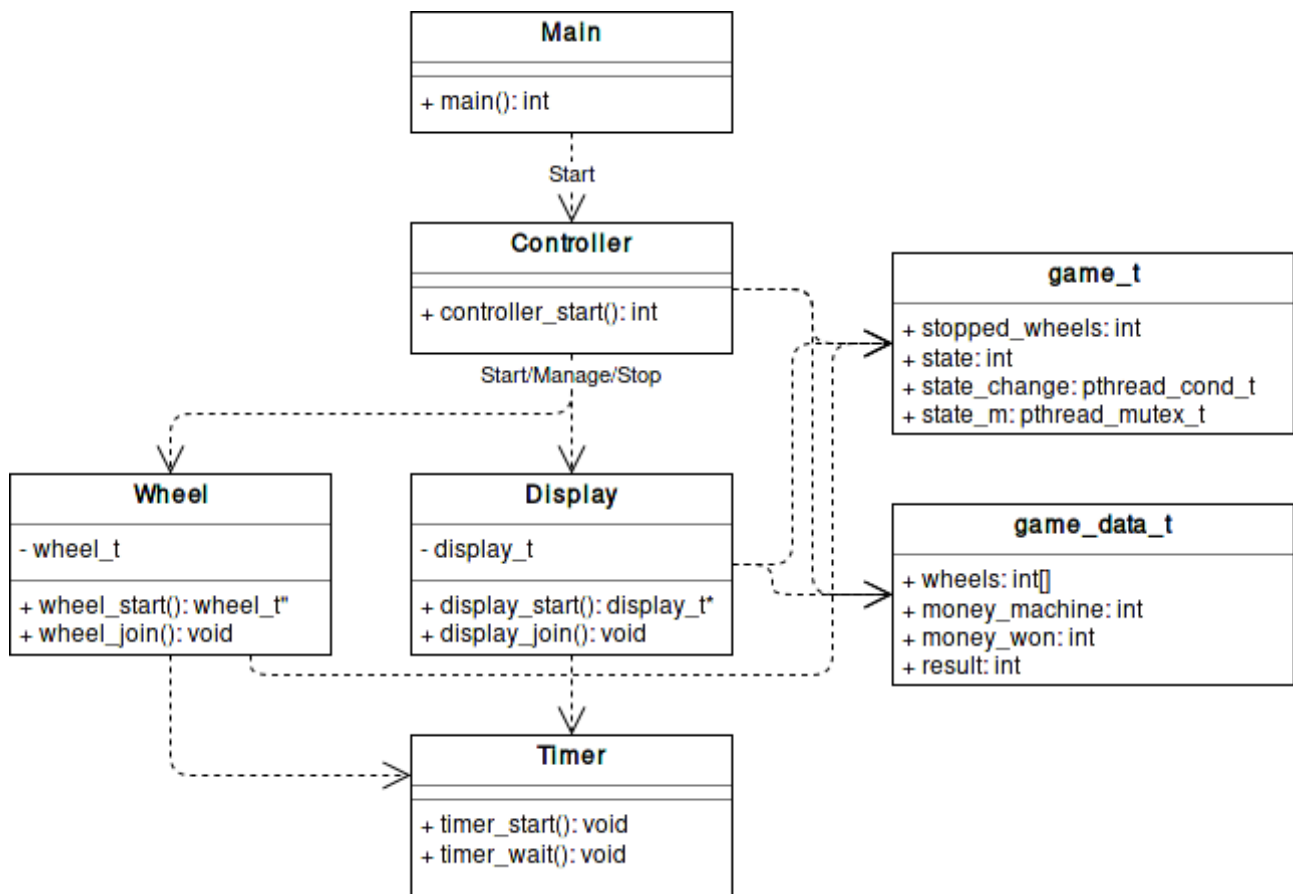


FIGURE 1 – Architecture du Jackpot

2.1.1 `game_t` et `game_data_t`

Ces deux structures représentent toutes les données partagées entre les 3 modules principaux, qui sont : *Controller*, *Wheel*, *Display*.

La première (*game_t*) contient les données d'état des roues ainsi que les données de synchronisation.

La deuxième (*game_data_t*) contient les données du jeu en tant que telle : la valeur de chaque roue, l'argent restant et l'état final du jeu (gagné, perdu).

2.1.2 `Main`

La fonction principale du programme ne fait que lancer le *Controller*.

2.1.3 Controller

Ce module est le coeur du programme. Il est responsable d'instancier le module *Display* ainsi que toutes les instance de *Wheel* (3 par défaut). Il est également chargé de contrôler l'avancement du jeu selon la machine d'état ci-dessous :

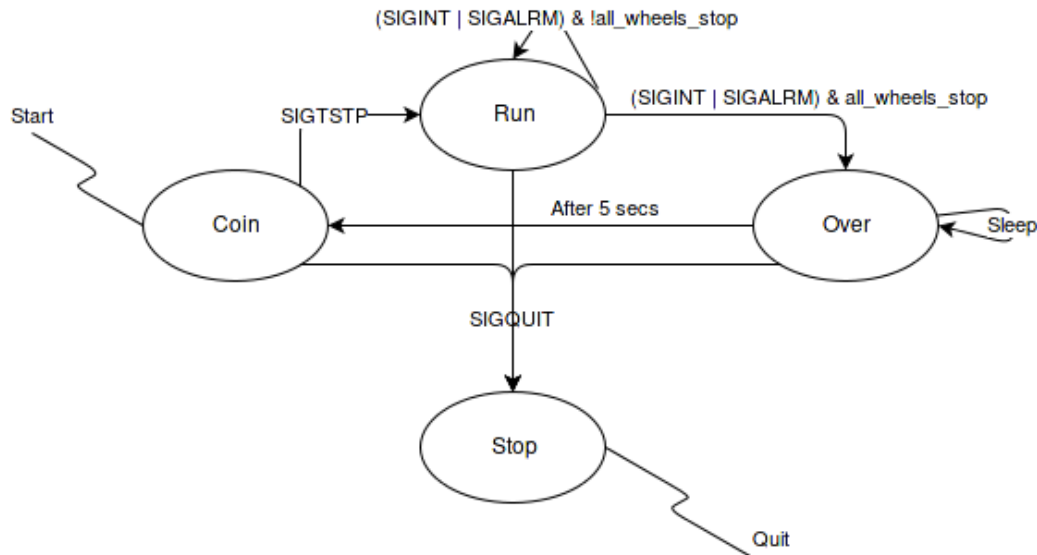


FIGURE 2 – Machine d'état du contrôleur

Comme le dit la spécification, les entrées utilisateurs sont gérées par des signaux. Le *Controller* est donc le seul module (et *thread*) à recevoir et à traiter ces signaux.

Les communications sont faites aux travers des deux structures citées (*game_t* et *game_data_t*), qu'il est chargé d'instancier et d'initialiser.

Au niveau des *threads*, le *Controller* crée lui-même son propre *thread* durant son instantiation.

2.1.4 Wheel

Le module *Wheel* est responsable de faire tourner 1 roue (changer le chiffre) dans un *thread* séparé à une certaine fréquence.

2.1.5 Display

Le module *Display* a le rôle de gérer l'affichage du jeu dans son *thread*. Donc :

- le message de début de parti ;
- les roues lorsque la parti est en cours ;
- le résultat de cette parti ;
- le message lorsqu'on quitte le jeu.

2.1.6 Timer

Ce module contient deux fonctions utilitaires permettant de mesurer le temps et d'attendre à une certaine fréquence.

2.2 Flux d'exécution et synchronisation

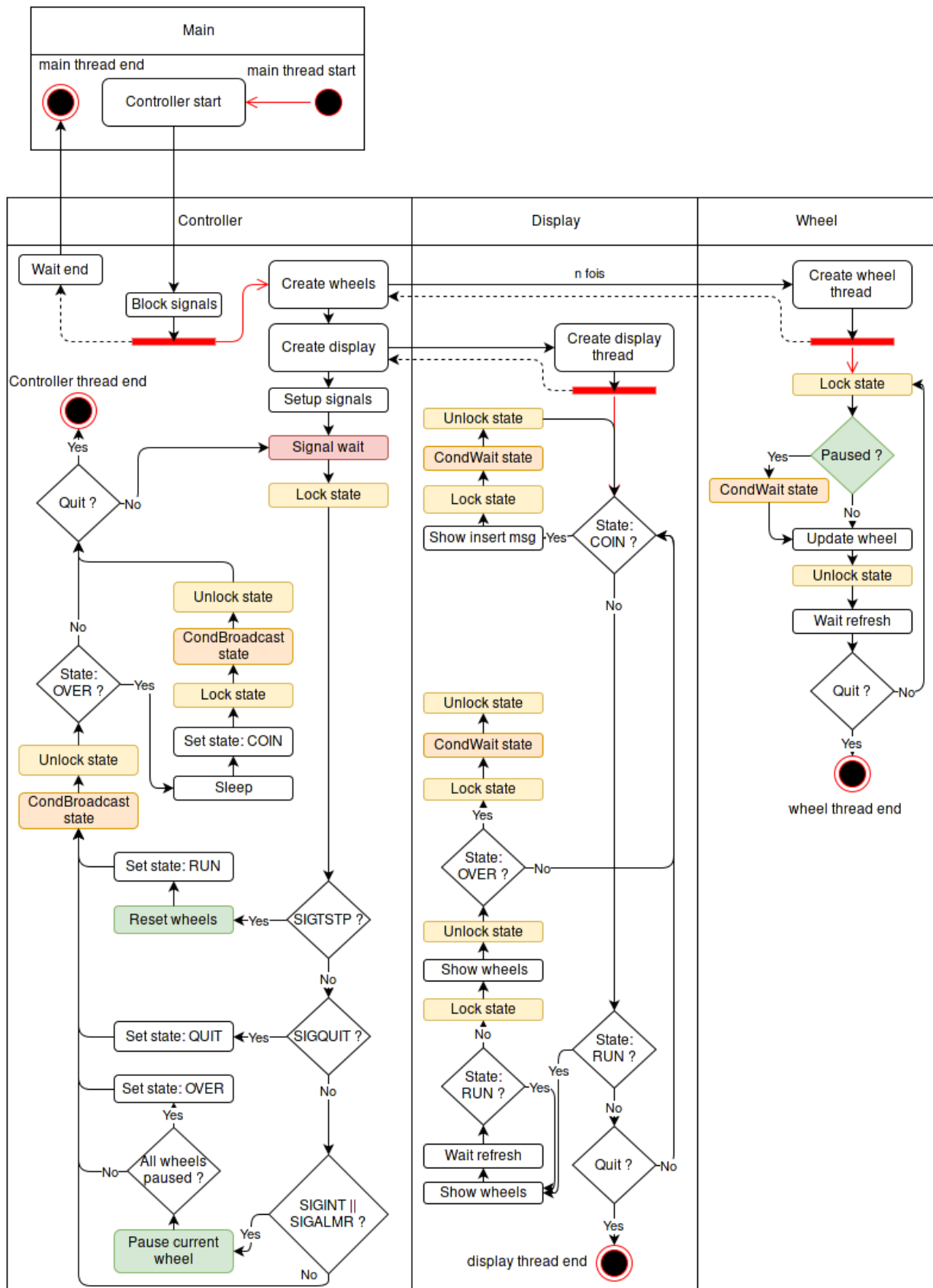


FIGURE 3 – Flux d'exécution et synchronisation

2.3 Méthodologie de travail

2.3.1 Répartition du travail

Ce travail a été effectué à deux.

Nous avons commencés par réfléchir sur papier sur deux éléments :

- architecture du programme : modules et interfaces de bases ;
- premier jet de synchronisation entre les différents threads.

Ensuite, le travail a été réparti ainsi :

- ...

Finalement, nous avons mis en commun les modules et finalisés la synchronisation des threads des différents modules.