Rapport LO21 : Trading Simulator

Gaetan CARABETTA Estia MALIQARI
Nu Huyen Trang PHAM Nhat-Minh DOAN
Zilu ZHENG Doris FEJZA

Juin 2019

Table des matières

1 Contexte

Le but de l'application *Trading Simulator* est de simulé un environnement d'échange entre deux monnaies. D'une part nous retrouvons la monnaie dite de *Base* et d'autre part la monnaie de *Contre-partie*.

L'utilisateur pourra acheter de la monnaie de base avec sa monnaie de contre-partie, selon le cours d'ouverture du jour, et réciproquement, il pourra vendre sa devise de base pour de la contre-partie selon le cours d'ouverture ou fermeture du jour.

L'affichage des cours est réalisé à l'aide de *bougies* et celui des volumes via des *bâtons*.

Nous retrouvons les indicateurs suivants, ayant chacun une représentation graphique propre :

— EMA [1]
— MACD [2]
— RSI [3]

2 Architecture

Nous avons séparé notre architecture en plusieurs modules [4] :

— Affichage
— Lecture des fichiers de données
— Stratégies
— Indicateurs techniques
— Editeur de texte
— Historique
— Gestion des sauvegardes

Ces modules sont connectés entre eux via l'application elle-même. Lorsque l'utilisateur décide d'acheter, de vendre ou de passer à la date suivante, l'application doit gérer l'affichage du cours, des volumes, de la mise à jour des stratégies et indicateurs.

3 Argumentation

Notre architecture est basé sur une classe centrale *MainWindow*. Nous avons ensuite décider de diviser chaque traitement de l'information dans dif-

1. Exponential moving average
2. Moving average convergence/divergence
3. Relative strength index
4. Liste non exhaustive

férentes classes.

Premièrement, il faut récupérer les données des cours dans un fichier. Actuellement, l'application ne supporte que la lecture de fichier au format CSV. Pour conserver le caractère modulaire de l'application, nous pouvons ajouter une classe abstraite *FileReader* en tant qu'interface et définir des classes spécifiques pour lire les autres formats de fichier. Le design pattern à appliquer serait *strategy*.

Deuxièmement, dans le cas où il sera nécessaire d'ajouter un autre graphique, pour afficher d'autres données, il suffira d'ajouter l'implementation d'une classe qui hérite de la classe abstraite *Graphique*.

Troisièmement, pour ajouter une stratégie, il suffit d'hériter de la classe abstraite *Strategie*.

Pour conclure, notre architecture permettra une évolution facile pour les différentes parties, que ce soit des stratégies, des indicateurs ou de la lecture de données.

4 Description du projet

Ci-dessous nous détaillerons le travail que nous avons réalisé sur chaque module.

3 mai	Création gitlab
3 - 21 mai	Etude conceptuelle individuelle
21 mai	Mise en commun des UML
21 - 30 mai	Recherche/Codage/Test individuel
30 mai	Débriefing
3 juin	Début de la mise en commun des modules
3 - 14 juin	Codage en commun
14 - 15 juin	Debug, test, vidéo, documentation
15 juin	Relecture et finalisation

TABLE 1 – Planning

4.1 Affichage des données

Estia et Gaétan ont restructuré l'affichage des données pour prendre en compte le volume d'échange journalier et borner l'affichage des cours, allant du cours d'il y a 10 jours jusqu'à celui de la veille.

La création du graphique des volumes a été fait par Estia, et les bornes ont été implémentées par Gaétan, pour les deux graphiques.

La mise à jour de la partie graphique, lors d'un changement de date a été réalisée par Gaétan.

4.2 Choix des paramètres

Gaétan a intégré la partie *option* après avoir réalisé la lecture du fichier de données, au format CSV. Elle permet entre autre de sélectionner une date de début pour la simulation, un solde initial pour la *base* et la *contrepartie*, la part du *broker* et le mode *manuel* ou *automatique*.

Lorsque le mode automatique est sélectionné une autre fenêtre s'ouvrira pour choisir la stratégie à appliquer.

4.3 Stratégie de trading

Trang a réalisé les stratégies de trading qui permettent à l'utilisateur de faire les transactions en mode automatique. Pour chaque indicateur technique, une stratégie est proposée, en plus détaillée, EMA Convergence (EMA), RSI Stratégie (RSI) et MACD Stratégie (MACD).

4.3.1 EMA Convergence

On doit d'abord récupérer le solde de base et le solde de contrepartie de la paire de devise, pour laquelle on veut faire la transaction.

On fait ensuite la comparaison de la valeur de l'indicateur EMA et celle du prix d'ouverture du cours à la date courante.

— Si Prix Ouverture $>$ valeur EMA et solde de contrepartie > 0, on achète
— Si Prix Ouverture $<$ valeur EMA et solde de base > 0, on vend
— Si non, on ne fait rien.

4.3.2 RSI Stratégie

Dans cette stratégie, on compare seulement la valeur de l'indicateur RSI avec deux valeurs (overbought value et oversold value)

On décide de l'achat ou de la vente en comparant les valeurs de RSI avec un seuil défini comme ci-après :

— Si RSI $<=$ oversold (30), on achète
— Si RSI $>=$ overbought (70), on vend
— Si non, on ne fait rien.

4.3.3 MACD Stratégie

Afin de décider de l'achat ou de la vente des devises, la stratégie va voir l'intersection des diagrammes courte période du MACD et longue période du MACD, et aussi les soldes de base et de contrepartie.

— Si courte >= longue et solde de contrepartie > 0, on achète
— Si courte <= longue et solde de base > 0, on vente
— Si non, on rien fait.

Pour la construction des stratégies, le design pattern *stratégie* a été utilisé avec une classe mère abstraite *Stratégie* et ses classes filles.

En observant les valeurs des indicateurs techniques, les stratégies déciderons de l'achat ou de la vente en envoyant un signal et un montant à *MainWindow* qui gère directement le solde et met à jour les montants.

Pour les valeurs de trading (différence de base, de contrepartie, ROI), ils sont tous calculés dans *MainWindow*.

A améliorer :
Intégrer les stratégies qui font les ventes de "stop lost" et aussi les achats "take profit".

4.4 Modes de jeu

Doris a implémenté la classe abstraite *Transaction* contenant une méthode virtuel pure *updateBalance* (qui fait la mise à jour des montants de devises après chaque transaction), definie dans chaque classe fille *Purchase* et *Sale*. Pour cette implémentation, nous utilisons le design pattern *Template*, permettant d'implémenmter la partie invariante une seule fois et laissant les comportements qui diffèrent au niveau des sous-classes.

Le mode automatique a été implémenté par Gaétan et Trang. La configuration de la stratégie et des dates de début et fin se fait au travers d'une fenêtre, chargée après le choix des options. Pour réalisé les transactions, nous avons définit une classe *robot*, qui s'aidera des indicateurs. Selon la stratégie choisie au début, l'instance de *robot* déterminera le montant pour la vente ou l'achat.

Le mode manuel laisse le choix à l'utilisateur d'acheter ou de vendre, selon ses soldes. Une seule transaction est autorisée par jour, c'est-à-dire soit un achat, soit une vente.

4.5 Editeur de texte

Lorsque cette fenêtre est ouverte, les notes précédemment enregistrées par l'utilisateur sont chargées à partir du fichier local XML et affichées dans cette

fenêtre. Les utilisateurs peuvent prendre des notes dans cette fenêtre. Lorsque l'utilisateur souhaite quitter l'ensemble de l'application, il devra appuyer sur le bouton *Sauvegarder* dans la barre des menus pour permettre à ses notes et au tableau des transactions d'être écrits dans le fichier local XML. De plus, l'utilisateur peut appuyer sur le buton Delete pour effacer les notes prises.
Doris a travaillé sur l'ajout de l'editeur de texte comme un widget et ses fonctions *cut, copy, paste, undo, redo, save, delete*. Zilu a travaillé sur les fonctions de la classe LoadSave pour sauvegarder et charger les notes en xml.
A améliorer :
L'editeur permettra d'arranger et formater des tableaux, nécessaires pour contenir des informations sur les achats et ventes réalisés par l'utilisateur. On peut aussi envisager d'imprimer et d'exporter en format PDF.

4.6 Indicateurs techniques

Minh a réalisé les indicateurs techniques. Ils permettent de générer les données spéciales à partir d'un fichier chargé, ainsi que de les visualiser pour aider l'utilisateur dans sa décision d'achat ou de vente dans mode manuel, et aider le robot à choisir la transaction à appliquer selon sa stratégie. Dans le cadre de ce projet LO21, les indicatuers techniques choisis sont EMA, MACD, RSI.

4.6.1 Hiérarchies des classes

Une classe *ValueIndicator* a été déclarée contenant une date et la valeur calculée de chaque cours. La classe *Indicateur* est abstraite donc il faudra créer soit un EMA, soit un RSI, soit un MACD. Chaque indicateur contient des iterators permettant de parcourir le tableau des adresses des objets *ValueIndicator*.

4.6.2 Exponential Moving Average (EMA)

Pour calculer une valeur EMA d'un jour :

$$EMA_{today} = (Price_{today} \times K) + (EMA_{yesterday} \times (1 - K))$$

Dont :

- $K = 2 \div (N + 1)$
- N = période d'EMA (ex : 5 jours, 10 jours ou 10 heures, 20 heures. . .) ici dans notre application c'est jour
- $Price_{today} = CLOSE_{today}$
- $EMA_{yesterday} = EMA_{jourprcdent}$

Le calcul du EMA commence par le $(N+1)^{eme} jour$, et $EMA_N = \frac{\sum_{i=1}^{N}(close_i)}{N}$

4.6.3 Moving Average Convergence Divergence (MACD)

Pour calculer le MACD :
— $MACD = EMA(12) - EMA(26)$
— $MACD_{signal}$ = EMA(9) de MACD
— $MACD_{histogram} = MACD - MACD_{signal}$

4.6.4 Relative Strength Index (RSI)

Pour calculer le RSI :

$$RSI = 100 - \frac{100}{(1 + RS)}$$

Dont RS = (average of 14 day's closes UP)÷(average of 14 day's closes DOWN).

On a choisi la periode standard comme 14.

La 1^{ere} RSI est calculée à partir du $14^{me} jour$

c : le prix de fermeture du jour

Calculer UP pour un jour :
si $c_{hier} < c_{aujourd'hui} => UP_{aujourd'hui} = c_{aujourd'hui} - c_{hier}$
si $c_{hier} >= c_{aujourd'hui} => UP_{aujourd'hui} = 0$

Calculer DOWN pour un jour :
si $c_{hier} <= c_{aujourd'hui} => DOWN_{aujourd'hui} = 0$
si $c_{hier} > c_{aujourd'hui} => DOWN_{aujourd'hui} = c_{hier} - c_{aujourd'hui}$

$UP_{AVERAGE}$ = average of 14 day's closes UP = la somme des 14 premiers UP (y compris le 0) ÷ 14

$DOWN_{AVERAGE}$ = average of 14 day's closes DOWN = la somme des 14 premiers DOWN (y compris le 0) ÷ 14

Calculer à partir de la 2^{eme} RSI :
— $UP_{AVERAGE} = (UP_{AVERAGE}[hier] \times 13 + UP_{aujourd'hui}) \div 14$
— $DOWN_{AVERAGE} = (DOWN_{AVERAGE}[hier] \times 13 + DOWN_{aujourd'hui}) \div 14$

4.6.5 Afficher des indicateurs

Pour chaque indicateur, on donne à l'utilisateur la possibilité de les afficher.

4.7 Analyse de forme

Estia a réalisé *l'analyse des formes* des bougies prennant en compte le prix minimal et maximal des données affichées. Pour chaque bougie, on contrôle la taille du corps et des mèches. Ensuite, des comparaisons avec des bornes raisonnables sont faites pour trouver la forme appropriée. Si demandé par l'utilisateur, le nom de la forme est affiché après avoir envoyé un signal *click()* sur la bougie.

4.7.1 Les Toupies

Premièrement, on analyse si une bougie est une *toupie* ou pas. On compare la taille du corps *(openPrice-closePrice)* avec la taille des données affichées *(maxPrice-minPrice)*. On suppose que la taille de la bougie a au plus 20% de la taille du graphique affiché.

Une fois confirmé la forme *toupie* pour la bougie, on continue à analyser les autres formes :

- *Doji* : si *openPrice* = *closePrice*
- *Marteau* : si on suppose que la mèche basse est dans une tendance baissière *(closePrice-lowPrice)* est au moins 2 fois plus grande que le corps
- *Etoile filante* : si on suppose que la mèche haute est dans une tendance haussière *(highPrice-closePrice)* est au moins 2 fois plus grande que le corps
- *Pendu* : si on suppose que la mèche basse dans une tendance haussière *(openPrice-lowPrice)* est au moins 2 fois plus grande que le corps

4.8 Gestion partie

Zilu a travaillé sur la partie *sauvegarde* et *chargement* du projet.

Le but de la sauvegarde est d'écrire toutes les données de transaction générées par l'application et les notes écrites par l'utilisateur depuis la mémoire vers un fichier local XML, pour un stockage permanent. Par conséquent, cette classe fournit une interface à la fenêtre principale pour l'appel aux fonctions.

Le but du chargement est de lire le fichier XML local, d'initialiser la mémoire avec des transactions et des notes. Après ils peuvent utiliser ces données pour les afficher.

4.9 Mode pas à pas

Le mode pas à pas nécessite deux boutons supplémentaires à rajouter à l'interface : *Précédent* et *Suivant*.

Notre architecture actuelle ne devra pas être modifiée pour intégrer la possibilité de revenir en arrière, car nous conservons un historique de toutes les transactions avec le montant et la date. De plus, la date est accessible à tous les objets de notre application et il nous faudrait uniquement rajouter des fonctions pour revenir dans le passé.