Software Requirements Specification

for a

Scenario-based Financial Instrument Learning Tool

Version 1.0

Prepared by Mathilde Simoni, Peter Mahhov, Lachlan Pham

New York University Abu Dhabi

March 1st, 2022

Table of Contents

1. Introduction

1.1 Purpose

This document identifies the requirements for a scenario-based financial literacy learning tool. It is a new and self-contained product independent from any other system. This first version of the SRS describes all functional and nonfunctional requirements for the implementation of the product.

1.2 Intended Audience and Reading Suggestions

This document is intended for all individuals involved in the development process: the development team, the testers, and the stakeholders. Part 2 gives an overall description of the product consisting of its positioning in the market, its main functions, and constraints related to design and operating environment, as well as user classes and user documentation. Part 3 specifies the requirements regarding external interfaces with users, and software that will be important for both developers and stakeholders. Part 4 is meant for the development team and testers by providing a detailed description of each functional requirement. Finally, part 5 contains a list of nonfunctional requirements for the product related to performance, safety, security, maintenance, and usability.

1.3 Product Scope

This product aims at educating individuals and especially young people in financial literacy. By simulating financial instruments in the stock market and providing various scenarios of increased complexity, it will address the barriers in learning about investing and help users get a fundamental understanding of the financial market. This product will also teach risk management skills by allowing users to play in different timescales, access information about the historical behavior of financial instruments and see how short-term and long-term consequences can differ according to their actions. Thus, users might learn how to responsibly trade financial instruments and how to make thoughtful and informed financial decisions to avoid making mistakes when investing in the real-world stock market.

2. Overall Description

2.1 Product Perspective

This is a new, self-contained product. It is an independent original project that has no ties to existing products or already-developed software beyond those used in its creation. As a financial simulator and learning tool, it will be similar to other such products offered in the market in a broader financial education category. However, it will contain unique scenarios and a never-before-seen "timeline splitting" mechanic, which can serve to increase player agency and thus boost both engagement and learning.

2.2 Product Functions

- The user should be able to play through various financial simulation scenarios of increasing complexity.
    - The user shall be able to progress through time by pressing the appropriate button, the system shall then automatically update any time-variable attributes of the presently simulated financial instruments.
- The system shall simulate various financial instruments (stocks, loans) with differing risk profiles (changing trends, volatility).
    - The user shall be able to buy and sell stocks, the system shall then automatically update the user's portfolio.
    - The user should be able to take out loans, the system should then automatically update the user's portfolio.
    - The price of the financial instruments should fluctuate semi-randomly.
- The user should be able to access information about the historical behavior of the financial instruments they can invest in.
    - The user should be able to open up a pop-up window for each financial instrument available for purchase which shows the historical price of that instrument.
- The user should be able to learn the new mechanics from helpful information included in each scenario.
- Within a simulation scenario, the user shall have the ability to split and merge "timelines", allowing them to see how short-term and long-term consequences arise from their differing actions.
    - When only one timeline is currently active, the user shall be able to duplicate it, creating a new independent instance identical to the original.
    - When two timelines are active, the user shall be able to make independent decisions in either timeline without affecting the other one.
    - When two timelines are active, the user shall be able to delete one timeline and return the system to a situation with one active timeline (which would be identical to the other, non-deleted timeline).

2.3 User Classes and Characteristics

The users are meant to interact with the software for two different primary purposes. First, it can be used as a learning tool, a way to safely interact with various types of financial instruments and gain experience in their functioning with no personal risk to the user. This type of user will likely have a low initial level of financial education to begin with and will benefit from there being a wide variety of simulated financial instruments available.

While learning is the primary goal for the development of this tool, the process will be enhanced if the user is having fun during the process. Therefore as the software is meant to be entertaining, it can also serve as a way to have fun and pass the time. Hence the second user class are the kind of people who might be less interested in developing their financial acumen, and are rather looking for a new interesting experience. It is assumed that users who already have a high level of financial literacy will likely belong to this user class, or they would not be interacting with the software in the first place.

2.4 Operating Environment

The system must be able to run on both Windows and Macintosh, offline and at any moment. It should not interact nor interfere with any other software nor with files on the user’s computer beyond those that are needed to run the product.

2.5 Design and Implementation Constraints

The user must have a working Python environment installed in the machine where they intend to run the software on.

2.6 User Documentation

Tutorials will be included for each scenario. They will provide information about scenario context, newly introduced mechanisms and financial instruments, as well as describing any winning and losing conditions.

3. External Interface Requirements

3.1 User Interfaces

Within a simulation session there will be three primary components to the user interface: the background, the timeline windows and the information popups.

The background will serve as the backdrop to the other windows, having spaces to organize the other User Interface components into a user-friendly layout. The background will also consist of buttons to advance time as well as to split or merge timelines. The few information boxes present in the background layer serve to show the kind of information that is independent of the status of the individual timelines, like the present date in the simulation or a reminder of the win condition in the current scenario.

The timeline windows will present information about the current portfolio of each timeline: total value, volume of each financial instrument owned and cash available. Additionally, each timeline might have a dropdown menu to select among the financial instruments available for a scenario if there are too many to display. Upon selection, the current value of the instrument will be displayed along with an input interface which allows the user to sell and buy a specified volume of that instrument. A button will also allow the user to open an informational pop-up window.

Upon request for a particular financial instrument, a pop-up window can be generated. There, a graph will be presented showing that instrument's "historical" price leading up to the current in-game moment.

3.2 Hardware Interfaces

The tool will require standard input devices: a mouse or trackpad and a keyboard. These will allow the user to interact with the software's user interface.

3.3 Software Interfaces

The system will be developed using the latest version (as of now, 2.1.2) of the pygame library. The Graphical User Interface will be implemented using the resulting pygame_gui (version 0.6.4) library. Matplotlib 3.5.1 will handle visualizations of the financial instruments and historical data with graphs. In order to simulate randomness and probabilistic distributions, the software will use mathematical functions from the numpy library (version 1.22.2)

3.4 Communications Interfaces

Before initial startup, the system will need to communicate with Python 3.10.2 repositories to install the necessary libraries described above. Subsequently, no further external connection will be required.

4. Functional requirements

The list of functional requirements will be organized by object classes.

4.1 Background class

This class will generate and contain all instances of the objects used for each scenario.

4.1.1 Attributes

- 4.1.1.1 Timelines: An array of three timeline objects, two will be active during the split state and one during the merged state.
- 4.1.1.2 Financial instruments: An array of all instances of financial instruments (Stock and Loan type objects) available for the user to trade.

4.1.2 Functions

- 4.1.2.1 Load Data: Automatically called at the beginning of each scenario. Takes a data file containing all of the information required to generate the instances of the scenario objects including initial financial instrument parameters and simulated historical prices for each financial instrument.
- 4.1.2.2 Progress Time: The user will be able to click a button which causes the scenario to progress time to a predetermined interval. This will iterate through the financial instruments and call their respective Progress Time methods.
- 4.1.2.3 Drop Timeline: The user will be able to specify which timeline to drop by clicking on a button on the appropriate timeline interface. It will copy the information from the other active timeline to the 'center' timeline, overwriting its previous values.This will deactivate the two active timelines and activate the 'center' timeline, the one which is only shown during the merged state.
- 4.1.2.4 Split Timeline: When only the 'center' timeline is active and displayed, the user will be able to click on a button to split the timeline. This will copy the information from the 'center' timeline to the other two. Then the 'center' timeline will deactivate, and the other two timelines will activate.

4.2 Timeline class

Each scenario will consist of three instances of Timeline, two shown during the split state ('left' and 'right') and one shown during the merged state ('center'). This class will store and modify the information about the portfolio in each of these states.

4.2.1 Attributes

- 4.2.1.1 Is Active: A boolean flag that tracks whether this timeline is displayed on the screen and interactable by the user or not.
- 4.2.1.2 Money: The current amount of money owned by the player in this instance of the timeline.

- 4.2.1.3 Portfolio: A dictionary that stores all the financial instruments existing in the scenario and the information of how many of each the player currently possesses in the timeline.
- 4.2.1.4 Loan: An instance of Loan which contains information about money that is borrowed or has the potential to be borrowed by the user.
- 4.2.1.5 Net Worth: The net worth of the player in this instance of the timeline. Calculated as Money + (for each stock): stock price * amount of that stock owed.

4.2.2 Functions

- 4.2.2.1 Buy Stock: The user will be able to click on a button to buy a stock and optionally enter the volume for a custom amount. It will decrease the Money attribute of the timeline by the number of stocks bought multiplied by the price of an individual stock. If the total value of purchased stocks is greater than the current amount of money owned, the user won't be able to buy them. Otherwise, the transaction goes through and the possession count in the Portfolio dictionary for that particular stock increases by the number bought.
- 4.2.2.2 Sell Stock: The user will be able to click on a button to sell a stock and optionally enter the volume for a custom amount. It will increase the Money attribute by the total amount of stocks sold multiplied by the individual price of that stock. Then it will reduce the possession count in the Portfolio dictionary for that particular stock by the number sold. If the number of instances of the sold stock owned in that timeline is smaller than the number to be sold, the system will only sell as many instances as the timeline has.
- 4.2.2.3 Take a loan: The user will be able to input a desired loan amount (up to a maximum depending on the value of the Money attribute in the timeline), click on a button to take a loan and call the borrow method in the loan instance, setting the amount owed.
- 4.2.2.4 Pay off loan: This user will input a desired amount to pay back, click on a button pay off loan and call the pay off method in the loan instance, reducing the amount owed.
- 4.2.2.5 Activate/Deactivate: When this method is called, the boolean flag Is Active will switch values. If the timeline is being activated, it will be shown on the screen, if the timeline is being deactivated, it will be hidden.

4.3 Stock class

This class contains the information about a specific stock and the associated methods to access or modify this information.

4.3.1 Attributes

- 4.3.1.1 Price: current value of the stock that changes across time based on an appropriate statistical distribution.
- 4.3.1.2 Historical Prices: a table which stores the historical prices of the stock. This includes the historical prices procedurally generated at the start of each scenario. The price changes throughout the scenario will be included as time progresses.
- 4.3.1.3 Volatility: a behavioral parameter which determines the deviation of the statistical distribution from which to take the next price.

- 4.3.1.4 Trend: a behavioral parameter which, along with price, determines the mean of the statistical distribution from which to take the next price.

4.3.2 Functions

- 4.3.2.1 Progress Time: This is called when the Progress Time function is called in the background class. The price of the stock will be updated, randomly selected from an appropriate statistical distribution created from the stock's Price, Volatility and Trend. Historical Prices is updated with the new price.
- 4.3.2.2 Display Info: On the timeline window, the user will click on a button to request information about the particular stock. This will create an instance of the Information Pop-Up class for which the stock's current price and the historical prices table are passed as arguments.

4.4 Loan class

This class contains the information about a specific loan and the associated methods to access or modify this information.

4.4.1 Attributes

- 4.4.1.1 Offered Interest Rate: current interest rate of the offered loan that changes across time based on an appropriate statistical distribution.
- 4.4.1.2 Historical Interest Rates: a table which stores the historical interest rates of the loan offer. This includes the predetermined historical interest rates generated at the start of each scenario; the interest rate changes throughout the scenario will be included as time progresses.
- 4.4.1.3 Volatility: a behavioral parameter which determines the deviation of the statistical distribution from which to take the next offered interest rate.
- 4.4.1.4 Trend: a behavioral parameter which, along with interest rate, determines the mean of the statistical distribution from which to take the next offered interest rate
- 4.4.1.5 Amount owed: the total amount of money owed in this timeline instance.
- 4.4.1.6 Interest at Borrowing: the interest rate at which the currently taken loan was taken at. This is displayed only if this loan offer has been taken (if amount owed is nonzero).
- 4.4.1.7 Maximum Loan Amount: the maximum value possible to borrow at once: it is a direct function of the Net Worth value of the Timeline that the Loan object is in

4.4.2 Functions

- 4.4.2.1 Progress Time: This is called when the Progress Time function is called in the Background class. The offered interest rate will be updated, randomly selected from a statistically appropriate distribution created from the current offered interest rate, volatility, and trend. The Historical Interest Rates table is updated with the new price. The amount owed is proportionally increased depending on the interest at borrowing.
- 4.4.2.2 Borrow: This is called after 'take a loan' is called in a timeline instance. Firstly, 'amount owed' is checked. If it is not zero, a message is displayed to the user informing them that they cannot borrow money until all their debt is paid off. If amount owed is zero, sets amounts owed based on passed value (to a maximum

value of Maximum Loan Amount) and sets 'interest at borrowing' at the current value of the offered interest rate.

- 4.4.2.3 Pay off: This is called after 'pay off loan' is called in a timeline instance, reduced 'amount owed' based on passed value. Passed value must be lower or equal than 'amount owed'.

4.5 Information Pop-Up class

This class contains a method to display visualizations about a particular financial instrument.

4.5.1 Functions

- 4.5.1.1 Display Graph: The user will click on a button to get visualizations about the historical behavior of a financial instrument available for purchase. The information needed from the financial instrument class will be passed as arguments to this class. It will then open a pop-up window and display a graph using the Matplotlib library.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

There are no particular performance requirements as of now.

5.2 Safety Requirements

An initial disclaimer message will be displayed specifying that the developers who made this game are not financial experts, and that therefore none of the information in the game should constitute financial advice and the behaviour of the financial instruments is not necessarily perfectly reflective of reality.

5.3 Security Requirements

Beyond the initial download and library installation requirements, the system will not require any connection to the internet or further access to the computer's storage. The system will also not ask for nor store any sensitive information. As such, there will be no form of identity authentication; as long as a user is able to run the application, there should be no barriers to access all of its features.

5.4 Software Quality Attributes

- 5.4.1. The software should run on-demand, it does not have any systems that must be up and running 24/7.
- 5.4.2 The software and its user interface should be operable by 90% of our test users without external help.
- 5.4.3 The software should not crash while it is running for more than 2% of the playthroughs.
- 5.4.4 Unexpected inputs should not interrupt the functioning of the system.