Software Documentation including (Requirement, design, test cases, and test results)

for

Fin Sim

Version 2

Prepared by Team 9 (Lauren Elzeer, Chris Bierley, Thomas Wykle, & Taylor Jensen)

DATE: 04/29/2023

Revision History (2 pts)

Name	Date	Reason for Changes	Version
Lauren Elzeer	03/29/23	Set up for version 1/Started on section 1	1.1
Chris Bierley	03/30/2023	Added diagrams (i.e., use case, context, etc.)	1.2
Chris Bierley	03/31/2023	Input information into section 2	1.3
Lauren Elzeer	03/31/2023	More work on section 1 and the beginning of 5	1.4
Thomas Wykle	03/31/2023	Added references to Python and Tkinter libraries	1.5
Chris Bierley	04/03/2023	Added first draft to section 2	1.6
Wykle, Bierley, Elzeer	04/04/2023	Added to the functional requirements sections and completed the non-functional requirements	1.7
Taylor Jensen	4/04/2023	Added to the external interface requirements	1.8
Lauren Elzeer	4/04/2023	Revised 1.3	1.9
Thomas Wykle	4/11/2023	Expanded the detailed design section. I added the saveFile and Leaderboard sections and	1.91

		expanded the explanation of FinSimController	
Taylor Jensen	4/27/2023	Grammar and spelling error corrections and changed the colors of the text (grey is description of what to do, red are notes from the TA/Professor)	1.92
Lauren Elzeer	4/28/2023	Fixed spelling and grammar issues over the whole document, went over and fixed things from the TA advice, rewrote sections 2.5, 2.1, 2.6, 1.4, 2.4, 2.3, 2.2, glossary, table of contents, I also made sure the formatting of the document was correct this time so that all diagrams could be see	2
Chris Bierley	4/28/2023	Added Use Cases	2
Taylor Jensen	4/28/2023	Revised 1.4, 2.4, 2.5	2
Chris Bierley	4/28/2023	Added Test Cases	2
Taylor Jensen	4/28/2023	Grammar and spelling corrections	2
Thomas Wykle	4/28/2023	Finished the High-Level Architecture and Detailed Design Sections	2.1

1. Introduction

1.1 Purpose (5 pts)

Our overall purpose is to teach young adults how to start financing better. This project is a financial simulation (Fin Sim) that will take its users over a 5-year period of financial situations. Each turn is a month and allows the user to choose what they do with their money. In total there are 60 turns. The objective is to get through the five years and not go bankrupt. Not going bankrupt is the main thing we want to teach our users not to do. We hope that by playing this game users will learn how to knowledgeably allocate their money so that when they are going through college, renting apartments, or paying any other bill they will still have enough money left to get food and other necessities.

1.2 Intended Audience and Reading Suggestions (2 pts)

Those who want to have a better understanding of what we are doing or those who want to know how we have set up our code should read this document. Those who want a better understanding of what we are trying to accomplish should read this section, section two, and section four. This current section goes over what our project aims to do, what we aim to accomplish, what we want to get done in the future and what outside resources we used. Section four goes over the requirements (what we got done) in more detail. For those interested in our code, we highly recommend looking through sections two, three, five, and six. The sections focus on how the code is set up and what the code does.

1.3 Product Scope (3 pts)

Our simulation will follow through several months up to 5 years. Each month you will have a certain number of options based on whether you want either car payments, house rent, or both. Other options consist of spending your monthly budget on various categories such as bills, food, and entertainment among other potential variables. You will want to see how much money you saved by the end of the time/simulation. It will be your ultimate score. There will also be a saving system.

The program will help user learn about ideas such as:

- Saving money
- Where to spend money optimally
- How to balance a budget

Things that we would like to include in the future:

- Investment section
- Food budgeting
- Taxes understanding
- Servers
- Getting it into an Appstore

1.4 References (2 pts)

Both references below were the coding library (Tkinter) and language (python) we used to create the Sim Fin application. Both sources were used throughout the entirety of the coding process but were particularly relevant when setting up our initial Tkinter file as we had no experience with Python or any of its libraries and we used the sources as a means of getting familiar with it.

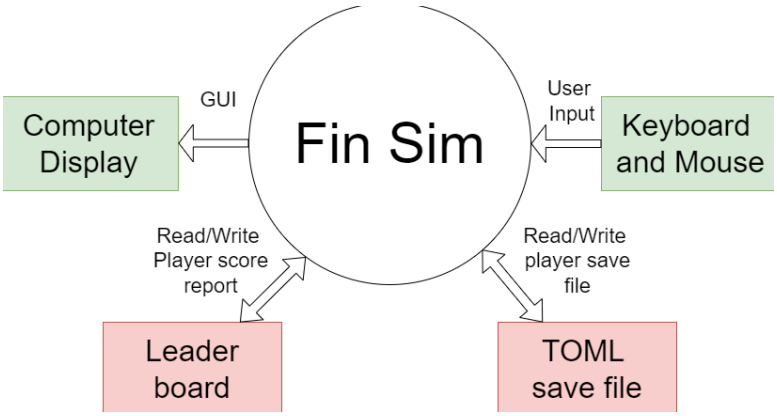
Lundh, F. (1999). An introduction to tkinter. URL: www.Pythonware.Com/Library/Tkinter/Introduction/Index.Htm.

Van Rossum, G. (2020). The Python Library Reference, release 3.8.2. Python Software Foundation.

2. Overall Description

2.1 Product Perspective (10 pts)

This is the context diagram. The context diagram shows how different components work with the main program.



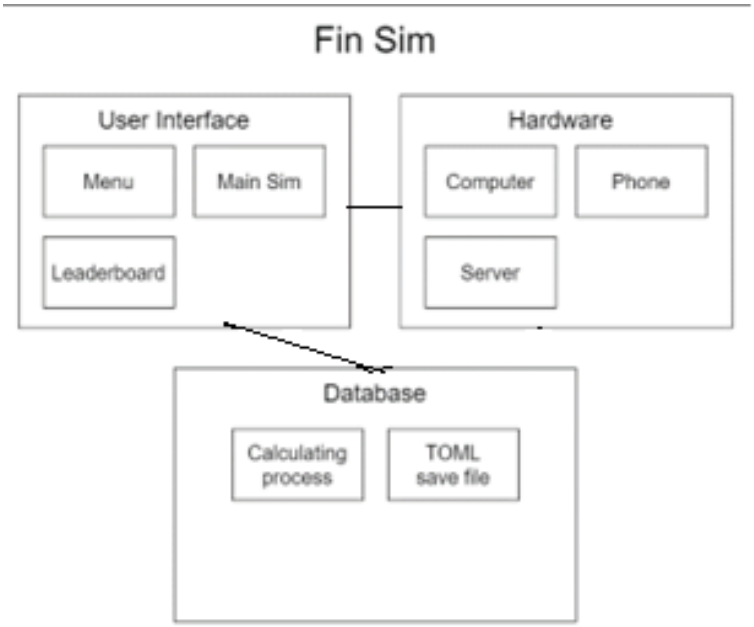
The mouse interacts by giving the program input. The mouse is used click buttons and text fields telling the program what the user is trying to access. The keyboard works in an analogous way. The keyboard is used to fill in information in text fields, so that when the mouse clicks a button the program will have information to receive.

The program gives information to the computer display by outputting a graphical user interface. This interface can be seen and interacted with by the user.

The leader board component writes to the program while the program reads it.

The save file works the same way. The save file writes to the program, and the program reads the file.

This is the component diagram. It displays an abstract and breakdown of our "big" items. These items include our user interface, the hardware, and the database.



The user interface interacts with both the hardware and the database. The hardware will give input to the user interface, the user interface will then send the information to the database. The database will save the information to a save file and communicate it back to the interface when needed.

As of right now we do not have servers set up. However, we hope to have them in the future. Whenever servers get set up, the only information that will be sent from the user interface to the server is the player's name and score.

In the diagram we have the phone listed as hardware. While we do not currently have the program set up as an app, we hope that in the future we will be able to implement such.

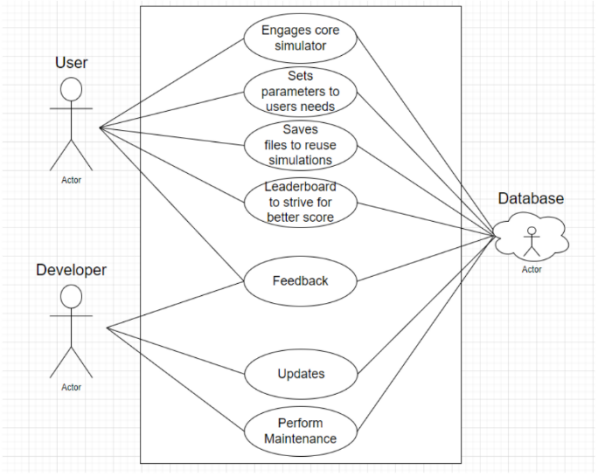
2.2 Product Functions (5 pts)

The product should allow the user to start a new game or continue a previously saved game. The user can alter settings to choose if they will have car and house payments. The program will also allow the user to input how much they would like to spend on food. From there the program should allow the user to go through 60 cycles to emulate 60 months (about 5 years). Each turn is one month. The program will also give the user a random event after each turn. The events will either be negative (cost the user money) or positive (give the user money). The money should be added to either the user's money saved or to their owed money.

2.3 User Classes and Characteristics (10 pts)

Users that want to increase their awareness of financial literacy can utilize the simulator to increase their exposure to financial experiences.

Use Case Diagram:



This is a use case diagram. This shows how the program interacts with the user, the developer, and the database. The user can use the menu to navigate into the main simulation. This starts the main problems the user must solve by budgeting. The user can also check settings for setting up information pertinent to them. Can also check the leaderboard to see the scores of all the players who finished the simulation. Finally, they can save their current simulation and close the game.

The Developer can access the feedback, program, and manage any updates, and perform maintenance on the code.

The Database can access and interact with everything. The database links the updates and maintenance performance from the developer to the user.

2.4 Operating Environment (5 pts)

This simulation can run on any version of Windows and any version of Mac OS operating systems. This is a desktop app specifically intended for those two operating systems. Linux distributions would also likely be able to run this simulation but cannot be said for certain as that has not been tested by our team. Geographically, this can be run anywhere that a working computer can be taken and intact as it does not require a Wi-Fi or internet connection. There are no other environmental necessities for running this application other than you would need some sort of IDE (Integrated Development Environment) to run it, and python version 3.7 or higher installed as these come with the Tkinter Library.

2.5 Design and Implementation Constraints (5 pts)

The constraints that the code has are that the user needs to have a laptop that can run python and run window or IOS. We also need an IDE (Integrated Development Environment) or Text Editor to write and run the Python code.

2.6 Assumptions and Dependencies (5 pts)

The assumptions the code has is that if the user chooses to pick a place to live that they will be between three specific places. Similarly, if the user chooses to make a car payment, it is assumed that they will have one of the three payment options that is predetermined by the program.
The program also assumes that the user knows a base of how to do addition and subtraction. The program is set up so that the user must figure out the math on their own to help them learn how to do those calculations on their own. Thus, preparing them for the real world.

3. External Interface Requirements

3.1 User Interfaces (5 pts)

The only interface required for this system is a mouse and keyboard. The user uses the mouse to click on the specific buttons required to progress through the simulation. Examples of this include the start button, the exit button, the different tabs, etc. The user also uses the keyboard to type the amount of money that will be used in the simulation.

3.2 Hardware Interfaces (2 pts)

Monitor
Keyboard
Mouse

3.3 Software Interfaces (2 pts)

Toml File: stores data that is inputted in the text boxes (player name, money saved, month iteration)

4. System Features

4.1 Functional Requirements (40 pts)

Use Case ID: UC1
Use Case Name: New Game Start
Goal: Get the player's name and do one turn
Actors: User, System
Pre:

- There is an active connection between user the and the name input.
- There are enough funds to complete the transaction.

Post:

- The new Game is made successfully.

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	Load Sim Fin applications	2	System loaded application
		3	Displays main menu
4	User selects new game	5	Displays new game page
6	Input username		
7	Submit username	8	System saves username
		9	Displays main app page
10	User inputs rent amount		
11	User inputs food amount		
12	User inputs over all amount		

13	User submits data	14	System calculates data
15	User selects new month	16	System displays new month
17	User verifies new month data is correct		

Exceptions (Alternative Scenarios of Failure):

- User does not create new game.
- User does not input correct info on main app.
- User does not input a username.
- User fails to submit a new month.

Requirements:
Req: #1, #2, #3, #7, #8, #11, #12, #13, #14, #15, #16

Use Case ID: UC2
Use Case Name: Change Settings
Goal: Change settings and verify game is using selected settings.
Actors: User, System
Pre:

- **Load Simulation**
  - **Setting option available.**

Post:

- **Simulation has active tabs that are based on the options chosen.**

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	Select Settings button on main menu	2	Loads the settings menu
3	Select the settings user wishes to use	4	Show graphically choices selected
		5	System updates choices for next sim internally
6	User returns to main menu	7	Displays main menu
8	User selects new game	9	Displays new game screen
10	User inputs desired name		
11	User submits name	12	System saves name
13	User hits start	14	Displays main app page
		15	Displays tabs that have been selected from the settings
16	Select an added settings tab	17	Displays the tab with updated information
18	Input data into tab		
19	Submit data	20	Updates correctly off data received

Exceptions (Alternative Scenarios of Failure):

- **User does not create new game.**
- **User continues game.**
- **Users do not interact with the new tabs.**

Requirements:
Req: #1, #2, #3, #5, #7, #8, #13, #19

Use Case ID: UC3
Use Case Name: Verify Save File
Goal: Create Character with New Game. Confirm data in save file.

Actors: User, System, TOML
Pre:

- **Load application**
- **TOML file set**

Post:

- **Successful TOML file updated.**

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	Load new application	2	Displays main menu
3	Select new game	4	Displays new game page
5	Input username		
6	Submit username	7	System saves username to TOML
8	Select start	9	Displays main app page
		10	Loads page with name on main tab
11	User inputs any data		
12	User selects save and exit	13	System saves data to TOML
14	Open TOML	15	Loads TOML in notepad
16	Verify Name and data updated		
17	Load Sim Fin	18	Displays main menu
19	Select Continue	20	Load main app with displaying save name

Exceptions (Alternative Scenarios of Failure):

- **Not save and exiting correctly**
- **Not loading and continuing previous file**
- **Selection New Game erasing old, saved data.**

Requirements:
Req: #1, #2, #3, #4, #6, #7, #8, #9, #10, #17, #18

- Req. 1: The program shall display a graphical interface.
- Req. 2: The graphical user interface shall contain a main menu.
- Req. 3: The main menu shall have a New Game button.
- Req. 4: The main menu shall have a Continue button.
- Req. 5: The main menu shall have a Settings button.
- Req. 6: The main menu shall have an Exit button.
- Req. 7: The New game button shall initiate a new simulation instance.
- Req. 8: The New game button shall prompt the user to enter their name which shall be character string.
- Req. 9: The New game button shall start a new save file containing the users input name and the users progress through the sim.
- Req. 10: The Continue game button shall call a function to read in a save file and the users progress through a previous instance of the sim.
- Req. 11: The Core Sim page shall allow you to see your bills.
- Req. 12: The Core Sim page shall allow you to input money to pay bills.
- Req. 13: The Core Sim page shall have tabs based on your settings.
- Req. 14: A tab on the Core Sim page shall have information about housing.
- Req. 15: A tab on the Core Sim page shall have information about food and sustenance.
- Req. 16: A tab on the Core Sim page shall have information about Car bills.
- Req. 17: The Core Sim page shall have a save and exit button.
- Req. 18: The Save and Exit button shall save the variables to a Toml file.

- Req. 19: The settings button shall allow you to turn on options for your simulation.

4.2 Non-functional Requirements

4.2.1 Performance Requirements (5 pts)

PE.1 The program shall not crash while operating in its intended environment.

4.2.2 Safety Requirements (5 pts)

SA.1 The program shall not directly cause any damage to the hardware it is run on.

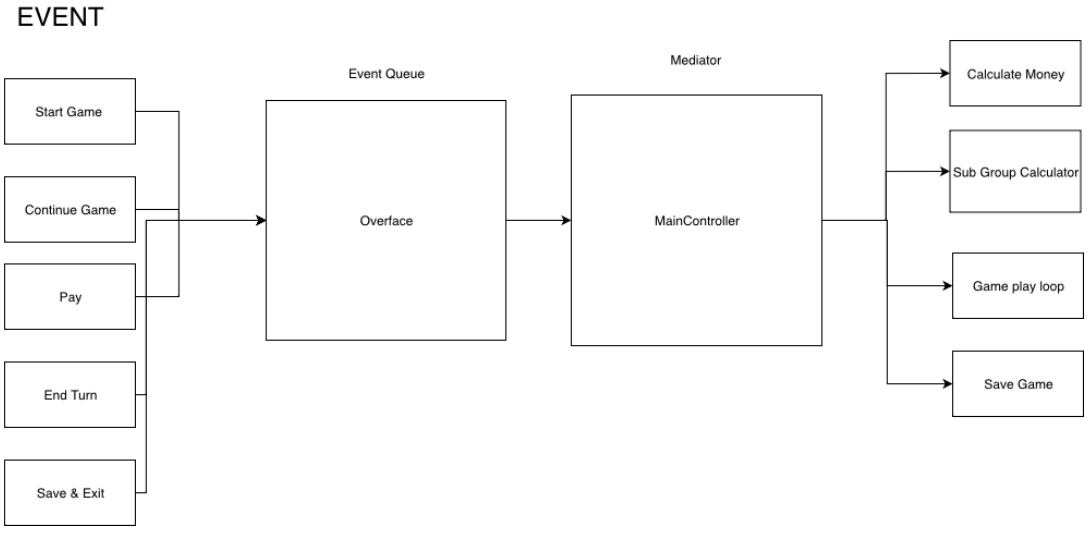
SA.2 The program shall not manipulate files outside of its intended files.

4.2.3 Security Requirements (5 pts)

SE.1 The program shall have read-write access on the computer it is run on only for reading and writing save files and leaderboard files.

5. Software Design

5.1 High Level Design (10 pts)

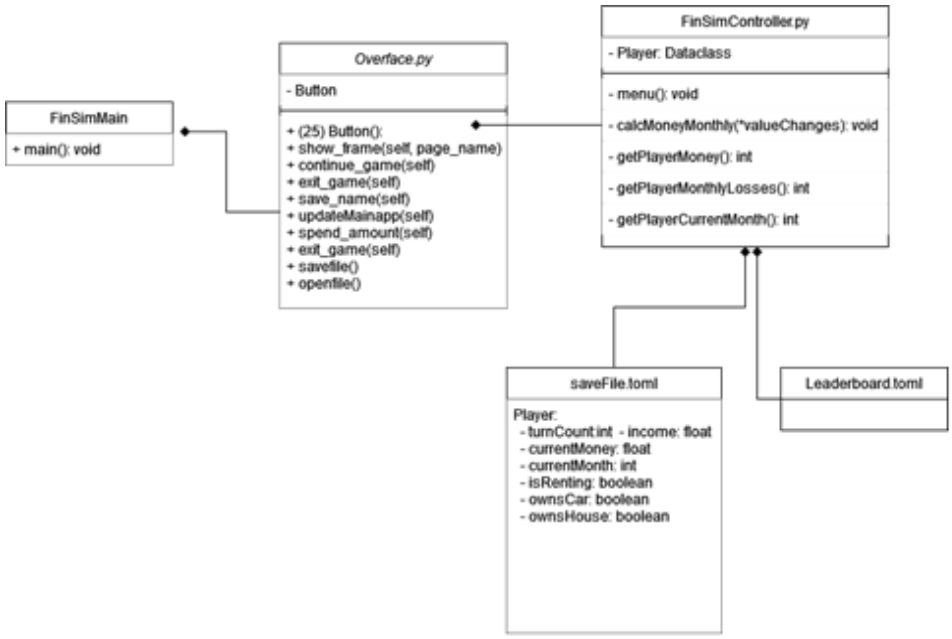


Event Drive Architecture

Our software design for FinSim incorporates an ‘Event Driven’ architecture. Players make decisions through Overface (GUI) which is our event router which calls functions contained within MainController.py/ FinSimController.py (These are the same). For example, if a user selects the

option to ‘pay’ for something such as their bills it routes the appropriate event to the user. (In this case it would be Calculate Money). When a user selects a page/ tab to view or move to (event) Overface then takes this event and displays the appropriate page with the appropriate modifications based on user action. (Routes to service)

5.2 Detailed Design (30 pts)



saveFile:

This is not a class, but this is a .toml format file that stores all the information contained in the Player data class and is updated at the end of every turn.

Leaderboard:

This is also not a class and is another .toml file that instead stores the players' names and scores who have performed the best in the sim. This is updated at the end of a playthrough and stores the top 20 players.

FinSimMain:

The first class is FinSimMain (FSM). FSM is responsible for instantiating an instance of the Overface (GUI) class. This is the only role that FinSimMain performs.

Overface:

Overface is the graphical user interface (GUI) for FinSim. This class is responsible for displaying the application to the user and is comprised of GUI widgets using the Python TKinter library. Overface also calls functions from the FinSimController class.

FinSimController:

FinSimController (FSC) is responsible for all of the calculations and serves as the backend for FinSim. FSC notably contains a class named Player which is a Python "dataclass" which is like a struct in C.

Overface:

show_frame(self, page_name):

show_frame receives the parameter self so that the actions it performs can access the methods and attributes of the class it is called in. It then receives the page_name parameter which is used to determine which page/ part of the GUI it will display.

continue_game(self):

continue_game receives the parameter self so that the actions it performs can access the methods and attributes of the class it is called in. continue_game is used to resume a game that was previously in progress. continue_game calls a method named open_file() which:

open_file():

open_file uses the tomllib library and reads in a toml containing a save state of the game if one is available. If not the continue_game method will reject the attempt to resume a game state since a previous one is not detectable by the program.

continue_game then takes the information read in from the toml file and places their values into the 'player' instance of the 'Player' Dataclass.

exit_game(self):

exit_game receives the parameter self so that the actions it performs can access the methods and attributes of the class it is called in. The method exit_game is used to 'destroy()' the GUI and end the program.

save_name(self):

The save_name method receives the parameter self so that the actions it performs can access the methods and attributes of the class it is called in. save_name takes the name (the name of the user) and passed it to FinSimController which places it into the name attribute into the 'player' instance of the 'Player' Dataclass.

spend_amount(self):

The spend_amount method receives the parameter self so that the actions it performs can access the methods and attributes of the class it is called in. The spend_amount method is responsible for retrieving the float value entered by the user into the spend_entry widget and passing it to FinSimController where it is added to the monthlyLosses attribute of the 'player' instance of the 'Player' Dataclass.

savefile():

The savefile() method takes all the attributes of the 'player' instance of the 'Player' Dataclass and writes it to a .toml format file to be read in if/ when the continue_game method is called.

FinSimController:

calcMoneyImmediate(*valueChanges):

This method is not currently called in any other methods or class constructors but is a method that immediately changes the player's money with positive or negative changes based on the arbitrary number of arguments it receives.

calcCurrentMoneyMonthly():

This method adds the income attribute from the 'player' instance of the 'Player' Dataclass to the currentMoney attribute and then subtracts the value contained in the monthlyLosses attribute of the same dataclass instance.

getPlayerMonthlyLosses():

This method is a basic 'getter' function. It is used by Overface to grab the monthlyLosses attribute from 'player' so that it may display it to the user through the GUI.

getPlayerMoney():

This method is a basic 'getter' function. It is used by Overface to grab the currentMoney attribute from 'player' so that it may display it to the user through the GUI.

getPlayerMoney():

This method is a basic 'getter' function. It is used by Overface to grab the currentMoney attribute from 'player' so that it may display it to the user through the GUI.

randomMonthlyDecisions():

This method is responsible for generating a random integer value and using that value to select a random event to present to the player. Examples include paying medical bills, car accident payments or in some cases no random event at all.

5.3 Detail Code

Code located at https://github.com/ERAU-SE300-DEV-TEAM/FinSim

6. Test Cases and Results (25 pts)

Test ID: SF_01	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**
Purpose	The program shall display a graphical interface.	
Dependencies	None	

Pass/ Fail Conditions	Display UI successfully		
Pre-conditions:	System must be up and running		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Run Program	Program runs	P
2.	View Display	Display viewed	P
Comments	Simple Display		

Test ID: SF_02	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**	
Purpose	The graphical user interface shall contain a main menu.		
Dependencies	ATM_01		
Pass/ Fail Conditions	The first displayed shall show a main menu. Main menu shall display five buttons		
Pre-conditions:	The system must be up and running. Must have a display		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Run Program	Program runs	P
2.	View Display		P
3.	View Main Menu	Main Menu is centered, buttons are available	P
Comments	Main Menu view varies based on screen size		

Test ID: SF_03	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**	
Purpose	The main menu shall have a New Game button.		
Dependencies	ATM_01		
Pass/ Fail Conditions	Button is displayed on the main menu. Button can be pressed and reacts		
Pre-conditions:	The system must be up and running. Must have a display		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Run Program	Program runs	P
2.	View Main Menu		P

3.	View New Game button	Button viewable	P
4.	Press New Game button	Initiates the new game page	P
Comments			

Test ID: SF_04	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**	
Purpose	The main menu shall have a Continue button.		
Dependencies	ATM_01		
Pass/ Fail Conditions	Button is displayed on the main menu. Button can be pressed and reacts		
Pre-conditions:	The system must be up and running. Must have a display		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Visit login page	Program runs	P
2.	View Main Menu		P
3.	View Continue button	Button viewable	P
4.	Press Continue button	Initiates the main app page	P
		Loads the past saved data to main app page	F
Comments	No saved data was created. No data could be loaded.		

Test ID: SF_05	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**	
Purpose	The main menu shall have a Settings button.		
Dependencies	ATM_01		
Pass/ Fail Conditions	Button is displayed on the main menu. Button can be pressed and reacts		
Pre-conditions:	The system must be up and running. Must have a display		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Run Program	Program runs	P
2.	View Main Menu		P
3.	View Settings button	Button viewable	P
4.	Press Settings button	Initiates the settings page	P

Comments	

Test ID: SF_06	**Designed by: Chris B.**	**Executed by (on date): Chris B. (04/28/23)**	
Purpose	The main menu shall have an Exit button.		
Dependencies	ATM_01		
Pass/ Fail Conditions	Button is displayed on the main menu. Button can be pressed and reacts		
Pre-conditions:	The system must be up and running. Must have a display		
Test Data:			
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/F**
1.	Run Program	Program runs	P
2.	View Main Menu		P
3.	View Exit button	Button viewable	P
4.	Press Exit button	Exits the simulation app	P
Comments			

6.1 Traceability Matrix (10 pts)

Requirement	Test case	Results
Req. 1	SF-01	Pass
Req. 2	SF-02	Pass
Req. 3	SF-03	Pass
Req. 4	SF-04	Pass
Req. 5	SF-05	Pass
Req. 6	SF-06	Pass

7. Appendix A: Glossary (5 pts)

7.1 Requirements

Core Sim page: the main page that the user will see while playing. This page includes, where the player will play, a turn iteration button, and the save game page. This page also has access to all tabs the user can click through to make changes to their payment and look at their debt.

Simulation instance: For our game, each instance is linked to the player's name. The simulation produces a data set based on the input given by the users for that instance. It is an execution of the simulation that generates and saves the data to the instance.

7.2 Design

Component Diagram: an abstract and breakdown of big items that focuses on divisions of tasks. It shows association through lines to describe the systems that directly interact.

Context Diagram: a high-level overview of the system that represents data flow between the system and external entities.

High level design: a type of visual representation that outlines the overall structure of our product. It provides a top-level view of the system's components, modules, interfaces, and their relationships.

IDE: Integrated Development Environment that is used to write, debug, and run code

Sim Fin: Name of the application. A financial Simulation. Also referred to as Fin Sim.

Use Case Diagram: a visual representation of how actors interact with the system. It outlines the different tasks or functions the system performs based on who or what is interacting with it.