Project Title: The Budget Game: Digital Edition

Name: Evan Hill

Degree and Major: Applied Computing: Business Concentration

Project Advisor Name: Michael J. O’Neill

Statement of Purpose & Subject Matter Background:

Picture this: Every student who goes into college leaves with the knowledge, skills, and passion necessary to manage their personal finances well, not just while in college but also after they graduate and get out into the workforce.

However, at present, we are far from this ideal state of affairs. While there are some students (such as myself) who are passionate about personal finance and are prepared/preparing to manage our finances well after college, students like this are definitely in the minority. Even though CSU offers personal finance classes and has personal finance peer coaching programs where student coaches can meet with other students one-on-one to help them with their personal finances and can go out to freshman seminar classes to present on personal finance, the coaches’ current presentation and student engagement methods during the presentation are not the most engaging and definitely could use an upgrade to make them more engaging in order to get the students being presented to more passionate about personal finance while still empowering them with the knowledge they need to succeed.

I know this because I’ve seen it firsthand. I’ve worked as a personal finance peer coach here at CSU and did both one-on-one coaching and presenting to freshman seminar classes. While most of the one-on-one coaching recipients are passionate about getting their finances in order (because if they weren’t then they wouldn’t sign up for a meeting), the students being presented to in the freshman seminar classes are definitely more of a mixed bag. Some are fairly passionate about their finances, but most seem to be more indifferent or even outright uninterested in learning these essential skills. I think this problem is not helped much by the peer coaching program’s current implementation of the budget game which they give at these presentations.

My solution, therefore, is to use my computer science skills to create a digital budget game for the peer coaching program with the CSU Center for Personal Financial Management (CPFM) to use in their freshman seminar programs to better engage the students and get them more passionate about learning personal finance skills.

Research & Background: I needed to learn more about Oracle APEX, Python, and web scraping in general in order to make this project successful. I was already familiar with Oracle APEX from CSU’s Database Management class, but web scraping was new to me, and I only had a passing familiarity with Python. However, through my research, I discovered that Oracle APEX would not allow me in any way to link a database to an external application such that the application can pull data from the database. As a result, I had to cancel the database and the web scraper because there would be no point for either going forward from this point in the project development, with the final application that the user would interface with being the sole product of this project.

Personal Motivation: My personal motivation for undertaking this project is largely because I want to give back to the CPFM by giving them a new and improved budget game for them to use in their freshman seminar presentations in order to better connect with the incoming freshmen and better help them get interested in personal finance.

Project Requirements:

(Note: Most of these requirements are now obsolete due to the heavily altered project scope. They are still included here for historical documentation purposes.)

Requirement #: 1 (Whole Project)  
Requirement Type: Security

Description: We will not store any personal data in this project because doing so would deviate from the original intent.

Rationale: There is no need to store personal information for this project, so I want to clarify that upfront here.  
Fit Criterion: Make sure that all data stored is obtained from publicly-available sources and that there is no personal user information in the project database.

Priority: Top priority

Dependencies: None. This requirement should be done first and upheld throughout the entire project.

Requirement #: 2 (Create Database Schema and Tables)  
Requirement Type: Functional

Description: The database schema and tables need to effectively store all the data required to make this project work, which includes average salary data for the different jobs available in the simulation game as well as all the prices and costs for things like places to live, cars, clothes, technology, and other expenses that would be included in the simulation game. Should also be compatible with data insertion from .csv files.

Rationale: If the data is not easy to access or insert, that will make the game run slower for end users and be more difficult than necessary to program and maintain, both of which are undesirable.

Fit Criterion: The data should be stored in a way that is easy for the game program to access, and the table structure should be able to easily accept new data from .csv files.

Priority: High

Dependencies: R#1

Requirement #: 3 (Create Database Schema and Tables)  
Requirement Type: Performance

Description: Database schema and tables must have quick access to data (that is, they must have fast performance)

Rationale: Data should not take a long time to access.

Fit Criterion: If testers notice that data is taking a noticeable amount of time to access, investigate the cause of that to determine if it’s related to the database schema and tables themselves.

Priority: High

Dependencies: R#1, R#2

Requirement #: 4 (Create Database Schema and Tables)  
Requirement Type: Look and Feel

Description: The database schema and tables need to look orderly, but the presentation of the schema and tables is ultimately not that important.

Rationale: The database schema and tables will not be viewed by end users, so they need to look good only to the staff which will be maintaining them behind the scenes.

Fit Criterion: If I (the staff maintaining the schema and tables behind the scenes) think that the schema and tables look good, then this requirement can be considered a success.

Priority: Low

Dependencies: R#1, R#2, R#3

Requirement #: 5 (Create Database Schema and Tables)  
Requirement Type: Maintainability and Support

Description: The database schema and tables should be easy to maintain and support over the long run.

Rationale: This program may run for several years and may undergo several transfers of responsibility.

Fit Criterion: Over the long run and in the moment, we can reflect on whether the schema and tables are easy to maintain and support. There may be some hurdles introduced by the Oracle APEX software, but overall, it should not be my fault if we run into any maintainability issues.

Priority: High

Dependencies: R#1, R#2, R#3

Requirement #: 6 (Create & Run Web Scraper, then upload scraped data to database)  
Requirement Type: Functional

Description: The web scraper should be able to gather all the necessary data from the internet and compile that data into a .csv file which I can then upload to the database schema to insert the values. As such, the .csv file must be compatible with the database.

Rationale: This program may need to be run every few years for several years and may undergo several transfers of responsibility, plus if the web scraper does not work as designed then the whole project falls apart.

Fit Criterion: Check the formatting of the text in the .csv file while developing the web scraper to ensure compatibility with the database. Also, check the web scraper’s programming to make sure that it can scrape the correct data in a proper manner.

Priority: High

Dependencies: R#1

Requirement #: 7 (Create & Run Web Scraper, then upload scraped data to database)  
Requirement Type: Performance

Description: Web scraper should be reasonably fast and work properly each time it is run.

Rationale: If the web scraper is not fast and/or not reliable, it will be difficult to maintain over the long term, which violates R#6.

Fit Criterion: Trained admins should agree that the web scraper is reasonably fast and works properly each time it is run.

Priority: High

Dependencies: R#1, R#6

Requirement #: 8 (Create & Run Web Scraper, then upload scraped data to database)  
Requirement Type: Usability

Description: Backend software that should be usable to trained admin(s).

Rationale: This will only be used by trained admins, not end users.

Fit Criterion: The web scraper will be easy for trained admins to use if we can just set it and forget it each time we have to run it.

Priority: Medium

Dependencies: R#1, R#6, R#7

Requirement #: 9 (Create & Run Web Scraper, then upload scraped data to database)  
Requirement Type: Look and Feel

Description: Backend software that should be presentable to trained admin(s).

Rationale: This will only be used by trained admins, not end users.

Fit Criterion: The trained admin can understand at a glance how to navigate and use the software.

Priority: Medium

Dependencies: R#1, R#6

Requirement #: 10 (Create Game Application)  
Requirement Type: Functional

Description: The game application should not only be a re-creation of the original budget game, but should also improve the game where it’s most needed like with visuals, for example.

Rationale: This project will be a replacement for the old budget game, so it needs to do all the same things that the old budget game did, but do them better than the old version.

Fit Criterion: Compare new budget game to the old one to see whether it is a faithful re-creation along with whether what needed to be improved was indeed improved upon.

Suggested Improvements:

1. Consolidate the entire game into one application without any extra discrete parts.

2. Integrate coin-flipping for the game into the software.

3. Integrate the necessary math calculations that user needs to perform into the software.

4. Integrate the user interactivity elements (such as choosing a career, etc.) and interface into the software.

Priority: High

Dependencies: R#1

Requirement #: 11 (Create Game Application)  
Requirement Type: Look and Feel

Description: The game app should be visually appealing and feel good to use for end users.

Rationale: The game app is the part of this project that will actually be used by end users, so the game’s visuals must be good in their eyes.

Fit Criterion: I can have people play the game app for testing purposes and during the testing, I can have them rate how much they like or dislike the visuals.

Priority: High

Dependencies: R#1, R#10

Requirement #: 12 (Create Game Application)  
Requirement Type: Performance

Description: The game app should perform well, that is, it should not be slow to run on a wide variety of devices.

Rationale: If the game app runs slowly, users will not have a fun time playing the game. Even if a user runs the game on a slow device, the game app should still run at a playable speed.

Fit Criterion: During the testing phase, I can have users with a variety of different devices test the game app and report how well they perceived the performance to be.

Priority: High

Dependencies: R#1, R#10, R#11

Requirement #: 13 (Create Game Application)  
Requirement Type: Performance

Description: The game app should also be built well in the sense that the code should be robust and fault-tolerant. The game’s code should not crash easily, but ideally not crash at all.

Rationale: If the game crashes while running and/or is not fault-tolerant, then it will cause a bad experience for the user and be problematic to try to fix on the fly.

Fit Criterion: During the testing phase, I can test for bugs as well as have users test for bugs. Based on the results of these tests, I can fix the code as needed to produce a more stable result.

Priority: High

Dependencies: R#1, R#10

Requirement #: 14 (Test project thoroughly as per test plan document)  
Requirement Type: Functional

Description: Any and all tests I run on this project should be good tests that return valuable information.

Rationale: Tests that do not return valuable information are a waste of time to run.

Fit Criterion: Any test that is run should return at least some information that contains value, which means that the information can be used to help improve the project.

Priority: High

Dependencies: All requirements in this document

Requirement #: 15 (Deliver Finished Product)  
Requirement Type: Functional

Description: The finished product should be delivered in a way that is easy for end users to use and is easy for me to maintain.

Rationale: If the finished product is not easy to use and/or is difficult to maintain, then that creates unnecessary problems and headache for all parties involved, which should be avoided.

Fit Criterion: Run tests with end users to get their opinions on whether the game is easy to use and navigate, and check up with admins every so often to get their opinions on whether the project is easy to maintain.

Priority: High

Dependencies: All requirements in this document

Requirement #: 16 (Deliver Finished Product)  
Requirement Type: Security

Description: Again, we need to make sure that the project doesn’t store any user data, as described in R#1.

Rationale: We’re checking for this again here in order to limit scope creep.

Fit Criterion: Make sure that all data stored is obtained from publicly-available sources and that there is no personal user information in the project database.

Priority: High

Dependencies: R#1, and the project needs to be finished by this point.

Project Description: My project will be a digital version of the budget game that will engage students better than the existing budget game.

Proposed Implementation Languages: C++ for the game application.

Libraries, Packages, Development Kits, etc. to be used in in the proposed implementation languages(s): N/A

Additional Software/Equipment Needed: N/A

Project Implementation Description & Explanation:

Source Code Repository Link: <https://github.com/echill76/CSU_Senior_Project>

Screenshots of Project:

A black screen with white text

AI-generated content may be incorrect.

*Fig. 1 shows the very beginning of the game.*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 2*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 3 shows the first unforeseen event. Currently, these happen at preset intervals, but the exact event that ends up occurring is chosen at random by the software.*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 4*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 5*

*A black background with white text

AI-generated content may be incorrect.*

*Fig. 6 shows the short-term and long-term savings amounts. These are chosen by the user and can be any double-type numerical input.*

*A black background with white dots

AI-generated content may be incorrect.*

*Fig. 7*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 8 shows the user’s technology choices. The user can say yes or no to any of them and the impact on the user’s in-game budget is adjusted according to their choices. If the user inputs any character other than ‘y’ or ‘n’, the program continues by treating the erroneous input the same way as an input of ‘n’.*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 9*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 10*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 11*

*A black screen with white text

AI-generated content may be incorrect.*

*Fig. 12 shows the good ending, which occurs when you end the game with a budget surplus or by breaking even. There is also a bad ending, which you get by ending the game with a budget deficit.*

Test Plan:

(Note: Most of these tests are now obsolete due to the heavily altered project scope. They are still included here for historical documentation purposes.)

Test #: 1 (Whole Project)

Expected Result: No user data is stored in the final version of the project. In other words, all data stored is obtained from publicly-available sources and there is no personal user information in the project database.

Actual Result: Success

Comments: None

Test #: 2 (Create Database Schema and Tables)

Expected Result: The data should be stored in a way that is easy for the game program to access, and the table structure should be able to easily accept new data from .csv files.

Actual Result: Success

Comments: None

Test #: 3 (Create Database Schema and Tables)

Expected Result: Testers have no complaints regarding data access speed.

Actual Result:

Comments:

Test #: 4 (Create Database Schema and Tables)  
Expected Result: I think that the data and tables are organized in a way that makes sense to me.

Actual Result: Success

Comments: None

Test #: 5 (Create Database Schema and Tables)

Expected Result: The database schema and tables will be easy to maintain and support over the long run.

Actual Result:

Comments: I’m not sure if I should say now if this test has succeeded or failed or if I will need to wait and see over the long run.

Test #: 6 (Create & Run Web Scraper, then upload scraped data to database)

Expected Result: The web scraper is able to gather all the necessary data from the internet and compile that data into a .csv file which can then be uploaded to the database schema to insert the values. As such, the .csv file is compatible with the database.

Actual Result: Success

Comments: None

Test #: 7 (Create & Run Web Scraper, then upload scraped data to database)

Expected Result: The web scraper is reasonably fast and works properly each time it is run.

Actual Result:

Comments:

Test #: 8 (Create & Run Web Scraper, then upload scraped data to database)  
Expected Result: The web scraper will be easy for trained admins to use if we can just set it and forget it each time we have to run it.

Actual Result:

Comments:

Test #: 9 (Create & Run Web Scraper, then upload scraped data to database)  
Expected Result: The trained admin can understand at a glance how to navigate and use the software.

Actual Result:

Comments:

Test #: 10 (Create Game Application)  
Expected Result: The game application is not only a re-creation of the original budget game, but also improves the game where it’s most needed like with visuals, for example.

Actual Result: Success

Comments: None

Test #: 11 (Create Game Application)

Expected Result: The game app is visually appealing and feels good to use for end users.

Actual Result: Success

Comments: None

Test #: 12 (Create Game Application)  
Expected Result: The game app performs well, that is, it is not slow to run on a wide variety of devices.

Actual Result: Success

Comments: None

Test #: 13 (Create Game Application)  
Expected Result: The game app is built well in the sense that the code is robust and fault-tolerant. The game’s code does not crash easily.

Actual Result: Success

Comments: None

Test #: 14 (Test project thoroughly as per test plan document)  
Expected Result: Any test that is run returns at least some information that contains value, which means that the information can be used to help improve the project.

Actual Result: Success

Comments: None

Test #: 15 (Deliver Finished Product)

Expected Result: The finished product is delivered in a way that is easy for end users to use and is easy for me to maintain.

Actual Result: Success

Comments: None

Test #: 16 (Deliver Finished Product)

Expected Result: Again, all data stored is obtained from publicly-available sources and there is no personal user information in the project database.

Actual Result: Success

Comments: None

Challenges Overcome: The main challenge of this project was discovering late into the development that the database platform I was using would not work with the end user application at all, resulting in me having to overhaul the scope of the entire project.

Future Enhancements: There are many possible future enhancements, including, but certainly not limited to, adding a GUI, finding a database that will actually work the end user application and support regular updates as needed, adding an separate ending case for breaking even on the budget, and including other frontend embellishments to keep users even more engaged with the end user application.