

**Santa Clara University**  
**DEPARTMENT of COMPUTER ENGINEERING**

**Date: June 11, 2009**

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY  
SUPERVISION BY

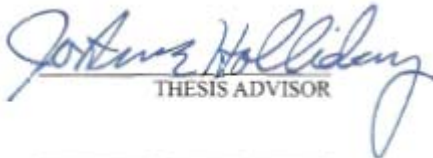
**Marco Echandi, Jason Goetsch, and James Lewis**

ENTITLED

**Educational Tool for Finances**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF

**BACHELOR OF SCIENCE IN COMPUTER ENGINEERING**

  
THESIS ADVISOR

DEPARTMENT CHAIR

# **EDUCATIONAL TOOL FOR FINANCES**

by

Marco Echandi, Jason Goetsch, and James Lewis

## **SENIOR DESIGN PROJECT REPORT**

Submitted in partial fulfillment of the requirements  
for the degree of  
Bachelor of Science in Computer Engineering  
School of Engineering  
Santa Clara University

Santa Clara, California

June 11, 2009

## **Abstract**

The Independence Network, an institution which educates mentally and physically disabled adults, observed the lack of educational software concerning daily personal finances. In particular, educators at the Independence Network struggled teaching the different values associated with distinct currency denominations – for example, the value ascribed to a five dollar bill versus a one dollar bill. The educators were especially concerned that when purchasing products or services, many of the students would be unable to correctly calculate the change which they ought to receive back from a vendor. As a result, Brian Darby, an instructor at the Independence Network, contacted the Santa Clara Engineering Department about developing software to assist the students in their lessons on personal finances.

Our software product has been designed to teach personal finances to students at the Independence Network in a way which engages the students by incorporating elements from their classroom environment. The product has a similar presentation and flow to Starfall.com educational tools, which are used frequently at the Independence Network for reading exercises. Similar to these educational tools, our product emphasizes dynamic interaction with the user, complemented by an intuitive graphical interface. Furthermore, our software features activities which the students regularly complete at the Independence Network, such as riding in a bus or purchasing groceries at a supermarket. By employing this educational tool in conjunction with related classroom exercises, the students are able to reinforce their knowledge of personal finances.

## **Acknowledgements**

Our group is thankful to all of the people who helped take this project from a vision into a reality. First, we would like to take the opportunity to thank Mr. Brian Darby, who is the originator of this project. Mr. Darby, who educates students every week at the Independence Network, was the first to take note that his students could greatly benefit from an educational financial tool to assist them in daily financial tasks. Furthermore, this project could not have been successful without the assistance and support of Dr. JoAnne Holliday, who helped guide us through this project from the design phase until the finished product.

We would also like to thank Dr. Shoba Krishnan and Patti Rimland for listening to Mr. Darby's concerns and subsequently raising awareness of the issue and notifying our group about this community-based project. We would also like to thank Santa Clara University, for facilitating the senior design project, as well as for purchasing the Adobe Flash Suite.

## Table of Contents

	Page
Abstract.....	1
Acknowledgements.....	2
Table of Contents.....	3
List of Figures.....	4
Introduction.....	5
Development Timeline.....	7
User Guide.....	10
List of Requirements.....	17
Design Rationale.....	19
Technologies Used.....	21
Use Cases.....	22
System Sequence Chart.....	26
System Architecture.....	28
Project Risks.....	30
Test Plan.....	33
Societal Issues.....	35
Conclusion.....	40
References.....	42
Appendix A.....	43

## List of Figures

	Page
Figure 1 Main Menu Graphical Interface.....	11
Figure 2 Payment Screen Graphical Interface.....	12
Figure 3 Grocery Graphical Interface.....	13
Figure 4 Bus Graphical Interface .....	14
Figure 5 Bus Graphical Interface After Paying Fare.....	14
Figure 6 ATM Graphical Interface.....	15
Figure 7 Change Screen Graphical Interface.....	16
Figure 8 System Sequence Chart.....	25
Figure 9 System Architecture.....	27
Figure 10 Risk Identification and Prioritization .....	29

## **Introduction**

The Independence Network provides an environment for mentally challenged adults to learn and to become educated in everyday matters that may come easier to people that are not mentally challenged. One of the ways in which the network attempts to educate the students is with regard to financial matters. Due to the variety of mental disabilities experienced by the students at the Independence Network, teaching each individual requires a wider approach to teaching methods, since some students learn in different ways. At the Independence Network, most students are unable to process more abstract concepts like numbers and money without a good example.

The students' special needs demand that the school accommodate their learning abilities, and to accomplish this, the school is looking for a way to teach the students about financial transactions in an easy to use and pictorial interface. The Independence Network approached Santa Clara University through Professor Shoba Krishnan to request some students with the necessary skills required to help on this project. Professor Krishnan contacted Marco Echandi and we collectively decided to pursue assisting the Independence Network for our senior design project.

The core of the project revolves around creating a web interface that allows the students to practice everyday banking and commerce situations. The tool will make use of large, detailed pictures to help the students learn about monetary denominations and values, so that they can take these skills into real life situations with more confidence and knowledge. Sample situations for the tool to demonstrate include traveling by bus, shopping at a grocery store, and checking a bank account. The banking aspects will contain real pictures of various denominations of dollar bills so that the students can familiarize themselves with the characteristics of bills, the numbers, the colors, and the presidents.

The program we have created uses Flash browser plug-ins and is compatible with most current browsers. The use of Flash ensures that those students without access to Internet connections at home will still be able to run and use the program on their own time. Flash should provide an easy to use, easy to maintain front for the tool, as well as allow for additional functionality to be added which may aid the students, such as text to speech abilities and sound plug-ins for students who have difficulty learning visually. Other technologies were considered for the project, but it was decided that other technologies may require frequent updates or limit those students without access to certain hardware and software.

The scope of this project encompasses the financial tool for the students, allowing them to play and learn at the same time. After each implementation change, continuous feedback testing assisted us in improving the project as well as providing the Independence Network with a core learning tool that can be expanded upon in the future, potentially adding more features and programs on top of the existing financial tools. Future projects could encompass more needs of the school with regards to learning tools and incorporate more feedback sessions.

The project was completed on June 5, 2009 and provides the Independence Network with a functional tool to assist in teaching mentally challenged students life skills that will benefit them throughout their lives.

## **Development Timeline**

Every quarter requires a certain amount of work to be completed so that we may deliver our product on time. Listed below are the timeline details of our project. Each quarter demonstrates the focus of that specified week and helped our group in organizing what needs to be done.

### Summer Quarter 2008: Requirements Analysis and Mock-Up

In our first quarter on the project we met with our customer, Mr. Brian Darby, to discuss the requirements of the project. We then began our first step of requirements analysis and made sure to ask questions in order to clarify each requirement. We received enough input from the customer that would allow our team to have a good idea of what the project is about and how we were going to be able to deliver a solution to the customer's problem. Next, our job was to begin the creation of our mock-up to allow for a good foundational view of the project and to agree with all features and other necessities.

### Fall Quarter 2008: (Week 1 – 3) Problem Statement

We began the new academic year by working on our problem statement which encompasses the fundamentals of the project and how we are intending to design and implement the project. We then began to work on the design document for our project, which involved solidifying our decision to use Adobe Flash as our programming suite and the scope of the features that we believed we could finish in the time we have.

### Fall Quarter 2008: (Week 4 - 7) Design Analysis and Design Document

We then began looking and researching on how to complete the design document and prioritize our work. The creation of the design document allowed us to begin to envision how our project should be implemented and let us start splitting the work to beginning implementing the project. During this week, we turned in a rough draft of the design document and began preparing for the initial implementation and design review of the project.

### Fall Quarter 2008: (Week 8 - 10) Preparation for Design Review and Initial Implementation

We have checked over all the requirements and necessities that the project entails as well as other features that we believe we can finish by the end of the quarter to help with ease of use in the interface. We began working on the initial implementation and have equally allocated work to ensure efficiency and progress of the implementation while we are away during winter break. We have also completed a PowerPoint for the design review and have prepared what to say for the design presentation.

### Winter Quarter 2009: (Week 1 - 3) Initial Operational System

The initial operational system demonstrated the earliest version of our flash application. In this version, all requirements deemed critical have been completed and ready to use. We will adjust requirements and prioritize critical features as we receive feedback from the customer.

### Winter Quarter 2009: (Week 4 - 7) Revise Design

After the design review, the original design document has been revised and adjusted after talking with the customer to receive comments and criticisms. Based on the newly formatted design document, we will begin implementing the four-year schedule web application.

### Winter Quarter 2009: (Week 8 - 10) Maintenance

Finish implementing the requirements for the project, organize deployment instructions, and start developing user scenarios, involving multiple test cases to increase usability and reliability. Test cases consist of multiple inputs along with expected outputs to test all aspects of the product implementation. Record all test results and prepare to present a functional product.

Spring Quarter 2009: (Week 1 - 4) Final System / Presentations

We finished implementation of the product and finalize all requirements of high priority. Moreover, we delivered a flash application demonstrating functionality of an Educational Tool for Finances to be tested by the students. Also, sat down with Mr. Darby several times to ensure project requirements were met during feedback.

Spring Quarter 2009: (Week 5 - 10) Final Report

Met with customer and finalized the product from feedback of beta testing with the students. We presented our project at the senior design conference on May 8, 2009. Completed the final report and added any remaining low-level details.

## **User Guide**

The user guide may be used as a guide for a first-time user when becoming acquainted with our product. All functionality of the product will be described in detail and with screenshots. By the completion of our project, we found that the GUI and software continuity was subject to the most frequent changes of any aspect of our software after reviews with our customer and feedback from usability tests. These two aspects were significant in regards to the educational value the students found in our product.

## Main Menu

The screenshot below depicts our educational software's main menu. This menu is the only component of the application which allows the user to access all other activities in the program. The main menu also serves the purpose of giving the user a task to complete in order to provide continuity for the user. In order for this new task to be displayed, the user must click on the "New Task" button. For the example shown below, the user is supposed to purchase two cartons of eggs, one box of macaroni, and four bottles of ketchup. In order to proceed, the user must click on the "Bus Stop" button. The goal of the "New Task" is to keep the user interacting with the application, and in fact the software actually checks to see if the user has purchased the correct groceries (although the user may purchase other groceries in addition to the "New Task" groceries). The Main Menu will be the first screen a user sees when executing this program, and will return to the Main Menu after completing tasks or pressing the "EXIT" button from the bank, grocery store, or bus. Pressing the "EXIT" button from the Main Menu will exit the program entirely.

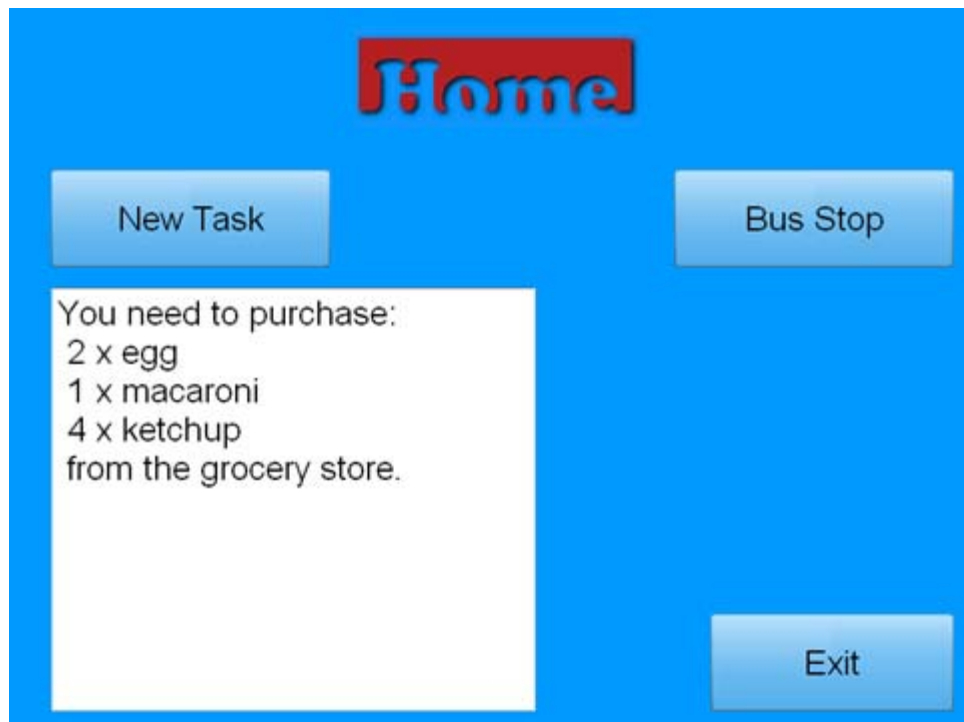


Figure 1: Main Menu Graphical Interface

## The Payment Screen

The Payment Screen depicted below serves multiple purposes for the user. The Payment Screen is the application's main tool for teaching users about US currency. As is shown in the example below, the Payment Screen provides the user with their bank balance, along with the actual breakdown of the physical currency which is contained in the user's wallet. The purpose of showing the breakdown of the physical currency is to increase the user's understanding of the actual economic value attached to each denomination (at the Independence Network, currency recognition is a major roadblock to understanding personal finance among many students there). Although the user may have enough money in their account to complete a daily task, the user must also actually withdraw the amount they desire from their bank account into their wallet in order to pay for goods or services in the game. When the user pays for their groceries, the user will be taken to the Change Screen, where they are shown the exact change they are owed back from the Grocery Store. The "EXIT" button the Payment Screen menu will return the user to the Main Menu.



Figure 2: Payment Screen Graphical Interface

## The Grocery Store

The purpose of the Grocery Store is to serve as one of the user's possible tasks. The user will be able to browse the store's inventory and select items they wish to purchase. The user will also have a shopping cart which consists of a list of the items they have currently selected, the ability to remove these items, and a running total of how much they will pay at checkout with their current shopping cart. The grocery aisles are represented by a three-tabbed menu, which the user can browse as if browsing through different grocery aisles. If the user has clicked "New Task" at the main menu, the user is then also presented with the specific items they need to buy as a reminder (similar to a grocery list). The user would also have the ability to purchase other items at their discretion. The purpose of allowing users to do this is to teach through experience about spending and to increase the user's ability to set spending priorities. Since the user has a finite money supply, they will need to be able to set spending priorities in order to avoid situations where they cannot complete a daily task due to not having the required amount of money. If the user did not bring enough physical cash to pay for their groceries, the user will have to go to the Bank and withdraw money.



Figure 3: Grocery Graphical Interface

## The Bus

The Bus menu serves a similar purpose to that of the Grocery Store, by contributing to user understanding of spending. Riding the bus is required in order for the user to travel to the Grocery Store and the Bank. Similar to the Grocery Store, the user will need to have enough cash on hand to pay for their fare. If the user does not have enough cash on hand, they will have to return to the Bank in order to withdraw the correct amount of money. Unlike the Grocery Store, the user does not receive change back, as the public bus system in Santa Clara does not provide change back. Figure 4 depicts the Bus Stop screen before the user has paid their fare, while Figure 5 depicts the Bus Stop screen after the user has paid their fare and is able to go to the Bank or the Grocery Store.



Figure 4: Bus Interface

Figure 5: Bus Interface (After Paying Fare)

## The ATM Scenario/Bank

To contribute to the user's understanding of ATM machines, this application includes an ATM interface for the Bank scenario. The interface has been designed to most closely resemble a real life ATM interface, to allow users to become comfortable with using ATMs before using them in real situations. As shown in the below figure, the user is requested to enter a pin – any four digit pin number works. The user is then able to access their bank account and withdraw money as they please, which is deposited into the user's wallet.



Figure 6: ATM Graphical Interface

## The Change Screen

In order to assist in reinforcing the relation of money to goods and services, this product also features a Change Screen. After a user pays for their groceries, they are automatically taken to the Change Screen which displays the change which the user ought to receive back, visually represented by the actual US currency denominations a cashier would be required to provide. In this example, the user has paid with a \$20 dollar bill for a grocery bill of \$8.25, and thus should receive \$11.75 in return.

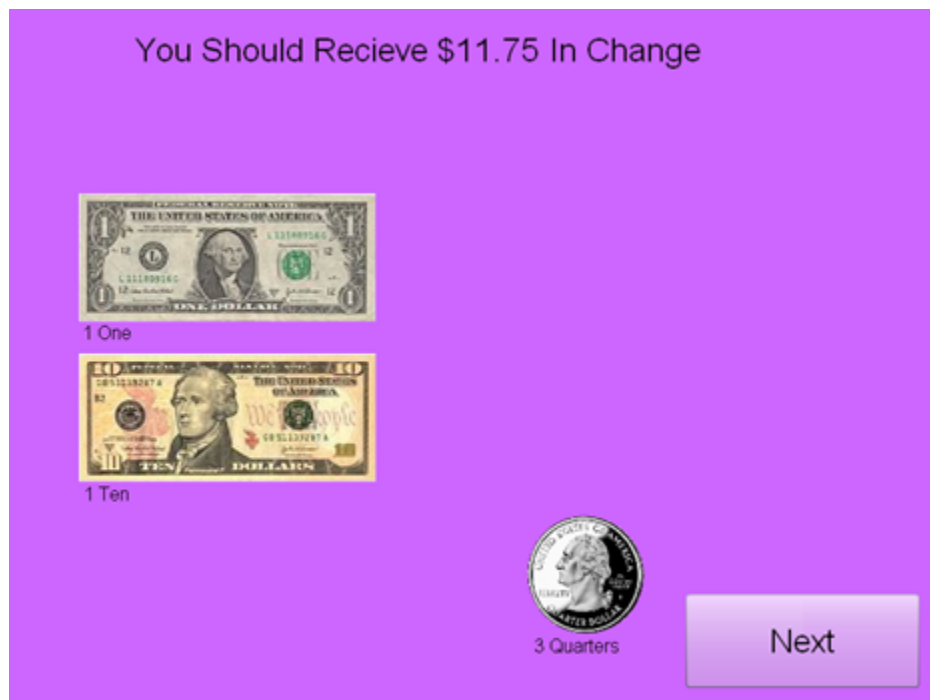


Figure 7: Change Screen Graphical Interface

## List of Requirements

After discussing requirements with the customer, we collected a list of functional and non-functional aspects necessary to complete the product. To differentiate between functional and non-functional requirements, definitions are included below along with the list. The bullet points in the lists start from high to low importance as they are numbered from one to three.

Functional – Requirements that are related to the system and specifies its behaviors and how it functions.

Non-Functional – Requirements that are related to usability and the overall look and feel.

Functional Requirements listed in highest importance (1) and lowest importance (3)

1. (1) Visual Educational Tool for Finances to assist with the understanding of monetary value and financial responsibilities
2. (1) Account Calculator/Management to provide for assistance with finances
3. (1) Mode Selection creating the user interface
  - Banking – simulate personal finance management
  - Grocery Store – simulate transaction between items and money through the use of a shopping cart and the banking modules
  - Bus – simulate interaction between services and currency.
  - ATM – simulate a real life situation where the user would be able to withdraw money
4. (1) Consistent and recognizable icons for each mode to provide for user friendliness
5. (2) Ability to copy/paste to other applications such as online banking websites

Non-Functional Requirements listed in highest importance (1) and lowest importance (3)

1. (1) Web-based standalone application to provide for easy and portable accessibility
2. (1) Graphical Interaction that allows for user relations
3. (2) The application will be user-friendly to cater toward mentally challenged adults
4. (3) Supported Web Browsers: Mozilla FireFox and Internet Explorer

## Design Rationale

Following the requirements elicitation and analysis phases, our software's design became restricted due to our customer requirements and needs. However, after analyzing the variety of ways in which these requirements could be implemented, our team realized that many feasible approaches could be used to correctly implement our software solution. To evaluate a given design decision, our group used multiple criteria to compare the decision to its alternatives. First, we evaluated whether the decision would increase the quality of our end product. Second, we determined if the decision would increase the overall utility of our software in order to benefit more individuals. Finally, we also looked to see if a certain decision could help our team members gain experience in relevant software practices for our future as computer engineers.

The first design decision which we made was whether our software would be a standalone executable program or a web based solution. Analyzing this decision based on the above criteria, we found that this decision would likely have no effect on the overall quality of our end product. However, implementing our project as a web based solution has many advantages for the second and third criteria. Chiefly, a web based solution allows a user to access our application without needing to download and install our application. Furthermore, web applications are generally considered more independent of specific platforms. These factors have allowed us to better reach our goal of creating a more widely accessible product, while gaining experience in web programming.

The second design decision we made was selecting the programming language through which we will implement our software solution. We have chosen to use ActionScript, which is a scripting language which leverages the Adobe Flash Player platform. We believe that this language is preferable for our project's purposes for a number of reasons and satisfies the stated criteria. ActionScript is considered one of the

most efficient means of animation, and one major feature of our application is animation. Furthermore, ActionScript will help to render our application on a variety of operating systems and web browsers. This is because ActionScript utilizes the Adobe Flash Player, which is a multiplatform application that should appear the same on all browsers as long as the proper plug-in is installed. This should thus reduce extra efforts made towards cross-browser compatibility. This will contribute to each team member's computer engineering knowledge by allowing us to gain experience with an established web programming language which we had no experience with prior to this project.

Another important design decision was the selection of our color scheme and font sizing. Doing some research through the web, we were able to see a particular trend of softer and warm colors as well as larger and clear sizes and fonts. We decided to make sure that the color of the buttons in the main menu was also the same as the color of the background in each section. We also associated particular colors that people would be able to recognize, for example, we used green as the color of the button and background to relate the color of the bills to the banking section. The large and clear font was a critical part of our design rationale as we were catering to mentally challenged adults. Clarity, familiarization and attractiveness were key design decisions.

## **Technologies Used**

Our project utilizes the Adobe Flash web technology suite as well as a standalone technology through the use of the browser to facilitate the creation of our Educational Tool for Finances. Since most operating systems are incorporated with web browsers, we have been able to provide ease of portability and a more simplified structure to reach the maximum amount of students. We decided to use Adobe Flash to provide for an interactive experience as the students learn how to relate a dollar denomination to a value as well as understand when and where to use currency. The Flash program will allow for students with or without an internet connection to use our program.

Adobe Flash – is a multimedia programming platform that allows for animations and interactivity in web pages.

ActionScript – is a scripting programming language that is used to create Adobe Flash programs.

Copy/Paste – is a function that we will be using to facilitate with the online banking aspect.

## Use Cases

### Practicing purchasing groceries

Scope: Simulate a store purchase

Actor: Special needs student

Goal: Teach the student how to properly make change and purchase items

Preconditions: None

Postconditions: A dialog box should explain to the student the outcome and, if need be, what went wrong. It informs the student of the remaining amount of money.

Typical Sequence of Events:

1. Select the store icon from the interface.
2. The store situation appears.
3. Student selects items and adds them to shopping cart.
4. A randomly generated sum within a range appears, representing the price of the groceries.
5. The student must, using a calculator in the interface, along with visual representations of the various denominations of money, make proper change for the groceries.
6. The student may repeat as necessary in order to procure the proper denominations to pay for the groceries.

### Taking the bus

Scope: Simulate a bus trip

Actor: Special needs student

Goal: Teach the student how to properly ride the bus

Preconditions: None

Postconditions: A dialog box should explain to the student the outcome and, if need be, what went wrong. It informs the student of the remaining amount of money.

Typical Sequence of Events:

6. Select the bus icon from the interface.
7. The bus situation appears.
8. The student is presented with a sum representing the cost of the trip.
9. The student must, using a calculator in the interface, along with visual representations of the various denominations of money, make proper change for the trip.
10. The student may repeat as necessary in order to procure the proper denominations to pay for the bus ticket.

ATM banking situation: withdrawal

Scope: Simulate an ATM withdrawal

Actor: Special needs student

Goal: Teach the student how to use an ATM

Preconditions: None

Postconditions: A dialog box should explain to the student the outcome of the bank withdrawal and, if need be, what went wrong. It informs the student of the remaining amount of money.

Typical Sequence of Events:

1. Select the ATM icon from the interface.
2. The ATM situation appears.
3. The student is presented with a typical ATM interface.
4. The student must enter a pin number to access their account.
5. The student is presented with a sum of money in the bank account.
6. The student is allowed to withdraw money, but not above the sum in their account.
7. The student enters the amount of money, represented by pictorial as well as numerical figures, to withdraw from the account.
8. The student's avatar receives the amount of money withdrawn.

### ATM banking situation: deposit

Scope: Simulate an ATM deposit

Actor: Special needs student

Goal: Teach the student how to use an ATM

Preconditions: None

Postconditions: A dialog box should explain to the student the outcome of the bank withdrawal and, if need be, what went wrong. It informs the student of the remaining amount of money.

Typical Sequence of Events:

1. Select the ATM icon from the interface.
2. The ATM situation appears.
3. The student is presented with a typical ATM interface.
4. The student must enter a pin number to access their account.
5. The student is presented with a sum of money in the bank account.
6. The student is allowed to deposit money, but not above the sum on their person.
7. The student enters the amount of money, represented by pictorial as well as numerical figures, to deposit to the account.
8. The student's avatar deposits the amount of money selected to the bank.

### Real world online banking assistance

Scope: To assist a student with their real world bank account

Actor: Special needs student

Goal: To help students manage finances in real life.

Preconditions: The student has a real world bank account and has access to the account.

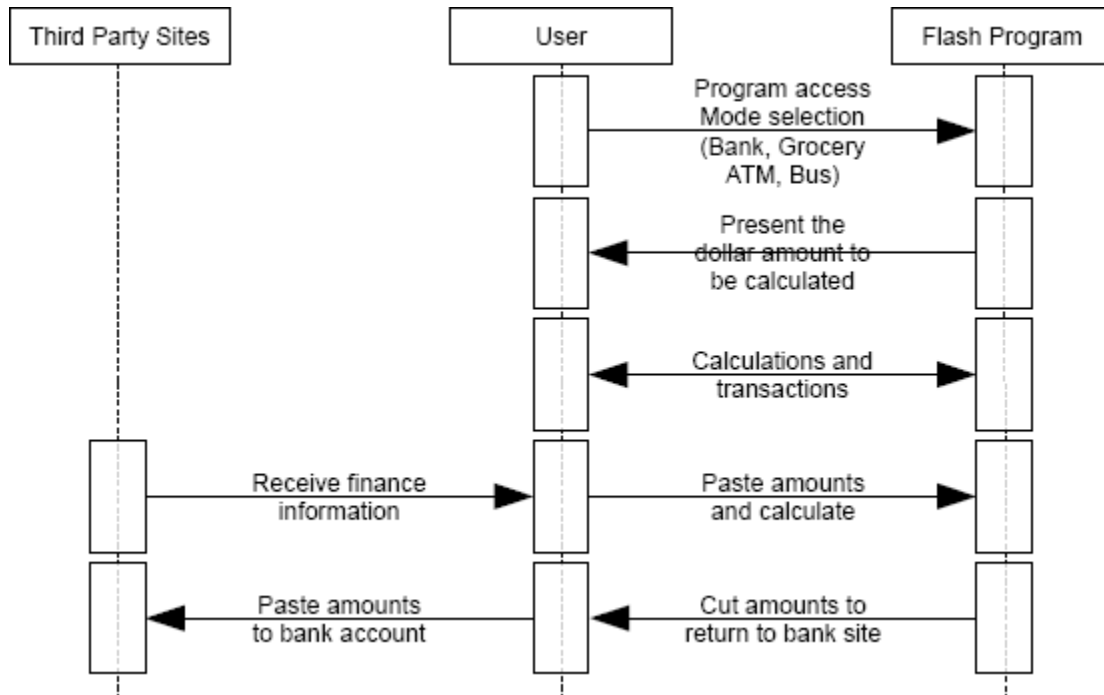
Postconditions: The student's finances are in order.

Typical Sequence of Events:

1. The student must be logged into both the game and the real world bank account

- online.
2. Select the Real bank account icon from the interface.
  3. The student is presented with a calculator.
  4. The student is allowed to enter the current real world balance.
  5. Using the calculator, the student can calculate what is owed for their real world bills.
  6. A cut and paste icon allows the student to copy the calculated numbers and transfer them to their bank account page.

## System Sequence Chart



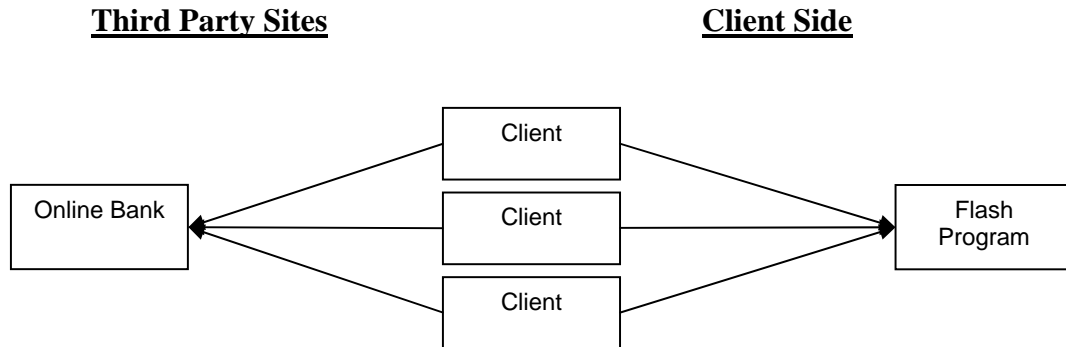
The system begins with the user accessing the program, wherein the user will be presented with their options as far as what they would like to practice or work on at that point. If one of the practice games is selected (the purchasing, traveling, or ATM situations) the user will be presented with that situation and randomized values, to practice for similar real world situations. If the user is simultaneously logged in to a banking program, they have the option of using the Flash program to help calculate their finances and manage their real-world bank account. The primary method of interaction between the two will be cut and paste features, as security on real world banking sites make it too difficult to interface with our program. The user's responsibilities on this level will be much higher than at the games level, as the program will have restricted access to the banking sites.

The program has been implemented primarily using a system of modules, so that

the code can be reused. To achieve this end, the program has several independent, cohesive modules that can interact with each other efficiently. As the different modes of the program may use certain modules, and ignore others, we will maintain low coupling between the modules so that extraneous features and functions are not carried over from different modes.

The modules have been created to fulfill the functions that are represented on the System Sequence Chart as arrows between the columns. The presentation of amounts to be calculated, the actual calculations to be made, the transaction through a graphical interface with dollar denomination representations, the cut and paste between programs, these features have been implemented independently of each other to ensure low coupling, but they have been implemented in a way that allows them to interact and pass data with ease, ensuring high cohesion.

## System Architecture



The system has been designed as a single, standalone program, without any need for Internet access. The program runs on any platform with the latest versions of Internet Explorer or Mozilla FireFox installed. All of the computation, processing, and other implementation concerns are done in ActionScript.

The students have access to several functions within the program to assist with third party sites, namely banking sites, to help manage their real life finances. Cut and paste features between the two programs will allow students to transfer dollar amounts back and forth between the two programs and the students will then be able to calculate transactions in a more familiar and user-friendly environment. After the calculations, the amounts can be moved back to the online banking program, to manage finances in the real world with their actual bank account.

The program overall consists of several situation that use similar components in their operation. Each situation, such as banking, grocery shopping, and bus riding, has a component where the user must calculate their total, and pay out using a graphical dollar denomination system. These tools, as they will be reused in all of the situations, are best implemented in a modular way. As the tools also may need to perform slightly different functions in each situation, ideally they should not be coupled together extraneously. The goal for the modules is high cohesion, so that the different modules work together

well and are able to share information in a logical and efficient manner, yet maintain low coupling so that when we are reusing them in different situations, unnecessary functions and attachments to other modules do not hinder performance.

The modular tools will need to interact smoothly, yet retain independence for individual reuse. For example, as most of the modes use the calculator functionality, the calculator has been modularized and is able to be accessed by the students in the different modes, without the need to recreate the calculator module for each mode. However, the bank mode may include such functions as an interest calculator, something that is unnecessary for the grocery shopping mode. This calculator functionality should be easily implemented due to the modularity of the system, and easily removed from modes where it is not needed.

The calculator module is a graphical front end, with large buttons and characters for easy viewing for the students. It stores the inputs of the user and displays the current calculation for the student to read and review. ActionScript will hold the values the student inputs and store them, along with any functions, such as division, addition, subtraction, multiplication, that the student uses, and perform the calculations when necessary.

Additional modules that will be implemented and reused for the various modes include a transportation module, a graphical denominational payout system, and others as deemed necessary. These modules must retain individual functionality, while being able to interact with each other smoothly and efficiently.

## Project Risks

### Overview

In the following section, the project's risks are analyzed in order for our team to be able to anticipate any possible obstacle during the design and implementation of our final product. The goal of this analysis is to identify all risks, to prioritize them according to their significance, and finally to plan a mitigation strategy to eliminate or reduce these risks. In order to assign a priority to each risk, our team decided to use a risk methodology which attaches an overall risk factor to each risk. This risk factor is calculated by multiplying the probability of a risk occurring by the impact of the risk's consequences.

Figure 6: Risk Identification and Prioritization

Risk # 1: Lack of Familiarity with Flash			
Description	Probability	Impact	Total Factor
Our team has made the design decision to implement our product using the Adobe Flash programming suite. This has been identified as a risk due to no team member having prior experience with this language.	.8	.9	.72
Risk # 2: Time Constraints			
Description	Probability	Impact	Total Factor
This risk has been identified since our project's success is tied to our team's ability to meet the deadlines stated above in the timeline.	.5	.7	.35
Risk # 3: Effective Collaboration			
Description	Probability	Impact	Total Factor
Our group acknowledges the variation in each member's personal schedule and its constraint on group work. Our project's efficiency and our end	.4	.6	.24

product will depend on group communication.			
Risk #4: Incomplete Requirements			
Description	Probability	Impact	Total Factor
The requirements gathered from the customer at the initial contact may not be correct or requirements maybe too vague to make design decisions on. Furthermore, sometimes customers do not know exactly what they want until they are able to test a prototype or initial operating system.	.6	.9	.54

**Probability:** This variable between 0 and, represents the likelihood that a risk will occur.

**Impact:** This factor, between 0 and 1, is the magnitude with which a risk could potentially harm the developmental and planning stages of the project.

**Risk Factor:** This represents the overall significance of the risk, with the higher risk factors receiving higher priority for planning the mediation of the corresponding risk.

#### Mediation Strategy

The first risk, “Lack of Familiarity with Flash,” is our largest potential risk. However, this risk happens to also have attributes that make it the easiest to mitigate. With an abundance of Flash information and tutorials available, each team member is able to read, practice, and ultimately gain experience with this language. This risk’s mitigation plan is to begin studying Flash during the design phase of this project, and to consult resources both on campus and via the Internet with problems during the implementation of our product. If technical knowledge shortfalls do impact the project, we will reduce the complexity of the implementation.

The second risk, “Time Constraints,” is a significant risk for a few reasons. This project has a total duration of thirty weeks, with roughly three months devoted to implementation. Each member faces time constraints from other courses and activities. Furthermore, our project contains deadlines which constrain the project further. These deadlines are our chief and most preferred mitigation strategy, which we believe will best

keep our project on track. However, if time constraints do become a significant obstacle, reducing the scope of our project will be our team's alternative. This is best done by prioritizing our project's requirements, so that lower priority requirements will not be implemented in order to devote time to the most important objectives. At four week intervals of our project, we have conducted an assessment of our progress and the further complexity of the project will be determined by the remaining time.

Our third risk, "Effective Collaboration," has been identified as a major, yet not significant risk. Our team has decided to schedule formal team meetings at regular intervals of our project to make any decisions regarding the design of our project. Furthermore, we have leveraged each member's wide accessibility to communication devices in order to discuss any issue a member needs help with or has questions on. Considering that some implementation work will be done individually by team members, we will notify each other of changes through both email and detailed commenting. In order to ensure that our software remains reliable throughout our project, version control and unit testing has been used whenever our software is modified.

The final risk, "Incomplete Requirements," has been identified as a major and significant risk. Given that our project is intended for use as an educational tool, our project must be usable enough so that users from a wide range of computing skills can effectively use our application. The application should contribute to the user's personal finance education, and pass rigorous acceptance tests. In order to ensure this, our team has scheduled frequent visits with the customer to receive feedback from the users who will be utilizing our application. These visits will occur anytime user interface changes have been made, new functionality introduced, or after any major implementation change. By keeping an active dialogue with our customers, we can be reasonably confident that we are meeting the requirements for this project, as well as have ample time to adjust if our project is not meeting certain requirements.

## **Test Plan**

This project has a time frame of approximately three months for the implementation phase. Considering the short timeframe for implementation, our group realized the importance of testing throughout the implementation process in order to ensure that our product remains reliable from its prototype stage until the finished product. Furthermore, scheduling conflicts may reduce the time in which our group will be able to actively develop this product together. Thus, our team has agreed to a thorough testing procedure which will consist of both white and black box testing.

### White Box

To conduct proper white box testing, we have written unit tests for each module of our software in order to be able to perform automated white box testing. The unit testing process consists of writing test methods which test certain assertions about our code, working against established test fixtures. After conducting unit tests after any implementation change, our group plans to also perform functional testing. In functional testing, we organize all related units into a functional unit, so that we can test the whole functional unit as one and thus determine if our individual units are aligned properly with one another. By automating this portion of our test plan, we can receive rapid feedback after any code has been altered to ensure that we have not regressed previously implemented portions of our code.

### Black Box

Black box testing is an integral component of our test plan. This testing involves intuition on the part of our team when testing, and we have examined the quality of our application by following the project's specifications. We have performed equivalence-class partitioning and boundary-value analysis. Furthermore, much of our black box testing efforts have been conducted in order to ensure that our application works well for both Mozilla FireFox and Internet Explorer. For cross browser issues, we will also enlist

the help of ActionScript debugging tools.

### Integration and System Testing

Due to the fact that a large portion of our implementation has been done individually by group members, integration and system testing was considered highly important whenever a new module was implemented into our system. For system testing, our plan was to test the entire system's functionality whenever a new module was merged into the system.

## **Societal Issues**

### Ethical

Our project has a myriad of ethical implications. We are attempting to help adults with disabilities to move through their daily lives, encountering the same situations we encounter, and to perform at a level at which we are able to perform. The ethical implications of our project included our successful and accurate representation of these situations for the students. We need to treat the encounters as realistic and something that we would be familiar with. These ethical implications meant that our project had to meet all of the needs of the students in the various ways they need to be catered to because of their disabilities, and the needs of the real world, exposing the students to situations they are likely to encounter in order to properly prepare them. We had to take the program as literally and directly as possible to ensure that the students would achieve the desired learning from using the tool.

### Social

Socially, our project was performing a service to the local community, as well as reaching out to adults with disabilities in a way that humanizes them. We show that we empathize with them and we are aware of their situation. Interacting with them and showing our support and willingness to provide assistance allows them to feel more comfortable in their situation and realize that people do want to help and better their situation. Santa Clara University instills in its students social responsibility, and we took that to heart in our project, focusing on the local community and those in most need. Our project was not suggested as a theoretical exercise or a product that could be sold and made profitable, our project was intended as outreaching, teaching, and compassionate. As a university with strong Catholic ties, and all the intellectual and financial assets behind it, we must extend our reach and influence to help those in need and those local to us.

### Political

The political implications and consequences of our project reflect our social concerns. We are enabling adults with disabilities and by our interaction; we are bringing them to the forefront of people's attention. The project focused on a group of people that, unfortunately, is frequently marginalized or forgotten when it comes time to assess their status in the political arena. In helping out these adults with disabilities, we hoped to bring more attention to their cause and to assist them and increase their political stock.

### Economic

The development costs of our product were paid for by Santa Clara University, in the form of providing us copies of Adobe Flash. The product is intended to be free to use for the adults with disabilities, thus it provides no revenue for the University. In this situation, we must consider the other implications and benefits of our project, to justify the economic costs. As the school is able to reach out and help the local community and people in need, fulfilling the Jesuit and Catholic charge for helping those in need, the project ideally pays for itself in return. Our personal stake in this then turned to ensuring that we provided a quality project and a functional solution for the school, to justify the expenses the school incurred to make this project happen.

### Health and Safety

Our project had health and safety issues associated with the disabilities the students suffer from. The entire project was supervised by those familiar with the students' disabilities and what kinds of interaction and presentation the students are comfortable with and can experience safely. For example, due to some students' visual impairments, we needed to ensure that the project was easily viewed and interacted with. Forcing students with fine motor skill disabilities to perform very precise mouse action would be detrimental to both their motor skills and could be harmful psychologically. The project focused on these and other health and safety issues, as the students' needs had to be tended to first, while ensuring the project delivered the desired functionality.

### Manufacturability

Manufacturability constituted a small factor in our project, as the project was software based, and thus does not have much of a manufacturing component. However, manufacturing in one sense was important to our project, because it was intended that the students be able to work with the financial tool from home on their own time. As the students' homes and financial situations differed, we could not assume that all students would even have access to all potential facilities to use the tool. We took this into consideration and we decided early on in the design process to develop on the Flash platform, as most computers are bundled with Flash-enabled browsers, so most students with access to computers would be able to use the program on their own time.

### Sustainability

Sustainability factored into our design process at several points. The project was intended to be a stand-alone application, with minimal and ideally no maintenance requirements from the people involved in the project or Santa Clara University. In addition to the economic and manufacturing concerns, the sustainability concern cemented our decision to make the project Flash-based, and stand alone. Hosting this project on a website would allow for wider accessibility, but would also entail maintaining the website and hosting fees, which were not planned for nor in the budget for the project. Making the project a Flash-based deliverable allowed us to circumvent these issues. However, using Flash brought up another issue with sustainability, and that is the lack of functionality to reverse-engineer the project from the .fla file, without the original Flash file. If there are issues encountered with the project in months to come, after heavy use by the students, correcting these issues becomes more and more difficult. Two ideas to solve this issue came up, one of which was passing the project on to another senior design group in the next few years, to allow them to update and maintain the project, while expanding its usefulness. The second idea involved providing the source material to Mr. Brian Darby, to allow him to explore his own options in expanding and

maintaining the project on his own.

### Environmental Impact

Environmentally, our project did not have much of an impact. Aside from the environmental issues of using computers, which involves the power consumption as well as the materials used to make the components of a computer, the project did not expand on the environmental impact of the currently existing technology.

### Usability

The most important aspect of the project for our group was the ability of the students to use the project and to benefit from it. We engineered the project from the beginning with the application's usability as the main focus. The wide variety of disabilities suffered by the students and the wide array of backgrounds and options for using the program available to the students forced us to consider every aspect of our design, from interface, to deliverables, to the platform we developed on. Immediately in our first test group with the students, we discovered that we had underestimated the students' visual and interface needs. We designed the buttons and graphics too small for the students to read and interact with clearly. Our first set of changes after meeting with the students was largely usability changes, increasing font and graphic size, changing colors to make the slides easier on the eyes, and making the instructions clearer for the students to read and comprehend. With this as our main focus, and one of the key issues in designing the product, we were able to quickly adapt to these needs and ensure that our project was useful to the students.

### Lifelong learning

The project provided a learning experience both for us as designers and for the students with whom we were working. The students were provided with a product designed to familiarize them with financial situations and currency dealings, and allow them to work and perform in an increasingly complicated world. On our half, we learned

the basics of designing a project from start to finish, and it allowed us to experience an aspect of the computer industry that is neglected in our studies and our focuses after school, and that is socially- and education-oriented software. These experiences contributed to our learning both in school and for years to come.

### Compassion

Defining compassion as awareness of suffering and a desire to alleviate it, the project encompassed this goal in its entirety. When we were informed of the situation at the school, and the locality and relevance to both the school and our personal experiences, we decided that action on our part was for the best for everyone involved. The students have a glaring need, and are in a state of suffering, from both physical and mental disorders, as well as negligence both on the side of the software industry and the community at large. These students are living their lives with a quality of life in some way unequal to the quality of life we all experience as students at Santa Clara University, gifted with our social situation and our health. To reach out to these students and to ensure that they are aware that people are looking out for them and actively trying to help them was important to us as engineers and human beings. The students, on our frequent visits, expressed nothing but graciousness and pleasure at the work we were doing to help them learn and live better. They might have gotten confused and hindered by the tool at points, but they had nothing but good things to say both for us and the project, and they were more than happy to explain what their issues were, and grateful for our work to help make the project more usable and accessible to them. The project was focused entirely on the students and their needs, and we hope we achieved all our goals and met all their needs. The students would have us know that they are more than pleased with even the most basic of deliverables we provided, which only makes the compassion aspect of our project that much more poignant, because these students are so marginalized that even the simplest extension of our helping hand pleased them more than they could express.

## Conclusion

Our project sought to provide a tool for adults with disabilities, to teach them about money and purchases, and the value of various currency denominations. The project entailed frequent customer interaction with the students, both to ensure the project was bug-free, and to ensure that the project was providing the students with the tools they needed to learn about money and become more adept at interacting with people. The target audience for our project was a frequently forgotten, marginalized group, so the project had additional motives and pressures, those being working with and assisting those in need in the local community.

Throughout the project, we learned the value of customer interaction, as our initial vision for the project was radically different from how the project ending up appearing. We underestimated and overestimated the students' needs at various junctures, and through our interaction with them, we were able to correct these issues and provide the students with a product that was acceptable and useful to them.

We learned about the development process along with the one aspect missing from lab and homework exercises, the customer interaction. We frequently met with Mr. Brian Darby and went over our current ideas and plans, to ensure that they fell in line with his ideas and his vision for the project. The give and take involved in the customer interaction was a large portion of the learning process for us. We were used to being given a task and achieving it however we felt comfortable, but with a customer, frequently what the designer is comfortable with and envisions is not quite what the customer wants.

We also learned about the local community and the needs of these adults with disabilities. We saw their situations concerning both their personal health issues and their living and learning situation, the school rooms they worked in, and the computers they used. The state of the art technologies we take for granted are miles out of reach for some of these students and even some schools. We took this into account and factored it into our design process, to help reach a wider audience with our product.

Our project turned out fairly simplistically technology-wise, but we feel that this was a strength of our project, as it allowed greater flexibility in distribution and easier maintenance and updating. Our project hit our target for providing an interface wherein the students were presented with a realistic, plausible situation, and had to use the tools at their disposal to resolve the situation. We feel that we had some shortfalls in our interactivity, as we wished we had more experience designing interactive Flash to make the tools more engaging and exciting for the students. However, considering this downfall, and in light of the fact that many of our ideas for further interaction and flair to the project would have taken away from the core utility and been distracting to the students - potentially to the point of negating its usefulness - we consider this shortfall not extremely detrimental to the project as a whole.

We would like to pass this project on to future senior design groups. The continuation of this project would serve a dual purpose. The minimal service would be to maintain the project in its current incarnation, resolving errors and bugs encountered down the road. The second goal would be to expand on the project's usefulness for the students, potentially adding features and other situations the students might find themselves in.

The financial tool we provided to the students accomplished all of our initial design goals, providing realistic situations for the students to resolve and work through, all while learning more about the value of currency. The project had to be easily accessible and distributable, and we met this goal in our implementation using Adobe Flash and delivering a Flash Movie file that allowed the students with computers to play with the program at home on their own. We feel that we hit all our marks and provided a product that both we as designers and Mr. Brian Darby as the customer are happy with, and look forward to seeing where the project can go in the future.

## References

1. "ActionScript." *Adobe Flash Developer Center*. 2009. Adobe Systems Incorporated. 5 Jun 2009 <<http://www.adobe.com/devnet/flash/actionscript.html>>.

## Appendix A

Source Code:

```
/*
 * File: Finanical_Tool.fla
 * -----
 * This program is an education tool for finances designed to
 * assist students with the understanding of monetary value
 *
 * Author: Marco Echandi, Jason Goetsch, James Lewis
 * Date: June 5 2009
 *
 *
 * Frame 1: Contains all the global functions, variables and
 * holds the home page of the flash program.
 * Responsible for new task function and direction
 * to the bus.
 */

/*
 * Flash libraries.
 */

import flash.events.*;
import fl.data.DataProvider;
import fl.controls.DataGrid;
import fl.controls.ScrollPolicy;
import fl.events.ListEvent;
import flash.display.BitmapData;
import flash.net.URLRequest;
import flash.display.Loader;

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Button to function assignment dependent on event.
 */
```



```

function swapSceneBus(e:MouseEvent):void{
    gotoAndStop(3);
}

function swapSceneGrocery(e:MouseEvent):void{
    gotoAndStop(4);
}

function swapSceneExit(e:MouseEvent):void{
    gotoAndStop(1);
}

function swapScenePayments(e:MouseEvent):void{
    gotoAndStop(5);
}

/*
 * Function used to round the floating point number into
 * something readable.
 */

function rounding(num, precision){
    var splitChar = ".";

    if((precision = Math.abs(precision)) == 0) return
Math.round(num);
    if(splitChar == null) splitChar = ".";
    if(num == 0) return "0.00";

    num = parseFloat((Math.floor(num + 0.01) + splitChar +
Math.round(num*Math.pow(10,precision)).toString().substr(-precision)));
    return (Math.floor(num + 0.01) + splitChar +
Math.round(num*Math.pow(10,precision)).toString().substr(-precision));
}

/*
 * Function used to assign a task randomly, this is to
 * ensure the incorporates all the items in the store
 * as well as keep the program interesting.
 */

```

```

function assignRandomTask(e:MouseEvent):void{

    var numItemsToGet:uint = 3;
    var amountToGet:uint = 4;
    var i:uint = 0;
    var amt:uint = 0;
    var item:uint = 0;
    var previous1 = -1;
    var previous2 = -1;

    toGetArray = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0];

    dailyTaskText.text = "You need to purchase:\n";

    while(i < numItemsToGet){

        amt = Math.ceil(Math.random()*amountToGet);
        item = Math.floor(Math.random() * groceryItems.length);

        while(item == previous1 || item == previous2){
            item = Math.floor(Math.random() *
groceryItems.length);
        }

        if (previous1 == -1){

            previous1 = item;

        }
        if (previous2 == -1){

            previous2 = item;

        }
        dailyTaskText.text += " " + amt + " x " +
groceryItems[item] + "\n";
        toGetArray[item] -= amt;
        toGetArrayTemp[item] -= amt;
        trace(" " + toGetArray[item]);
        i++;

    }

    dailyTaskText.text += " from the grocery store.";
    dailyTask = dailyTaskText.text;
    trace(dailyTask);

}

```

```
/*  
 * This is assigning the style of the text and buttons.  
 */  
  
var tf:TextFormat = new TextFormat("_sans", 28);  
var labeltf:TextFormat = new TextFormat("_sans", 16);  
var atmtf:TextFormat = new TextFormat("_sans", 36);  
  
assignTaskButton.setStyle("textFormat", tf);  
dailyTaskText.setStyle("textFormat", tf);  
exitButton.setStyle("textFormat", tf);  
busStopButton.setStyle("textFormat", tf);
```

```

/*
 * Frame 2: This frame is the bank page which holds the
 *          functionality of the ATM machine. It accepts
 *          any pin (for privacy reasons) and you can
 *          withdrawl an amount betwenn 1 to 999.
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Variable assignment.
 */

var inputType = 0;
atmText.text = "Please input PIN";
var pinText = "";
var withdrawlAmt = "";

/*
 * Button to function assignment dependent on event.
 */

busStopButton.addEventListener(MouseEvent.CLICK, swapSceneBus);
exitButton.addEventListener(MouseEvent.CLICK, swapSceneExit);
oneButton.addEventListener(MouseEvent.CLICK, atmInputOne);
twoButton.addEventListener(MouseEvent.CLICK, atmInputTwo);
threeButton.addEventListener(MouseEvent.CLICK, atmInputThree);
fourButton.addEventListener(MouseEvent.CLICK, atmInputFour);
fiveButton.addEventListener(MouseEvent.CLICK, atmInputFive);
sixButton.addEventListener(MouseEvent.CLICK, atmInputSix);
sevenButton.addEventListener(MouseEvent.CLICK, atmInputSeven);
eightButton.addEventListener(MouseEvent.CLICK, atmInputEight);
nineButton.addEventListener(MouseEvent.CLICK, atmInputNine);
zeroButton.addEventListener(MouseEvent.CLICK, atmInputZero);
noButton.addEventListener(MouseEvent.CLICK, atmInputNo);
yesButton.addEventListener(MouseEvent.CLICK, atmInputYes);

/*
 * This is assigning the style of the text and buttons.
 */

```

```

exitButton.setStyle("textFormat", tf);
busStopButton.setStyle("textFormat", tf);
atmText.setStyle("textFormat", atmtf);
oneButton.setStyle("textFormat", atmtf);
twoButton.setStyle("textFormat", atmtf);
threeButton.setStyle("textFormat", atmtf);
fourButton.setStyle("textFormat", atmtf);
fiveButton.setStyle("textFormat", atmtf);
sixButton.setStyle("textFormat", atmtf);
sevenButton.setStyle("textFormat", atmtf);
eightButton.setStyle("textFormat", atmtf);
nineButton.setStyle("textFormat", atmtf);
zeroButton.setStyle("textFormat", atmtf);
noButton.setStyle("textFormat", atmtf);
yesButton.setStyle("textFormat", atmtf);

/*
 * Function checks the input from the pin pad and provides
 * its functionality to withdraw money
 */

function checkInput(num){
    switch(inputType){
        case 0:
            if(num == "yes"){
                return;
            }
            else if(num == "no"){
                pinText = "";
                atmText.text = "PIN: ";
            }
            else if(pinText.length >= 3){
                inputType = 1;
                atmText.text = "Withdrawl:"
            }
            else{
                pinText += num;
                atmText.text = "PIN: " + pinText;
            }
            break;
        case 1:
            if(num == "0" && withdrawlAmt.length == 0){

```

```

        return;
    }
    else if(num == "no"){
        withdrawlAmt = "";
        atmText.text = "Withdrawl amount:";
        return;
    }
    else if(num == "yes"){
        addBillsToWallet();
        atmText.text = "Thank you";
        withdrawlAmt = "";
        pinText = "";
        inputType = 0;
        return;
    }
    else if(withdrawlAmt.length < 3){
        withdrawlAmt += num;
        atmText.text = "$" + withdrawlAmt + ".00"
    }
    break;
}
return;
}

/*
 * Function that processes the input from checkInput function into
 * the appropriate bills for the payment page representation.
 */

function addBillsToWallet(){
    var total:uint = withdrawlAmt;

    walletTwenty += total / 20;
    total = total % 20;
    walletTen += total / 10;
    total = total % 10;
    walletFive += total / 5;
    total = total % 5;
    walletOne += total;
}

```

```
}

/*
 * Functions that respond to the users input into the pin pad of the
 * ATM.
 */

function atmInputOne(e:MouseEvent):void{
    checkInput("1");
}

function atmInputTwo(e:MouseEvent):void{
    checkInput("2");
}

function atmInputThree(e:MouseEvent):void{
    checkInput("3");
}

function atmInputFour(e:MouseEvent):void{
    checkInput("4");
}

function atmInputFive(e:MouseEvent):void{
    checkInput("5");
}

function atmInputSix(e:MouseEvent):void{
    checkInput("6");
}

function atmInputSeven(e:MouseEvent):void{
    checkInput("7");
}

function atmInputEight(e:MouseEvent):void{
```

```
        checkInput("8");
    }
    function atmInputNine(e:MouseEvent):void{
        checkInput("9");
    }
    function atmInputZero(e:MouseEvent):void{
        checkInput("0");
    }
    function atmInputNo(e:MouseEvent):void{
        checkInput("no");
    }
    function atmInputYes(e:MouseEvent):void{
        checkInput("yes");
    }
}
```

```

/*
 * Frame 3: This frame is the bus page which allows the
 *         user to move to either the bank or the store,
 *         but first by going through the payment page to
 *         pay the fare or go home if you need a new task.
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Variable assignment.
 */

total = 2.25;

/*
 * Button to function assignment dependent on event.
 */

homeButton.addEventListener(MouseEvent.CLICK, swapSceneHome);
exitButton.addEventListener(MouseEvent.CLICK, swapSceneExit);
fareButton.addEventListener(MouseEvent.CLICK, farecheck);

/*
 * This is assigning the style of the text and buttons.
 */

exitButton.setStyle("textFormat", tf);
homeButton.setStyle("textFormat", tf);
fareButton.setStyle("textFormat", tf);

/*
 * Function that checks whether the fare needs to paid to move to
 * right page.
 */

function farecheck(e:MouseEvent):void{

```

```
    bus = 0;  
    gotoAndStop(5);  
}
```

```

/*
 * Frame 4: This frame is the store page which allows
 *         the user to complete the daily task and/or
 *         buy other items that exist in the store.
 *
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Button to function assignment dependent on event.
 */

exitButton.addEventListener(MouseEvent.CLICK, swapSceneExit);
busStopButton.addEventListener(MouseEvent.CLICK, swapSceneBus);
checkoutButton.addEventListener(MouseEvent.CLICK, swapScenePayments);
milkButton.addEventListener(MouseEvent.CLICK, addMilk);
tomatoButton.addEventListener(MouseEvent.CLICK, addTomato);
carrotButton.addEventListener(MouseEvent.CLICK, addCarrot);
breadButton.addEventListener(MouseEvent.CLICK, addBread);
honeyButton.addEventListener(MouseEvent.CLICK, addHoney);
juiceButton.addEventListener(MouseEvent.CLICK, addJuice);
appleButton.addEventListener(MouseEvent.CLICK, addApple);
cheeseButton.addEventListener(MouseEvent.CLICK, addCheese);
meatButton.addEventListener(MouseEvent.CLICK, addMeat);
lettuceButton.addEventListener(MouseEvent.CLICK, addLettuce);
eggButton.addEventListener(MouseEvent.CLICK, addEgg);
potatoButton.addEventListener(MouseEvent.CLICK, addPotato);
bananaButton.addEventListener(MouseEvent.CLICK, addBanana);
relishButton.addEventListener(MouseEvent.CLICK, addRelish);
ketchupButton.addEventListener(MouseEvent.CLICK, addKetchup);
mustardButton.addEventListener(MouseEvent.CLICK, addMustard);
hotdogbunButton.addEventListener(MouseEvent.CLICK, addHotdogbun);
hamburgerbunButton.addEventListener(MouseEvent.CLICK, addHamburgerbun);
pancakeButton.addEventListener(MouseEvent.CLICK, addPancake);
sodaButton.addEventListener(MouseEvent.CLICK, addSoda);
waterButton.addEventListener(MouseEvent.CLICK, addWater);
lemonadeButton.addEventListener(MouseEvent.CLICK, addLemonade);
macaroniButton.addEventListener(MouseEvent.CLICK, addMacaroni);
gumButton.addEventListener(MouseEvent.CLICK, addGum);
candyButton.addEventListener(MouseEvent.CLICK, addCandy);
removeList.addEventListener(MouseEvent.CLICK, removeItemFromList);
tab1.addEventListener(MouseEvent.CLICK, toggleAisle1);
tab2.addEventListener(MouseEvent.CLICK, toggleAisle2);
tab3.addEventListener(MouseEvent.CLICK, toggleAisle3);

```

```

/*
 * Variable assignment.
 */

var cartDp:DataProvider = new DataProvider();
total = 0;
var prices:Array = [ 0.25, 0.75, 0.25, 0.75, 1.00, 0.75, 2.75, 2.75,
2.50, 0.75, 2.00, 2.00, 0.75, 1.25, 1.25, 0.75, 0.25, 2.25, 0.50, 1.75,
1.00, 3.50, 5.00, 1.25, 2.50];
dailyTaskGroceryText.text = dailyTask;
myCart.dataProvider = cartDp;

/*
 * This is assigning the style of the text and buttons.
 */

totalTitle.setStyle("textFormat", tf);
removeList.setStyle("textFormat", tf);
checkoutButton.setStyle("textFormat", tf);
exitButton.setStyle("textFormat", tf);
busStopButton.setStyle("textFormat", tf);
cartTitle.setStyle("textFormat", tf);
dailyTaskGroceryText.setStyle("textFormat", labeltf);
var cartFormat:TextFormat = new TextFormat();
cartFormat.size = 20;
cartFormat.color = 0xCC3033;
myCart.setRendererStyle("textFormat", cartFormat);
dailyTaskGroceryText.setStyle("textFormat", cartFormat);

/*
 * Function that empties the cart and reset the amount of groceries
 * to buy.
 */

function refreshCart(){

    var i;
    for(i = 0; i < toGetArray.length; i++){

        toGetArrayTemp[i] = toGetArray[i];

    }

}

```

```
/*  
 * Functions that toggles the clicked on aisle to display the  
 * items, allowing the user to access more items in a organized  
 * fashion inside the store.  
 */
```

```
function toggleAisle1(e:MouseEvent) :void{  
  
    carrotButton.visible = false;  
    lettuceButton.visible = false;  
    tomatoButton.visible = false;  
    potatoButton.visible = false;  
    bananaButton.visible = false;  
    appleButton.visible = false;  
    relishButton.visible = false;  
    ketchupButton.visible = false;  
    mustardButton.visible = false;  
    lemonadeButton.visible = false;  
    pancakeButton.visible = false;  
    honeyButton.visible = false;  
    gumButton.visible = false;  
    juiceButton.visible = false;  
    waterButton.visible = false;  
    sodaButton.visible = false;  
    candyButton.visible = false;  
    macaroniButton.visible = false;  
    breadButton.visible = true;  
    hamburgerbunButton.visible = true;  
    hotdogbunButton.visible = true;  
    cheeseButton.visible = true;  
    meatButton.visible = true;  
    milkButton.visible = true;  
    eggButton.visible = true;  
    labelCarrot.text = "";  
    labelHoney.text = "Hamburger Bun: $1.00";  
    labelTomato.text = "Hotdog Bun: $1.00";  
    labelJuice.text = "";  
    labelBread.text = "Bread: $0.50";  
    labelApple.text = "";  
    labelSteak.text = "Steak: $5.00";  
    labelCheese.text = "Cheese: $3.50";  
    labelMilk.text = "Milk: $1.25";  
    labelEgg.text = "Egg: $2.50";  
  
}
```

```
function toggleAisle2(e:MouseEvent) :void{  
  
    carrotButton.visible = true;  
    lettuceButton.visible = true;  
    tomatoButton.visible = true;  
    potatoButton.visible = true;  
    bananaButton.visible = true;
```

```

appleButton.visible = true;
relishButton.visible = true;
ketchupButton.visible = true;
mustardButton.visible = true;
pancakeButton.visible = false;
honeyButton.visible = false;
gumButton.visible = false;
juiceButton.visible = false;
waterButton.visible = false;
lemonadeButton.visible = false;
sodaButton.visible = false;
candyButton.visible = false;
macaroniButton.visible = false;
breadButton.visible = false;
hamburgerbunButton.visible = false;
hotdogbunButton.visible = false;
cheeseButton.visible = false;
meatButton.visible = false;
milkButton.visible = false;
eggButton.visible = false;
labelCarrot.text = "Carrot: $0.25";
labelHoney.text = "Lettuce: $0.75";
labelTomato.text = "Tomato: $0.25";
labelJuice.text = "Potato: $0.75";
labelBread.text = "";
labelApple.text = "Apple: $0.75";
labelSteak.text = "Relish: $2.75";
labelCheese.text = "Banana: $1.00";
labelMilk.text = "Ketchup: $2.75";
labelEgg.text = "Mustard: $2.50";
}

function toggleAisle3(e:MouseEvent) :void{

```

```

    carrotButton.visible = false;
    lettuceButton.visible = false;
    tomatoButton.visible = false;
    potatoButton.visible = false;
    bananaButton.visible = false;
    appleButton.visible = false;
    relishButton.visible = false;
    ketchupButton.visible = false;
    mustardButton.visible = false;
    lemonadeButton.visible = true;
    pancakeButton.visible = true;
    honeyButton.visible = true;
    gumButton.visible = true;
    juiceButton.visible = true;
    waterButton.visible = true;
    sodaButton.visible = true;
    candyButton.visible = true;
    macaroniButton.visible = true;
    breadButton.visible = false;

```

```

    hamburgerbunButton.visible = false;
    hotdogbunButton.visible = false;
    cheeseButton.visible = false;
    meatButton.visible = false;
    milkButton.visible = false;
    eggButton.visible = false;
    labelCarrot.text = "Pancake = $2.00";
    labelHoney.text = "Honey: $2.00";
    labelTomato.text = "Gum: $0.75";
    labelJuice.text = "Juice: $1.25";
    labelBread.text = "Lemonade: $0.75";
    labelApple.text = "";
    labelSteak.text = "Soda: $0.75";
    labelCheese.text = "Water: $1.25";
    labelMilk.text = "Candy: $0.25";
    labelEgg.text = "Macaroni: $2.25";

}

/*
 * Functions to add each item within the store to the shopping cart.
 * Also adds the appropriate amount for item and handles quantity.
 */

function addCarrot(e:MouseEvent):void{

    cartDp.addItem( { label: "carrot", data:0 } );
    toGetArrayTemp[0]++;
    total += prices[0];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addLettuce(e:MouseEvent):void{

    cartDp.addItem( { label: "lettuce", data:1 } );
    toGetArrayTemp[1]++;
    total += prices[1];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addTomato(e:MouseEvent):void{

    cartDp.addItem( { label: "tomato", data:2 } );
    toGetArrayTemp[2]++;
    total += prices[2];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

```

```

function addPotato(e:MouseEvent):void{

    cartDp.addItem( { label: "potato", data:3 } );
    toGetArrayTemp[3]++;
    total += prices[3];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addBanana(e:MouseEvent):void{

    cartDp.addItem( { label: "banana", data:4 } );
    toGetArrayTemp[4]++;
    total += prices[4];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addApple(e:MouseEvent):void{

    cartDp.addItem( { label: "apple", data:5 } );
    toGetArrayTemp[5]++;
    total += prices[5];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addRelish(e:MouseEvent):void{

    cartDp.addItem( { label: "relish", data:6 } );
    toGetArrayTemp[6]++;
    total += prices[6];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addKetchup(e:MouseEvent):void{

    cartDp.addItem( { label: "ketchup", data:7 } );
    toGetArrayTemp[7]++;
    total += prices[7];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addMustard(e:MouseEvent):void{

    cartDp.addItem( { label: "mustard", data:8 } );
    toGetArrayTemp[8]++;
    total += prices[8];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

```

```

function addLemonade(e:MouseEvent):void{

    cartDp.addItem( { label: "lemonade", data:9 } );
    toGetArrayTemp[9]++;
    total += prices[9];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addPancake(e:MouseEvent):void{

    cartDp.addItem( { label: "pancake mix", data:10 } );
    toGetArrayTemp[10]++;
    total += prices[10];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addHoney(e:MouseEvent):void{

    cartDp.addItem( { label: "honey", data:11 } );
    toGetArrayTemp[11]++;
    total += prices[11];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addGum(e:MouseEvent):void{

    cartDp.addItem( { label: "gum", data:12 } );
    toGetArrayTemp[12]++;
    total += prices[12];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addJuice(e:MouseEvent):void{

    cartDp.addItem( { label: "juice", data:13 } );
    toGetArrayTemp[13]++;
    total += prices[13];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

function addWater(e:MouseEvent):void{

    cartDp.addItem( { label: "water", data:14 } );
    toGetArrayTemp[14]++;
    total += prices[14];
    totalTitle.text = "Total is $" + rounding(total, 2);
}

```

```

function addSoda(e:MouseEvent):void{

    cartDp.addItem( { label: "soda", data:15 } );
    toGetArrayTemp[15]++;
    total += prices[15];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addCandy(e:MouseEvent):void{

    cartDp.addItem( { label: "candy", data:16 } );
    toGetArrayTemp[16]++;
    total += prices[16];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addMacaroni(e:MouseEvent):void{

    cartDp.addItem( { label: "macaroni", data:17 } );
    toGetArrayTemp[17]++;
    total += prices[17];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addBread(e:MouseEvent):void{

    cartDp.addItem( { label: "bread", data:18 } );
    toGetArrayTemp[18]++;
    total += prices[18];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addHamburgerbun(e:MouseEvent):void{

    cartDp.addItem( { label: "hamburger bun", data:19 } );
    toGetArrayTemp[19]++;
    total += prices[19];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addHotdogbun(e:MouseEvent):void{

    cartDp.addItem( { label: "hotdog bun", data:20 } );
    toGetArrayTemp[20]++;
    total += prices[20];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

```

```

function addCheese(e:MouseEvent):void{

    cartDp.addItem( { label: "cheese", data:21 } );
    toGetArrayTemp[21]++;
    total += prices[21];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addMeat(e:MouseEvent):void{

    cartDp.addItem( { label: "steak", data:22 } );
    toGetArrayTemp[22]++;
    total += prices[22];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addMilk(e:MouseEvent):void{

    cartDp.addItem( { label: "milk", data:23 } );
    toGetArrayTemp[23]++;
    total += prices[23];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

function addEgg(e:MouseEvent):void{

    cartDp.addItem( { label: "egg", data:24 } );
    toGetArrayTemp[24]++;
    total += prices[24];
    totalTitle.text = "Total is $" + rounding(total, 2);

}

/*
 * Function that allows the user to remove a selected item
 * from the shopping cart.
 */

function removeItemFromList(e:MouseEvent):void{

    if (myCart.length == 0)
    {
        return;
    }
    if (myCart.selectedItem == null)
    {
        return;
    }
    total -= prices[myCart.selectedItem.data];
}

```

```

        toGetArrayTemp[myCart.selectedItem.data]--;
        cartDp.removeItem(myCart.selectedItem);
        totalTitle.text = "Total is $" + rounding(total, 2);
    }

    /*
     * Assigning text to the corresponding display
     */

    labelCarrot.text = "";
    labelHoney.text = "Hamburger Bun: $1.00";
    labelTomato.text = "Hotdog Bun: $1.00";
    labelJuice.text = "";
    labelBread.text = "Bread: $0.50";
    labelApple.text = "";
    labelSteak.text = "Steak: $5.00";
    labelCheese.text = "Cheese: $3.50";
    labelMilk.text = "Milk: $1.25";
    labelEgg.text = "Egg: $2.50";

    /*
     * Assigning the visibility of each item when entering the store.
     */

    breadButton.visible = true;
    hamburgerbunButton.visible = true;
    hotdogbunButton.visible = true;
    cheeseButton.visible = true;
    meatButton.visible = true;
    milkButton.visible = true;
    eggButton.visible = true;
    carrotButton.visible = false;
    lettuceButton.visible = false;
    tomatoButton.visible = false;
    potatoButton.visible = false;
    bananaButton.visible = false;
    appleButton.visible = false;
    relishButton.visible = false;
    ketchupButton.visible = false;
    mustardButton.visible = false;
    lemonadeButton.visible = false;
    pancakeButton.visible = false;
    honeyButton.visible = false;
    gumButton.visible = false;
    juiceButton.visible = false;
    waterButton.visible = false;
    lemonadeButton.visible = false;
    sodaButton.visible = false;
    candyButton.visible = false;

```

```
macaroniButton.visible = false;  
refreshCart();
```

```

/*
 * Frame 5: This frame is the payment page that is
 *         designed to show how to pay for items or
 *         the fare. There functionality to show the
 *         wallet total, amount of bills you have and
 *         amount owed. Also the fare doesnt give
 *         change back as in a real life situation.
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Variable assignment.
 */

walletChange = 0;
var savedwalletOne:Number = walletOne;
var savedwalletFive:Number = walletFive;
var savedwalletTen:Number = walletTen;
var savedwalletTwenty:Number = walletTwenty;
var savedwalletTotal:Number = walletTotal;
var tempTotal = total;

OneDollarText.text="Total = "+walletOne;
FiveDollarText.text="Total = "+walletFive;
TenDollarText.text="Total = "+walletTen;
TwentyDollarText.text="Total = "+walletTwenty;
walletTotal =
walletOne+(walletFive*5)+(walletTen*10)+(walletTwenty*20);
sumTotal.text= "Wallet Total = $" +rounding(walletTotal,2);
amountOwed.text = "You owe $" + rounding(total, 2);

/*
 * This is assigning the style of the text and buttons.
 */

payButton.setStyle("textFormat", tf);
clearButton.setStyle("textFormat", tf);
storeButton.setStyle("textFormat", tf);
statusValue.setStyle("textFormat", tf);
var payFormat:TextFormat = new TextFormat("_sans", 20);
payFormat.color = 0xFFFFFFFF;

```

```

OneDollarText.setStyle("textFormat", payFormat);
FiveDollarText.setStyle("textFormat", payFormat);
TenDollarText.setStyle("textFormat", payFormat);
TwentyDollarText.setStyle("textFormat", payFormat);
sumTotal.setStyle("textFormat", payFormat);
amountOwed.setStyle("textFormat", payFormat);
statusValue.setStyle("textFormat", payFormat);

/*
 * Button to function assignment dependent on event.
 */

oneDollarButton.addEventListener(MouseEvent.CLICK, incrementOneDollar);
fiveDollarButton.addEventListener(MouseEvent.CLICK,
incrementFiveDollar);
tenDollarButton.addEventListener(MouseEvent.CLICK, incrementTenDollar);
twentyDollarButton.addEventListener(MouseEvent.CLICK,
incrementTwentyDollar);
payButton.addEventListener(MouseEvent.CLICK, checkPayments);
clearButton.addEventListener(MouseEvent.CLICK, clearPayments);
storeButton.addEventListener(MouseEvent.CLICK, clearComplete);
storeButton.addEventListener(MouseEvent.CLICK, checkWhere);

/*
 * Functions to increment the dollar bill amounts (One, Five, Ten and
Twenty)
 * and wallet total. Handles the payment process while count the bills
used.
 */

function incrementOneDollar(e:MouseEvent):void{

    if ( total <= 0 ) {

        return;

    }

    walletOne = walletOne - 1;
    if(walletOne < 0 || walletOne > 1000){

        walletOne = 0;

    }else{

        OneDollarText.text="Total = "+walletOne;
        walletTotal=walletTotal-1;
        total = total - 1;
        walletChange = total;
    }
}

```

```

        if(total < 0){
            total = 0;
        }

        amountOwed.text = "You owe $" + rounding(total, 2);
        sumTotal.text = "Wallet Total = $" + rounding(walletTotal, 2);
    }
}

function incrementFiveDollar(e:MouseEvent):void{
    if ( total <= 0 ) {
        return;
    }

    walletFive = walletFive - 1;
    if(walletFive < 0 || walletFive > 1000){
        walletFive = 0;
    }else{
        FiveDollarText.text="Total = "+walletFive;
        walletTotal=walletTotal-5;
        total = total - 5;
        walletChange = total;

        if(total < 0){
            total = 0;
        }

        amountOwed.text = "You owe $" + rounding(total, 2);
        sumTotal.text = "Wallet Total = $" + rounding(walletTotal, 2);
    }
}

function incrementTenDollar(e:MouseEvent):void{
    if ( total <= 0 ) {
        return;
    }

    walletTen = walletTen - 1;

```

```

if(walletTen < 0 || walletTen > 1000){
    walletTen = 0;
}
else{
    TenDollarText.text="Total = "+walletTen;
    walletTotal=walletTotal-10;
    total = total - 10;
    walletChange = total;

    if(total < 0){
        total = 0;
    }

    amountOwed.text = "You owe $" + rounding(total, 2);
    sumTotal.text = "Wallet Total = $" + rounding(walletTotal,2);
}
}

function incrementTwentyDollar(e:MouseEvent):void{
    if ( total <= 0 ) {
        return;
    }

    walletTwenty = walletTwenty - 1;

    if(walletTwenty < 0 || walletTwenty > 1000){
        walletTwenty = 0;
    }
    else{
        TwentyDollarText.text="Total = "+walletTwenty;
        walletTotal=walletTotal-20;
        total = total - 20;
        walletChange = total;

        if(total < 0){
            total = 0;
        }

        amountOwed.text = "You owe $" + rounding(total, 2);
        sumTotal.text = "Wallet Total = $" + rounding(walletTotal,2);
    }
}

```

```

    }
}

function clearPayments(e:MouseEvent):void{

    walletTotal = savedwalletTotal
    walletOne = savedwalletOne;
    walletFive = savedwalletFive;
    walletTen = savedwalletTen;
    walletTwenty = savedwalletTwenty;
    OneDollarText.text="Total = "+walletOne;
    FiveDollarText.text="Total = "+walletFive;
    TenDollarText.text="Total = "+walletTen;
    TwentyDollarText.text="Total = "+walletTwenty;
    total = tempTotal;
    sumTotal.text= "Wallet Total = $" +rounding(walletTotal,2);
    statusValue.text = "";
    amountOwed.text = "You owe $" + rounding(total, 2);

}

function clearComplete(e:MouseEvent):void{

    walletTotal = savedwalletTotal
    walletOne = savedwalletOne;
    walletFive = savedwalletFive;
    walletTen = savedwalletTen;
    walletTwenty = savedwalletTwenty;
    OneDollarText.text="Total = "+walletOne;
    FiveDollarText.text="Total = "+walletFive;
    TenDollarText.text="Total = "+walletTen;
    TwentyDollarText.text="Total = "+walletTwenty;
    sumTotal.text= "Wallet Total = $" +rounding(walletTotal,2);
    amountOwed.text = "You owe $" + rounding(total, 2);

}

/*
 * Function that checks the payment and items purchased to move to the
 * confirmation page.
 */

function checkPayments(e:MouseEvent):void{

    var fail:uint = 0;
    var i:uint = 0;
    trace("wallettotal:" + walletTotal + " total:" + total);

    if(total <= 0){

        if(bus == 1){

```

```

        for(i = 0; i < toGetArrayTemp.length; i++){
            if(toGetArrayTemp[i] < 0){
                fail = 1;
            }
        }
    }

    if(fail == 0){
        trace("Payment correct" + bus);
        if(bus == 0){
            gotoAndStop(7);
        }
        else {
            gotoAndStop(6);
        }
    }
    else{
        statusValue.text = "Wrong groceries, please
return to the store";
        trace("Incorrect groceries");
    }
}

else{
    trace("Payment incorrect");
}
}

/*
* Function checks whether you came for the bus fare or to pay
* for items from the store.
*/

```

```
function checkWhere(e:MouseEvent):void{
    trace("checking for bus " + bus);
    if(bus == 0){
        gotoAndStop(3);
    }
    else{
        gotoAndStop(4);
    }
}
```

```

/*
 * Frame 6: This frame is the confirmation page that
 *         is displayed when the correct items and
 *         payments are done, also shows functionality
 *         of change that is recieved back.
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop fucntion.
 */

stop();

/*
 * Button to function assignment dependent on event.
 */

goHomeButton.addEventListener(MouseEvent.CLICK, destcheck);

/*
 * This is assigning the style of the text and buttons.
 */

goHomeButton.setStyle("textFormat", tf);
changeTotal.setStyle("textFormat", tf);
labelTwenty.setStyle("textFormat", labeltf);
labelTen.setStyle("textFormat", labeltf);
labelFive.setStyle("textFormat", labeltf);
labelOne.setStyle("textFormat", labeltf);
labelQuarter.setStyle("textFormat", labeltf);
labelDime.setStyle("textFormat", labeltf);
labelNickel.setStyle("textFormat", labeltf);
labelPenny.setStyle("textFormat", labeltf);

/*
 * Variable assignment.
 */

walletChange = walletChange * -1;
changeTotal.text="You Should Recieve $" + rounding(walletChange, 2) + " In
Change";
var tempChange:Number = walletChange;
var changeArray:Array = new Array();
var twenties:uint = 0;

```

```

var tens:uint = 0;
var fives:uint = 0;
var ones:uint = 0;
var quarters:uint = 0;
var dimes:uint = 0;
var nickles:uint = 0;
var pennies:uint = 0;
var changeNames:Array = ["twenty.png", "ten.png", "five.png",
"one.png", "quarter.png", "dime.png", "nickle.png", "penny.png"];

/*
 * Function that checks whether to display change or not since the bus
 * does not give change back.
 */

function destcheck(e:MouseEvent):void{

    if(bus == 0){

        gotoAndStop(7);

    }

    if(bus == 1){

        gotoAndStop(1);

    }

}

/*
 * Function that displays the correct change from the payment page.
 */

displayChange();

function displayChange():void {

    makeChangeCount();
    displayChangeCount();

}

/*
 * Function that calculate the amount of change.
 */

```

```

function makeChangeCount(){

    twenties += tempChange / 20;
    changeArray[0] = twenties;
    tempChange = tempChange % 20;
    tens += tempChange / 10;
    changeArray[1] = tens;
    tempChange = tempChange % 10;
    fives += tempChange / 5;
    changeArray[2] = fives;
    tempChange = tempChange % 5;
    ones += tempChange / 1;
    changeArray[3] = ones;
    tempChange = tempChange % 1;
    quarters += tempChange / .25;
    changeArray[4] = quarters;
    tempChange = tempChange % .25;
    dimes += tempChange / .1;
    changeArray[5] = dimes;
    tempChange = tempChange % .1;
    nickles += tempChange / .05;
    changeArray[6] = nickles;
    tempChange = tempChange % .05;
    pennies += tempChange / .01;
    changeArray[7] = pennies;

}

/*
 * Function that displays the change in bill format with the correct
 count.
 */

function displayChangeCount(){

    if (changeArray[0] > 1)
    {

        twentyDollarButton.visible = true;
        labelTwenty.text = changeArray[0] + " Twenties";

    }

    if (changeArray[0] == 1)
    {

        twentyDollarButton.visible = true;
        labelTwenty.text = changeArray[0] + " Twenty";

    }

    if (changeArray[0] == 0)
    {

```

```

                twentyDollarButton.visible = false;
                labelTwenty.text = "";
            }
    if (changeArray[1] > 1)
        {
            tenDollarButton.visible = true;
            labelTen.text = changeArray[1] + " Tens";
        }
    if (changeArray[1] == 1)
        {
            tenDollarButton.visible = true;
            labelTen.text = changeArray[1] + " Ten";
        }
    if (changeArray[1] == 0)
        {
            tenDollarButton.visible = false;
            labelTen.text = "";
        }
    if (changeArray[2] > 1)
        {
            fiveDollarButton.visible = true;
            labelFive.text = changeArray[2] + " Fives";
        }
    if (changeArray[2] == 1)
        {
            fiveDollarButton.visible = true;
            labelFive.text = changeArray[2] + " Five";
        }
    if (changeArray[2] == 0)
        {
            fiveDollarButton.visible = false;
            labelFive.text = "";
        }
}

```

```
if (changeArray[3] > 1)
{
    oneDollarButton.visible = true;
    labelOne.text = changeArray[3] + " Ones";
}

if (changeArray[3] == 1)
{
    oneDollarButton.visible = true;
    labelOne.text = changeArray[3] + " One";
}

if (changeArray[3] == 0)
{
    oneDollarButton.visible = false;
    labelOne.text = "";
}

if (changeArray[4] > 1)
{
    quarterButton.visible = true;
    labelQuarter.text = changeArray[4] + " Quarters";
}

if (changeArray[4] == 1)
{
    quarterButton.visible = true;
    labelQuarter.text = changeArray[4] + " Quarter";
}

if (changeArray[4] == 0)
{
    quarterButton.visible = false;
    labelQuarter.text = "";
}

if (changeArray[5] > 1)
{
    dimeButton.visible = true;
    labelDime.text = changeArray[5] + " Dimes";
}
```

```

    }

    if (changeArray[5] == 1)
    {

        dimeButton.visible = true;
        labelDime.text = changeArray[5] + " Dime";

    }

    if (changeArray[5] == 0)
    {

        dimeButton.visible = false;
        labelDime.text = "";

    }

    if (changeArray[6] > 1)
    {

        nickleButton.visible = true;
        labelNickel.text = changeArray[6] + " Nickels";

    }

    if (changeArray[6] == 1)
    {

        nickleButton.visible = true;
        labelNickel.text = changeArray[6] + " Nickel";

    }

    if (changeArray[6] == 0)
    {

        nickleButton.visible = false;
        labelNickel.text = "";

    }

    if (changeArray[7] > 1)
    {

        pennyButton.visible = true;
        labelPenny.text = changeArray[7] + " Pennies";

    }

    if (changeArray[7] == 1)
    {

        pennyButton.visible = true;

```

```
        labelPenny.text = changeArray[7] + " Penny";
    }
else
{
    pennyButton.visible = false;
    labelPenny.text = "";
}
}
```

```

/*
 * Frame 7: This frame is the second bus page which
 *         is displayed once the bus fare is paid
 *         and allows the user to move to either
 *         the grocery store or the bank.
 */

/*
 * Function used to make sure the user stops that this page when
 * using the gotoAndStop function.
 */

stop();

/*
 * Variable assignment.
 */

bus = 1;

/*
 * Button to function assignment dependent on event.
 */

bankButton.addEventListener(MouseEvent.CLICK, swapSceneBank);
exitButton.addEventListener(MouseEvent.CLICK, swapSceneExit);
groceryStoreButton.addEventListener(MouseEvent.CLICK,
swapSceneGrocery);

/*
 * This is assigning the style of the text and buttons.
 */

bankButton.setStyle("textFormat", tf);
exitButton.setStyle("textFormat", tf);
groceryStoreButton.setStyle("textFormat", tf);

```