**CPSC 381  
12/9/2019  
Assignment 5  
Final Project Paper  
Sami**   
  
 My contributions during the project was basically that of the algorithm designer, reviewer and implementer. I lead my team throughout the project and guiding each function they created while still ensuring that each part would properly coincide with the other. By the time each function and class was created, I went through all the code and combined it into one fully functional program. A good portion of the work I did went into debugging algorithm design and debugging after compiling the code. I am a firm believer that if you cannot write the code yourself, you should not use a library unless you understand the core mechanics behind it. That being the case I put a large emphasis internally to write every aspect of the program we required to perform the task. As the Team Leader, to put into context the amount of work I feel each member did, I would say I(Sami) completed roughly 40%, Erick 25%, Ethan 25% and Frank 10% of the coding. Since we focused on the delegation of work between myself, Erick, and Ethan, well agreed that Frank would be better suited for research, documentation, and commenting of the code to better keep track of the progress as well.

**Erick**

To create the AI of Majeks Currency we used a predictive model. This model uses changes in the exchange rate of other currencies to predict an increase or decrease in the exchange rate of Majeks Universal Exchange, or MUX. Unfortunately, we didn’t have access to API calls to get the real exchange rates for our currencies. Instead, we made our own random currency exchange rate generator. The main part of our AI is the scaling values module. Scaling values uses the new exchange rate of a currency to calculate the difference it has on MUX.

The random generator first generates a small random change between 0 and .005, then it is given a certain probability of being negative and positive. For our demonstration we decided to keep this at 50/50. This random value represents a change in the exchange rate of a currency.

Our scaling values module uses the value provided by the random generator to calculate the difference it has on MUX. It uses a scaling function to calculate the predicted impact each change has on the exchange rate of the MUX. The scaling function returns a value between -1 and 1. This value represents how much long term impact the change had on the exchange rate of MUX. The function is sensitive to how many times a positive and negative change occurs. We call these consecutive positive or consecutive negative numbers hits. The more hits we have in the same direction, the more certain we are that the exchange rate of MUX will go in that direction. Our scaling function uses this with the actual magnitude of the change to come up with its prediction.

Since we are not certain that the predictions are correct, we divided the decision-making process to 9 levels. Depending on the value returned by our predictive AI, we can suggest a user if it is good to invest in the MUX. This decision is based on the probability that the exchange rate of the MUX will go up or down. In general, it is a good idea to invest when the probability of gaining a profit is high.

**Frank**

We originally were going to have API integration for our program, but as we did more and more research we realized that it would require us to spend money to acquire the API so we decided to do as much as we can without it. When I was doing research for the API integration we were thinking about using an API for real time exchange rates for over 100 currencies. We could’ve done it every 10 minutes or every hour if we decided to do it. Since we couldn’t integrate the API we decided to just locally assign the currencies. So instead of fetching the data from an API we would just fetch it locally and then we would tell the AI how many times to run it and then it would determine if our universal currency would go up or down.

By consisting telling the AI to keep running it allows the AI to truly identify whether or not the currency is increasing or not because it allows for more sample size from all the data it gathered which allows it to accurately assess the currency and tell the user whether or not they this is the time to invest in it. Ideally we would have loved to have an API constantly running tests on a daily basis, but being able to test it as many times as we wanted is good for now since we can allow the AI to run it a bunch of times.  
 **Ethan**  
  
When creating the Currency class to work with MyMajeks program, we needed several requirements to make it work. We needed access to data, a way to manipulate that data, implement the AI to evaluate our data, and the allow the user to manipulate or use the data.

Since we were working with Java, we needed to make a structure to store sets of data into. The equivalent of that is a class in Java, so we stored three data types in it: Currency name, Currency ISO, and Currency value. We can recall and manipulate data this way. We stored each currency as a node in a linked list. Originally, we had planned to pull the data from an online source so that the data would be accurate and update frequently. However, that would cost money per access. To simulate this, we stored the data locally so that we can manipulate it as we see fit. To work with our AI, we created a universal currency based off the top five currencies based originally off the dollar conversions. Our program would average it a store it in our data as a Universal currency.

We also created the RunIterations function that would take in a linked list and randomly increase and decrease the values in the table based on how many iterations we implemented. The users have the ability to determine how many iterations the program needs to change our data. These iterations would be stored locally and will be accessed when we need to create new iterations to help with our AI. The more iterations that are created, the more accurate our AI is, but large numbers exponentially increase processing time. Also, as a preventative measure, values were not allowed to be negative, so they were either increasing or stayed at a default zero.

Outside of our original functions, the users have options to alter our data. The CompareValues function specifically recalls two specific currencies for comparison. This allows users to see the value of currencies based off their local currency or any other currencies. However, if the user wants to change their own value for the universal currency, they can manually input their own value. This is done through our AddCurrency function which takes in a linked list and the Currency class data types. Doing so will manipulate the iterations and data that our AI will use.

Finally we do have an error checking function named CheckList. It will take in the list we used and check if it is valid and within our parameters. This is returned as a Boolean value.