**Vedic**

**Concept of Operations**

Team Name: Vedicoders

Team Members:

* William Funk
* Jorge Rodriguez
* James Vinson

Modification history:

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | Comment |
| v0.0 | 02/14/2016 | W. Funk | Initial Plan |
| v1.0 | - | - | - |
| ... |  |  |  |

Contents of this Document

The Current System

* Unreal Engine 4’s VR Game Creation Software
* Node-Based GUI in Unity 3D
* “A Framework for Design and Implementation of Virtual Languages” by Ando Saabas

The Proposed System

* Needs
* Users and Modes of Operation
* Operational Scenarios
* Operational Features
* Expected Impacts
* Analysis

**The Current System**

There exists no current system that performs the specific functionality that VEDIC will provide. However, there are examples of software that possess some of the peripheral functionality that VEDIC will incorporate. Among the list of those software examples are: Unreal Engine 4’s VR game-making software currently in development, Node-based GUI creation in Unity 3D, and a master thesis by Ando Saabas on how to implement visual languages.

Unreal’s VR programming environment uses two hand-held controls that translate into two laser pointers. A user can move and manipulate 3D objects in a gaming environment much like the 2D version does with a mouse pointer. This software is still under development, and only demo videos are available as examples of how it performs. From those videos, it doesn’t appear to add any additional function or intuitive value for the programmer that the Unreal Engine 4 doesn’t already offer in the 2D editor. The relationship between these objects is still hidden from view in three-dimensional space, while the node representation available to account for this is still in the 2D style.

Unity’s ability to customize a node-based GUI helps create tools and controls for the user in 3D space. This presents only a fraction of the functionality that VEDIC will give, but may serve to expedite development time by applying it to the VEDIC system.

The master thesis detailing how to translate an object-oriented language--such as Java--into a visual language, provides a conceptual blueprint on how VEDIC might go about designing its transpiler from typical C# (which is similar to Java in many ways, and object-oriented) into VEDIC’s virtual environment, and back to C# again.

**The Proposed System: Needs**

The main component that VEDIC will require is access to a virtual environment. This will require an Oculus Development Kit 2.0 as this is the latest technology that is still compatible with all three of our developers’ computers. It exceeds the first version by reducing the “screen door effect” and avoiding motion sickness, which is usually due to a lower frame-rate.

To make this project feasible within the proposed deadline, it will be necessary to employ a graphics engine, rather than building one from scratch. Both the Unity 5 and Unreal 4 engines have VR capability and a large support network. However, all three developers have had some experience with Unity and none with Unreal, making Unity ideal for a shorter ramp-up time. For now, Unity will serve as VEDIC’s development environment. It’s free version also makes it ideal for the group’s narrow budget.

Later on, VEDIC will require additional hardware for user input. The available options include the Myo Band, Leap Motion control, and XBOX/PS4 controller. At the start, these components won’t be necessary until a later phase of the project.

**The Proposed System: Users and Modes of Operation**

Users of the VEDIC system will typically be programmers working with C#, though Java could easily be a later edition. These software developers would range anywhere from the novice, learning to program in an object-oriented language, to the experienced developer working on a complex project that involves numerous files and classes.

VEDIC will have a total of four modes of operation: the viewing of a database (SQL), the creation or manipulation of a database (SQL), the viewing of a program with its multiple files (C#), and the creation or manipulation of a program (C#).

In the viewing of a database mode, the user will see a multitude of different shapes of varying colors, some interconnected, some stand-alone. These will consist of SQL tables with distinct sections representing columns within these tables. Both tables and columns will be labeled and their colors will be specific to the type of variable they contain (integer, float, char, etc.). Colored pipes or wires will connect these tables where relationships exist (joinings, shared variables, etc.). The number of rows in the table will either exist as a number adjacent to the table’s title, or in the relative height of the object in scale with the other table shapes. User will be able to rotate the scene or a specific table, expand a table for better view of its individual columns and possible list of row contents, zoom in and out, explode scene (increase distances between objects), implode the scene (decrease distances between objects), and move through the scene in a flying-style of motion.

The creation and/or manipulation of databases mode will have user seated at a work-bench. On a wall directly in front of the desk/bench, and within arms-reach, will be a series of shelves with cabinets, bins, drawers, or boxes labeled with the corresponding object(s) they contains with the image of the representative shape(s) on the front (ie. a cyan-colored, flat, rectangular cube representing a column that contains an integer type). On the desk/work-bench in front of the user will be the complex object they are building or manipulating (much like a 3-dimensional circuit diagram). This object/SQL-table will hover above the work-bench and can be connected using specifically colored wires/pipes to other tables and their columns. User can change between view and edit modes at will. This mode will comply with the standard C.R.U.D. array of database operations: database tables can be Created, data can be Read from rows by columns (more easily in the view mode). Tables, columns, and rows can be Updated while any of those can be Deleted, as well. The system will log these in a series of transactions and committed to the database as a batch when user elects to do so.

Third VEDIC mode is the viewing of a C# project. This mode will contain colored shapes and connections similar to the viewing databases mode. The difference will be the number of different shapes and colors as a programming project has a more diverse set of components than that of an SQL database. This view will share many similarities with a UML diagram, both structural and behavioral, with a third dimension to better illustrate the complexity and interaction that the objects possess.

The fourth and final VEDIC mode of operation is the creation and/or manipulation of C# files and classes within a project. Similar to the create/edit mode for databases, there will be many more shapes, colors, and connections to add into the 3D circuit floating above the work-bench. These classes, their functions and variables, can be transpiled into an export format when the user elects to do so, and converted to C# files in a project folder for later compiling by user outside of the VEDIC system.

VEDIC software will ultimately be free to use under the MIT license. It is this group’s belief that freely offering the software will promote its use, and bolster its circulation in a shorter amount of time.

**The Proposed System: Operational Scenarios**

Features Available:

Most of the features have been mentioned in earlier sections, but can be found in greater detail below:

Recipe suggestion is the key component to this system. This somewhat random meal chooser will use an algorithm fueled by a taste profile and any health considerations entered in by user, to offer anywhere from individual mealtime suggestions to a coordinated meal plan. Calories can be tracked, and nutritional information offered at a single click.

Health advice is offered in general to users if that is something they elect to have, or it can be more specific based on a condition mentioned in the user’s profile. This data could directly affect which recipes are chosen. A person with high cholesterol probably won’t receive a suggestion for a red meat recipe with a side of buttery shrimp scampi. A person with a peanut allergy will have a lower chance of getting a suggestion for deconstructed peanut butter pie. Disclaimers will be in place to inform all users that it is impossible to completely eliminate conflicts with their health problems, and caution is advised with every meal.

Online grocery orders will be funneled through the system, linked with a list of ingredients for the chosen recipe, and can be automatically ordered if user entered in their access info to a cooperating site such as is available with Walmart.

Ingredient inventory tracker. If user chooses to track the ingredients they already possess (to avoid purchasing more than what is needed), they can manually enter in what they have or use the barcode reader available through mobile (UPC number via web-based version).

Social networking will be used (ie. Facebook) to connect users with a wider community where supportive posts can be made or received to help users achieve their goals. This would be done through existing API available with these third-party systems, or made from scratch if one of the versions (Android) requires a more custom approach.

Typical Scenarios:

User requests a recipe at random. They are given a recipe based off of taste profile, and filtered by health conditions. They are shown an image of the finished product, a list of ingredients, and a series of instructions on how to prepare the meal. The user is offered the chance to purchase the ingredients online or print out the recipe info package to a local printer.

User requests recipes for anywhere from a whole day to an entire week. They are given a varied list of recipes based on taste profile, filtered by health conditions, and also filtered to ensure recipes are not repeated during the week to avoid redundancy. The same offers would be made as in the individual recipe request mentioned above.

User posts a recipe using the form template provided. Their input is verified and incorrect information alerts the user and waits for correction. This will store the proposed recipe into the database, where it will be labeled as untested, and awaits the bold user to try it. Once it has been rated a minimum of three times, it will be released into the main recipe collection for common use. User will be rewarded with points for their contributions as will those who used and rated it.

User posts the need for support through social media connection. The post is processed, and filtered for obviously negative content. After successful filter, it is posted on user’s registered media networks. Responses from “friends” and fellow Phoodies will be scraped, formatted, and placed in their social network folder inside Phood Buddy (this is a “Would Like To Have” feature).

Atypical Scenario:

User requests a recipe containing conflicts with their health profile. Disclaimers have already been read and agreed to via automatic hard-coding for all users. However, user notices issue and posts to support using the available “report error” feature. Programmer will inspect recipe manually, comparing it against list of medical issues listed by user. If conflict exists, corrections will be made to filtering criteria. If issue is due to user error (ie. didn’t actually enter that health data into profile, or misspelled, etc.), then user will be notified of this mistake and informed on how to rectify it for future use.

User receives erroneous health tips (bad input). The user will have been warned via hard-coded disclaimer and agreed to it. They can notify us via “report error” feature and Programmer will verify if concern is legitimate or user is misinformed. If system error, filter will be corrected. If user error, user will be notified and additional resources provided for supplemental health education.

User tries to order ingredients to a recipe via the system, but order won’t go through. User will be offered a chance to re-enter their sign-in information for the third-party, online grocery order application. If that fails to rectify the situation, user is given the chance to open third-party site in a different browser tab where they can manually enter their order using list provided by Phood Buddy recipe. “Report error” feature will alert programmer to issue, and will be subsequently resolved.

User is told they don’t have the ingredients needed for a recipe, though they are certain their stock is full. “Report error” feature alerts programmer. The database is checked and tested to ensure there is no data loss. If database is functioning within parameters, programmer will check to see if error notifications are being made to user (whether software related or connection related) when an input to database (barcode or UPC entry) is made by user. If this doesn’t resolve concern, programmer shall try to use similar device as that of user’s to repeat issue. Further investigation is required beyond this point.

**The Proposed System: Operational Features**

Must Have:

* Recipe Suggestion
* Recipe Ratings
* Recipe Posting
* Health Tips
* Online Grocery Orders
* Ingredient Inventory Tracker

Would Like to Have:

* Social Support Networking
* Advertising
* Point Reward System

**The Proposed System: Implementation**

This system will be developed on three separate platforms: Android and Windows Mobiles, and Web-based. In their respective orders, the languages used will be Java, C++, C#, JavaScript, PHP, SQL. This system will use a Parse database backend. For the most part, business logic will be conducted on client-side, while a large amount of user data will be stored server-side.

This system will require an internet connection to use (especially the Web-based version) as most of the user data is located server-side, but also because many of the features are accessing online resources (ie. Online grocery purchases). The Android and Windows Mobile developers currently have some experience with their respective platforms, but a short learning period will still be required. The Web-based languages are well known to those working in this section, and the learning period should be minimal.

This system will be both mobile and stationary via mobile, tablet, and desktop/laptop. For now, this application will be specific to a North American audience, though it will be accessible to users worldwide. The cultural and language focus, however, will be for US and Canada.

The disadvantages to both Android and Windows mobile platforms consist of difficult development periods and a rapid upgrade pace. There are many versions of the Android mobile platform floating around and any serious app must try to accommodate as many of them as possible. Windows mobile has issues related to its app store and limited developer support network due to the mere 6% worldwide user base. The advantages to using these platforms are a rich cross-platform design, and Androids 60+% user base worldwide. Our Web-version is meant to capture the remaining market of users whether without smart phones or iO-based. As this platform has been around for ages, the disadvantages are anticipated to be minimal.