**V.E.D.I.C. – Virtual Environment for Developing Interactive Code**

**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 (http://motherboard.vice.com/read/developers-will-soon-code-vr-games-inside-virtual-reality-epic-games-unreal-editor?utm-source=mbtwitter), Node-based GUI creation in Unity 3D (https://www.udemy.com/creating-a-node-based-editor-in-unity-3d/), and a master thesis by Ando Saabas on how to implement visual languages (https://cocovila.github.io/files/documentation/saabas\_thesis.pdf).

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:

It is essential that the user be able to “move through” the viewing environment (database or file system). Initially this will be performed with two keyboard keys for backward and forward movement, two more for ascending and descending, motion-tracking of the user’s head will adjust the facing direction, while the mouse-pointer and button will select/grab 3D objects. Later in the project’s life cycle, additional user input devices will give the user greater control over the environment (Myo Band, Leap-Motion Control, Xbox/PS4 controller, voice control, etc.).

User can create tables or classes by selecting relevant objects from the 3D node-based GUI, and placed in the scene. Column, function, and variable objects can be selected and brought into the scene, to be inserted into larger objects (building the complexity of these objects by encapsulation).

Similarly, user can edit and update existing objects by selecting them, opening them, and manipulating their components. If deletion of an object is desired, they have only to move the object over to a virtual trach bin available at the side of their work space.

A virtual keyboard will be present where labels and values can be entered and attached to the virtual objects. Later, audio input may expedite this process.

A new transpiler, unique to this project, will be able to convert a JSON string to a series of virtual objects. This same transpiler will also convert those same virtual objects into a series of SQL queries (JSON string format).

Another transpiler will take C# files (classes, interfaces, etc.) and convert them into a series of virtual objects. This same transpiler can convert the virtual components back into C# files, where they will be packaged for export.

VEDIC will have a basic warning and error checking system built-in. Much of this will involve the restriction of number and type of objects available to the user. Otherwise, the user will be alerted visually of any invalid actions or import compatibility flaws.

Typical Scenarios:

* User enters database connection information, and calls import. The scene is populated by the tables from user’s database. These shapes (cubes) are suspended in air, a rainbow of colors marks the number, type, and order of the columns in each table. Vibrantly colored pipes connect a few tables here and there, where the “user\_id” column (cyan-colored) of one table is connected to the “user\_id” (also cyan-colored) column of another table by a pipe (carnation red) signifying it’s the same variable in both tables. User ascends to one of the tables with the press of a keyboard key and a turn of the head, selects and expands its list of rows and the data contained within each column by point-and-clicking the mouse while dragging.
* User sits at the virtual work-bench, a hovering cube slowly rotates on the surface directly in front. Several colored slices mark the different columns and their valid data types. From the shelf in front, the user selects a flat, rectangular cube, pink in color (Date type), and slides it between the hovering cube’s cyan-colored slice (integer type) and the yellow slice (char type). Using the virtual keyboard, the user labels the new column as “PayDate” and selects the “Submit” GUI button at the corner of their work-bench. The Update operation is transpiled into an SQL query and sent to the database.
* User imports a C# project from an accepted source location. The scene is populated with numerous shapes and a network of colored wires. User approaches an orange cylinder (Stack for integer values), selects it, and watches the map of wires glow brightly, leading the eyes to all the other objects related to the orange stack cylinder. The user can immediately determine the type of connections based on the color of those wires. A canary yellow wire connects to a peach-colored cube (interface class) denoting the stack was a mandatory implementation of the interface, where a bright green wire leading to a silver cube denotes the class that did the implementing.
* User sits at work-bench, and selects a peach-colored cube from the shelf (interface class) and places it to hover above the bench in front. A few keys on the virtual keyboard allow the user to label the interface. The user takes a purple cylinder from the shelf (stack of double type variables) and sets it into the peach cube’s private quadrant, thereby selecting its level of protection. Another few keys on the virtual keyboard label the cylinder. User repeats the routine with a gold-colored atom shape, the number of nodules in the nucleus equaling the number of parameters it takes (colors denoting the expected variable types of those parameters), the electron-like nodule and its color depicting the return type (black == void). Once complete, the user places the completed cube in an outbox on their workbench. A modal appears with a UML-style description of the interface class and its variables/functions. User clicks the “submit” GUI button in the corner of their work-space and the cube is added to the project.

Atypical Scenario:

* User sits at their virtual work-bench, a silver-colored cube (class) hovers in front of them. With a click and drag of the mouse pointer, the user connects a canary yellow wire (implements) to the cube and tries to attach it to an existing peach-colored cube (interface class). Immediately the yellow wire flashes in slow rhythmic pulses to indicate there is a warning/error. A purple cylinder (stack of type Double variables) inside the peach-colored cube glows a little brighter. User is thus alerted to the fact that they must implement that interface variable to have a valid implementation. To rectify the error, user clicks the glowing purple cylinder with the mouse pointer, and a duplicate cylinder appears inside their silver cube, awaiting a label, which the user enters using the virtual keyboard.
* User enters database connection information and selects import. Nothing happens. A modal appears in the form of a red sphere, slowly rotating; silvery text on its surface informs the user that a connection could not be made with the database, and offers a list of possible solutions to this problem, to include verifying the accuracy of the connection information entered.
* After importing a C# project into the VEDIC environment, the system uses the files to populate the scene. Ten objects become fifty, fifty become two-hundred, two-hundred become a thousand, and suddenly the objects vanish. Another red, error sphere appears. The silvery text informs the user that the hardware they are running the VEDIC environment on, can’t support the number of 3D objects they are trying to import. They must either upgrade their system (graphics card), or import fewer files.

**The Proposed System: Operational Features**

Must Have:

* Ability to receive data from local and remote databases.
* Parse JSON strings into expected component types.
* Ability to automate generation of 3D objects from data string.
* Need to recognize, visually, all the components involved in the database through colors, shapes, and labels.
* Need to navigate through the scene, and interact with objects using all three dimensions.
* Has to allow alteration of tables, columns, and data contained within individual rows.
* Capable of importing and interpreting C# files.
* Automated generation of 3D objects from those C# files.
* Must be able to navigate in all three dimensions within the scene, interacting with the various shapes and colors created from the imported C# files.
* Ability to create new, edit or delete old, classes and their functions/variables.
* Must be able to convert 3D objects into C# files and package them for export.

Would Like to Have:

* Compatible with multiple user input devices (Myo Band, Leap-Motion control, Xbox/PS4 controller, audio, etc.).
* Ability to compile inside VEDIC environment.
* Ability to execute programs inside VEDIC environment.
* The creation of a database within VEDIC environment.
* Can add entire row content entries into database.
* Ability to create new tables from two or more table joins/unions.
* Show object instantiation of program to demonstrate size and scope of project.
* Include a separate runtime environment for C# programs.

**The Proposed System: Implementation**

The system will focus mostly on a Windows desktop platform, but may eventually support an Android mobile environment, depending on available technology. User’s computer must have internet connection to import from a remote database, and a local installation of a C# compiler if they wish to compile and run their C# project.

User will need equipment to operate in virtual reality. Any device equal to, or more advanced than, the Oculus Development Kit 2.0 should be capable of running the VEDIC environment. The computer running the virtual space needs to have at a minimum: Windows 7 or better, 2 USB 2.0 ports (at least one must be powered), and a dedicated graphics card at the level of or better than the Nvidia GTX 600 series or AMD Radeon HD 7000 series, with DVI-D or HDMI graphics output. Ideally, the user’s system should be capable of running 3D games at 1080p quality at a framerate of 75frames/second or higher. This information and more can be located at <https://support.oculus.com/hc/en-us/articles/201835987-Oculus-Rift-Development-Kit-2-FAQ>

The user will need to have at least a rudimentary understanding of how to program in an object-oriented programming language. They will need at a minimum: a keyboard and mouse for user import. A C# compiler installed on their desktop would assist in the compiling of their programs, while possession and installation of a remote SQL database will make VEDIC’s full functionality available to the user.